Journal of Theoretical and Applied Information Technology
10th May 2013. Vol. 51 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

HIGH-DIMENSIONAL HIERARCHICAL OLAP : A PREFIX–INDEX HIERARCHICAL CUBING APPROACH

[1]KONGFA HU, [2]ZHE SHENG, [3]LING CHEN

[1] Prof., Department of Computer Science and Engineering, Yangzhou University, 225009,China
[2] Master , Department of Computer Science and Engineering, Yangzhou University, 225009,China
[3] Prof., Department of Computer Science and Engineering, Yangzhou University, 225009,China
E-mail: kfhu05@126.com

ABSTRACT

The pre-computation of data cubes is critical for improving the response time of OLAP(online analytical processing) systems and accelerating data mining tasks in large data warehouses. However, as the sizes of data warehouses grow, the time it takes to perform this pre-computation becomes a significant performance bottleneck. In a high dimensional OLAP, it might not be practical to build all these cuboids and their indices. In this paper, we propose a multi-dimensional hierarchical cubing algorithm, Prefix-index hierarchical cubing, based on an extension of the previous minimal cubing approach. This method partitions the high dimensional data cube into low dimensional cube segments. Such an approach permits a significant reduction of CPU and I/O overhead. Experimental results show that the proposed method is significantly more efficient than other existing cubing methods.

Keywords: *Data cube, High dimensional OLAP, Prefix-Indexing cubing*

1. INTRODUCTION

OLAP refers to the technologies that allow users to efficiently retrieve data from the data warehouse for decision support purposes [1]. A lot of research has been done in order to improve the OLAP query performance and to provide fast response times for queries on large data warehouses. A key issue to speed up the OLAP query processing is efficient indexing and materialization of data cubes [2,3,4]. Recently, many data cubing algorithms, such as BUC [5], H-cubing [6], quotient cubing [7], and star-cubing [8], have been proposed.

A key challenge for efficient data cubing is that, in large data warehouse applications, data usually has a high dimensionality (e.g. more than 100 dimensions) and each dimension has multiple hierarchy levels. Since data cube grows exponentially with the number of dimensions and number of hierarchy levels, it is generally too costly in both computation time and storage space to materialize a full high-dimensional data cube.Although some new algorithms, such as condensed cube [9], dwarf cube[10], or star cubes [8], can delay the explosion, they do not solve the fundamental problem[11]. The minimal cubing approach by Li and Han [11] can alleviate this problem, but it does not consider the dimension hierarchies and cannot efficiently handle OLAP queries. In this paper, we develop an efficient cubing algorithm that supports dimension hierarchies for high-dimensional data cubes and answers OLAP queries efficiently.

The proposed cubing algorithm has the following salient features. 1) It supports not only high-dimensional data cubes but also hierarchical data cubes with multiple levels in a dimension. 2) The decomposition of the data cube space leads to significant reduction of processing and I/O overhead for many queries by restricting the number of cube segments to be processed for both the fact table and bitmap indices. 3) The prefix bitmap index is designed to support efficient OLAP by allowing fast look-up of relevant tuples. 4) The proposed cubing algorithm supports parallel I/O, parallel processing, and load balancing among disks and processors.

2. SHELL CUBE SEGMENTATION

To illustrate the method ,a tiny warehouse, Table 1, is used as a running example.

For a cube of d dimensions, it will create 2d cuboids. If we consider the dimension hierarchies of each dimension, the cube will create $\prod_{i=1}^{d}(h_i+1)$ cuboids (where h_i is the number of hierarchy levels of dimension Di). For example, the cube in

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 1 has three dimensions: DimProduct, DimRegion and DimTime. The DimProduct dimension has three hierarchies as (Class,Item,Product), the DimRegion dimension has three hierarchies as (Country,Province,City),and the DimTime dimension has three hierarchies as (Year,Month,Day). Therefore, this cube with three dimensions will generate in total $\prod_{i=1}^{d}(h_i+1)=(3+1)*(3+1)*(3+1)=64$ cuboids such as {(Product,City,Day),(Product,City,Month),(Product,City,Year), (Product,City,All) ,..., (All,All,All)}. But in a high-dimensional warehouse, there is a substantial I/O overhead for accessing a fully materialized data cube.

Table 1. A Sample Warehouse

TID	DimProduct			DimRegion			DimTime			Measure	
	Class	*Item*	*Product*	*Country*	*Province*	*City*	*Year*	*Month*	*Day*	*Count*	*SaleNum*
1	Class1	Item1	Exploder	China	Jiangsu	Nanjing	2010	1	1	1	20
2	Class1	Item1	Exploder	China	Jiangsu	Nanjing	2010	1	2	1	60
3	Class1	Item1	Exploder	China	Jiangsu	Yangzhou	2010	1	2	1	40
4	Class1	Item1	Exploder	China	Jiangsu	Yangzhou	2010	1	3	1	20
...	...	...	...	...	...	...	...	...	...	...	...
367	Class1	Item1	Exploder	China	Jiangsu	Nanjing	2011	1	2	1	60
...	...	...	...	...	...	...	...	...	...	...	...

A partial solution which has been implemented in some commercial data warehouses is to compute a thin shell cube. For example, one might compute all the cuboids with 3 dimensions or less in a 30-dimension data cube. There are two disadvantages of this approach. First, it still needs to compute $C_{30}^3+C_{30}^2+C_{30}^1$ = 4525 cuboids if there is no hierarchies and it needs to compute 2h*($C_{30}^3+C_{30}^2+C_{30}^1$)=23*4525=36200 cuboids when each dimension has h=3 levels dimension hierarchies. Second, it does not support OLAP in a large portion of the high-dimensional cube space.

In this paper, we propose an orthogonal way to partition the cube space. We partition all the dimensions of a cube into subsets called the cube segments. For example, for a warehouse of 30 dimensions, D_1, D_2,...,D_{30}, we first partition the 30 dimensions into 10 Cube segments of size 3: (D_1,D_2,D_3), (D_4,D_5,D_6),...,(D_{28},D_{29},D_{30}). For each cube segment, we compute its full data cube. For example, in Cube segment(D_1,D_2,D_3),we compute the eight cuboids:{(D_1,D_2,D_3),(D_1,D_2,All),(D_1,All,D_3),(All,D_2,D_3), (D_1,All,All),(All,D_2,All),(All,All,D_3),(All,All,All)}. If we consider that each dimension of the 3-D cube (D_1,D_2,D_3) has three hierarchy levels as $D_1(L_1^1,L_2^1,L_3^1)$,$D_2(L_1^2,L_2^2,L_3^2)$, $D_3(L_1^3,L_2^3,L_3^3)$, we will compute 64 cuboids:{(L_3^1,L_3^2,L_3^3), (L_3^1,L_3^2,L_2^3) ,...,(All,All,All)}.

The benefit of this model can be seen by a simple calculation. For a cube of 30 dimensions without hierarchy, if we partition it into 10 segments, each with 3 dimensions, each segment will have 8 cuboids and there are only 8×10 = 80 cuboids to be computed. If each dimension has three hierarchy levels, then each segment will have 64 cuboids as shown above, and there are in total 64×10 = 640 cuboids to be computed. Comparing this to the 36200 cuboids needed by the shell cube technique, the savings in cubing time and space are significant.

Lemma 1. Given a warehouse of T tuples and d dimensions, the entire shell Cube segment will create $\sum_{i=1}^{f}C_f^i*\lceil d/f\rceil$ =($2^f*\lceil d/f\rceil$) cuboids and needs O(T* $\sum_{i=1}^{f}C_f^i*\lceil d/f\rceil$)=O(T* ($2^f*\lceil d/f\rceil$)) storage space, while the partial cube will create $\sum_{i=1}^{f}C_d^i$ cuboids and needs O(T* $\sum_{i=1}^{f}C_d^i$) storage space, and the full cube will create 2d cuboids and needs O(|T|*2^d) storage space.

Rational. In the shell Cube segment method, the cube partition into $\lceil d/f\rceil$ cube segments. For each cube segment will create C_f^1 cuboids of 1-dimension, C_f^2 cuboids of 2-dimension,..., C_f^f cuboids of f-dimension and one cuboid(All,All, ...,All). Thus each cube segments will create ($C_f^1+C_f^2+\cdots+C_f^f+1$) = $\sum_{i=1}^{f}C_f^i$ cuboids. So the entire shell Cube segment will create $\sum_{i=1}^{f}C_f^i*\lceil d/f\rceil$ =($2^f*\lceil d/f\rceil$) cuboids and needs O(T*$\sum_{i=1}^{f}C_f^i*\lceil d/f\rceil$)= O(T*($2^f*\lceil d/f\rceil$)) space.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

In partial cube, we select f dimensions from the d dimensions to create the partial cube . It will create C_d^1 cuboids of 1-dimension, C_d^2 cuboids of 2-dimension,..., C_d^f cuboids of f-dimension. Thus the f partial cube will create $C_d^1 + C_d^2 + \cdots + C_d^f = \sum_{i=1}^{f} C_d^i$ cuboids and needs O(T*$\sum_{i=1}^{f} C_d^i$) storage space.

In full cube, for each dimension *D*, the dimension of its aggregate cuboids is *D* or All. For every dimension $\{D_1,...,D_d\}$, the dimension of its aggregate cuboids is chosen form the 2-values $\{D_i$,All$\}$. So for the entire full cube, it will create $\prod_{i=1}^{d} 2 = 2^d$ cuboids and needs O(|T|*2^d) storage space.

Lemma 2. If we consider each dimension has h hierarchies, our prefix-index cubing method will create $\prod_{i=1}^{f}(h_i+1)*\lceil d/f \rceil$ =($(h+1)^f * \lceil d/f \rceil$) cuboids, while the minimal cubing method of Li's and Han's will create $\sum_{i=1}^{f} C_f^i (\sum_{j=1}^{h_i} C_{h_i}^j) * \lceil d/f \rceil$ =($2^{(f+h)} * \lceil d/f \rceil$) cuboids.

Rational. In prefix-index cubing method, each dimension D_i has hi hierarchies, the dimension hierarchies of its aggregate cuboids is chosen form the (h_i+1)-values { $L_1^i, L_2^i, \cdots, L_h^i$,All}. So it will create $\prod_{i=1}^{f}(h_i+1)*\lceil d/f \rceil$=($(h+1)^f * \lceil d/f \rceil$) cuboids.

In the minimal cubing method, the cube partition into $\lceil d/f \rceil$ cube segments. For each cube segments will create $\sum_{i=1}^{f} C_f^i$ cuboids for the f dimensions cube segments. For each dimensions of these cube segments have h_i hierarchies and create $\sum_{j=1}^{h_i} C_{h_i}^j$ dimensional hierarchy cuboids. So the entire minimal cubing will create $\sum_{i=1}^{f} C_f^i (\sum_{j=1}^{h_i} C_{h_i}^j) * \lceil d/f \rceil$ =($2^{(f+h)} * \lceil d/f \rceil$) cuboids.

3. PREFIX BITMAP INDEXING

As we will see, our multi-dimensional fragmentation permits eliminating some bitmaps, thus improving storage and access overhead. We propose this novel hierarchical encoding on each dimension table. The encoding is implemented through the assignment of a special surrogate key on each dimension table tuple, called dimension hierarchical encoding.We can create the DimRegion,DimTime and DimProduct dimension hierarchy encoding shown in Table 2,Table3 and Table4.

Table 2. Dimtime Dimension Hierarchy Encoding

TimeID	Year	Month	Day	B^{TimeID}
	yyy	mmmm	ddddd	yyymmmmddddd
1	2010	Jan	1	001000100001
2	2010	Jan	2	001000100010
3	2010	Jan	3	001000100011
...	...	...	...	...

Table 3. The Dimregion Dimension Hierarchy Encoding

RegionID	Country	Province	City	$B^{RegionID}$
	uuuuuuu	vvvvv	cccc	uuuuuuuvvvvvcccc
1	China	Jiangsu	Nanjing	0000001000010001
2	China	Jiangsu	Yangzhou	0000001000010010
...	...	...	...	...

Table 4. The Dimproduct Dimension Hierarchy Encoding

roductID	Class	Item	Product	$B^{ProductID}$
	gggg	aaaaa	ppppppp	gggaaaaapppppp
1	Class1	Item1	Exploder	00100001000000
2	Class1	Item1	Detonator	00100001000001
...	...	...	...	...

Lemma 3. Our prefix-index cubing method needs O(T*$(h+1)^f * \lceil d/f \rceil$)*$\lceil \log_2 m \rceil$/8) storage space, while the minimal cubing method of Li's and Han's needs O(T*($2^{(f+h)} * \lceil d/f \rceil$)* $\lceil \log_{10} m \rceil$) storage space.

Rational. In prefix-index cubing method , the member of each dimension needs $\lceil \log_2 m \rceil$ bits dimension hierarchical encoding. So the storage space of prefix-index cubing method needs O(T*($(h+1)^f * \lceil d/f \rceil$)* $\lceil \log_2 m \rceil$ /8) bytes. In the minimal cubing method , the member of each dimension needs $\lceil \log_{10} m \rceil$ integer indices. So the storage space of the minimal cubing needs O(T*($2^{(f+h)} * \lceil d/f \rceil$)$\lceil \log_{10} m \rceil$*) bytes.

By using dimension hierarchical encoding, we can register a list of tuples IDs (tids) associated with the dimension members for each dimension. For example, the TID list associated with the DimProduct , DimRegion and DimTime dimension are shown in Table 5, Table 6 and Table 7 in turn.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

To compute a data cube for this database with the measure avg() (obtained by sum()/count()), we need to have a tid-list for each cell: {tid1,..., tidn}. Because each tid is uniquely associated with a particular set of measure values, all future computations just need to fetch the measure values associated with the tuples in the list. In other words, by keeping an array of the ID-measures in memory for online processing, one can handle any complex measure computation.Table 8 shows what exactly should be kept, which is substantially smaller than the database itself.

Table 5. Dimproduct Dimension TID

$B^{ProductID}$	TID List
0001000010000001	1-2-3-4-367
...	...

Table 6. Dimregion Dimension TID

$B^{RegionID}$	TID List
0000001000010001	1-2-367
0000001000010010	3-4
...	...

Table 7. Dimtime Dimension TID

B^{TimeID}	TID List
001000100001	1
001000100010	2-3
001000100011	4
...	...

Table 8. TID- Measure Array Of Table 2

tid	Count	SaleNum
1	1	20
2	1	60
3	1	40
4	1	20
...	...	...

4. PARALLEL HIERARCHICAL AGGREGATION ALGORITHM

4.1 Parallel Construction of Shell Cube Segments

The data cube can be distributed across a set of parallel computers by parallel constructing the Cube segments. Therefore, for the end-user and other potential applications, we consider this data cube as one large virtual cube, which is distributed across a set of parallel computers, which manage the creation, updates and querying of the associated cube portions. To develop appropriate scheduling mechanisms for these management tasks, we consider that the virtual cube is split into several smaller parts, called Cube segments. But a Cube segment could furthermore also be split into smaller segments and so on, till we achieve the level of chunks. They can then be assigned to parallel computers, having sequential or parallel computing power, which are responsible for their management. The algorithm for shell prefix cube segment parallel computation can be summarized as follows.

Algorithm 1 (Parallel computation of cube segments)

Input: A base cuboid BC of n dimensions:(D_1; ... ;D_n).

Output: (1) A set of Cube segment partitions {P_1;..., P_k} and their corresponding (local) Cube segments {CS_1; ... ; CS_k}, where P_i represents some set of dimension(s) and $P_1 \bigcup ... \bigcup P_k$ are all the n dimensions, and (2) an ID measure array if the measure is not tuple-count such as {sum, avg}.

{ partition the set of dimensions :(D_1; ... ;D_n) into a set of k Cube segments {P_1;..., P_k};

scan base cuboid BC once and do the following with parallel processing

{ insert each <tid, measure> into ID-measure array;

for each attribute value a_i of each dimension D_i

build an dimension hierarchy encoding index entry: <*B*: TID list>;}

parallel processing all segment partition P_i as follows

build a local Cube segments CS_i by intersecting their corresponding tid-lists and computing their measures;}

We can parallel construct the high dimensional cube with the Cube segments parallel construction. The system architecture of these shell Cube segment parallel construction is shown in Figure 1.

The Cube *Constructor* reads one tuple after the other, passes over the items to the index warehouse, retrieves its global index and then passes the (raw) measure and its associated global index to the data cube structure. The *Querying Cube operator* is some kind of highly sophisticated, recursively nested loops for aggregation of measures. Because the number of computational operations of nested aggregation depends on the size of the dimensions and thereby on the order in which dimensions are aggregated, the engine uses a kind of query plan optimization to select dimensions in a “good” way.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

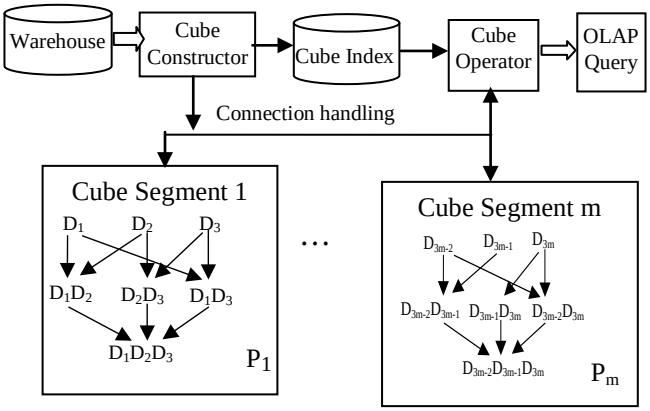

Fig. 1. The System Architecture of Parallel Construction of Shell Cube Segments

4.2 Efficient OLAP query handling

Based on the bitmap indexing, we can efficiently retrieve the matching hierarchy levels of each dimension, evaluate the set of query ranges for each dimension, and improve the efficiency of OLAP queries. A key property of our encoding is that it is a prefix indexing scheme that allows one to quickly retrieve a path prefix for each dimension.

The path prefix of the member d_k^i of the hierarchy level L_j^i is defined as DMPrefixpath(DTree, d_k^i)= $\cup_{j=i}^{1}$ DMPrefixpath(DTree,Parent(d_k^j))={ Ancestor(d_k^i)}, where Ancestor(d_k^i) is the all ancestors of the member d_k^i according to its dimension hierarchy tree. The encoding prefix of the member d_k^i is defined as Bprefix($B^{d_k^i}$, L_{m-1}^i)= $B^{d_k^i} >> \sum_{l=m}^{j}$ (Bit $^{L_l^i}$), where m={1, ...,j}.

By using encoding prefix, we can register the dimension hierarchy encoding and its TID list for every dimension hierarchy for each dimension. For example, the dimension hierarchy encoding and its TID list associated with the dimension hierarchies *Month* and *Province* are shown in Table 9, and so on.

For each fragment, we compute the complete data cube by intersecting the TID-lists in the dimension and its hierarchies in a bottom-up depths-first order in the cuboid lattice (as seen in [8]). For example, to compute the cell {0001000010000001, 0000001000010001, 0010001}, we intersect the TID lists of $B^{ProductID}$ =0001000010000001, $B^{RegionID}$ =0000001000010001, and *Bprefix*(B^{TimeID},*Month*)= 0010001 to get a new list of {1,2}. The algorithm of efficient OLAP query can be summarized as follows.

Table 9. Month hierarchy encoding Prefix AND its TID

B^{TimeID}	*Bprefix*(B^{TimeID},*Month*)	TID List
001000100001	0010001	1-2-3-4
001000100010		
001000100011		
...	...	...
010000100001	0100001	367
...	...	...

Algorithm 2 (OLAP Query)

Input: A set of precomputed shell Cube Segments for partitions {P_1,..., P_k}; an TID measure array; and a OLAP query Q<a_1,...,a_n,M>.(The a_i is attribute for the dimension A_i and M is the measure of the query.)

Output: The computed measure

{ ascertain all Cube Segment CS_i according as the each dimension attribute of the query Q<a_1,...,a_n,M>;

for each CS_i

{ compare the CS_i with query Q<a_1,...,a_n,M> using the Lattice and find the all dimension D_i of $CS_i \cap \{a_1,...,a_n\}$ with parallel processing;

compute the TID List of the all BC_i cells of $CS_i \cap \{a_1,...,a_n\}$ in D_i and its aggregate Cuboids;

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

intersect the TID List of th BC_i and compute the query result set RQ{TID List};

compute the aggregate with each TID of the TID-measure array from the set RQ{TID List};}}

This method uses the small dimension hierarchical encoding and their prefix path, so it can rapidly retrieve the matching dimension member hierarchical encoding and evaluate the set of query ranges for each dimension. It rapidly aggregated the clustered fact data that is clusteringly stored by the dimension hierarchical encoding, so that it can drastically reduce the multi-table join effort and so much as could remove completely one or more join operations. As a result, the algorithm can greatly reduce the disk I/Os and highly improve the efficiency of OLAP queries.

5. PERFORMANCE STUDY

There are two major costs associated with our proposed method: (1) the cost of storing the shell fragment prefix-index cubes and their intelligent dimension hierarchical encoding, and (2) the cost of retrieving dimension hierarchical encoding and computing the queries online. In this section, we perform a thorough analysis of these costs. In our experimentation we generated a large number of synthetic data sets which in terms of the following parameters: d— number of dimensions, h_i — number of hierarchy levels of dimension D_i, m— maximum number of distinct members of the hierarchy L_j^i, T—number of tuples , f — size of the shell cube segment.All experiments were conducted on an Intel Pentium IV 2.8 GHz system with 512MB main memory , running Microsoft Windows-XP Server.

The performance results of prefix-index cubing and the minimal cubing of Li's and Han's[11] are reported from Fig. 2 to Fig. 5. Fig. 2 shows the storage size of the two methods on the cube had T=10^6 tuples and h =1 level hierarchy and with shell fragment size f=3 , and the storage size of the two methods on the cube had h=3 levels hierarchies is shown in the Fig. 3. Fig. 4 shows the average I/O page access for online query of the two methods on the cube had different levels of hierarchy and with shell fragment size f=3 and T=10^6 tuples. Fig. 5 shows the average I/Os of prefix-index cubing method with different dimensions .

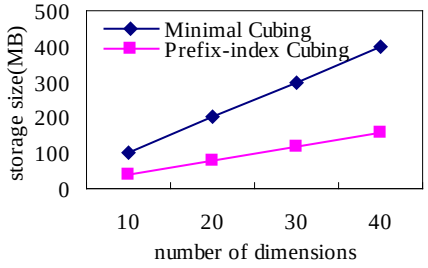

Fig. 2. Storage size of shell segment with h=1

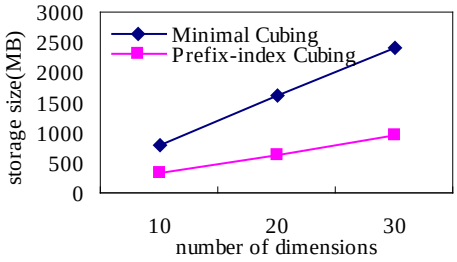

Fig. 3. Storage size of shell segment with h=3

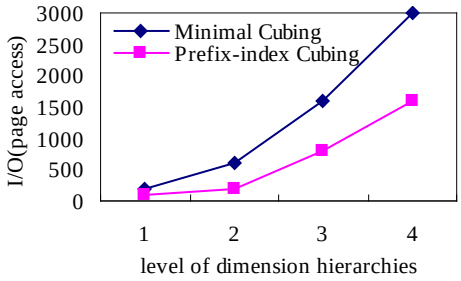

Fig. 4. Average I/Os with f=3 and d=10

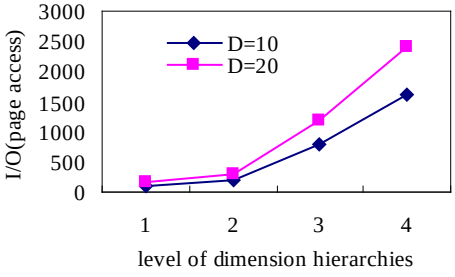

Fig. 5. Average I/Os with f=3 of prefix-index cubing

6. CONCLUSION

Data cube has been playing an essential role in fast OLAP in many multi-dimensional data warehouses. The pre-computation of data cubes is critical to improving the response time of OLAP

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

systems and accelerating data mining tasks in large data warehouses. But in a high-dimensional hierarchical OLAP, it might not be practical to build all these cuboids and their indices In this paper, we propose a multi-dimensional hierarchical cubing algorithm, Prefix-index hierarchical cubing. It partitions a high dimensional cube into a set of disjoint low dimensional cubes. OLAP queries are computed online by dynamically constructing the cuboids from these cube segments. The analytical and experimental results show that proposed Prefix-index hierarchical cubing algorithm is significantly more efficient in time and space than the other leading cubing methods.

ACKNOWLEDGEMENTS

The research in the paper is supported by the National Natural Science Foundation of China under Grant No. 61070047, 61003180; the "Six Talent Peaks Program" of Jiangsu Province of China; the "333 Project" of Jiangsu Province of China.

REFERENCES

[1] S. Chauduri, and U. Dayal, "An overview of data warehousing and OLAP technology", *SIGMOD Record*, Vol. 26, No. 1,1997,pp. 65-74.

[2] K. Wu, E. J. Otoo, and A. Shoshani, "A performance comparison of bitmap indexes", *Proceedings of the 10th International Conference on Information and Knowledge Management*, ACM Press ,New York, NY, 2001, pp.559-561.

[3] H. Mistry, P. Roy, and S. Sudarshan, "Materialized view selection and maintenance using multi-query optimization", *Proceedings of the 2001 ACM SIGMOD*, ACM Press ,New York, NY, 2001, pp. 307-318.

[4] J. Gray, S. Chaudhuri, A. Bosworth , A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. "Datacube: A relational aggregation operator generalizing group-by, cross-tab and subtotals", *Data Mining and Knowledge Discovery*, 1997, pp. 29-54.

[5] K. Beyer, and R. Ramakrishnan, "Bottom-up computation of sparse and iceberg cubes", *Proceedings of the 1999 ACM SIGMOD*, ACM Press ,New York, NY,1999, pp. 359-370.

[6] J. Han, J. Pei, G. Dong, and K. Wang, "Efficient computation of iceberg cubes with complex measures", *Proceedings of the 2001 ACM SIGMOD* , ACM Press ,New York, NY, 2001, pp.1-12.

[7] L. V. S. Lakshmanan, J. Pei, and J, Han, "Quotient cubes: how to summarize the semantics of a data cube", *Proceedings of 28th International Conference on Very Large Data Bases*, Morgan Kaufmann, San Fransisco, 2002, pp. 778-789.

[8] D. Xin, J. Han , X. Li, and B. W Wah, "Star-cubing:computing iceberg cubes by top-down and bottom-up integration", *Proceedings of 29th International Conference on Very Large Data Bases* ,Morgan Kaufmann, San Fransisco, 2003, pp. 476-487.

[9] L. V. S. Lakshmanan, J. Pei, and Y. Zhao, "QC-trees: An efficient summary structure for semantic OLAP", *Proceedings of the 2003 ACM SIGMOD* **,**ACM Press ,New York, NY, 2003, pp. 64-75.

[10] Y. Sismanis, A. Deligiannakis, Y. Kotidis, and N. Roussopoulos, "Hierarchical dwarfs for the rollup cube", *Proceedings of 30th International Conference on Very Large Data Bases*, Morgan Kaufmann, San Fransisco, 2004, pp. 540-551.

[11] X. Li, J. Han, and H. Gonzalez, "High-dimensional OLAP: A minimal cubing approach", *Proceedings of 30th International Conference on Very Large Data Bases* . Morgan Kaufmann, San Fransisco, 2004, pp. 528-539.