XML COMPACTION IMPROVEMENTS BASED ON BINARY STRING ENCODINGS

Ramez Alkhatib

Department of Computer Technologies, Hama University, Hama, Syria

ABSTRACT

Due to the flexibility and the easy use of XML, it is nowadays widely used in a vast number of application areas and new information is increasingly being encoded as XML documents. Therefore, it is important to provide a repository for XML documents, which supports efficient management and storage of XML data. For this purpose, many proposals have been made, the most common ones are node labeling schemes. On the other hand, XML repeatedly uses tags to describe the data itself. This self-describing nature of XML makes it verbose with the result that the storage requirements of XML are often expanded and can be excessive. In addition, the increased size leads to increased costs for data manipulation. Therefore, it also seems natural to use compression techniques to increase the efficiency of storing and querying XML data. In our previous works, we aimed at combining the advantages of both areas (labeling and compaction technologies), Specially, we took advantage of XML structural peculiarities for attempting to reduce storage space requirements and to improve the efficiency of XML query processing using labeling schemes. In this paper, we continue our investigations on variations of binary string encoding forms to decrease the label size. Also We report the experimental results to examine the impact of binary string encoding on the query performance and the storage size needed to store the compacted XML documents.

KEYWORDS

*XML Compaction, XML Labeling, XML Storage, B*inary encoding

1. INTRODUCTION

The ability to efficiently manage XML data is essential because the potential benefits of using XML as a representation method for any kind of data. There have been many proposals to manage XML documents. However, XML Labeling and compaction techniques are considered as two major approaches able to provide robust XML document storage and manipulation. Since the logical structure of an XML document is an ordered tree consisting of tree nodes that represent elements, attributes and text data, establishing a relationship between nodes is essential for processing the structural part of the queries. Therefore, tree navigation is essential to answer XML queries. However standard tree navigations (such as depth- first or breadth-first traversals) are not sufficient for efficient evaluation of XML queries, especially the evaluation of ancestor and descendant axes. For this purpose, many node labeling schemes have been made. The use of labeling schemes to encode XML nodes is a common and most beneficial technique to accelerate the processing of XML queries and in general to facilitate XML processing when XML data is stored in databases [15].

The power of XML comes from the fact that it provides self-describing capabilities. XML repeatedly uses tags to describe the data itself. At the same time this self-describing nature of XML makes it verbose with the result that the storage requirements of XML are often expanded and can be excessive. In addition, the increased size leads to increased costs for data manipulation. The inherent verbosity of XML causes doubts about its efficiency as a standard data format for data exchange over the internet. Therefore, compression of XML documents has become an increasingly important research issue and it also seems natural to use compression techniques to increase the efficiency of storing and querying XML data [3, 4, 6, 8]. In our works, we focused on combining the strengths of both labeling and compaction technologies and bridging the gap between them to exploit their benefits and avoid their drawbacks to produce a level of performance that is better than using labeling and compression independently.

In this paper, we continue our investigations on variations of binary encoding forms that would provide for opportunities to further minimize the storage costs of the labels. The rest of the paper is structured as follows: Section 2 and 3 review The CXQU and CXDLS compaction approaches respectively. In Section 4, we present variations of binary encoding schemes can be used to minimize the storage costs of the labels. Experimental results to study the impact of prefix free encoding schemes on reducing the storage size are presented in Section 5. Finally, we conclude and outline future work in Section 6.

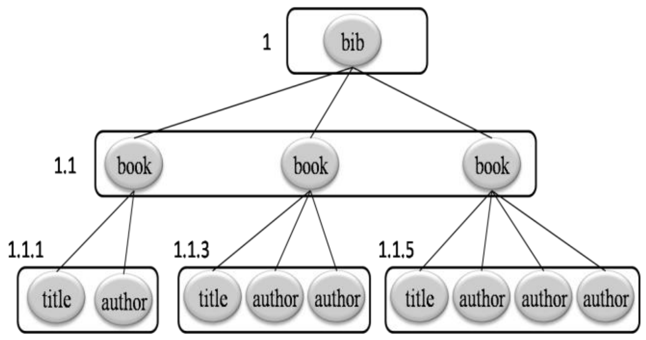


Figure 1. Simple XML document with cluster labels

2. THE CXQU COMPACTION APPROACH

CXQU is our proposed approach [1] to represent XML documents. It not only supports queries and updates but also compacts the structure of an XML document based on the exploitation of repetitive consecutive tags in the structure of the XML documents by using our proposed labeling scheme called Cluster Labeling Scheme (CLS) [1]. CLS assigns a unique identifier to each group of elements which have the same parent (i.e. sibling element nodes). CLS preserves the hierarchal structure of XML documents after the compaction and supports the managing compacted XML documents efficiently. It allows insertion of nodes anywhere in the XML tree without the need for the subsequent relabeling of existing nodes. To compact an XML document with CXQU, first, it separates its structural information from the content to improve query processing performance by avoiding scans of irrelevant data values. CXQU then compacts the structure using our algorithm, which basically exploits the repetition of similar sibling nodes of XML structure, where “similar” means: elements with the same tag name. CXQU stores the compacted XML structure and the data separately in a robust compact storage that includes a set of access support structures to guarantee fast query performance and efficient Updates. Figure 1 displays the cluster labels and Figure 2 displays the compacted structure of a simple XML document, where the crossed-out nodes will not be stored.

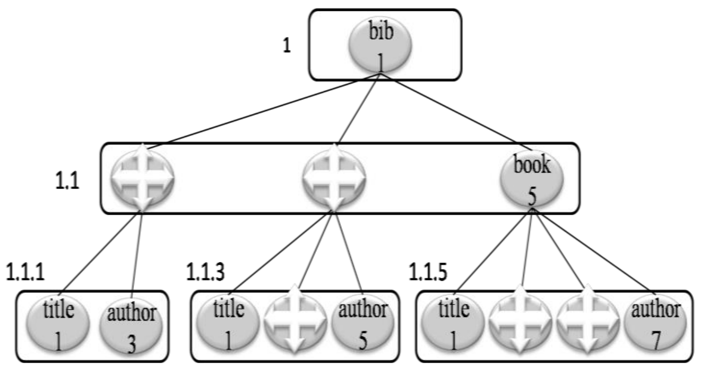


Figure 2. The compacted structure using CXQU

3. THE CXDLS COMPACTION APPROACH

We also proposed an improved technique called CXDLS [2] combining the strengths of both labeling and compaction techniques. CXDLS bridges the gaps between numbering schemes and compaction technology to provide a solution for the management of XML documents that produces better performance than using labeling and compaction independently. CXDLS compacts the regular structure of XML efficiently. At the same time, it works well when applied to less regular or irregular structures. While this technique has the potential for compact storage, it also supports efficient querying and update processing of the compacted XML documents by taking advantage of the ORDPATH labeling scheme. ORDPATH [14] is a particular variant of a hierarchical labeling scheme, which is used in Microsoft SQL Server's XML support. It aims to enable efficient insertion at any position of an XML tree, and also supports extremely high performance query plans for native XML queries.

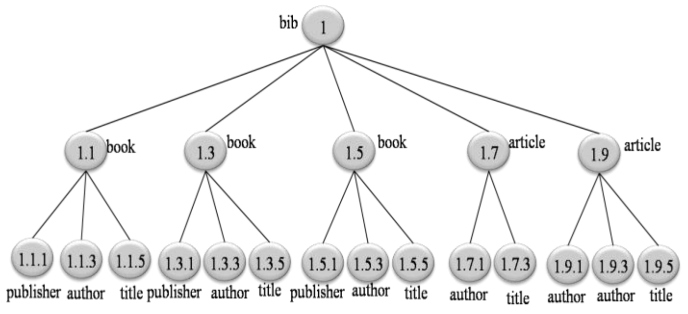


Figure 3. Simple XML document with ORDPATH labels

CXDLS helps to remove the redundant, duplicate subtrees and tags in an XML document. It takes advantage of the principle of separately compacting structure from data and it also uses the ORDPATH labeling scheme for improving the query and update processing performance on compacted XML structures.

In CXDLS, the XML structure is compacted based on the basic principle of exploiting the repetitions of similar nodes in the XML structure, where two nodes N and N' of XML structure are said to be „similar“ if they are consecutive elements, i.e. sibling nodes, in the structure and have exactly the same tag name. Another principle is to exploit the repetitions of identical subtrees, where two subtrees S and S' of XML structure are said to be „identical“ if they are consecutive and have exactly the same structure. Figure 3 shows the ORDPATH labels and Figure 4 displays the compacted structure using CXDLS.

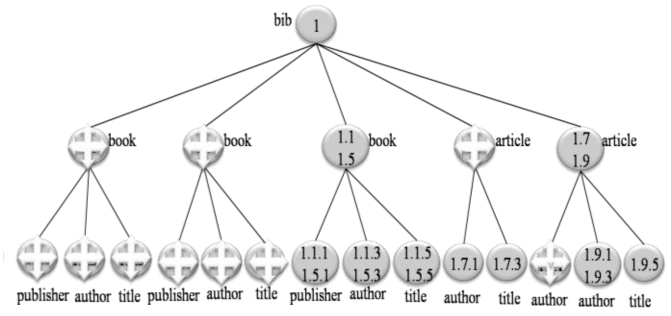


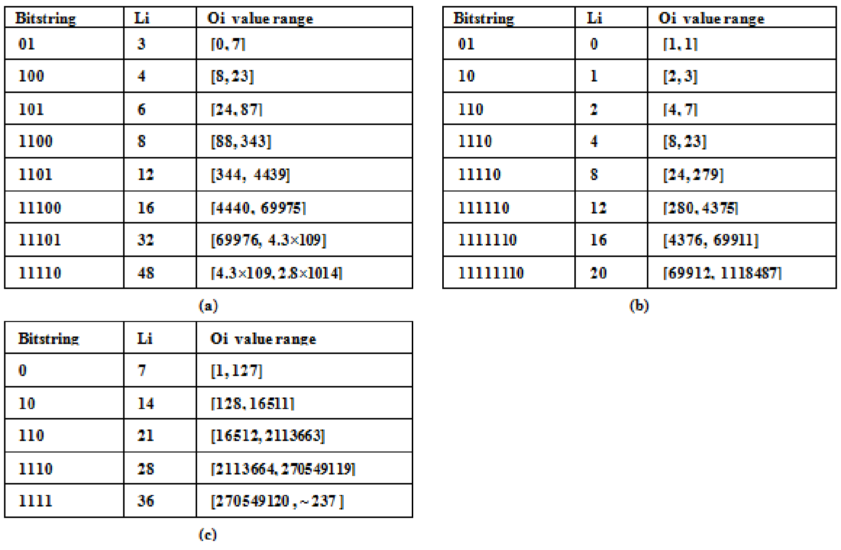
Figure 4. The compacted structure using CXDLS

4. BYTE REPRESENTATION OF THE LABELS

To achieve low storage consumption for XML documents, we have to reduce the size of node labels. Therefore, both ORDPATH and Cluster labeling schemes used Unicode-like compact representation that consists of a compressed binary representation and a prefix free encoding. It uses successive variable length Li/Oi bitstrings and is generated to maintain document order and allow cheap and easy node comparisons. One Li/Oi bitstring pair represents a component of a label. Li bitstring specifies the number of bits of the succeeding Oi bitstring. The Li bitstrings are represented using a prefix free encoding that can be constructed using a Huffman tree, an example for a prefix free encoding shown in figure 5(a). The binary encoding of a label is produced by locating each component value in the Oi value ranges and appending the corresponding Li bitstring followed by the corresponding number of bits specifying the offset for the component value from the minimum Oi value within that range.

Example: Let us consider the bitstring pairs translation for the label (1.3.22). Note that the first component '1' is located in the Oi value range of [0, 7]. So that the corresponding L0 bitstring is 01 and the length L0 = 3, indicating a 3-bit O0 bitstring. We therefore encode the component "1" with L0 = 01 and O0= 001. Similar to that the binary encoding of the component "3" is the bitstring pair L1 = 01, O1 = 011. The component 22 is located in the Oi value range of [8,23] and its corresponding L2 bitstring 100 and the length L2= 4. Thus the O2 bitstring is 1111 that is the offset of 15 from 8 specified in 4 bits. As final result the bitstring 01001010111001111 is the binary encoding of the cluster label (1.3.22).

Variations of prefix free encoding schemes can be created using the idea of Huffman trees, Figure 5 show different forms of prefix free encoding schemes.
Because the labels are binary encoded and stored in a byte array, in the case of use the codes in Figure 5(a) or the codes in Figure 5(b), the last byte may be incomplete. Therefore, it is padded on the right with zeros to end on an 8-bit boundary. This padding can lead to an increase in the storage requirements. For example, by using codes in Figure 5(a), the binary encoding of 1.9 is 010011000001 but its total length in bytes is 2 bytes and will be stored as the following bitstring 0100110000010000. Also by using codes in Figure 5(a), the label 1.9, for example, results in the bit sequence 0111100001, but it is padded by zeros to store it in 2 byte arrays as 0111100001000000 bitstring. In order to avoid padding with zeros, prefix free encoding scheme, shown in figure 5(c), was designed in a way that each division already observes byte boundaries.



Bitstring	Li	Oi value range
01	3	[0, 7]
100	4	[8, 23]
101	6	[24, 87]
1100	8	[88, 343]
1101	12	[344, 4439]
11100	16	[4440, 69975]
11101	32	[69976, 4.3×109]
11110	48	[4.3×109, 2.8×1014]

(a)

Bitstring	Li	Oi value range
01	0	[1, 1]
10	1	[2, 3]
110	2	[4, 7]
1110	4	[8, 23]
11110	8	[24, 279]
111110	12	[280, 4375]
1111110	16	[4376, 69911]
11111110	20	[69912, 1118487]

(b)

Bitstring	Li	Oi value range
0	7	[1, 127]
10	14	[128, 16511]
110	21	[16512, 2113663]
1110	28	[2113664, 270549119]
1111	36	[270549120 , ~237]

(c)

Figure 5. Variations of prefix free encoding schemes

5. The Impacts of Prefix Free Encoding Schemes

5.1. Storage Requirements

In order to examine the impact of prefix free encoding schemes, mentioned in the previous section, on reducing the storage size needed to store the XML documents that are compacted using our approaches CXQU and CXDLS. We did experiment to measure and compare the storage requirements of our approaches and our cluster labeling scheme with other labeling schemes, such as OrdPath and Dewey [7,14]. In the experiment, each approach is suffixed with a number that refers to a prefix free encoding scheme, where number 1 refers to Figure 5(a) and so on respectively. We conducted our experiment using a variety of both synthetic and real datasets that covered a variety of sizes [5, 9, 10, 11, 12, 13, 16], we select 23 XML documents. Table 1 shows the different structural properties of the used datasets, including the document name, document Topic, document size, the number of elements and the maximum depth. The XML document sizes range from 897 KB to 1.09 GB.

Table 1. XML datasets used in the experiments

Datasets	**File name**	**Topics**	**Size**	**No. of elements**	**Max depth**	**No. of dist. elements**
D1	Shakespeare	Shakespeare's play	7,53MB	179727	9	59
D2	XMark	XML benchmark	1,12MB	17132	12	74
D3	XMark	XML benchmark	11,1MB	167865	12	74
D4	XMark	XML benchmark	113 MB	1666315	12	74
D5	XMark	XML benchmark	1.09GB	16703210	12	74
D6	Treebank	Wall Street Journal	85,4MB	2437666	36	250

D7	NCBI	Biological data	427,47 MB	2085385	5	24
D8	SwissPort	DB of protein sequences	112 MB	2977031	6	84
D9	Part	TPC-H benchmark	6,02 MB	200001	4	11
D10	Lineitem	TPC-H benchmark	30,7 MB	1022976	4	18
D11	Customer	TPC-H benchmark	5,14 MB	135001	4	10
D12	Orders	TPC-H benchmark	5,12 MB	150001	4	11
D13	Mondial	Geographical database	1,77MB	22423	6	23
D14	Medline02n0378	Bibliography medicine science	120 MB	2790422	8	78
D15	medline02n0078	Bibliography medicine science	38,71MB	1079702	8	67
D16	medline02n0001	Bibliography medicine science	58,13 MB	1895193	8	48
D17	medline01660167	Bibliography medicine science	196MB	5123499	9	75
D18	TOL	Organisms on Earth	5,36MB	80057	243	4
D19	OT	Religion	3,32 MB	25317	6	21
D20	NT	Religion	0,99 MB	8577	6	15
D21	Quran	Religion	897KB	6709	6	16
D22	BOM	Religion	1,47 MB	7656	6	22
D23	Nasa	Astronomical Data	24,4MB	476646	9	61

It is clearly visible from the results of the experiment in Figures 6, 7, 8, 9, 10 and 11, that the use of third prefix free encoding scheme in our approaches made them more efficient in term of storage requirements for various XML data sets, when compared to other labeling schemes. These results confirm that the success rate of the use of our approaches (SCQX, CXDLS and cluster labeling scheme) is very high and they can dramatically reduce the storage requirements for almost all the datasets, even for a large document such as XMark or Swiss-protein datasets.
From result in Figure 9, it can be observed that the storage requirements, by using the approach CXDLS, are very small for the documents such as PART, Lineitem, Order and Customer because they have a regular structure and CXDLS focuses on compacting regular XML structures. At the same time the storage requirements are still relatively small for other documents that have either an irregular structure or less regular structure.

Experimental results show that the use of third prefix free encoding scheme could lead to extra improvements in storage costs.

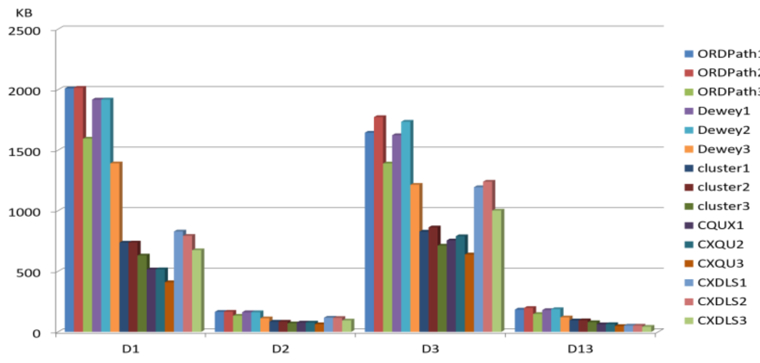


Figure 6:The storage requirements

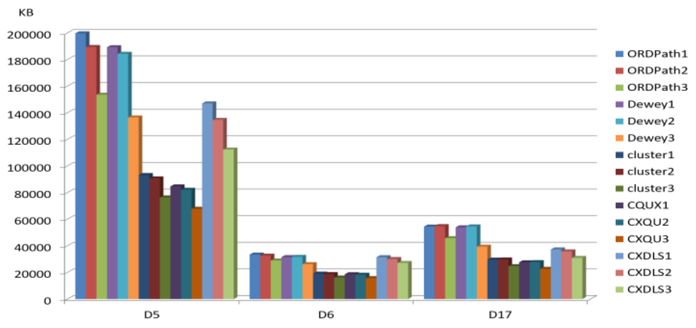


Figure 7:The storage requirements

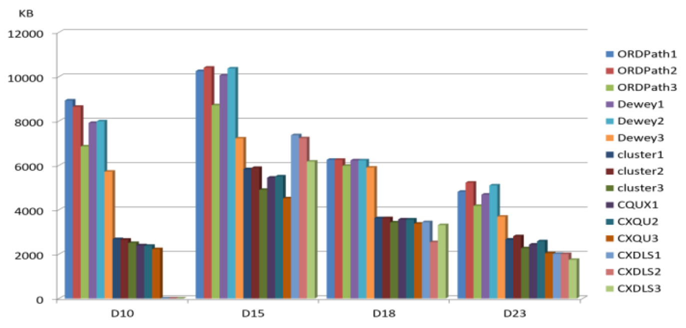


Figure 8:The storage requirements

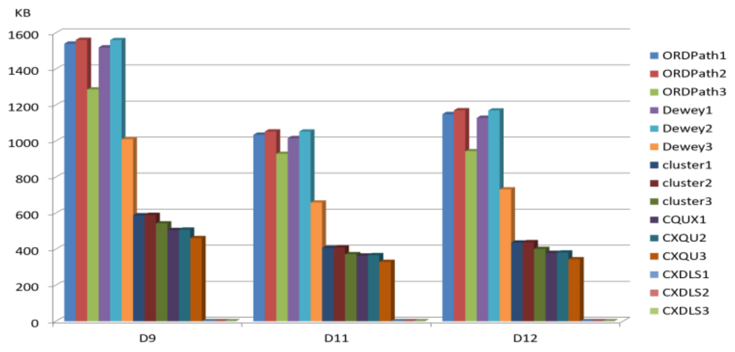


Figure 9:The storage requirements

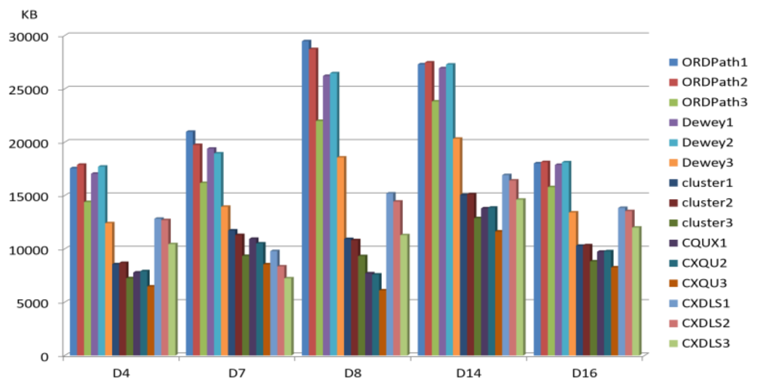


Figure 10:The storage requirements

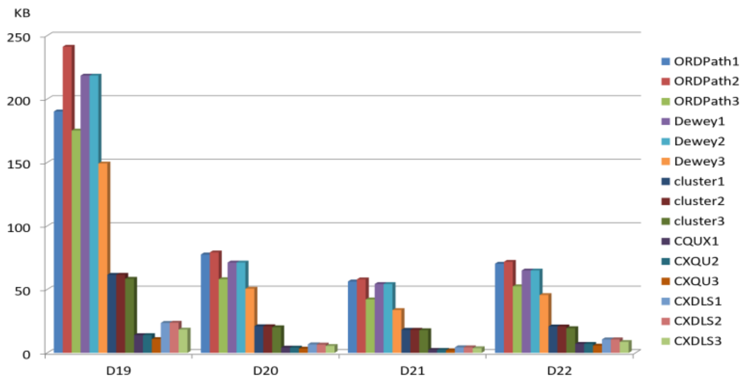


Figure 11:The storage requirements

5.2. QUERY PERFORMANCE

The second experiment is performed to examine the impact of prefix free encoding schemes on the query performance, of our approaches CXQU and CXDLS, for all axes in XPath [17]. We compacted XMark and Shakespeare datasets using our approaches CXQU and CXDLS and then ran different kinds of queries, such as short queries, long queries, simple twig queries and complex twig queries, etc. The queries used in our experiments are described in Table 2. The first character in a query name refers to the data set on which the query is executed where 'X' is a symbol of XMark, and 'S' is for Shakespeare. The experiments focus on the pure query evaluation time, excluding the time for parsing, compiling and optimizing the queries as well as serialization times. To gain a better insight into the query performance of all approaches. We repeated the query 10 times and recorded the averages of the processing times.

Table 2: The Query set for XMark

Queries
XQ1 site/regions XQ2 site/closed auctions/closed auction XQ3 site/people/person XQ4 site/regions/europe/item XQ5 site/open auctions/open auction/initial XQ6 site/regions/asia/item/payment XQ7 site/regions/namerica/item/name XQ8 site/closed auctions/closed auction/annotation/ description/parlist/listitem

XQ9 site/regions/africa/item/description/parlist/listitem/text/keyword
XQ10 site/regions/australia/item//parlist/listitem/text/emph/keyword
XQ11 site//closed auction//description/parlist/listitem/parlist//keyword
XQ12 site/regions/europe/item/*
XQ13 site/*/person
XQ14 site/closed auctions/closed auction/annotation/description/parlist/*/parlist/listitem/text/*/keyword
XQ15 site//item//mailbox/*
XQ16 //closed auction//parlist/*
XQ17 //closed auction//*//keyword
XQ18 //*//description//listitem/*
XQ19 site/*//description/*/listitem/parlist//text//*/keyword
XQ20 site//*/description/parlist/listitem/*//keyword
XQ21 site/regions[asia]//item//name
XQ22 site/closed auctions/closed auction[annotation /description[parlist/listitem/text[keyword[bold]]]]/price
XQ23 site/people[person[profle[education]/age]]/person/phone
XQ24 //australia/item[payment='Cash']
XQ25 //address[zipcode='16']
XQ26 //africa//item[location = 'United States']
XQ27 site/people/person/profile[education='College']
XQ28 //item[payment = 'Creditcard']
XQ29 site/people/person[profile/business = 'Yes']
XQ30 //person[profile/business = 'Yes']

Table 2:The Query set for Shakespeare

Queries
SQ1 /SHAKESPEARE/A AND C/PLAY/ACT/SCENE/SPEECH/SPEAKER
SQ2 /SHAKESPEARE/TAMING/PLAY//SCENE//SPEAKER
SQ3 //SPEAKER
SQ4 //FM/P
SQ5 //SCENE/STAGEDIR
SQ6 //PROLOGUE/STAGEDIR
SQ7 //PLAY//SCENE//STAGEDIR
SQ8 //PLAY/*/*
SQ9 //PLAY/ACT[2]
SQ10 //PLAY/ACT[4]
SQ11 //PLAY/ACT/SCENE/SPEECH[2]
SQ12 //PLAY/ACT/SCENE/*[2]
SQ13 //PERSONAE/PGROUP[GRPDESCR='senators.']
SQ14 //SPEECH[SPEAKER='PHILO']/LINE
SQ15 //PLAY/ACT/SCENE/SPEECH[SPEAKER='Steward']/LINE
SQ16 /SHAKESPEARE//PLAY[TITLE = 'The Tragedy of Antony and Cleopatra']//PERSONA
SQ17 //*[TITLE = 'The Tragedy of Antony and Cleopatra']/ACT//LINE
SQ18 //*[TITLE = 'The Tragedy of Antony and Cleopatra']//PERSONA
SQ19 ////SPEECH/SPEAKER[text() = 'Mark ANTONY']//LINE
SQ20 /SHAKESPEARE/A AND C/PLAY/ACT/SCENE/SPEECH/SPEAKER[text() != 'ANTONY']

The results of the experiments are presented in Figures 12.13.14.15.16 and 17, where the elapsed query processing time is measured in milliseconds. It is shown in the results that the three forms of prefix free encoding schemes have fairly reasonable effect on the query performance of our approaches CXQU and CXDLS. Especially when the third prefix free encoding scheme are used; they guarantee query times less than 30 milliseconds, even for a large document such as Swiss-protein dataset. Regarding query performance, we believe that the fact that the use of prefix free encoding schemes offers an extra degree of freedom for query optimization adds another strong point to our approaches, besides reduced storage costs.

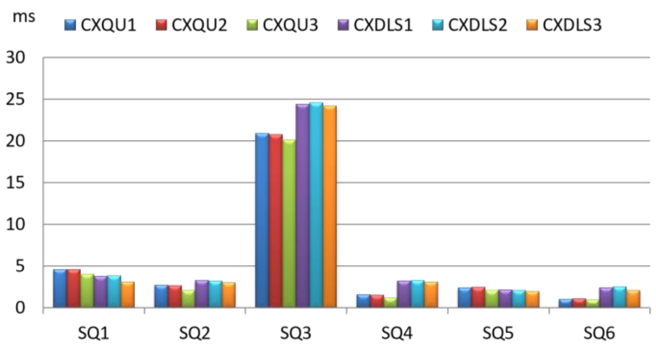


Figure 12: Query Performance of our approaches (Shakespeare Queries)

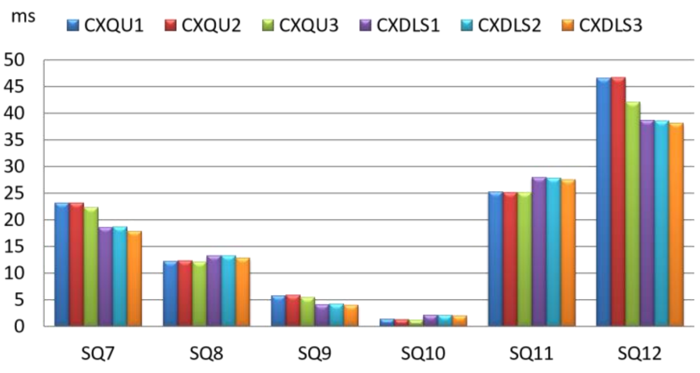


Figure 13: Query Performance of our approaches (Shakespeare Queries)

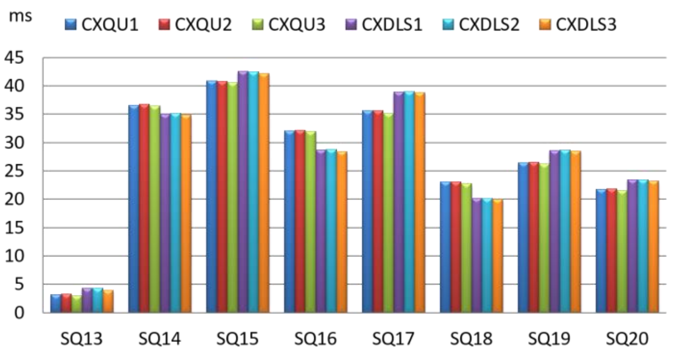


Figure 14: Query Performance of our approaches (Shakespeare Queries)

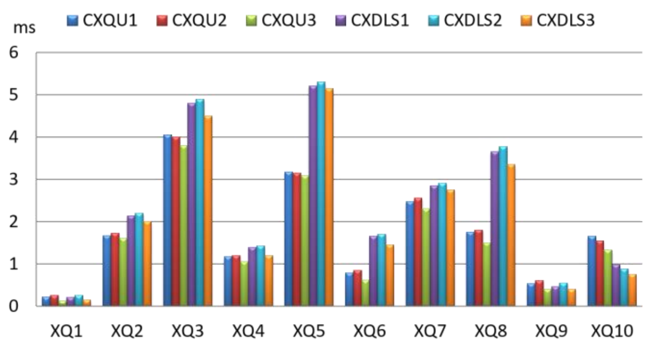


Figure 15: Query Performance of our approaches (XMark Queries)

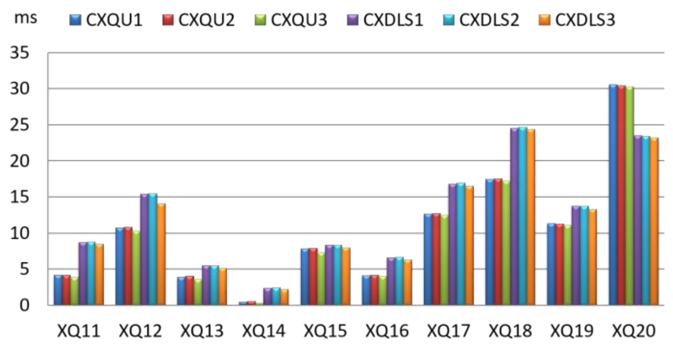


Figure 16: : Query Performance of our approaches (XMark Queries)

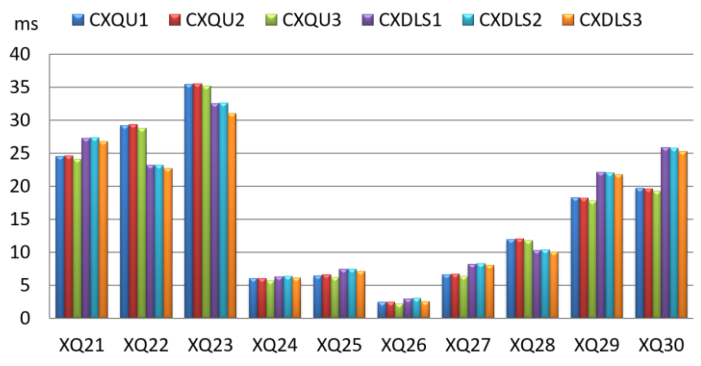


Figure 17: Query Performance of our approaches (XMark Queries)

6. CONCLUSIONS

This work investigated prefix free encoding technique created using the idea of Huffman trees. We have exploited three main ideas to improve the performance of XML namely, prefix free encoding, the XML labeling schemes and our XML compaction techniques. Our experimental results indicate that the use of prefix free encoding schemes offers an extra degree of freedom for query optimization adds another strong point to our approaches, besides reduced storage costs. The inference we make from the results is query performance can be significantly affected by how the data is stored. Since minimizing the storage costs can further improve update performance, one other possible future direction is to test the influence of prefix free encoding schemes on the update performance.

REFERENCES

[1] R. Alkhatib and M. H. Scholl. Cxqu: A compact xml storage for efficient query and update processing. In P. Pichappan and A. Abraham, editors, ICDIM, pages 605–612. IEEE, 2008.

[2] R. Alkhatib and M. H. Scholl. Compacting xml structures using a dynamic labeling scheme. In A. P. Sexton, editor, BNCOD, volume 5588 of Lecture Notes in Computer Science, pages 158–170. Springer, 2009.

[3] M. Ali, M. A. Khan: Efficient parallel compression and decompression for large XML files. Int. Arab J. Inf. Technol. 13(4):403-408, 2016

[4] H. AlZadjali, Siobhán North: XML Labels Compression using Prefix-encodings. WEBIST, pages 69-75, 2016

[5] J. Bosak. Xml-tagged religion. Oct 1998. http://xml.coverpages.org.

[6] S. Böttcher, R. Hartel, C. Krislin: CluX - Clustering XML Sub-trees. ICEIS, pages 142 150, 2010

[7] T.Härder, M. P. Haustein, C.Mathis, andM.W. 0002. Node labeling schemes for dynamic xml documents reconsidered. Data Knowl. Eng., 60(1):126–149, 2007.

[8] M.Lohrey, S. Maneth, R. Mennicke: XML tree structure compression using RePair. Inf. Syst. 38(8): 1150-1167, 2013

[9] D.R.MADDISON, K.-S. SCHULZ, and W. P. MADDI- SON. The tree of life web project. ZOOTAXA, pages 19–40, 20 Nov. 2007.

[10] W. May. Information extraction and integration with FLORID: The MONDIAL case study. Technical Report 131, Universität Freiburg, Institut für Informatik, 1999. Available from http://dbis.informatik.uni-goettingen.de/Mondial,.

[11] G. Miklau. Xml repository. http://www.cs.washington.edu/research/xmldatasets.

[12] NCBI. National center for biotechnology information(ncbi) xml data format. http://www.ncbi.nlm.nih.gov/index.html.

[13] NLM. National library of medicine (nlm) xml data format. http://xml.coverpages.org

[14] P. E. O'Neil, E. J. O'Neil, S. Pal, I. Cseri, G. Schaller, and N. Westbury. Ordpaths: Insert-friendly xml node labels. In G.Weikum, A. C. K¨ onig, and S. Deßloch, editors, SIGMOD Conference, pages 903–908. ACM, 2004.

[15] Tatarinov, S. Viglas, K. S. Beyer, J. Shanmugasundaram, E. J. Shekita, and C. Zhang. Storing and querying ordered xml using a relational database system. In M. J. Franklin, B. Moon, and A. Ailamaki, editors, SIGMOD Conference, pages 204–215. ACM, 2002.

[16] T. web project. the tol tree structure. 1998. http://tolweb.org/tree

[17] W3C. Xml path language (xpath). 'http://www.w3.org/TR/xpath.

AUTHORS

Ramez Alkhatib received the BSc degree in Informatics from University of Aleppo, Syria, the MSc degree in Computer Science from the University of Brunswick, Germany, and the Doctor of Eng. degree in Computer and Information Science from University of Konstanz, Germany. Currently he is an assistant professor of Information Technology at Hama University where he teaches Informatics courses. His current research interests are in Information Technology especially database management systems.