http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 9 (1): 116-123, 2010
ISSN 1812-5638

Research on Index Technology for Group-by Aggregation Query in XML Cube

[1]Yawei Zhao, [2]Tinghuai Ma and [3]Feng Liu
[1]College of Computing and Communication, Graduate University of Chinese Academy of Sciences, China
[2]School of Computer and Software Nanjing, University of Information Science and Technology, China
[3]Research Institute of Fiscal Science Ministry of Finance, China

Abstract: In order to enhance efficiency of group-by aggregation query, the purpose of this study is to explore a new XML data cube model and on this basis to research appropriate index technology. As group-by aggregation query is common in data cube, the query efficiency of an XML cube, especially for aggregation query, is critical to determine its usability. In this study, we introduce a new kind of format of XML cube and propose an index method of the cube for group-by aggregation query, the index of which is based on hash and tree index to rapidly attain the measurable paths in the cube. We describe the algorithms of building the new index and its corresponding group aggregation query in an XML data cube and estimate the cost of query and carry out related experiments.

Key words: XML cube, group-by, aggregation, hash index, query

INTRODUCTION

Data warehousing and Online Analytical Processing (OLAP) technologies are now moving onto handling complex data that mostly originate from the web (Boussaid *et al*., 2008). Building data warehouse based on relational database is a traditional technique. In recent years, with the development and application of Internet, XML has become an important criterion for data exchanging. XML was proposed by W3C and has been widely used for data exchange standard on the Internet. XML data has no fixed architecture and format, so it is named semi-structured data (Abiteboul *et al*., 1999). Due to an explosive increase of XML documents, it is imperative to manage XML data in an XML data warehouse. So, data warehouse based on XML has been a hot research topic and used in some data cube architectures. But XML warehousing and OLAP based on XML warehouse impose challenges, which are not found in the relational data warehouses (Rusu *et al*., 2009; Park Byung *et al*., 2005; Boussaid *et al*., 2008). Some traditional methods with star schema (Pokorný, 2002; Hümmer *et al*., 2003) to build XML data warehouse directly is inefficient to query.

As data cube is core component in data warehouse and is widely used in practical projects, XML cube is a new method describing and saving with XML data. Hümmer *et al*. (2003) proposed XCube which is data warehouse architecture including three documents: XcubeSchema, XcubeDimension, XcubeFact. XCubeSchema describes the whole XML data warehouse architecture; XCubeDimension defines all dimensions value and hierarchies of XML data warehouse and XCubeFact contains the values of facts for every data cube.

A method of building XML data warehouse without giving its architecture but extracting data required from original XML files by XQuery and saving more XML files (Taniar *et al*., 2005). Each dimension saves as one file and only one fact document and a dimension file is joined with a fact file by key. The advantage of the method is very simple. But if the quantity of data is very large, it would spend much time using a query by XQuery.

Paparizos *et al*. (2002) advances a method of transforming analysis XQuery expression into Pattern Tree, because the XML files allow some elements inexistence and its architecture is very flexible in opposition to relational database. Comparing and contrasting Pattern Tree and XML architecture, they concluded the right result being uniform to Pattern Tree (Jagadish *et al*., 2002).

The relative index methods of aggregation query are not introduced by the above methods. The low efficiency of aggregation query is not solved because the process of aggregation query is to look through the whole XML data cube or to use the XML index to deal with XML data cube as XML data.

Corresponding Author: Yawei Zhao, College of Computing and Communication,
Graduate University of Chinese Academy of Sciences,
No.19A Yuquan Road, Beijing, 100049, China Tel: 86-01-8825-6564

Although, some researchers have developed data models of XML cube, but the research is still in its initial phase. There are no successful data models and methods to build a XML data warehouse. The main reason is that there is significant difference between XML data and relational data. Relation schema is strict in the structure and uniform in the formats and XML schema is a typical semi-structure. Furthermore, the large amount of data also leads to the inefficiency in an XML data cube, especially for group-by aggregation query.

XML DATA CUBE SCHEMA

On the conceptual view, the data cube based on relational data is the same with XML data cube. Data cube consists of dimensions and facts. There are two types of cubes: star schema and snow flake schema. As the snow flake schema is a form of normalization of star schema, they are equivalent in logic. In this study, we only consider the cube with star schema.

Dimensions and facts both have balanced tree structure. The distance between the root node and each leaf node is the same in one dimension or in fact. The granularity of cube determines the attribute of a leaf node. Each measure value is corresponding to leaf nodes in all dimensions and the fact is the Cartesian-product linked measure.

XML data cube

Definition 1. Star schema: Let D = {D_s | s = 1, 2, …, r} be a set of independent dimensions. Each D_s has D_s.PK as a primary key. F is fact with r foreign keys, which are {FK_t | t = 1, 2, …, r}. A star schema is defined by the couple (F, D) that satisfies the conditions $F.FK_t = D_s.PK$. Each D_s.PK has one or many corresponding $F.FK_t$.

This definition is more suitable to build a relation data cube. For example, in a cube with star schema shown in Fig. 1, each dimension has one primary key which is equal to the foreign keys in the fact table, which has 4 foreign keys: Time_key, Branch_key, Location_key and Item_key.

The number of primary keys in a dimension depends on the number of attributes which is in the smallest granularity in the relational data cube. The number of records in fact table is equal to the Cartesian-product of all dimensions linked to fact. The more detail there is, the more the number of records in fact table.

In a similar way, XML cube definition has similar definition to relational cube but with its own specialties.

Definition 2. XML star schema: Let (F, D) be a star schema, where F is a fact and D is a set of dimensions. Both the fact and dimensions are described with XML data. F and D have tree structures. F is the Cartesian-product of leaf nodes of D and linked measured values.

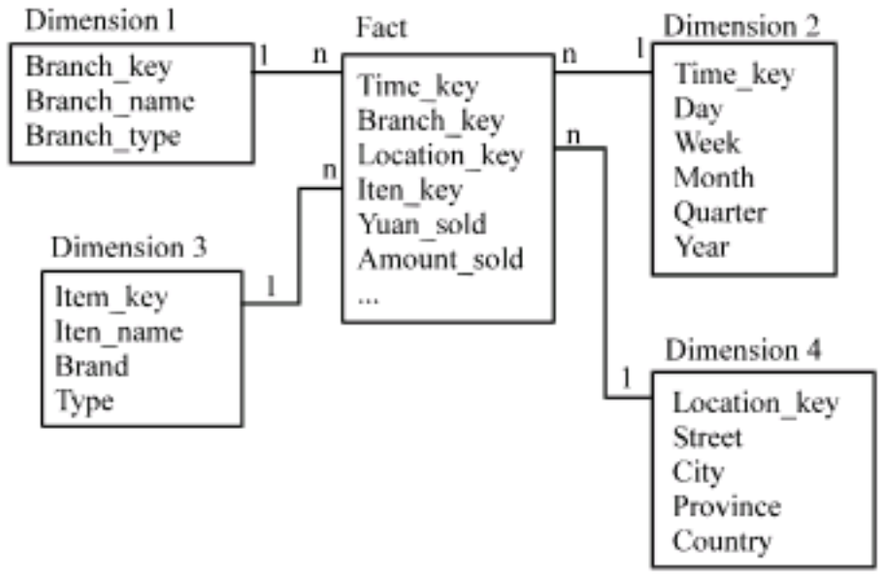

Fig. 1: Star schema

```
<xs: key name = "factFtimekey" refer = "dimesion1timekey">
    <xs: selector xpath = "/cube/fact"/>
    <xs: field xpath = "Branch_key"/>
</xs: key>
<xs: key name = "factFbranchkey" refer = "dimesion1timekey">
    <xs: selector xpath = "/cube/fact"/>
    <xs: field xpath = "Time_key"/>
</xs: key>
…
<xs: key name = "itemkey" refer = "dimesion1timekey">
    <xs: selector xpath = "/cube/fact"/>
    <xs: field xpath = "Item_key"/>
</xs: key>
```

Fig. 2: Fact schema about foreign keys

```
<xs: key name = "dimesion1timekey">
    <xs: selector xpath = /cube/dimension1"/>
    <xs: field xpath = "Time_key"/>
</xs: key>
```

Fig. 3: Dimension schema about primary key

There are keys in XML schema if the dimensions and facts have respective schemas. The fact could join dimensions with keys, but it would result in a rapid increase in the quantity of fact data.

For example, all of the foreign keys should be defined in the fact schema and all of the primary keys are also defined in dimension schema, which are shown in Fig. 2 and 3.

It is very inefficient for aggregation query in XML cube by the keys. In a fact XML file, each measure should cite all the dimensions keys. When being queried in XML cube, all keys in the fact document should be located. This will result in much more time cost. Furthermore, we

can not recognize the hierarchies in dimension merely through the keys. Therefore, it would be inefficient to build the XML cube by keys when the number of dimensions is too high.

The key issue is how to locate the fact data very quickly in cube paths. Meanwhile, the relative fact data should be stored together, which can help to improve the query efficiency.

XML data cube schema: Because the query is inefficient to take keys to join dimensions and facts in XML cube, we design a new XML cube model integrating Laura Irina Rusu's model with Wolfgang Hümmer's model, which does not use XML keys to join dimensions and facts but use the schema of cube described with code. The XML cube, which is called XMLCube for short, also has three parts: schema, dimension and fact.

Dimension code: The XML data has tree structure, so a query in general needs to look through all the nodes in the tree. In order to improve the efficiency, encoding of nodes in dimensions is necessary. The code of root node is exclusive and identifies a tree. A son node is encoded as follows:

$$\text{Node\_code} = \text{rn\_code} + \text{fn\_code} + \text{sn\_offset} \quad (1)$$

In Eq. 1, rn_code is the code of root node, fn_code is the code of father node and sn_offset is the offset of son node. All the sn_offsets can be computed with the fn_code and the fn_code also can be computed with a random one sn_offset. This coding method fits to the aggregation operation and is helpful for getting the high relative dimension nodes rapidly.

Hierarchy of a dimension is described by the semantics of code. The code of a node in dimension can be translated into the XML element path. Code is in a special order in dimension schema, such as from top to bottom or from left to right and so on. It is a good choice to code from left to right, which is named horizontal code. Fig. 4 shows the horizontal code of a dimension. The code indicates hierarchy of the dimension. For example, node code a021 means that its dimension identifier is a, its father code is 02 and its own code is 021. The code identifies the location of a node in dimension tree obviously. Because identification of root (such as a in the above example) is useless in dimension computation, it is ignored for simplicity.

Fact XML: Path in terms of each measure is one of element of Cartesian-product of leaf nodes' codes of all

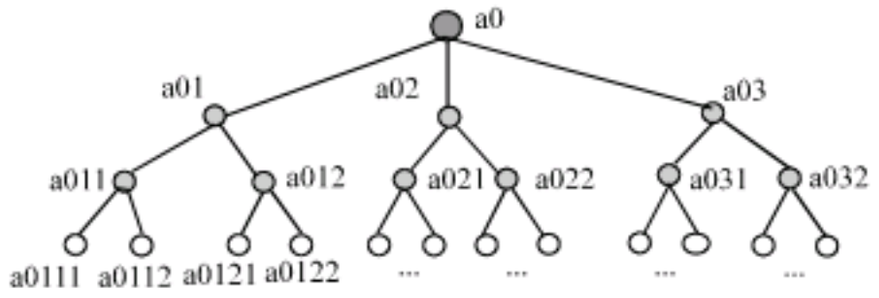

Fig. 4: Dimension coding

```
<ReatilFact fact="Count" dimension="3">
  <a0111 dimension="Department">
    <b111 dimension="Catalog">
      <c211 dimension="Date">
        <jxCount> 100 </jxCount>
        <lxCount> 200 </lxCount>
      </c211>
    </b111>
  </a0111>
</ReatilFact>
```

Fig. 5: A cell of fact XML

dimensions in XMLCube. The Cartesian-product described paths of all measures, that is to say, the path and measure value express fact XML. All paths in fact XML are as follows:

$$\text{Fact}_{\text{path}} = \{\text{dimension1.leaf.node\_code}\} \times \{\text{dimension2.leaf.node\_code}\} \times \ldots \times \{\text{dimension n.leaf.node\_code}\}$$

For example, Fig. 5 describes a cell of a fact XML. The XML cube has three different hierarchical dimensions: a, b and c. The dimension a has four levels and the other two dimensions b and c has 3 levels. The combination of leaf nodes of the three dimensions <a0111, b111, c211> represents the path of cell.

Cube schema: XMLCube schema is integrated with description of a XML cube and is oriented to a certain subject. Code of every dimension, ranking of each dimension code in fact XML and their measures are all defined in the schema, so the cube schema is regarded as metadata.

XML CUBE INDEX FOR GROUP-BY AGGREGATION QUERY

Group aggregation, which is base of roll-up and drill-down operations in a cube, is generally accepted in the cube created by relational database. In general, when the cube needs to be built, the raw data should be grouped (Gray *et al*., 1996; Jagadish *et al*., 2001; Bordawekar and Christian, 2005; Paparizos *et al*., 2002).

An appropriate index is the basis of improving query efficiency. In the relational databases, index technique is mature and there are also some similar index techniques in XML database (Barashev and Novikov, 2002; Li and Moon, 2001). The index techniques in XML database can be classified into two categories: node-record-style index and structural-summary-style index (Ling-Bo *et al*., 2005). These index techniques could improve the query efficiency effectively. Although, they could be used in query of XML cube directly, these techniques do not fit to aggregation operations in XML cube because they are not targeted for it.

The index which is suitable for an XML cube should make full use of its hierarchy characteristics. Based on the XMLCube model above, we design a new index integrating the hash index with the tree index. The purpose of building new index is to get measure paths rapidly for aggregation query. According to these paths, aggregation query can be completed with an appropriate query language (e.g., XQuery) to improve aggregation query efficiency.

Index structure: Based on the above XMLCube mode, schema, the index structure of group-by query is designed as follows:

- The index has two levels: XML cube level and dimension level. The former is called cube index or global index and the latter is called dimension index or local index
- Dimension index adopts the balanced tree structure: the distance between the root node and each leaf node is the same
- The structure of bucker of cube index and dimension index is as follows:

$$S = < search\_key, Node\_code> \quad (2)$$

$$Node\_code = fn\_code + sn\_offset \quad (3)$$

In Eq. 2, S is the address of hash bucket, each element of the bucket has two attributes, search_key is the node value for query and Node_code is the code of the search_key. The cube index adopts standard hash bucket and the dimension index adopts special hash bucket whose hash function is to calculate the father's code. Figure 6 is an example showing a structure of cube index and Fig. 7 is one example of dimension indexes of the cube.

There are two important parts for an index computation, one is to locate bucket including the search key and the other is to get the bucket address in dimension index. Only with the collaborative

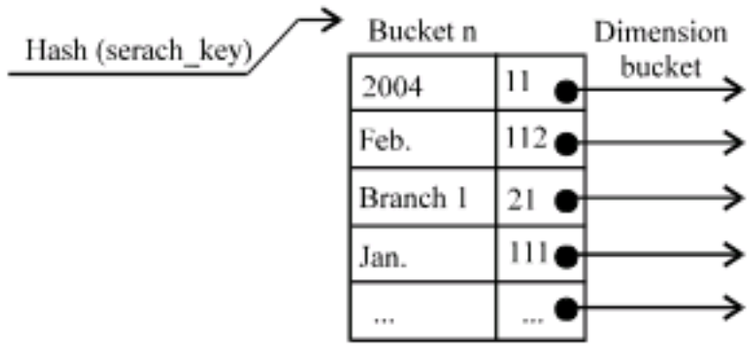

Fig. 6: A cube index structure

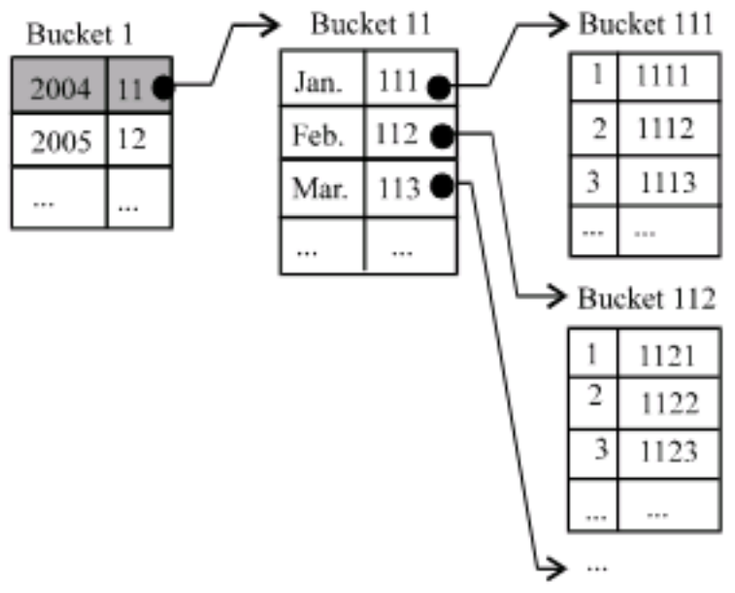

Fig. 7: A dimension index structure

computations using the two components can we determine the path of each fact cell for the aggregation query.

Calculation: As mentioned earlier, because the index is a two-level structure, the calculation for index includes two types: hash calculation and father-code calculation. The two types of calculations are used to get the bucket address. Hash calculation is to get the bucket address of search key and the father-code calculation is to get bucket address of a node and those of all its son nodes.

- **Hash calculation:** Apply general hash function to get bucket address of search key:

$$Hash (search\_key) = n \text{ (bucket address)}$$

- **Father-code calculation:** To get code of father node, the farther code is the bucket address of the current node. We describe the calculation using function get:

$$FatherCode (Node\_code) = fn\_code \text{ (bucket address)}$$

For example, a hash calculation is hash (Jan) = 03, which represents the address of bucket that saves the search key Jan. The value of Jan can be gotten with sequential search. The code of Jan is 111 which is also in

the bucket and binding with Jan. The value of function getFatherCode (111) is 11 and it is the address of bucket which saves code of 111 in dimension index. According to dimension index all of son nodes until the leaf nodes of the search key Jan can be located. The leaf node is an element of path of a fact cell. In the same way, when leaf nodes of other relational dimensions can be gotten. These leaf nodes rank by the order which is defined in XMLCube schema and to be all fact cell paths of Jan.

Index building algorithm: The index can be built by the structure of index as introduced above. The condition of building index is that the data has been organized with the XMLCube model. The process has two steps: initial phase and update phase. The first phase is referred to as the index building for the first time and the second phase is the index rebuilding when the data in the XML cube changes.

Algorithm 1: Index building algorithm:

- Search the dimensions and facts according to an XMLCube schema
- Read all nodes and edit codes, build the cube index, calculate the hash value of each code and save the result in the right bucket
- Save the son node value together with its code in the bucket whose address is its father's code to build the dimension index
- Build the links between cube index and dimension index and links among buckets in different hierarchies

When the dimension value changes, the index data needs to be rebuilt. The detailed procedures are as follows:

- When a dimension value is added, such as the data of a new year being added, a new branch is added because the structure of dimension index is a tree. Similarly, when the dimension value is deleted, its corresponding branch will be deleted from the index. It is rare to delete the dimension data in the cube
- When the dimension value is to be updated, the cube index will be updated with the updated value. For example, if the new dimension value is added, the added value will be allocated to the corresponding hash bucket; if a dimension value is deleted, the value will be deleted in the corresponding bucket

Because of adopting hash calculation, the efficiency of rebuilding the index is very high. Because the index is mainly used for group-by aggregation and a type of analytical query but not a real-time one, the frequency of index updating is low and the slow updating process is in general acceptable.

Index query algorithm: The path of each measure required by group-by query could be searched by index quickly. When all the paths are obtained, the group aggregation operation for the node could get its measure value.

Algorithm 2: Index query algorithm:

- According to a search key x, function hash (x) = B_0 (address of bucket) is performed and x.Node_code is searched and gotten in the bucket B_0
- Perform the function getFatherCode (Node_code) and get the value x.fn_code which is its father's code of x and is also the address of bucket saving x. We named the address B_1 for simplicity
- Search in bucket B_1 and calculate all son nodes of x until leaf nodes. The address set of leaf nodes can be gotten:

$$\{1.\ Node\_code_n \mid n=1, 2, \ldots, k\},\ 1 \text{ is leaf node}$$

- Integrate the codes of leaf nodes with the codes of other dimensions in the sequence provided by schema of XMLCube. The query path set in fact file can be gotten, such as:

$$path_k = /l_1.\ Node\_code_n / l_2.\ Node\_code_n / \ldots / l_m.\ Node\_code_n /$$

- Set the path set as import parameter and perform a kind of XML query language (e.g., Xquery). The aggregation query result of x could be calculated

For example, given a search node Jan., query all leaf nodes' codes. Jan. is a value of dimension tree. First compute hash (Jan.) and get the result n (bucket address) according to global index, locate all codes of the Jan. in the bucket which is shown as Fig. 8. Compute father code of each code, such as value of getFatherCode (111) = 11 is bucket address of Jan. Calculate codes of all son nodes until leaf nodes in dimension index, such as code 111 of Jan. is bucket address of its leaf node. If there is no constraint, then query all the codes of leaf nodes in other dimensions. Rank these leaf nodes' codes with codes in bucket 111 in order by the definition of XMLCube schema. Then all paths of measure cell for search key Jan. Figure 9 shows the paths of a cell in an aggregation described by XQuery.

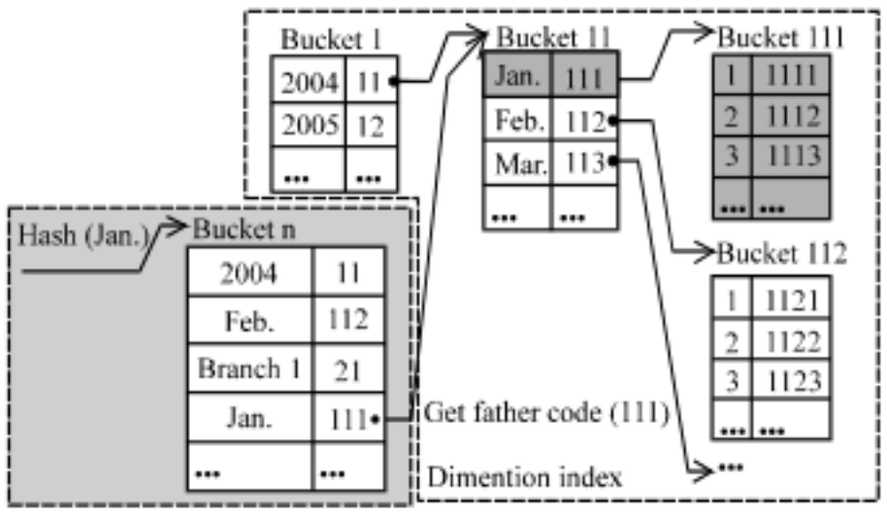

Fig. 8: Get the leaf nodes by the index

```
for $c in /ReatilFact/a0111/b111/c211/jxCount ,
    $d in /ReatilFact/a0111/b111/c212/jxCount ,
    $e in /ReatilFact/a0111/b112/c211/jxCount ,
    $f in /ReatilFact/a0111/b112/c212/jxCount
return
<jxCount>
  {$c+$d+$e+$f
  }</jxCount>
```

Fig. 9: An aggregation query with xquery

Cost estimating of index query: Appropriate index could improve the efficiency of query significantly, but the indexing cost determines its usability. In XMLCube, the cost of query based on index as introduced above includes two main parts. The first part is the query cost of indexing files and the second one is the cost of querying data files. For the second part, fact document of XML Cube could be regarded as general XML data and can apply XML index. So, we only consider the query cost of the first one.

For simplicity, some assumptions made are as follows:

- Consider the number of block transfers only
- Ignore the difference of sequence and random I/O and the cost of CPU
- Estimate the worst case only

The cost estimation of index is calculated as follows:

Cube index cost estimation: Cube index cost includes search code hash computation and index file I/O operation. Because hash computation is the cost of CPU, it can be ignored. Suppose the number of blocks to save one bucket is b_r, the worst case is linear search in the cube index and the cost of index is:

$$E_{A1} = (b_r /2),\ b_r \text{ is the number of blocks}$$

Dimension index cost estimation: Suppose the number of son nodes of the search key is ln and the number of blocks to save one bucket is b_r also, the cost is:

$$E_{A2} = (b_r /2) + \ln \times b_r,\ \text{where } b_r \text{ is number of blocks}$$

Total cost estimation: The total cost of the query is the sum of the above costs, namely, it is:

$$\begin{aligned} \text{Cost} &= E_{A1} + E_{A2} \\ &= (b_r /2) + (b_r /2) + \ln \times b_r \\ &= (\ln+1) \times b_r \end{aligned}$$

When the query needs to search more than one dimension, the cost of query is the sum of the above costs. According to the formula of total cost, when ln is smaller, Cost would be lower. For the height of dimension tree changes in relation to the size of bucket, reducing the value of ln would control the cost of index query. When the height of dimension tree is 3~4, the total cost are 5 blocks generally.

EXPERIMENTAL ANALYSIS

The purposes of this experiment are as follows: One purpose of the experiments is to attain the efficiency based on the XMLCube code; another is to analyze the cost and performance of XMLCube aggregation query.

Experiment settings: All the experiments in this study were done in a PC with P8400, 2.26 GHz of CPU, 1 GB of SDRAM and 200G of Disk. On the PC, the OS is Windows XP, the MDBS is SQL Server 2005, the developing tool of XML is Stylus Studio 2006 and the developing language of experimental system is C++.

Data set: A real data set Retail is selected to assure the accuracy of experimental data for there is no benchmarking data set until now. The data set Retail, with good distribution, is relational and has three dimension tables, one fact table and six measures. There are 4, 913, 818 data records in fact table and the other three dimension table has 91, 266 and 203 records. In the experiment, we transfer these data into XCube-format data and XMLCube-format data, respectively.

Process: Our experiment was designed for XCube schema and XMLCube schema, which was done with the same operation, scenario and process. The specific process is as follows:

- Build retail data cube based on platform SQL Server 2005 and ensure the accuracy of the data cube
- Transform the retail data cube into XMLCube-format cube according to the XMLCube schema
- Transform the retail data cube into XCube-format cube according to the XCube schema
- Build the XMLCube index according to the XMLCube schema
- For XMLCube, do different research to get the XMLCube aggregation path to validate the availability and efficiency according to the different number of leaf nodes
- Compare the query efficiency based on the XCube and XMLCube in the same scenario

Result and performance analysis: The paths in an XMLCube-format cube were gotten by index. The search efficiency is related to the number of the fact cell paths in the aggregation query, which is related to the number of leaf nodes. Suppose the searching efficiency through each path is the same, the time T for searching all the paths is equal to the searching time t for a path multiply the total number of paths n, T = t×n. For the above experiment, our results are as follows.

XMLCube efficiency of path getting: The computational efficiency of fact cell path is caused by the smallest granularity and all dimensions. The result is shown in Fig. 10. Real efficiency of path getting is between the largest efficiency of path getting and the smallest one. The shortest path and longest path is the maximum or minimum value at the specific number of leaf nodes.

The result shows that the efficiency of path searching is increasing in general when the number of leaf nodes is increasing because all paths are determined not only by the number of leaf nodes but also by the Cartesian-product of each dimension.

Query efficiency: Query efficiency can be measured by the time cost between an XMLCube query and an XCube query with the smallest granularity and all dimensions. The result is shown in Fig. 11.

The result shows that the query cost by XMLCube is lower than that by XCube and the gap between the query efficiencies becomes more obvious as the number of fact cells increases. The XCube query efficiency is low due to the following two major reasons. One is that query operation needs to join the fact data and dimension data, which will cause the significant decrease as the amount of data reaches a certain level. Another is that each cell has no obvious difference in the path for the XCube fact data, which will lead to cover the whole fact data document.

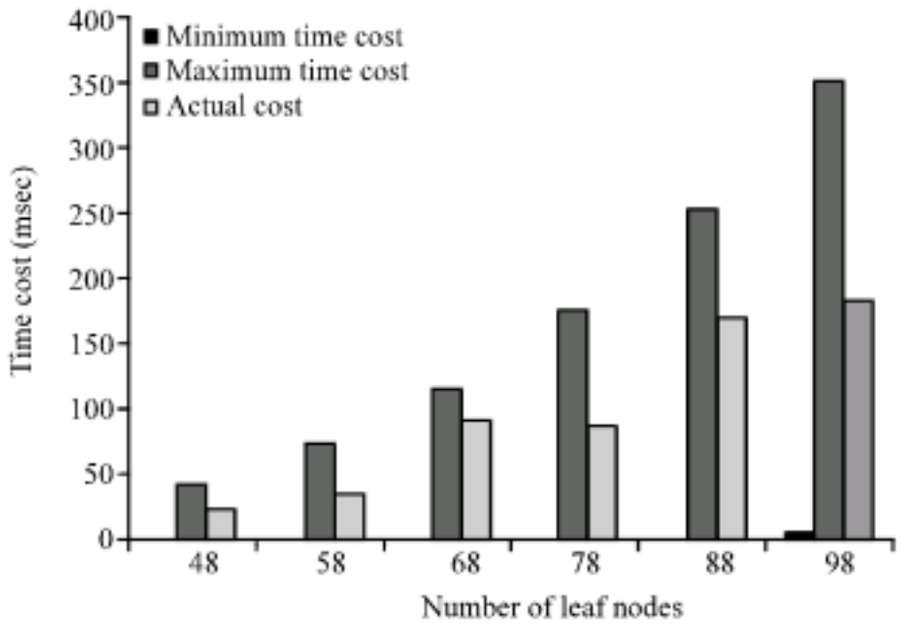

Fig. 10: Efficiency of path getting of XMLCube

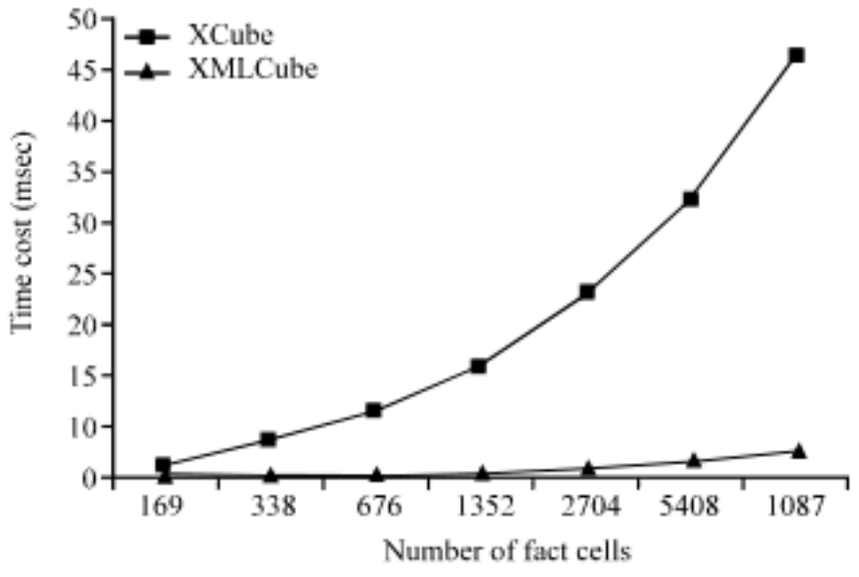

Fig. 11: Comparison of aggregation query efficiency due to XCube and XMLCube

According to the given comparison, a fact cell path in XMLCube can locate a specific cell very fast in the query process because the code method is basing on semantic and cell paths are different with codes. Furthermore, the cell path of aggregation query is created by dimension index, which can avoid the inefficient problem because the fact data joins the dimension data.

CONCLUSION

We have presented an XML cube model named XMLCube based on coding and an index structure which is applicable to aggregation query for XMLCube. The structure of the index looks up the XML query paths of aggregation query as the searching objects and integrates hash index with tree index. By analyzing the cost, the new index can improve the efficiency of query in a XML cube. The performance of XMLCube schema is better than XCube schema according to the experimental results. However, there are still some technical problems to be addressed, such as the control method of the index size cost, redundancy in the index. These are also the study for the next logical phase.

ACKNOWLEDGMENTS

This study is supported by Foundation of President of The Chinese Academy of Sciences (O65001H), Natural Science Foundation from Nanjing University of Information and Science Technology (20080302), the oversea study scholarship of jiangsu government and jiangsu youth project.

REFERENCES

Abiteboul, S., P. Buneman and D. Suciu, 1999. Data on the Web: From Relations to Semi-Structured Data and XML. 1st Edn., Morgan Kaufman Publishers Inc., San Francisco, CA. USA., ISBN: 155860622X.

Barashev, D. and B. Novikov, 2002. Indexing XML to support path expressions. Proceedings of the 6th East European Conf. on Advances in Databases and Information Systems (ADBIS), Bratislava, Sept. 8-11, Springer-Verlag, pp: 1-10.

Bordawekar, R.R. and A.L. Christian, 2005. Analytical processing of XML documents: Opportunities and challenges. ACM SIGMOD Record, 34: 27-32.

Boussaid, O., J. Darmont, F. Bentayeb and S. Loudcher, 2008. Warehousing complex data from the web. Int. J. Web Eng. Technol., 4: 408-433.

Gray, J., C. Surajit, B. Adam, L. Andrew and R. Don *et al*., 1996. Data cube: A relational aggregation operator generalizing group-by, cross-tab and sub-totals. Proceedings of 12th ICDE. New Orleans, Mar. 1, Morgan Kaufmann Publishers Inc., pp: 152-159.

Hümmer, W., B. Andreas and H. Gunnar, 2003. XCube: XML for data warehouses. Proceedings of the 6th ACM International Workshop on Data Warehousing and OLAP, Nov. 7, ACM, New Orleans, pp: 33-40.

Jagadish, H., L. Lakshmanan, D. Srivastava and K. Thompson, 2001. Tax: A tree algebra for XML. Proceedings of the International Conference on Database Programming Language, Sept. 8-10, IEEE Xplore, pp: 149-164.

Jagadish, H.V., S. Al-Khalifa, C. Adriane, V. Laks and S. Lakshmanan *et al*., 2002. TIMBER: A native XML database. Int. J. Very Large Data Bases, 11: 274-291.

Li, Q. and B. Moon, 2001. Indexing and querying XML data for regular path expressions. Proceedings of the 27th International Conference on Very Large Data Bases, Sept. 11-14, IEEE Xplore, London, pp: 361-370.

Ling-Bo, K., T. Shi-Wei, Y. Dong-Qing, W. Teng-Jiao and J. Gao, 2005. XML indices. J. Software, 16: 2063-2079.

Paparizos, S., S. Al-Khalifa, H.V. Jagadish, L. Laks, N. Andrew, S. Divesh and W. Yuqing, 2002. Grouping in XML. Proceedings of Workshop on XML-Based Data Management, Heidelberg, Mar. 24, Springer-Verlag, pp: 169-183.

Park Byung, K., H. Hyoil and S. Il-Yeol, 2005. XML-OLAP: A multidimensional analysis framework for XML warehouses. Proceedings of the 7th International Conference on Data Warehousing and Knowledge Discovery, Aug. 22-26, DaWaK, Copenhagen, Denmark, pp: 32-42.

Pokorný, J., 2002. XML data warehouse: Modelling and querying. Proceedings of the Baltic Conference, Baltic DB and IS, Tallinn, Jun. 3-6, Kluwer Academic Publishers, pp: 267-280.

Rusu, I.L., R. Wenny and T. David, 2009. Partitioning methods for multi-version XML data warehouses. Distributed Parallel Databases, 25: 47-69.

Taniar, D., J.R. Wenny and I.R. Laura, 2005. A methodology for building XML data warehouses. Int. J. Data Warehousing Mining, 1: 23-48.