International Journal of Computer and Communication Technology

Volume 1 | Issue 4 Article 13

October 2010

Evaluating XPath Expressions on Light Weight BitCube

Mrs. Pranali P. Chaudhari
Lecturer, I.T. Department Maharashtra Academy of Engineering, Alandi (D), Pune.,
pranalichaudhari@gmail.com

Evaluating XPath Expressions on Light Weight BitCube

Mrs. Pranali P. Chaudhari
Lecturer, I.T. Department
Maharashtra Academy of Engineering,
Alandi (D), Pune.
Email: pranalichaudhari@gmail.com
Mob: 09822357650

ABSTRACT- XML has become a popular way of storing data and hence has also become a new standard for exchanging and representing data on internet. Many techniques have been proposed for indexing and retrieval of XML documents such as X-Tree, BitCube. In this paper a indexing structure known as Light Weight BitCube is proposed. LWBC is an extension to the earlier BitCube technique which overcomes the memory management problem of BitCube while maintaining the same query processing efficiency as that of BitCube. Many XPath expressions and BitCube operations are evaluated on this LWBC to show the query processing efficiency. The results are also compared with XQEngine, a well known XML Query Processing Engine. The results also show that, Light Weight BitCube manages memory much more efficiently than the BitCube, without compromising on the query processing time.

Keywords: Light Weight BitCube, Bitwise Operations, Indexing XML Documents

1. INTRODUCTION

"Information Retrieval deals with the representation, storage and organization of, and access to information items. The representation and organization of the information items should provide the user with easy access to the information in which he is interested." [4]. Given the user query, the key goal of an IR system is, thus, to retrieve information, which might be useful or relevant to the user.

A majority of traditional business applications, transactional systems and enterprise applications rely on relational databases to maintain their data. But now a days we are having lots of medias such as portals, knowledge based systems, emails etc , for getting information or communication thus a typical organization's enterprise information is no longer maintained as structured data alone. Typically, structured data are the data with a repeated structure that can be easily stored in the data tables of a relational database. Semi-structured databases [1], unlike traditional databases, do not have a fixed schema known in advance. The eXtensible Markup Language (XML) [3] is a commonly used data modeling technique for such data.

The representation of documents in XML paved way for the possibility of content based retrieval. The widespread use of XML in digital libraries, product catalogues, scientific data repositories and across the web prompted the development of appropriate searching and browsing methods for XML documents. As enterprise applications (or, web services) continue to build upon XML, it is critical that they include a search functionality that is fully compatible with XML. In order to optimize query processing, the data need to be organized (indexed) in a way that facilitates efficient retrieval. Without indexes, the database may be forced to conduct a full data scan to locate the desired data record, which can be a lengthy and an inefficient process. There is an urgent need for an XML indexing and retrieval technique that aids in efficient query processing.

The rest of the paper is organized as follows. Section 2 describes some of the related work in this area. Section 3 provides introduction to bitmap index and BitCube, the earlier indexing techniques and some preliminary operations performed on BitCube. In Section 4 the proposed indexing approach is described. In section 5, experimental results on the approach and their comparison with previous approaches are mentioned. Section 6, summarizes the results of the study and draw conclusions and the potential future work in this area.

2. RELATED WORK

The conventional techniques used for document retrieval systems include stop lists, word stems, and frequency tables. The words that are deemed "irrelevant" to any query are eliminated from searching. The words that share a common word stem are replaced by the stem word. A frequency table is a

matrix that indicates the occurrences of words in documents. The occurrence here can be simply the frequency of a word or the ratio of word frequency with respect to the size of a document.

However, the size of frequency table increases dramatically as the size of the document database increases. To reduce frequency tables, the latent semantic indexing (LSI) [5] technique has been developed LSI retains only "most significant" of the frequency table.

Commercial database supports various types of indexes such as B+ trees, hash indexes, signature files, inverted files [1, 2]. These indexing techniques can be evaluated based on access/insertion/deletion time and disk-space needed. The above Indexing techniques, from the database and information retrieval communities, however, are still not satisfactory. It is partly because they cannot scale much beyond their current point to larger collections and partly because semantic and structural equivalencies are not efficiently checked and maintained in the indexes.

Index structures for semi-structured data have been developed in recent years. Examples of such indexes for semi-structured data are XQEngine [10], Dataguides [9], Toxin [7] and ViST[8]. A new data structure, called X-tree [6], has been introduced for storing very high dimensional data.

To overcome the problem of efficiently managing large collections of XML data, a technique called bitmap indexing was proposed by researchers which are used to optimize queries. The collection of bitmaps in a bitmap index [12] forms a 2-dimensional bit matrix. Later a 3-dimensional bit matrix is proposed known as BitCube, [11] on which bitwise logical operations are performed. A BitCube is conceptually defined to store information in the form of bits pertaining to the existence of relationships between documents, paths and words. It supports bit wise operations to handle various types of queries and this is what makes it highly efficient in terms of query processing. In spite of this advantage, it consumes large volumes of memory.

Again an extension to BitCube a indexing structure known as Quasi BitCube[13] was also introduced in which the density of XML document is first calculated and then the document having the highest density is placed at the bottom of the BitCube. This process is known as document ordering. Then by chopping the above zero bits the size of the BitCube is reduced.

In this paper the indexing structure known as Light Weight BitCube is introduced which is similar to the Quasi-BitCube proposed earlier but here no density calculations are involved rather the size reduction is achieved using the BitSet data structure which automatically ignores the top and the bottom zero bits. Light Weight BitCube manages memory much more optimally and at the same time retains the same query processing efficiency of a BitCube. I compare the new indexing time and size with an earlier work that uses BitCube [11]. I have compared the indexing structure with XQEngine [10], which is an open-source native XML database engine, and the traditional approach of indexing. From the retrieval perspective, also a comparison with query processing time of the new index enhancement schemes with BitCube and XQEngine for different types of Xpath expressions is given.

3. PRELIMINARIES

3.1 Bitmap Index:

An XML document is defined as a sequence of ePaths with associated element contents. An XML document database contains a set of XML documents. A bitmap index is 2-dimensional [12]. In a document-ePath bitmap index, a bit column represents an ePath, and a row represents an XML document. XML database can be given as :

d1:
```
<e0>
  <e1>v1</e1>
  <e2>
    <e3>v2 v3 v5 </e3>
    <e4> v3 v8 </e4>
  </e2>
</e0>
```

d2:
```
<e0>
  <e1>v1</e1>
  <e2>
    <e3>v2 v3 v5 </e3>
    <e4> v3 v8 </e4>
    <e5> v6 v7 v8 </e5>
    <e6> v4 </e6>
    <e7> v6 </e7>
  </e2>
  <e8> v7 </e8>
</e0>
```

d3:
```
<e0>
  <e1>v1</e1>
  <e2>
    <e3>v2 v7 </e3>
    <e4> v3 v9 </e4>
    </e5>
  </e2>
  <e9> v5 </e9>
</e0>
```

Figure 1. XML Documents Database

For the above XML database the epaths are defined as :

p0 = e0.e1, p1 = e0.e2.e3, p2 = e0.e2.e4, p3 = e0.e2.e5, p4 = e0.e2.e6, p5 = e0.e2.e7, p6 = e0.e8, p7=e0.e9.

Depending on these epaths for various documents d1, d2, d3 a bitmap index is constructed. If a document has ePath, then set the corresponding bit to 1.Otherwise, all bits are set to 0. Thus bitmap index is given by:

	P	P	P	P	P	P	P	P
d	1	1	1	1	0	0	0	0
d	1	1	1	0	1	1	1	0
d	1	1	1	1	0	0	0	1

Figure 2. Bitmap Index

3.2 BitCube:

A BitCube for XML documents is defined as BitCube = (d, p, v, b), where d denotes XML document, p denotes ePath, v denotes word or content for ePath, and b denotes 0 or I, the value for a bit in BitCube (if ePath contains a word, the bit is set to 1, and 0 otherwise).

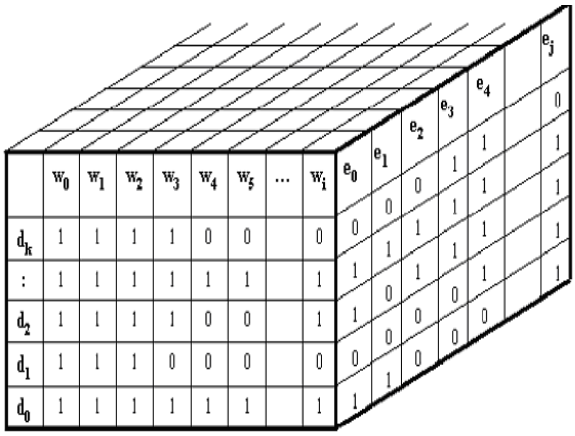

Figure 3. BitCube

Above figure shows a BitCube for k documents. Each document has a set of ePaths.

3.3 BitCube Operations:

There are three operations defined on BitCube. These operations after appling on a BitCube give us a bitmap index.

1. ePath Slice.
2. Word Slice.
3. Document Project.

3.3.1 ePath Slice :

Each bit for a particular ePath is sliced. This operation takes a Path as input and returns a set of documents with words associated with it.

P-Slice(ePath) = {(doc, word) I ePath is used in doc, and word is associated with the ePath}.

The outcome of this slicing is a bitmap index that represents a set of documents with a set of words. Typical web searches may not possible for ePath.

3.3.2. Word Slice :

Each bit for a particular word can be sliced. This operation takes a (search key) word as input and returns a set of documents.

W-Slice(word) = {(doc, ePath) 1 word is associated with the ePath which is in turn used in doc}.

The outcome is a bitmap index that represents a set of documents with a set of ePath with which the word is associated. Typical web searches are based on this word slice operation if they search XML documents.

3.3.3. Document Project :

Each row of a BitCube can be projected. This operation takes a document as input and returns a set of ePaths with words associated with those ePaths.

Project(doc) = {(ePath, word) I entire content and ePath pairs appeared in doc}

The outcome is a bitmap index that represents a set of ePaths with their content (or words). A typical method for this project operation is a web browsing.

4. LIGHT WEIGHT BITCUBE:

As discussed earlier An XML document is defined as a set of (e,w) pairs, where e denotes an element path and w denotes a word in a path. Light Weight BitCube is an extension to BitCube technique that retains many of its powerful features and at the same time has a lot more structural advantage. The main advantage of a BitCube (Light Weight BitCube) lies in its high-speed query processing ability.

Construction of Light Weight BitCube consists of three main steps: i. Construction of Bitmap Index, ii. Construction of BitCube, iii. BitCube Reduction. The main system architecture of constructing a Light Weight BitCube is shown below:

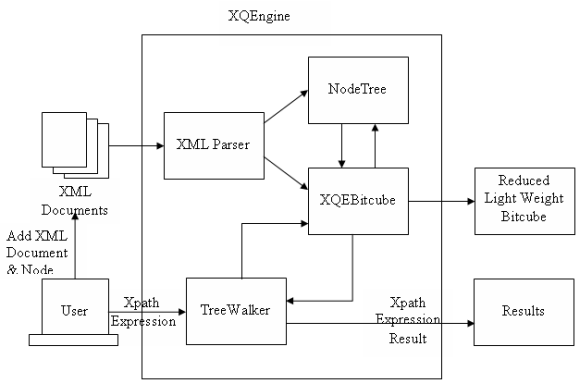

Figure 4. System Architecture

The BitCube structure is sparse. There is a documents bit vector (dbv) for each (e,w) pair. The bit is SET if the (e,w) pair exists in the corresponding document and RESET otherwise. In a real-time environment, it is very unlikely that a document will contain all the paths and the words contained in all the other documents. This means that numerous bits towards the top of each documents bit vector are consuming space and are superfluous (see Figure 5).

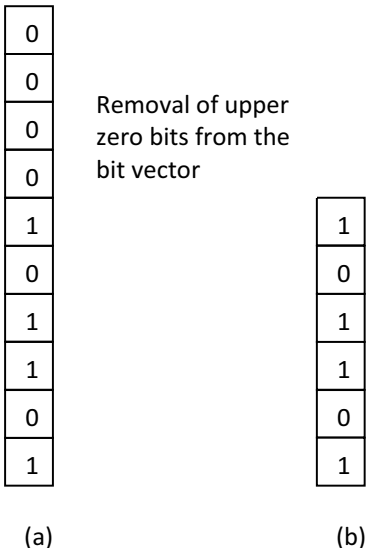

Figure 5. a. Original Bit Vector b. Reduced Bit Vector after removal of upper zero bits

The main characteristics of Light Weight BitCube index structure can be summarized as follows:
i. Simple structure
ii. Memory efficient structure
iii. Efficient query processing due to bit-wise operations

5. EXPERIMENTAL RESULTS

5.1 Experiment I (Space Measurement):

This experiment was performed to measure the effectiveness of the Light Weight BitCube index structure by comparing its memory size and index time with that of the BitCube for different data sets.

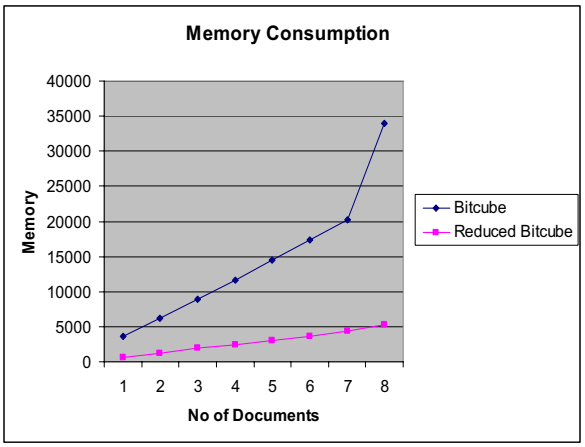

Figure 6. Memory Consumption

The index times of both the structures were about the same. Thus, when compared to BitCube, the Light Weight BitCube structure saves significant amount of index memory without compromising on the indexing time.

5.2 Experiment II (Query Processing Efficiency Measurement)

This experiment was performed to measure the retrieval efficiency of the Light Weight BitCube index structure by comparing it with BitCube and XQEngine for different types of query operations.

The query processing time for two different operations: word slice, path slice is compared. A path slice takes a path as input and returns a set of documents with words associated with the given path. A word slice takes a word as input and returns a set of documents with paths associated with the given word.

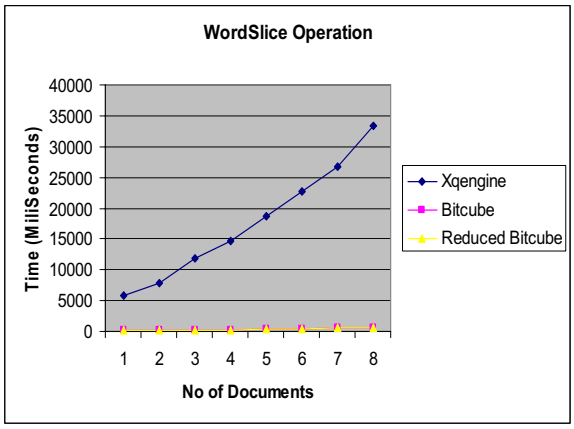

Figure 7. Word Slice Comparisons

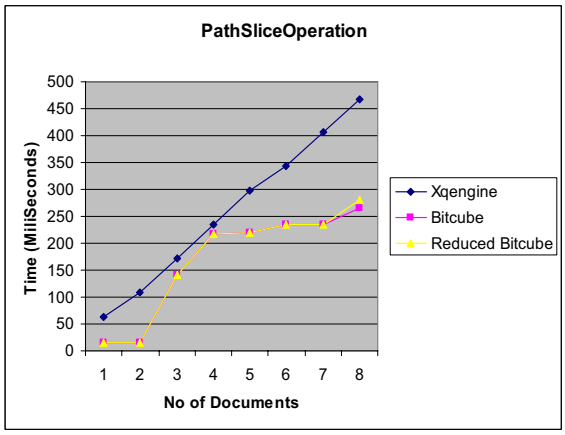

Figure 8. Path Slice Comparisons

The results show that the query processing using Light Weight BitCube is at least as efficient as BitCube for all the two types of query operations. The average word slice time of Light Weight BitCube and BitCube remains constant (0 ms) with the increase in the size of the document collection.

In the case of XQEngine, the word and path slice (equivalent Xpath expressions) times increase linearly with the increase in the size of the document collection, which shows that the query performance of Light Weight BitCube is significantly better than that of XQEngine.

5.3 Experiment III (XPath Expression Evaluation)

The test cases given below are performed for testing the performance of Light Weight BitCube. Following test cases are tested on eight xml files each greater than 3 MB size. Various Xpath expressions are evaluated and the time for processing those expressions, in milliseconds is observed for both XQEngine and Light Weight BitCube. Also some additional Xpath expressions are included which are not supported by XQEngine.

Test Case No.	Xpath Expression	XQEngine Execution Time (ms)	Light Weight BitCube Execution Time (ms)
1.	/data/MPROP/BASEMENT	344	265
2.	/data/MPROP[BASEMENT ='F']	20094	469
3.	/data/MPROP[BASEMENT='F' or BASEMENT='P']	Not Supported	375
4.	/data/MPROP[BASEMENT='F' and BASEMENT='P']	Not Supported	360
5.	Document Project	Not Supported	05
6.	Word Slice	Not Supported	31
7.	Path Slice	Not Supported	05

Table 1. Xpath Expression Evaluation Comparison

6. CONCLUSION AND FUTURE WORK

Light Weight BitCube, a memory efficient indexing scheme extended from BitCube, is proposed in this paper. Since the information stored is in the form of bits, the entire index structure fits into the main memory and hence I/O operations are no longer a concern during information retrieval. Results show that Light Weight BitCube manages memory much more effectively and at the same time retains the same query processing efficiency of a BitCube. The execution time of Light Weight BitCube for different query operations is much more efficient than XQEngine.

In this paper XQEngine version 0.69 is used. Also, there is a growing demand of XML in the areas relating to XLinks, XPointers and Security. Light Weight BitCube, in its current form, does not take into account relationships (parent-child, sibling, etc.) between paths. As a result, the current structure cannot support locating relative location paths. In future, this index structure can be extended to support these complex querying operations.

REFERENCES:

[1] Silberschatz, Korth, Sudarshan, Database System Concepts, 4th Edition, Mc Graw Hill, 2002.

[2] Ramakrishnan, Gehrke, Database Management Systems, 3rd Edition, Mc Graw Hill, 2003.

[3] T. Bray et al., "Extensible Markup Language (XML) 1.0 (Second Edition)," W3CRecommendation, 10 Nov. 2004; http://www.w3.org/TR/REC-xml.

[4] R. Baeza-Yates and B. Ribeiro-Neto, Modern Information Retrieval, Addison Wesley, 1999.

[5] C. Papadimitriou, H. Tamaki, P. Raghavan, S. Vempala, "Latent Semantic Indexing: A Probabilistic Analysis," In Proc. of 17th ACM Symp. on Principles of Database Systems, Seattle, WA, (1998) pp. 159-168.

[6] S. Berchtold, D. A. Keim, and H. P. Kriegel, The Xtree: An Index Structure for High-Dimensional Data, Proc. Intl. Conf. On Very Large Data Bases, Bombay, India, 1996, 28-39.

[7] F. Rizzolo and A. Mendelzon, "Indexing XML Data with ToXin," Proc. 4th Int'l Workshop Web and Databases, Springer-Verlag, 2001, pp. 49-54.

[8] H. Wang et al., "XML Indexing and Compression: ViST: A Dynamic Index Method for Querying XML Data by Tree Structures," Proc. ACM SIGMOD Int'l Conf. Management of Data, ACM Press, 2003, pp. 110-121.

[9] R. Goldman, J. Widom, "DataGuides: Enabling Query Formulation and Optimization in Semi-structured Databases", In Proc. of the Intl. Conference on Very Large Databases, Athens, Greece, (1997), pp. 436-445.

[10] H. Katz, "XQEngine," Apr. 2005; http://sourceforge.net/projects/xqengine/.

[11] J. Yoon, V. Raghavan and Venu Chakilam, BitCube: A Three Dimensional Bitmap Indexing for XML Documents, Thirteenth International Conference on Scientific and Statistical Database Management, FairFax, VA, 2001.

[12] J. Yoon, V. Raghavan and Venu Chakilam, Bitmap Indexing-based clustering & retrieval of XML documents, IEEE Trans. on Knowledge and Data Engg.

[13] B. Shah, A. Gummadi, J. Yoon, and V. Raghavan, "Efficient Dynamic Indexing and Retrieval of XML Documents Using Three-Dimensional Quasi-BitCube," Proc. First Int'l Workshop High Performance XML Processing, 2004.