Available online at www.elixirpublishers.com (Elixir International Journal)

Computer Science and Engineering

Elixir Comp. Sci. & Engg. 55A (2013) 13108-13112

ISSN: 2229-712X

Compression technique for XML - a new prototype

V. S. Gulhane and M. S. Ali

ARTICLE INFO

Article history:
Received: 30 May 2012;
Received in revised form:
13 February 2013;
Accepted: 18 February 2013;

Keywords
Prototype,
XML,
Web,
Compressor.

ABSTRACT

XML makes data flexible in representation and easily portable on the Web but it also substantially inflates data size as a consequence of using tags to describe data. Although many effective XML compressors, such as XMill, have been recently proposed to solve this data inflation problem, they do not address the problem of running queries on compressed XML data. More recently, some compressors have been proposed to query compressed XML data. However, the compression ratio of these compressors is usually worse than that of XMill and that of the generic compressor gzip, while their query performance and the expressive power of the query language they support are inadequate. In this paper we propose our approach a XML compressor which support querring compress XML data with partial de-compressor. Our approach address the compressor time and adaptive compression ratio of existing XML compressor.

Introduction

XML has become the de facto standard for data exchange. However, its flexibility and portability are gained at the cost of substantially inflated data, which is a consequence of using repeated tags to describe data. This hinders the use of XML in both data exchange and data archiving. In recent years, many XML compressors have been proposed to solve this data inflation problem. There are two types of compressions: *unqueriable compression* and *queriable compression*. The unqueriable compression, such as XMill [Liefke, H. & Suciu, D. 2000.et al], makes use of the similarities between the semantically related XML data to eliminate data redundancy so that a good compression ratio is always guaranteed. However, in this approach the compressed data is not directly usable; a full chunk of data must be first decompressed in order to process the imposed queries.

```
1. <site>
2. <open_auctions>
3. <open_auction id="open1">
4. <initial>$12.00</initial>
5. <bid>
6. <date>12/02/2000</date>
7. <increase>$2.00</increase>
8. </bid>
9. <bid>
10. <date>12/03/2000</date>
11. <increase>$1.50</increase>
12. </bid>
13. <seller person="person71"/>
14. </open_auction>
15. <open_auction id="open2">
16. <initial>$500.00</initial>
17. <seller person="person8"/>
18. </open_auction>
19. <open_auction id="open3">
20. <initial>$1.50</initial>
21. <bid>
22. <date>11/29/2002</date>
23. <increase>$0.50</increase>
24. </bid>
25. <seller person="person15"/>
26. </open_auction>
27. <open_auction id="open4">
28. <initial>$100.00</initial>
29. <seller person="person11"/>
30. </open_auction>
31. <open_auction id="open5">
32. <initial>$8.50</initial>
33. <bid>
34. <date>08/20/2002</date>
35. <increase>$5.00</increase>
36. </bid>
37. <seller person="person7"/>
38. </open_auction>
39. </open_auctions>
40. </site>
```

Fig. 1. A Sample Auction XML Extract

The queriable compression encodes each of the XML data items individually so that the compressed data item can be accessed directly without a full decompression of the entire file. However, the fine-granularity of the individually compressed data unit does not take advantage of the XML data commonalities and, hence, the compression ratio is usually much degraded with respect to the full-chunked compression strategy used in unqueriable compression.

The queriable compressors, such as XGrind [P. M. Tolani and J. R. Haritsa. XGRIND:] and XPRESS [J. K. Min, 2003 et al], adopts homomorphic transformation to preserve the structure of the XML data so that queries can be evaluated on the structure. However, the preserved structure is always too large (linear in the size of the XML document). It will be very inefficient to search this large structure space, even for simple path queries. For example, to search for bidding items with an

Tele:
E-mail addresses: v_gulhane@rediffmail.com

initial price under $10 in the compressed file of the sample XML extract shown in Fig. 1, XGrind parses the entire compressed XML document and, for each encoded element/attribute parsed, it has to match its incoming path with the path of the input query. XPRESS makes an improvement as it reduces the element-by-element matching to path-by-path matching by encoding a path as a distinct interval in [0.0,1.0], so that a path can be matched using the containment relationships among the intervals. However, the path-by-path matching is still inefficient since most paths are duplicate in an XML document, especially for those *data-centric* XML documents.

Proposed Xml Compression Methodology

The XML Compressor supports compression of XML documents. The compression is based on tokenizing the XML tags. The assumption is that any XML document has a repeated number of tags and so tokenizing these tags gives a considerable amount of compression. Therefore the compression achieved depends on the type of input document; the larger the tags and the lesser the text content, then the better the compression. The goal of compression is to reduce the size of the XML document without losing the structural and hierarchical information of the DOM tree. The compressed stream contains all the "useful" information to create the DOM tree back. The compressed stream can also be generated from the SAX events. XML Parser for Java can also compress XML documents. Using the compression feature, an in memory DOM tree or the SAX events generated from an XML document are compressed to generate a binary compressed output. The compressed stream generated from DOM and SAX are compatible, that is, the compressed stream generated from SAX can be used to generate the DOM tree and vice versa.

XML Serialization and Compression

An XML document is compressed into a stream by means of the serialization of an inmemory DOM tree. When a large XML document is parsed and a DOM tree is created in memory corresponding to it, it may be difficult to satisfy memory requirements and this can affect performance. The XML document is compressed into a stream and stored in an in-memory DOM tree. This can be expanded at a later time into a DOM tree without performing validation on the XML data stored in the compressed stream. The compressed stream can be treated as a serialized stream, but the information in the stream is more controlled and managed, compared to the compression implemented by Java's default serialization.

There are two kinds of XML compressed streams:

 DOM based compression: The in-memory DOM tree, corresponding to a parsed XML document, is serialized, and a compressed XML output stream is generated. This serialized stream regenerates the DOM tree when read back.

 SAX based compression: The compressed stream is generated when an XML file is parsed using a SAX parser. SAX events generated by the SAX parser are handled by the SAX compression utility, which handles the SAX events to generate a compressed stream. In addition to the above methodology the implemented proposed compression methodology compresses XML as well as HTML documents and works as follows:

 Any content within <pre>, <textarea>, <script> and <style> tags will be preserved and remain untouched (with the exception of <script type="text/x-jquery-tmpl"> tags which are compressed as HTML). Inline javascript inside tags (onclick="test()") will be preserved as well. You can wrap any part of the page in <!-- {{{ -->...<!-- }}} --> comments to preserve it, or provide a set of your own preservation rules (out of the box <?php...?>, <%...%>, and <!--#... --> are also supported)

 Comments are removed (except IE conditional comments). Multiple spaces are replaced with a single space.

 Unneeded spaces inside tags (around = and before />) are removed.

 Quotes around tag attributes could be removed when safe (off by default).

 All spaces between tags could be removed (off by default).

 Spaces around selected tags could be removed (off by default).

 Existing doctype declaration could be replaced with simple <!DOCTYPE html> declaration (off by default).

 Default attributes from <script>, <style>, <link>, <form>, <input> tags could be removed
(off by default).

 Values from Boolean tag attributes could be removed (off by default).

 javascript: pseudo-protocol could be removed from inline event handlers (off by default).

 http:// and https:// protocols could be replaced with // inside href, src, cite, and action tag attributes (tags marked with rel="external" are skipped).

 Content inside <style> tags could be optionally compressed using YUI compressor or your own compressor implementation.

 Content inside <script> could be optionally compressed using YUI compressor, Google Closure Compiler or your own compressor implementation.

 Any content inside <![CDATA[...]]> is preserved.

 All comments are removed. Could be disabled.

 All spaces between tags are removed. Could be disabled.

 Unneeded spaces inside tags (multiple spaces, spaces around =, spaces before />) are removed. With default settings your compressed layout should be 100% identical to the original in all browsers (only characters that are completely safe to remove are removed). Optional settings (that should be safe in 99% cases) would give you extra savings. Optionally all unnecessary quotes can be removed from tag attributes (attributes that consist from a single word: would become). This usually gives around 3% page size decrease at no performance cost but might break strict validation so this option is disabled by default. About extra 3% page size can be saved by removing inter-tag spaces. It is fairly safe to turn this option on unless you rely on spaces for page formatting. Even if you do, you can always preserve required spaces with &#20; or . This option has no performance impact

Architecture :

The following figure 2 shows the complete architecture of proposed implemented research methodology.

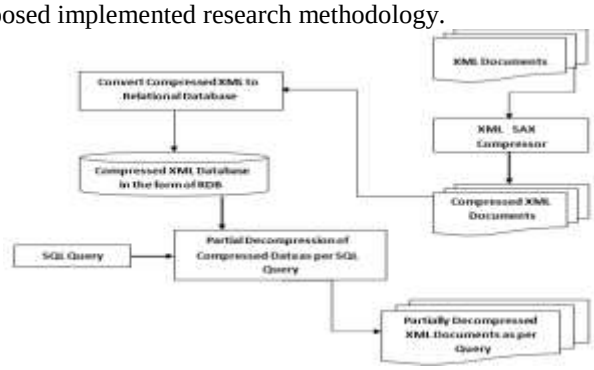


Figure 2: Complete Architecture of Proposed Implemented Research Methodology

In the proposed methodology initially all the XML documents are compressed using XML SAX parser. The graphical user interface is designed from where user can select their XML or HTML documents that he/she want to compress. The compressed XML and HTML file will be created in the current working directory with name Compressed XML.xml and Compressed HTML.html as per the file that has been selected by the user. Figure 3 shows the screenshot of HTML compressor where Image Acquisition Toolbox.html file is compressed. The original size of file was 69114 bytes. After compression the size of file is 49474 bytes. The total time required for compression is 234 ms. Figure 4 shows the screenshot where extracting frames from video.html is compressed. The original size of the file was 41762 bytes. After compression the size of file is 36645 bytes. The total time required for compression is 140 ms.

Figure 3: Compression of Image Acquisition Toolbox.html

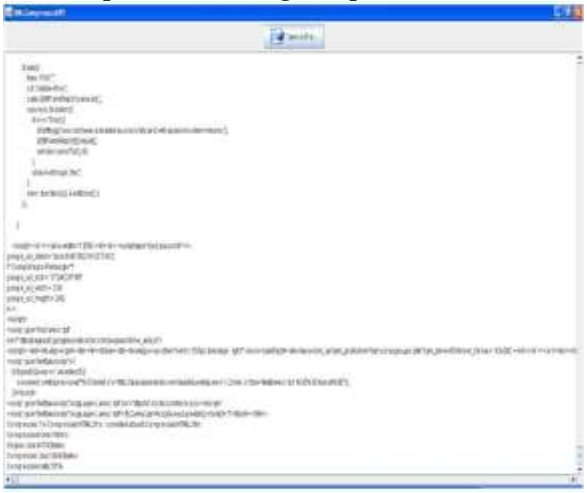


Figure 4: Compression of extracting frames from video.html

Experimental design and setup :

We compare the performance of our approach with that of the following four compressors:

(1) *gzip*, which is a widely used generic text compressor,

(2) *XMill*, which is a well-known XML-conscious compressor, and

(3) *XGrind*, which is a well known XML-conscious compressor that supports querying of compressed XMLdata. (4) *XCQ* - Querriable compressor.

All the experiments were run on a notebook computer with the following configuration:

– Core to Duo, machine with a clock rate of 600 MHz.

– 1GB RAM of main memory.

– 80GB hard disk.

During the experiments, the number of processes running on the machine was minimized in order to reduce unrelated influences. The time taken to compress documents is obtained by running the corresponding processes repeatedly three times and taking the average of the three runs. The main reason for doing this is to reduce the disk I/O influences on the results by loading the whole document into the physical memory if possible . To evaluate the performance of the compressors, we used five datasets that are commonly used in XML research (see the experiments in [W. Y. Lam, W. Ng, may 2003et al, Liefke, H. & Suciu, D. 2000. XMill) *SwissProt*, *DBLP*,ebay, yahoo, and *Shakespeare.*We now briefly introduce each dataset.

1.Ebay,yahoo : It consists of many XML documents that are used in online shopping processes through different e-shopping and auction web sites. These documents are converted from database systems and they contain many empty elements with neither data nor sub-elements inside them

2. *Swissprot* is the complete description of the DNA sequence is described in the XML document

3. *DBLP* is a collection of the XML documents freely available in the DBLP archive . that illustrates different papers published in proceeding of conferences and journals in the field of computer science.

4. *Shakespeare* is a collection of the plays of William Shakespeare in XML [AlHamadani, Baydaa (2011) et al].

The first four datasets given above are regarded as *data-centric* as the XML documents have a very regular structure, whereas the last one is regarded as *document centric* as the XML documents have a less regular structure.

Figure 5 shows the screenshot of the XML Compressor where shakespear.xml is compressed. The original size of file was 7894787 bytes. After compression the file size is 3947393 bytes. The time required for compression is 3047 ms.

Figure 5: Compression of shakespear.xml

Figure 6 shows the screenshot of the XML Compressor where SwissProt.xml is compressed. The original size of file was 94460066 bytes. After compression the file size is 84775077 bytes. The time required for compression is 25359 ms.

Figure 6: Compression of SwissProt.xml

Figure 7 shows the screenshot of the XML Compressor where dblp.xml is compressed. The original size of file was 92301286 bytes. After compression the file size is 644495524 bytes. The time required for compression is 25547 ms.

Figure 7: Compression of dblp.xml

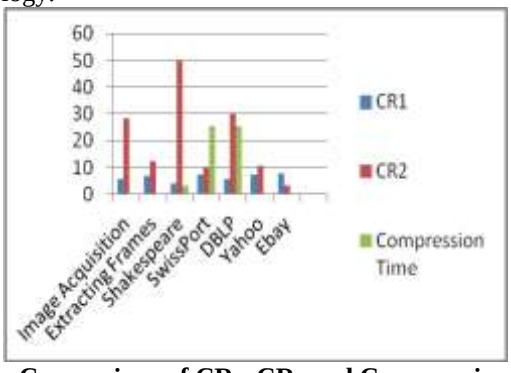
Figure 8 shows the screenshot of the XML Compressor where yahoo.xml is compressed. The original size of file was 25327 bytes. After compression the file size is 22694 bytes. The time required for compression is 125 ms.

Figure 8: Compression of yahoo.xml

Figure 9 shows the screenshot of the XML compressor where ebay.xml is compressed. The original size of file was 35469 bytes. After compression the file size is 34281 bytes. The time required for compression is 141 ms.

Figure 9: Compression of ebay.xml

The following graph 1 shows the computed values of CR_1, CR_2 and the compression time required for the implemented methodology.



Graph 1: Comparison of CR_1, CR_2 and Compression time on various datasets.

Compression performance

We now present an empirical study of our XML compressor performance with respect to compression ratio, compression time. All the numerical data used to construct the graphs can be found in the graph in(W. Y. Lam, W. Ng, et al)

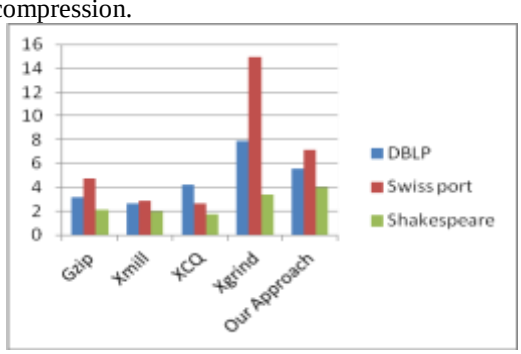
1] Compression Ratio :

The compression ratios are calculated for above discussed results by using the following equation. There are two different expressions that are commonly used to define the *Compression Ratio* (*CR*) of a compressed XML document.

$$CR_1 = \frac{\text{Size of compressed file X 8}}{\text{Size of Original file}} \text{ bits/byte}$$

$$CR_2 = \left(1 - \frac{\text{Size of compressed file}}{\text{Size of original file}}\right) \text{X } 100$$

The first compression ratio, denoted *CR*1, expresses the *number of bits required to represent a byte*. Using *CR*1 a better performing compressor achieves a relatively *lower* value. On the other hand, the second compression ratio, denoted*CR*2, expresses the *fraction of the input document eliminated*. Using *CR*2, a better performing compressor achieves a relatively *higher* value. Graph 2 shows the compression ratios that are achieved on the above-mentioned three datasets expressed in *CR*1 (bits/byte).Both XMill and XCQ consistently achieve a better compression ratio than gzip. Our approach compression ratio is better than XGrind and comparable with XCQ. The compression ratio achieved is relatively high for data-centric documents (i.e., SwissProt, DBLP, Ebay,Yahoo) and relatively low for document-centric documents (i.e., Shakespeare). This can be explained by the fact that the Shakespeare document does not have a regular structure, and therefore XMill , XCQ and our approach cannot take much advantage of the document structure during compression.

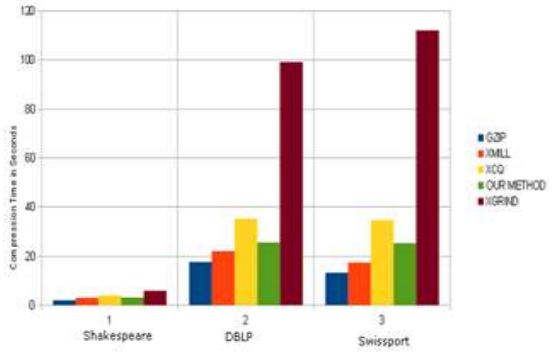


Graph 2: Comparison ratio for different data sets.

2] Compression Time :

Following Graph 3 shows the compression time (expressed in seconds) required by the compressors to compress the XML documents. From the observation it is clear that for our approach, we are getting better compression time as compared to other queribale XML compressor. It is clear that gzip out performs the other compressors in this experiment. XMill had a slightly longer compression time than gzip, and XCQ in turn had a slightly longer compression time than XMill. Our approach has slightly more compression time than Xmill but lesser compression time than a quriable XCQ and Xgrind. The time overhead can be explained by the fact that both XMill and XCQ introduce a pre-compression phase for re-structuring the XML documents to help the main compression process. The *grouping by enclosing tag* heuristic runs faster than the grouping method used in XCQ and thus XMill runs slightly faster than XCQ. It should be noted, however, that the data grouping result generated by XMill may not be as precise as our PPG data

streams. This complicates the search for related data values of an XML fragment in the separated data containers in a compressed file. In addition, the compression buffer window size in XMill is set at 8 MB, which is optimized solely for better compression [H. Liefke and D. Suciu. XMill et al]. Such a large chunk of compressed data is costly in full or partial decompression. On the other hand, the compression time required by XGrind is generally much longer than that required by gzip, XMill, XCQ and our proposed approach. XGrind uses Huffman coding and thus needs an extra parse of the input XML document to collect statistics for a better compression ratio, resulting in almost double the compression time required in a generic compressor.



Graph 3 : Compression time for different data sets for different techniques

Conclusion and future Scope:

We have presented here our approach for compression of XML database with the experimental evaluation we come to the conclusion that our compression time is better and compression ration with some of querible XML compressor. Still we found that there is a room for improvement in compression ration by applying schemes such as indexing.

References:

1] A. Arion, A. Bonifati, G. Costa, S. D'Aguanno, I. Manolescu, and A.Pugliese. Efficient Query Evaluation over Compressed XML Data. *Proceedings of EDBT* (2004).

2] A.Arion, A. Bonifati, G. Costa, S. D'Aguanno, I. Manolescu, and A. Pugliese. XQueC: Pushing Queries to Compressed XML Data. *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB'03)*, (2003).

3] Al-Hamadani, B. T., Alwan, R. F., Lu, J. & Yip, J. 2009. Vague Content and Structure (VCAS) Retrieval for XML Electronic Healthcare Records (EHR). Proceeding of the 2009 International Conference on Internet Computing, USA. P: 241-246.

4] Al Hamadani, Baydaa (2011) Retrieving Information from Compressed XML Documents According to Vague Queries. Doctoral thesis, University of Huddersfield.

5] Augeri, C. J., Bulutoglu, D. A., Mullins, B. E., Baldwin, R.O. & Leemon C. Baird, I. (2007). An analysis of XML compression efficiency. Proceedings of the 2007 workshop on Experimental computer science, ACM, San Diego, California.

6] Clarke J (2004) The Expat XML parser. Extensible Markup Language (XML) 1.0 (Second Edition) W3C Recommendation, October (2000). http://www.w3.org/TR/REC-xml/.

7] G. Antoshenkov. Dictionary-Based Order-Preserving String Compression. *VLDB Journal 6*, page 26-39, (1997).

Gerlicher, A. R. S. (2007), Developing Collaborative XML Editing Systems, PhD thesis, University of the Arts London, London.

8] Groppe, J.(2008), SPEEDING UP XML QUERYING, PhD thesis,Zugl Lübeck University, Berlin. H. Liefke and D. Suciu. XMill: An Efficient Compressor for XML Data. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 153-164 (2000).

9] Harrusi, S., Averbuch, A. & Yehudai, A. 2006. XML Syntax Conscious Compression.Proceedings of the Data Compression Conference (DCC'06),http://www.w3.org/TR/xquery.

10] J. Cheng and W. Ng. XQzip: Querying Compressed XML Using Structural Indexing. *Proceedings of EDBT* (2004).

11] J. Clark. XML Path Language (XPath), (1999). http://www.w3.org/TR/xpath.

12] J. Gailly and M. Adler. gzip 1.2.4. http://www.gzip.org/.

13] J. K. Min, M. J. Park, and C. W. Chung. XPRESS: A Queriable Compression for XML Data. *Proceedings of the ACM SIGMOD International Conference on Management of Data* (2003).

14]J.M.Martinez.MPEG-7Overview(version9). http://www.chiariglione.org/mpeg/standards/mpeg-7/mpeg-7.htm.

15] Liefke, H. & Suciu, D. 2000. XMill: an Efficient Compressor for XML Data. ACM.

16] Mark nelson, Prinipal of data compression, pub 1999. Moro, M. M., Ale, P., Vagena, Z. & Tsotras, V. J. 2008. XML Structural Summaries. PVLDB '08, Auckland, New Zealand.

17] Ng, W., Lam, W.-Y. & Cheng, J. (2006) Comparative Analysis of XML Compression Technologies. World Wide Web: Internet and Web Information Systems, Vol. 9, Pages 5-33

18] Norbert, F. & Kai, G. (2004) XIRQL: An XML query language based on information retrieval concepts. ACM Trans. Inf. Syst., 22, 313-356.

19] P. M. Tolani and J. R. Haritsa. XGRIND: A Query- friendly XML Compressor. *IEEE Proceedings of the 18th International Conference on Data Engineering* (2002). pkzip. http://www.pkware.com/.

20] S. Boag et al. XQuery 1.0: An XML Query Language, Nov. (2002).

21] Smith S. Nair XML compression techniques: A survey. Department of Computer Science ,University of Iowa, USA

22] T. M. Cover and J. A. Thomas. Elements of Information Theory. *Wiley-Interscience, John Wiley &S ons, Inc., New York*, (1991). The bzip2 and libbzip2 official home page. http://sources.redhat.com/bzip2/.

23] Violleau, T. (2001) Java Technology and XML. ORACLE.

24] W. Y. Lam, W. Ng, P. T. Wood, and M. Levene. XCQ: XML Compression and Querying System. *Poster Proceedings, 12th International World-Wide Web Conference (WWW2003)*, May (2003). Winzip. http://www.winzip.com/.