2012 4th International Conference on Computer Engineering and Technology (ICCET 2012)
IPCSIT vol.40 (2012) © (2012) IACSIT Press, Singapore

A New Multidimensional XML Storage Structure

Dhiaa Musleh, Salahadin Mohammed and Muhammed Al-Mulhem

Collage of Computer Sciences & Engineering
King Fahd University of Petroleum and Minerals
Dhahran 31261, Saudi Arabia
{dhiaa, adam and mulhem}@kfupm.edu.sa

Abstract. XML has emerged as a new standard for information representation and exchange on the Internet. Nowadays, the number of applications using XML data is increasing rapidly. As a result, it is important to develop efficient storage structure to store XML data .In this paper we propose a Multidimensional XML Storage Structure (MXSS) which supports the efficient evaluation of structural relationship among XML elements. MXSS stores XML data path-wise by using multidimensional storage structure. Our experimental results show that MXSS outperforms Inverted-list based storage structures in the number of disk blocks used to store benchmark datasets. Also, MXSS showed high storage density, which is a desirable attribute for any file structure.

Keywords: XML, storage system, query processing, multidimensional file, index structure.

1. Introduction

XML is considered as a new standard for the exchanging of information among various applications on the web[1] [2]. An XML document can be modelled as a rooted and labeled tree. Nodes in the tree represent elements, attributes, or values and edges representing Parent-Child (P-C) relationships between different nodes [3]. For example, the tree representation of the XML document in Figure 1 is shown in Figure 2.

The efficiency of XML query processing is affected significantly by the storage structure of the XML data.

Unlike relational database, querying XML data required values search and structure search. Evaluating structural relationship between XML elements has imposed a great challenge on efficient XML query processing.

XML query processing algorithms can be classified in to two approaches, namely, navigational approach and join-based approach. Navigational algorithms traverse the XML tree and check whether a tree node satisfies the constraints specified by the query [4]. Join-based algorithms [5] [6] [7] go through two steps. In the first they find the XML tree nodes that satisfy the constraints specified by the query, and in the second step, they join the matched nodes according to their structural relationship (e.g. parent-child or ancestor-descendant). Join-based algorithms are more efficient than the navigational-based algorithms especially with large XML datasets. The cost of join-based algorithms is significantly affected by the number of the structural join operation performed. In this paper we propose a new Multidimensional XML Storage Structure (MXSS) which reduces the cost of join-based algorithms by reducing the number of structural join operations. Also, MXSS takes less disk space when with the most popular XML storage structure, inverted-list.

The rest of this paper is organized as follows. Section 2 presents our proposed storage system. Experimental work is discussed in section 3 followed by the conclusion in Section 4.

2. MXSS

In MXSS, each labeled path in an XML-tree is represented as a point in an N-dimensional labeled path space, where N is the length of the longest labeled path in the XML-tree. Each dimension in MXSS represents a level in the XML tree. Thus number of dimensions (N) in MXSS is equal to the number of levels in the XML tree. Structural join operations are responsible for consuming a lion's share of the query processing time [8]. Thus, the main drawback of the existing query processing algorithms is the structural join operations. An effective improvement can be achieved in the query processing time if we can answer the XML query with efficient structural join operations.

The main objective of our research is to find an efficient way of organizing XML data in order to minimize structural joins.

The MXSS multidimensional space is divided into a number of hyper rectangles, called subspaces. Each subspace is either null or contains a pointer to a disk block.. For example, Figure 3 shows the MXSS representation of the XML tree in Figure 2. The labeled path numbers which are displayed inside circles in the multidimensional space are for demonstrational purpose only.

```
<Publishers>
  <Publisher>
    <name> MIT Press </name>
    <address> Cambridge
</address>
    <book>
      <title> database </title>
      <author>
        <name> Tom </name>
      </author>
    </book>
  </Publisher>
  <Publisher>
    <book>
      <title> Life </title>
      <author>
        <name> Smith </name>
        <age> 18 </age>
      </author>
    </book>
    <name> NY press </name>
  </Publisher>
</Publishers>
```

Fig.1 XML document

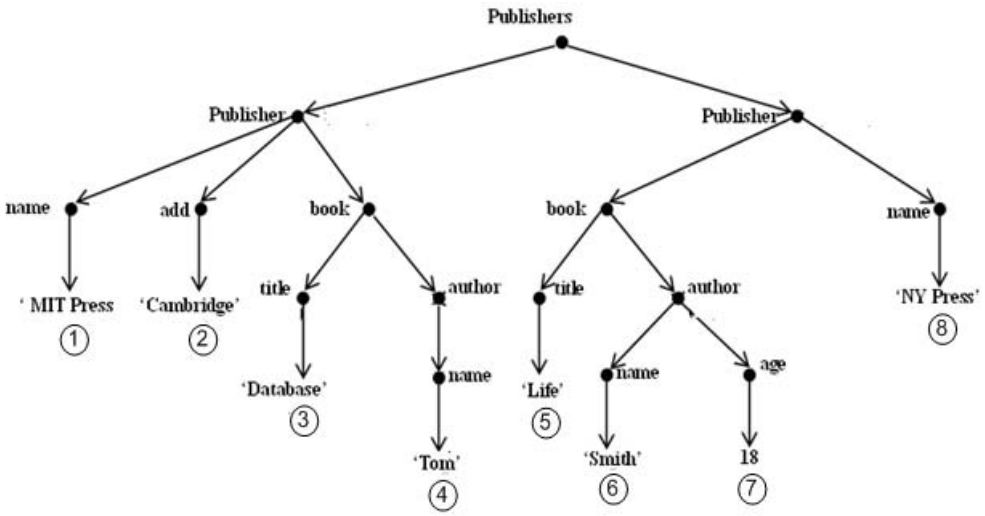

Fig.2 XML Tree

The multidimensional space is based on number of levels in the XML tree and the distinct element tags available in each level. For example Table 1 shows distinct tags in each level of the XML tree shown in Figure 2. The table shows that the XML tree has five levels, so number of dimensions in MXSS is five.

Table1 Distinct elements in each level

Level	**Number of tags**	Tag	Seq. #
0	1	Publishers	0

1	1	Publisher	0
2	3	Name	1
		Add	2
		Book	3
3	2	Title	1
		Author	2
4	2	Name	1
		Age	2

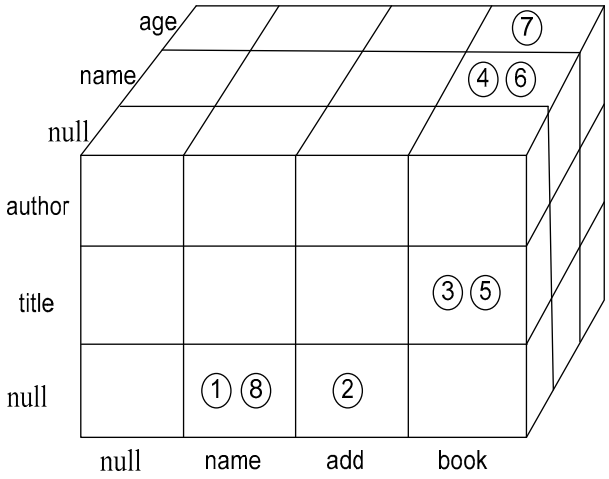

Fig.3 MXSS

The MXSS of Figure 2 is shown in Figure 3. Because levels 0 and 1 each contain one distinct tag, for demonstrational purpose, only dimensions 2, 3 and 4 are displayed in the Figure. The x–axis represents level 2, the y–axis represents level 3 and z-axis represents level 4. Also, the null tag is added to each dimension in case a labeled path has length less than the maximum depth of the XML-tree. For example, the labeledpath "/Publisher/publisher/name" doesn't have tags in levels 3 and 4, so we assume they are null tags. The circled numbers in the bottom of Figure 2 are the labeled path IDs.

MXSS stores XML data based on labeled paths. The multidimensional labeled path space will be used to define the correspondence between the labels paths and disk blocks where the nodes corresponding to these labeled paths are stored. The nodes are stored in a B+tree containing two fields. The first field is the Dewey ID of the node and the second field is the content of the node. MXSS assigns each tag in a dimension a sequence number. The sequence number of the null tag is 0. For example, Table 1 shows the sequence number of each tag in a dimension. So from the level and tag information of each node in a path, MXSS computes the subspace number of a labeled path in the multidimensional space. For example, path number 7 (in Figure 2) has the following elements: "publishers", "publisher", "book", "author" and "age". According to table 1, the (level, sequence number) information of these elements are (0, 0), (1, 0), (2, 3), (3, 2) and (4, 2) respectively. These numbers are used to compute the index of multidimensional subspace of the corresponding labeled path. This multidimensional subspace will store a pointer to the disk block, in which this path will be stored. Since paths are stored according to (level, index) information of their nodes; different paths will have different corresponding multidimensional subspace. On the other hand, paths that have nodes with the same (level, index) information will map to the same multidimensional subspace.

The MXSS is implemented on disk as a UB-tree and the contents and the Dewey IDs are implemented on disk as a B+-tree.

3. Experimental Results

In this section we present the performance analysis MXSS. We compared MXSS with the popular storage structure, inverted-list.

3.1. Datasets used

The datasets we used were the benchmark dataset obtained from University of Washington XML repository. We experimented with six datasets, namely SigmodRecord, Customer, Wsu, Dblp, Orders and Lineitem. These datasets were selected because they have different characteristics, i.e. size, depth.

```
<dblp>
        <mastersthesis key="ms/Brown92">
                <author>Kurt P. Brown</author>
                <title>PRPL: A Database Workload Specification Language, v1.3.</title>
                <year>1992</year>
                <school>Univ. of Wisconsin-Madison</school>
        </mastersthesis>
        <mastersthesis key="ms/Yurek97">
                <author>Tolga Yurek</author>
                <title>Efficient View Maintenance at Data Warehouses.</title>
                <year>1997</year>
                <school>University of California at Santa Barbara </school>
        </mastersthesis>
        <article key="tr/dec/SRC1997-018">
                <editor>Paul R. McJones</editor>
                <title>The 1995 SQL Reunion: People, Project, and Politics, May 29, 1995.</title>
                <journal>Digital System Research Center Report</journal>
                <volume>SRC1997-018</volume>
                <year>1997</year>
                <ee>db/labs/dec/SRC1997-018.html</ee>
        </article>
```

Fig.4 Sample data for Dblp dataset

3.2. Performance Evaluation

In these experiments, we compare MXSS against inverted-list based storage system. Different datasets, described in section 3.1 were used in these experiments. Each dataset was stored twice, first by as MXSS and then as inverted-list. Number of disk blocks used and storage density were used as our performance measures.

3.2.1 Number of disk blocks

For each dataset, MXSS outperformed Inverted-list based storage system in terms of number of disk blocks needed to store the dataset. Figure 5 show the number of disk blocks needed to store all datasets in both storage systems. For example, Inverted-list based storage system used 581 disk blocks to store Orders dataset, whereas MXSS needed 522 disk blocks only.

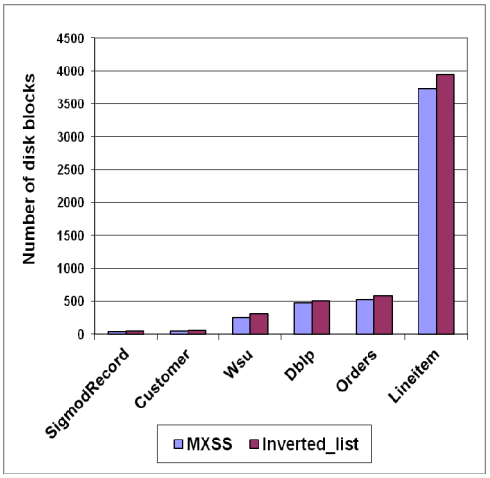

Fig.5 Number of disk accesses in MXSS and Inverted-list systems (all datasets)

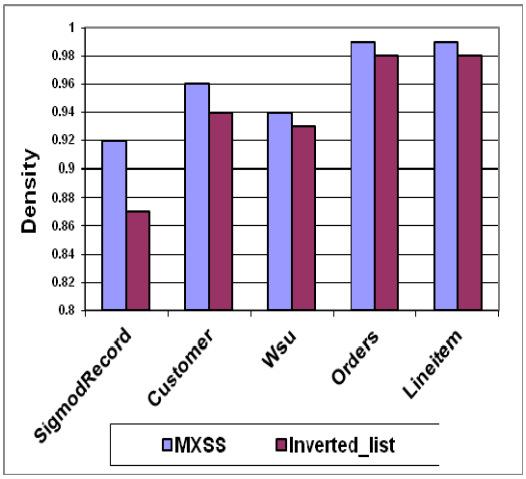

Fig.6 Storage Density

3.2.2. Storage density

This metric measures the number of records stored in specific disk block over the maximum number of records that can be stored in the same disk block. Higher density is more desirable. Figure 6 shows that MXSS storage density is always higher than Inverted-list storage density. For instance, with SigmodRecord dataset, MIXF density was 92% whereas Inverted-list density was 87%.

4. Conclusion

In this paper, we proposed Multidimensional XML storage structure called MXSS. Unlike most inverted list based storage system in which XML data is stored as elements, MXSS stores XML data as labeled paths. MIXF uses multidimensional space structure to define the correspondence between paths and disk block address, in which the corresponding paths are stored. Our experiment results show that MXSS outperforms Inverted-list based storage system in both the number of disk blocks needed to store all the XML datasets and in storage density.

5. Acknowledgements

The authors thank King Fahd University of petroleum and Minerals.

6. References

[1] G. Gou, and R. Chirkova. “XML Query Processing: A Survey”. Technical Report, North Carolina State University, 2005.

[2] D. Chamberlin. “XQuery: An XML Query Language”. IBM Systems Journal, Vol. 41, No. 4, 2002.

[3] XML Tutorial, w3schools.com, http://www.w3schools.com/xml/default.asp.

[4] N. Zhang, V. Kacholia, and M.T. Ozsu, “Succinct Physical Storage Scheme for Efficient Evaluation of Path Queries in XML,” Proc. 20th IEEE Int'l Conf. Data Eng. (ICDE '04), 2004.

[5] C. Zhang, J. Naughton, D. DeWitt, Q. Luo, and G.M. Lohman. “On Supporting Containment Queries in Relational Database Management Systems”. Proceeding of the ACM SIGMOD 2001 International Conferences, 2001.

[6] S. Al-Khalifa, H. Jagadish, N. Koudas, J. Patel, D. Srivastava, and Y. Wu, “Structural Joins: A Primitive for Efficient XML Query Pattern Matching,” Proceeding of the 18th IEEE International Conference on Data Engineering, 2002.

[7] N. Bruno, N. Koudas, and D. Srivastava. “Holistic Twig Joins: Optimal XML Pattern Matching”. Proceeding of the 2002 ACM SIGMOD International Conference, 2002.

[8] Y. Wu, J. Patel, and H. Jagadish. “Structural Join Order Selection for XML Query Optimization”. Proceeding of 19th IEEE International Conference on Data Engineering, 2003.