XML Retrieval More Efficient Using Compression Technique

Tanakorn Wichaiwong and Chuleerat Jaruskulchai

Abstract— **In this paper, we report experimental results of our approach for retrieval large-scale XML collection, to improve both efficiency and effectiveness of XML Retrieval. We propose new XML compression algorithm that allows supporting Absolute Document XPath Indexing and Score Sharing Algorithm by a Top-Down Scheme approach. It has been discovered that these steps reduce the size of the data down by 91.87 % compare to GPX, and reduce the length of Score Sharing processing time down to 44.18% when compared to before the compression. In terms of processing time, our system required an average of one second per topic on INEX-IEEE and an average of ten seconds per topic on INEX-Wiki better than GPX system. In addition, we explain the comprehensive description of our XML retrieval system, with performance experiments on large-scale corpora on INEX collections.**

Index Terms— **XML Retrieval, Compression Strategies, Ranking Strategies**

I. Introduction

THE widespread use of Extensible Markup Language (XML) [1] documents in digital libraries has led to the development of information retrieval (IR) methods specifically designed for XML collections. Most traditional IR systems are limited to the whole document retrieval; however, since XML documents separate content and structure, XML-IR systems are able to retrieve the relevant portions of documents. This means that users who interact with an XML-IR system potentially receive highly relevant and precise material.

Recently, the Initiative for the Evaluation of XML Retrieval (INEX) [2] has provided an excellent test corpus on XML information retrieval and queries [3]. The corpus contains marked up with context, and queries included articles from IEEE journals and Wikipedia. There are two main performance issues in Information Retrieval; effectiveness and efficiency. In the past, much research was mainly aimed to improve only effectiveness. In recent years, research has been focused on the efficiency of the trend of retrieval large-scale collection. In this paper, we present our approach toward improving the efficiency by using compression technique.

This paper is organized as follows; Section 2 reviews related works. Section 3 explains the implementation of our system overview and new XML compression algorithm. Section 4 show the experiment, conclusions and further work are drawn in Section 5.

Manuscript received January 20, 2011; revised February 07, 2011. This work was supported in part by the Graduate School of Kasersart University.

Wichawong T. is with the Kasetsart University, Bangkok, Thailand (phone: 6687-696-1333; e-mail: g5184041@ ku.ac.th).

Jaruskulchai C. is with the Kasetsart University, Bangkok, Thailand (e-mail: fscichj@ku.ac.th).

II. Related Works

A. XML Compression Schemes

Recently, researches on XML data compression stress the reduction of XML data size. Each method has its own techniques. XML data compression can be divided into three types: 1) data compression 2) tag compression and 3) data and tag compression. Several compression strategies have been developed in XML as follows;

XMill [4] is a technique which compresses both data and tag in order to reduce the size by starting with separating the tag, which is composed of elements and attributes, from the data, which is a character. After that, the data groups' relationships will be organized. The same data will be in the same group. The next step is the data compression by using gzip [5] so that the data will come out in the same file since grouping requires understanding of the data definitions which depend on the application type. XMill allows user check the data definition. The disvantage of XMill that data cannot be search through the compressed data. However, XMill is the first research that made researchers realizes the importance of the problem and how to solve it in XML data compression. Data that has been compressed is not in the form of XML schema structure.

XGrind [6] is a technique which compressed data and tag but the user can still search for data after the compression. This qualification results from the fact that the compressed data still maintain the structure of the old data. However, XGrind will compress only XML data that has DTD structure so some data set that does not have DTD will result in having the user waste time in creating DTD for XML data set that they wanted to compress.

XPRESS [7] uses the technique in compressing both the data and the tag. Its advantages are the same as XGrind: it can search for the data after the compression. Nevertheless, XPRESS does not use DTD. In addition, XPRESS presented a new idea which uses reverse arithmetic encoding, which is a method in organizing data so that the search for XPath expressions can be done effectively. Furthermore, XPRESS has developed the search of data type without having to use the information from users. However, the use of XPRESS is limited because XPRESS cannot understand documents that use ID and IDREF. It also does not have a way to decompress data back into normal XML.

XPACK [8] is a way to compress XML data, which uses grammatical approaches in XML data compression and decompresses. The main component of XPACK is the Grammar Generator, which creates the grammar. The second component is the Compressor which compresses the data. The last component is the Decompressor which

ISBN: 978-988-18210-3-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

decompresses the compressed data by using the old structure of the data. However, XPACK cannot manage XML data that has mixed content element (which is an element composed of element and characters), limiting users to search for data in compressed XML.

XSchemaTag [9] is a technique that compresses only XML tag and that technique still enables to search and maintain documents because the data is already in the form of XML. The quality comes from compressed data, which has the old data structure. However, with XSchemaTag scheme not take into account of the frequency of tag occurrences and the counter of tag position.

The GPX [10] search engine is using a relational database implement an inverted list data structure. It is a compromise solution provides the convenience of a DBMS at the cost of somewhat reduced performance, which may otherwise be possible. For example, the XPath as following:

/article[1]/bdy[1]/sec[5]/p[3]

This could be represented by two expressions, a Tag-set and an Index-set as below;

Tag-set: /article/bdy/sec/p
Index-Set: 1/1/5/3

The original XPath can be reconstructed from the tag-set and the index-set. The GPX assigns to each tag set and each index-set a hash code and create auxiliary database tables mapping the hash codes to the corresponding tag-set and index-set entries. These hash tables are small enough to be held in memory and so decoding is efficient. The GPX takes 15 seconds to load all table data and takes an average of 7.2 seconds per topic. Sometimes, it takes longer than 30 seconds, depending on the type of query on a 3GHz PC with 2 GB RAM. Unfortunately, this method has not been focused on the efficiency.

The relative inverted-path list (RIP list) [11], the list contains all the structure information and has uniqueness in preorder of XML nodes, which are traversed in depth first order. The list adopts all the distances between nodes and their child node as follows;

Node ID: {Distance, Term Frequency}

The RIP list has high accessibility to the parent node ID from each Node ID. The RIP merges the numbers of terms contained in every node, the scores of retrieved nodes and the numbers of query terms contained in each node.

III. XML Retrieval Model

A. Our System Overview (XMLIR)

Our system uses a relational DBMS as a storage back end and query processing methods are based on Full-Text Search (FTS). In the following, we discuss the schema setup using MySQL [12] engine generally available release: 5.1.51. In figure 1, depicts the overview of XML retrieval system. For the initial step, we consider a simplified XML data model, but disregarding any kind of Meta markup including comment, link in the form of XLink or ID/IDRef and attributes. The main components of the XML retrieval system are including;

1. The ADXPI Indexer, when new documents are entered, the Indexer parses and analyzes the tag and content data to build the list of leaf-nodes.
2. The cADXPI Compressor that analyzes the tag and counter to build the structure index store in MySQL database.
3. The ADXPI Indexer will get all of structure index to construct the leaf-node index store in MySQL database.
4. Score Sharing Algorithm, which allows assigning the parent scores by sharing score from leaf node to their parents by a Top-Down Scheme approach.

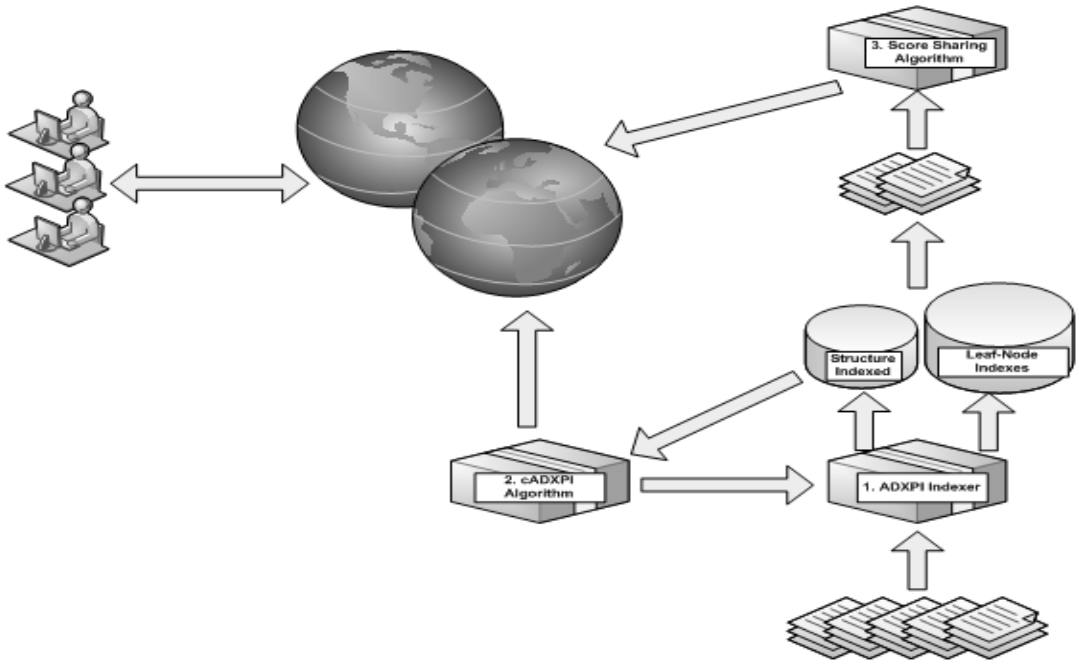

Figure 1. XML Retrieval System Overview

ISBN: 978-988-18210-3-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

B. Absolute Document XPath Indexing

In previous reports [13], a single inverted file can hold the entire reference list, while the suitable indexing of terms can support the fast retrieval of the term-inverted lists. To control overlap and reduce the cost of Joined on DBMS, we used the Absolute Document XPath Indexing (ADXPI) scheme to transform each leaf element level into a document level. For instance, take a document named x1.

```
<?xml version="1.0"?>
<article>
 <title>xml</title>
 <body>
    <section>
    <title>xml</title>
    <p>information</p>
    <p>retrieval</p>
    </section>
 </body>
</article>
```

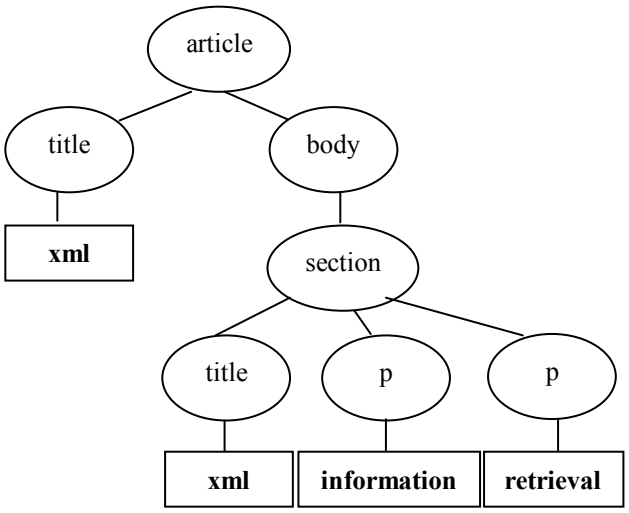

Figure 2. The Example of XML Element Tree

Figure 2 depicts the example of the XML element trees then we can build an index by ADXPI expression identifies a leaf XML node that has text contain within the document, relative to document and their parents are following;

x1/article[1]/title[1]: “xml”
x1/article[1]/body[1]/section[1]/title[1]: “xml”
x1/article[1]/body[1]/section[1]/p[1]: “information”
x1/article[1]/body[1]/section[1]/p[2]: “retrieval”

C. Compression of ADXPI Algorithm

The representation of the ADXPI is more problematic, because each unique XPath is repeated in the inverted list for each term in the same node, and the XPath repeated in many files. We find out the way to encoded tags and the compression algorithm like XMill might be effective, but we considered this again to be unnecessary, particularly given the processing overheads. We have adopted the following simple compression scheme using Dictionary Mapping and easy to reconstruct the original XPath. Finally, the database schema consists of the following tables and adding FTS index to LeafNode.Details and figure 3 depicts the example of data store in MySQL table as follows;

```
CREATE TABLE LeafNode (
 ID int(11) NOT NULL AUTO_INCREMENT,
 xPath varchar(1000) DEFAULT NULL,
 Details text,
 PRIMARY KEY (ID),
 UNIQUE KEY id (ID),
 FULLTEXT KEY Details (Details)
) ENGINE=MyISAM DEFAULT CHARSET=latin1$$;
CREATE TABLE Structure (
 sID int(11) NOT NULL,
 sName varchar(500) DEFAULT NULL,
) ENGINE=MyISAM DEFAULT CHARSET=latin1$$;

 Counter :=0;
 For each List in LeafNodeList
    For each Path in List.Split('/')
        If NodeList.ContainKey(Path) Then
            NodeList[Path] ← NodeList[Path] +1;
        Else
            NodeList.Add(Path,1);
        End If
    End For
 End For
 NodeList ← NodeList.SortbyValue();
 For each Path in NodeList
    FinalList.Add(Path,Counter);
    Counter ← Counter + 1;
 End For
 Return FinalList;
```

Figure 3. The details of Compression Algorithm

In figure 3, depicts the details of a compression algorithm and in the following algorithm description, indentation is used to denote the details of algorithm processing:

1. Fetch all leaf node entries from the collection list.
2. For each list, create data structure to store tag name and frequency, we call Dictionary<tag,freq> data type.
3. Split all tag and counter from the leaf and add to Dictionary<tag,freq>, for instance, the leaf node is: /article[1]/body[1]/section[1]/p[1].
   We can split them as follows;
   1st tag is “article[1]”, frequency is 1.
   2nd tag is “body[1]”, frequency is 1.
   3rd tag is “section[1]”, frequency is 1.
   and “p[1]”, frequency is 1.
4. For each tag has to check in Dictionary<tag,freq> list as follows;
   If Dictionary<tag,freq> has contain tag then freq is accumulate by freq = freq + 1
   Otherwise add new tag and 1 to Dictionary<tag,freq> list.
5. When already processed all of a list from 2 then create the Final Dictionary<tag,map> list by sorting freq from Dictionary<tag,freq>list. The map is a sequence of tag in Final list.
6. Return Final Dictionary<tag,map> list to store in DB.

ISBN: 978-988-18210-3-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Remind to our example, the compression algorithm processing is following;

x1/article[1]/title[1]: "xml"
x1/article[1]/body[1]/section[1]/title[1]: "xml"
x1/article[1]/body[1]/section[1]/p[1]: "information"
x1/article[1]/body[1]/section[1]/p[2]: "retrieval"

We can split all leaf-node and construct the dictionary list as follows;

1st tag is "article[1]", frequency is 4.
2nd tag is "title[1]", frequency is 2.
3rd tag is "body[1]", frequency is 3.
4th tag is "section[1]", frequency is 3.
5th tag is "p[1]", frequency is 1.
6th tag is "p[2]", frequency is 1.

Following the result list as above, we sort the dictionary list by frequency than the final dictionary with map as follows;

1st tag is "article[1]", frequency is 4.
2nd tag is "body[1]", frequency is 3.
3rd tag is "section[1]", frequency is 3.
4th tag is "title[1]", frequency is 2.
5th tag is "p[1]", frequency is 1.
6th tag is "p[2]", frequency is 1.

As a result, the indexes of leaf-node and structure store in DB as below;

The Leaf-node indices:
x1/1/4: "xml"
x1/1/2/3/4: "xml"
x1/ 1/2/3/5: "information"
x1/1/2/3/6: "retrieval"

The Structure indices;
1 : article[1]
2 : body[1]
3 : section[1]
4 : title[1]
5 : p[1]
6 : p[2]

D. Leaf-Node Scoring Scheme

The Leaf-Only indexing is closest to traditional information retrieval since each XML node is a bag of words of itself, and can be scored as ordinary plain text document then we calculate the leaf element score of its context using Vector Space Model of MySQL Full Text Search as following;

$$LeafScore(e,Q) = \sum_{t \in Q} \mathrm{Q_t} * \mathrm{W_t} * \mathrm{L_t} \qquad (1)$$

$$L_t = \log\left[\frac{(tf_t + 1)}{len(e)}\right] * \frac{U}{(1 + 0.0115 * U)}$$

$$W_t = \log\left[\frac{N - e_t}{e_t}\right]$$

$$Q_t = Qtf_t$$

Note that;
LeafScore(e, Q) measures the relevance of element e to a query Q.
W_t is the inverse element frequency weight of a term t.
tf_t is the frequency of a term t occurring in an element e.
len(e) is the length of an element e.
U is the number of unique terms in element e.
N is the total number of an element in the collection.
e_t is the total element of a term t occur.
Qtf_t is the frequency of a term t occurring in a query Q.

E. Score Sharing Function

In previous reports [14], we compute the scores of all elements in the collection that contain query terms. We must consider the scores of elements by accounting for their relevant descendents. The scores of retrieved elements are now shared between leaf node and their parents in the document XML tree according to the following scheme.

$$Score(PNode) \leftarrow Score(PNode) + [(LeafScore) * \beta^n] \qquad (2)$$

Note that;
PNode is a current parent node.
β is tuning parameter.

If {0 – 1}, then preference is given to the leaf node over the parents.

Otherwise, preference should be given to the parents.

n is the distance between the current parent node and the leaf node.

IV. Experiment Setup

In this section, we present and discuss the results that were obtained at INEX collections. We performed with the Wikipedia collection. This experiment was done on Intel Pentium Dual-Core 1.87 GHz with the memory of 1 GB, Microsoft Windows XP Professional and using Microsoft Visual C#.NET 2008 system on MySQL engine generally available release: 5.1.51.

A. INEX Collection Tests

The document collections are following the INEX-IEEE document collection contains total of 16,819 articles from 24 IEEE Computer Society journals, covering the period of 1995-2004 and totaling 764 megabytes in size and 11 million elements in its canonical form. The Wikipedia XML Corpus of the English Wikipedia in early 2006 [15] that contains 659,338 Wikipedia articles and the total size is 4.6 GB without images and 52 million elements. On average an article contains 161.35 XML nodes, where the average depth of a node in the XML tree of the document is 6.72. Indexing these collections took between 5 minutes for INEX-IEEE and 60 minutes for INEX-Wikipedia. After that, our system uses the index in experiments.

B. INEX Evaluations

As for INEX-IEEE effectiveness, we refer to the relative and absolute precision values as well as the non-interpolated mean average precision (MAP), which displays absolute (i.e., user-perceived) precision as a function of absolute recall using official relevance assessments provided by INEX. Furthermore, the following, more sophisticated and XML-specific metrics were newly introduced for the INEX-IEEE benchmark. The normalized extended Cumulated Gain

ISBN: 978-988-18210-3-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

(nxCG) metrics are an extension of the Cumulated Gain metrics that consider the dependency of XML elements (e.g., overlap and near-misses) within an evaluation.

As for INEX-Wikipedia effectiveness [16], we refer to the main ranking of INEX competition based on iP[0.01] instead of the overall measure MAiP, allowing us to emphasize precision at low recall levels.

Our experiment targets CO Task only as well as systems that accept CO queries. Note that CO queries are terms enclosed in the <title> tag. Then, only the Focused Task remains in the INEX during the period 2005-2008. Thus, the system is evaluated only using Focused Task according to the inex_eval and EvaJ tools provided by INEX.

In the experiment of data compression, the effectiveness in data compression is the proportion of compression, which can be found by using;

$$Size = 1 - \left[\frac{Compressed\ data\ size}{Actual\ data\ size}\right] \quad (3)$$

And the effectiveness of response time is the proportion which can be found by using;

$$Time = 1 - \left[\frac{Processing\ time\ in\ compressed}{Processing\ time\ in\ actual}\right] \quad (4)$$

C. Experiment Results

In this section, we present the results of evaluation of the Score Sharing scheme with and without cADXPI technique. Although, in principle, any XML document part can be retrieved, some document parts tend to be more likely to be relevant. Table II and Table III show the distribution of elements over tag-names and counter. In this case, the most frequently are mapping to the short number of compression method.

As shown in Table I and figure 4, the uses of cADXPI compression technique reduces the data size down by 91.87 % compare to GPX system, and reduce the length of Score Sharing processing time down by 44.18% when compared to before the compression as the show in Table VI and figure 5. We are using the appropriate parameter base on INEX measure for Focused Task at iP[0.10]. The total number of leaf node is 2,500 to compute the sharing score and the parameter for β is 0.10 then we report the effectiveness of our system for 29 topics of INEX 2005, 114 topics of INEX 2006, 99 topics of INEX 2007 and 70 topics of INEX 2008 as shown in Table IV and Table V.

TABLE I. COMPARE DATA SIZE AFTER COMPRESSION

Collections	Size (MB)			%
	GPX	ADXPI	cADXPI	
INEX-IEEE	2,048	629.91	579.35	71.71
INEX-Wikipedia	15,360	1,910.87	1,248.61	91.87

TABLE II. DISTRIBUTION OF TOP 10 ELEMENTS IN INEX-IEEE

Elements	INEX-IEEE	
	Frequency	Tag Mapping
article[1]	1,494,676	0
bdy[1]	1,370,545	1
sec[3]	317,390	2
sec[4]	262,472	3
sec[2]	258,482	4
p[1]	240,679	5
st[1]	239,648	6
ss1[1]	219,646	7
ss1[2]	215,361	8
sec[5]	173,315	9

TABLE III. DISTRIBUTION OF TOP 10 ELEMENTS IN INEX-WIKIPEDIA

Elements	INEX-Wikipedia	
	Frequency	Tag Mapping
article[1]	6,360,427	0
body[1]	5,704,185	1
section[1]	1,863,653	2
title[1]	1,545,969	3
section[2]	1,225,624	4
p[1]	1,066,149	5
section[3]	717,764	6
name[1]	656,295	7
p[2]	549,743	8
section[4]	420,855	9

TABLE IV. THE EFFECTIVENESS ON INEX-IEEE FOCUSED TASK OVERLAP=OFF, QUANT=GEN

TOPIC	nxCG@5	nxCG@10	nxCG@25	nxCG@50
2005	**0.2508**	0.1910	0.1603	0.0864

TABLE V. THE EFFECTIVENESS ON INEX-WIKI FOCUSED TASK

TOPIC	iP[0.00]	iP[0.01]	iP[0.05]	iP[0.10]	MAiP
2006	0.5580	**0.5126**	0.4072	0.3389	0.1290
2007	0.4800	**0.4169**	0.3186	0.2539	0.0987
2008	0.6838	**0.5740**	0.4262	0.3411	0.1187

ISBN: 978-988-18210-3-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

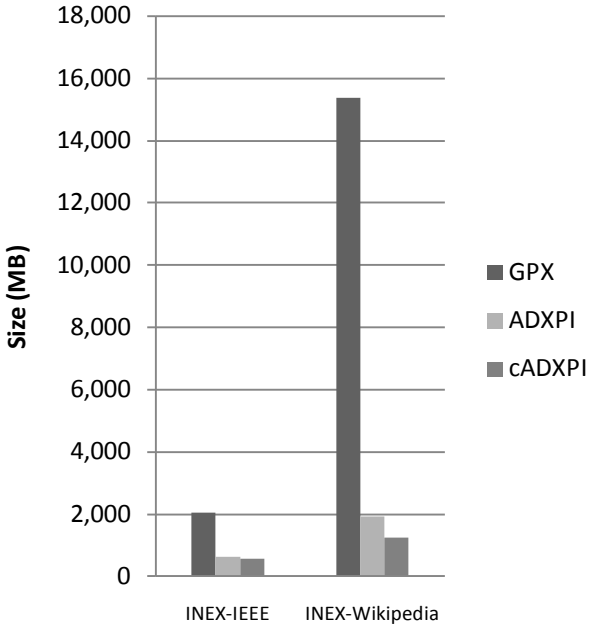

Figure 4. Graph showing the size of data

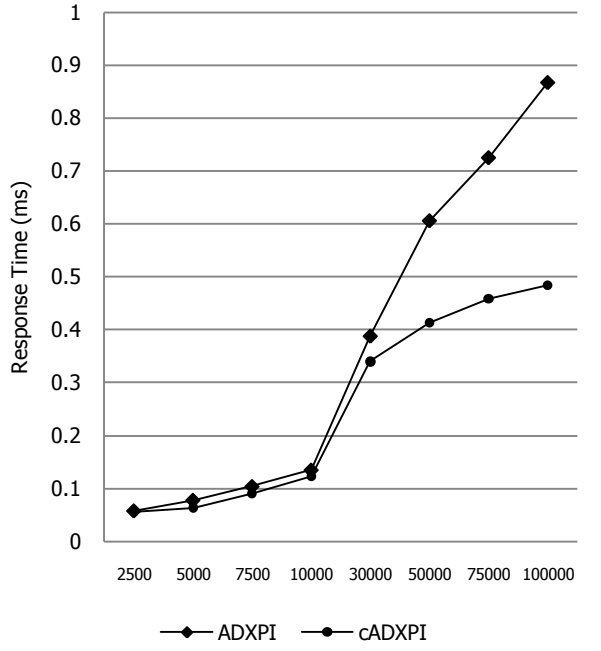

Figure 5. Graph showing the processing time

TABLE VI. COMPARE SCORE SHARING PROCESSING TIME (B=0.10)

Number of Leaf Node (N)	Response Time (ms)			%
	ADXPI	cADXPI	Decompress	
2500	0.058	**0.041**	**0.015**	**3.45**
5000	0.078	**0.043**	**0.021**	**17.95**
7500	0.104	**0.067**	**0.023**	**13.46**
10000	0.135	**0.092**	**0.031**	**8.89**
30000	0.388	**0.122**	**0.218**	**12.37**
50000	0.606	**0.147**	**0.266**	**31.85**
75000	0.725	**0.183**	**0.268**	**36.69**
100000	0.867	**0.216**	**0.276**	**44.18**

V. CONCLUSIONS

In this paper, we propose new XML compression algorithm that allows supporting ADXPI indexing and score sharing function by a Top-Down Scheme approach, and a comprehensive description of our system, with performance experiments on large-scale corpora on INEX collections. It has been discovered that these steps reduces the size of the data down by 91.87 % compare to GPX, and reduce the length of score sharing processing time down by 44.18% when compared to before the compression. In terms of processing time, our system required an average of 1 second per topic on INEX-IEEE and an average of 10 seconds per topic on INEX-Wikipedia better than GPX system.

As our future work, we are going to study how to infer structural hints from CAS queries and experiment more deeply on INEX 2009 collection.

REFERENCES

[1] Extensible Markup Language (XML) 1.1 (Second Edition). http://www.w3.org/TR/xml11/

[2] INitiative for the Evaluation of XML Retrieval (INEX). http://www.inex.otago.ac.nz/

[3] Geva, S. et al,. 2009. Overview of INEX 2009 Ad Hoc Track. The INEX 2009 Workshop Pre-proceeding. Schloss Dagstuhl, Germany, pp. 16-50.

[4] Liefke H. and Suciu D., "XMill: an Efficient Compressor for XML Data.," In Proceeding of the 2000 ACM SIGMOD International Conference on Management of Data, pages 153-164, May 2000.

[5] Gailly J. L. and Adler M., "gzip: The compressor data," Available at http://www.gzip.org/

[6] Tolani P. M. and Haritsa J. R., "XGRIND: A Query-friendly XML Compressor.," In Proceedings of 18th International Conference on Databases Engineering, February 2002.

[7] Min J.-K., Park M.-J., and C Chung.-W., "XPRESS: A Queriable Compression for XML Data.," In Proceeding of the 2003 ACM SIGMOD International Conference on Management of Data, pages 122-133, June 9-12, 2003.

[8] Maireang K. and Pleurmpitiwiriyavach C., "XPACK: A Grammar-based XML Document Compression," In Proceeding of NCSEC2003 the7th National Computer Science and Engineering Conference, October 28-30, 2003.

[9] Wichaiwong T. and Jaruskulchai C., "Improve XML Web Services' Performance By Compressing XML Schema tag," The 4th International Technical Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, Chiang Rai, Thailand, May 9-12, 2007.

[10] Geva, S. 2005. GPX - Gardens Point XML Information Retrieval INEX 2004. In: Fuhr, N., Lalmas, M., Malik, S., Szlavik Z. (eds.): Advances in XML Information Retrieval: Third International Workshop of the Initiative for the Evaluation of XML, Springer, Lecture Notes in Computer Science LNCS, pp. 211-223.

[11] Tanioka, H. 2008. A Fast Retrieval Algorithm for Large-Scale XML Data, Focused Access to XML Documents, LNCS, Vol. 4862, Springer-Verlag, pp. 129–137.

[12] MySQL Full-Text Search Functions, Available at http://dev.mysql.com/doc/refman/5.1/en/fulltext-search.html

[13] Wichaiwong T. and Jaruskulchai C., "XML Retrieval More Efficient Using ADXPI Indexing Scheme," The 4th International Symposium on Mining and Web, Biopolis, Singapore, March 22-25, 2011.

[14] Wichaiwong T. and Jaruskulchai C., "A Simple Approach to Optimize XML Retrieval," The 6th International Conference on Next Generation Web Services Practices, Goa, India, November 23-25, 2010.

[15] Denoyer L. and Gallinari P., 2006. The Wikipedia XML Corpus. SIGIR Forum, pp. 64–69.

[16] Kamps, J. Pehcevski, J. Kazai, G. Lalmas, M. and Robertson, S. 2007. INEX 2007 evaluation measures. In Focused access to XML documents.

ISBN: 978-988-18210-3-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)