Adaptive Compression Techniques and Efficient Query Evaluation for XML Databases

V.S. Gulhane
Dr. M.S. Ali

Abstract— XML has become the standard way for representing and transforming data over the World Wide Web. The problem with XML documents is that they have a very high ratio of redundancy, which makes these documents demanding large storage capacity and high network band-width for transmission. Because of their extensive use, XML document could be retrieved according to queries by users. The aim of this paper is to present the design of a system named "Adaptive Compression Techniques and Efficient Query Evaluation for XML Databases" which has the ability of compressing the XML document and retrieving the required information from the compressed version with less decompression required according to queries.

The system first compressed the XML document by proposed algorithm. The compressed file is divided into different relational databases doing so there is no need to decompress the complete file for retrieving the results of any query. Only the required information is decompressed and submitted to the user. The average compression ratio of the designed compressor is considered competitive compared to other queriable XML compressors. Based on several experiments, the query processor part had the ability to answer different kinds of queries ranging from simple exact match queries to complex ones that require retrieving information from several compressed XML documents.

I. INTRODUCTION

Extensible Markup Language (XML) [XML 1.0 (Second Edition) W3C Recommendation, October (2000)] is proposed as a standardized data format designed for specifying and exchanging data on the Web. With the proliferation of mobile devices, such as palmtop computers, as a means of communication in recent years, it is reasonable to expect that in the foreseeable future, a massive amount of XML data will be generated and exchanged between applications in order to perform dynamic computations over the Web. However, XML is by nature verbose, since terseness in XML markup is not considered a pressing issue from the design perspective [7]. In practice, XML documents are usually large in size as they often contain much redundant data. The size problem hinders the adoption of XML since it substantially increases the costs of data processing, data storage, and data exchanges over the Web. As the common generic text compressors, such as Gzip], Bzip2, WinZip, PKZIP, or MPEG-7 (BiM) , are not able to produce usable XML compressed data, many XML specific compression technologies have been recently proposed. The essential idea of these technologies is that, by utilizing the exposed structure information in the input XML document during the compression process, they pursue two important goals at the same time. First, they aim at achieving a good compression ratio and time compared to the generic text compressors mentioned above. Second, they aim at generating a compressed XML document that is able to support efficient evaluation of queries over the data. In the survey of XML-conscious compressors it has been found that the existing technologies indeed trade between these two goals. For example, XMill [H. Liefke et al] needs to perform a full decompression prior to processing queries over compressed documents, resulting in a heavy burden on system resources such as CPU processing time and memory consumption. At the other extreme, some technologies can avoid XML data decompression in some cases, but unfortunately only at the expense of the compression performance. For example, XGrind [P.M.Tolani et al] adopts a homomorphic transformation strategy to transform XML data into a specialized compressed format and support direct querying on compressed data, but only at the expense of the compression ratio; thus the XML size problem is not satisfactorily resolved. In regard to the importance of achieving a good level of performance in both compression and querying, it has been found that the current research work on XML compression does not adequately analyze the related features.

II. 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.

III. XML SERIALIZATION AND COMPRESSION

An XML document is compressed into a stream by means of the serialization of an in-memory 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 documents and works as follows:

XML Compression
Input: XML File
Output: Compressed XML File
Begin
1. Open XML file.
2. Store Current time instance.
3. If Compressor is enabled
   3.1 Parse the complete XML file with SAX parser
   3.2 Compress the file by removing comments, extra spaces and line breaks while preserving the content from the CDATA lock.
   3.3 Remove XML Comments
   3.4 Remove multiple whitespace characters
   3.5 Remove inter-tag whitespace characters
   3.6 Remove unnecessary tag attributes such as quotes
   3.7 Simplify existing doctype
   3.8 Remove optional attributes from script tag
   3.9 Remove optional attributes from style tag
   3.10 Remove optional attributes from link tag
   3.11 Remove optional attributes from form tag
   3.12 Remove optional attributes from input tag
   3.13 Remove values from Boolean tag attributes
   3.14 Remove java script from inline event handlers
   3.15 Replace http:// with // inside tag attributes
   3.16 Replace https:// with // inside tag attributes
   3.17 Preserve original line breaks
   3.18 Remove spaces around provided tags
   3.19 Compress inline css.
   3.20 Compress inline java script
4. Create and Store the resulting Compressed XML file in current working directory.
5. Get Current time instance and subtract it from the previously stored time instance to compute the total compression time required.
6. Compute the Compression ratio by considering the sizes of Original and Compressed files.
End

Figure 1: XML Compression Algorithm

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: <div id="example"> would become <div id=example>). 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.
The following figure 2 shows the complete architecture of Propose implemented research methodology

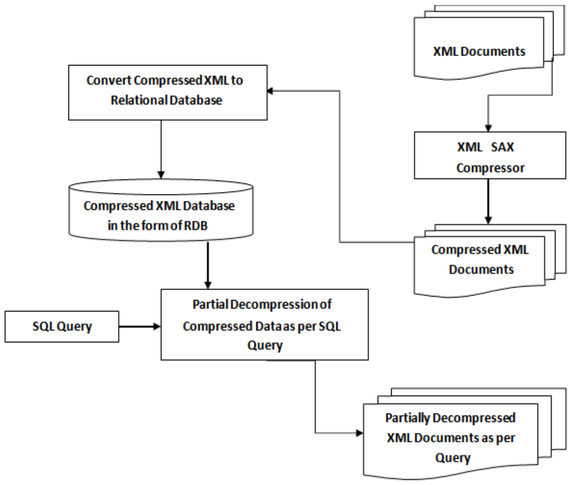


Figure 2: Complete Architecture of Proposed Implemented Research Methodology

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.

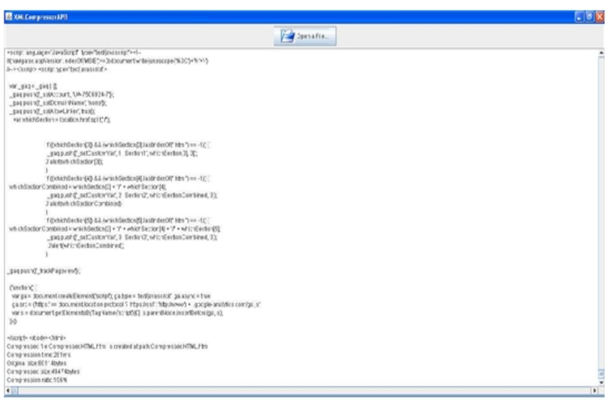


Figure 3: Compression of Image Acquisition Toolbox.html

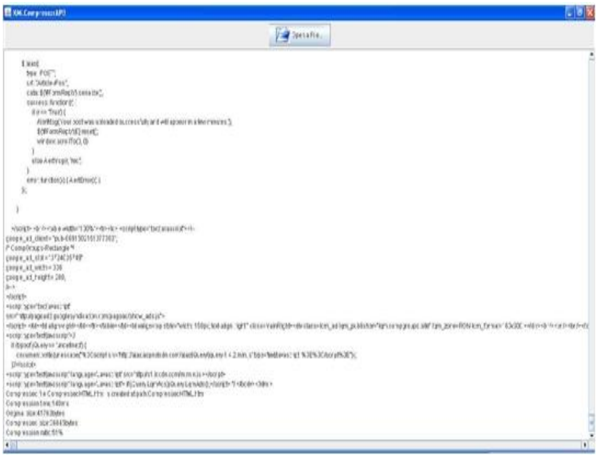


Figure 4: Compression of extracting frames from video.html

IV. 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 .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

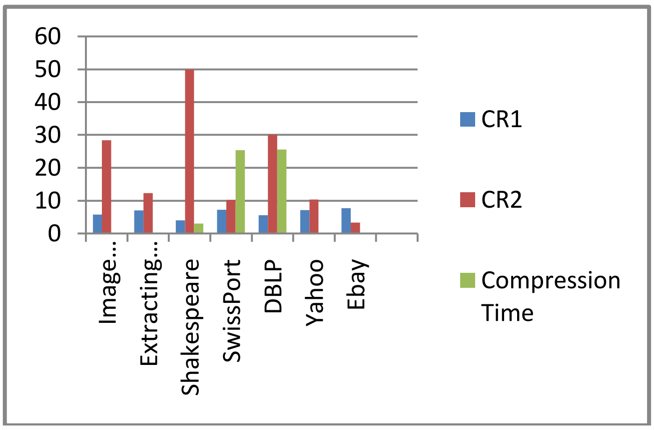
Figure 8 shows the screenshot of the XML Compressor where yahoo.xml is compressed. The original size of file was 25327 bytes. After compression the file size is 22694 bytes. The time required for compression is 125 ms.

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.

V. 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)

A. 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.

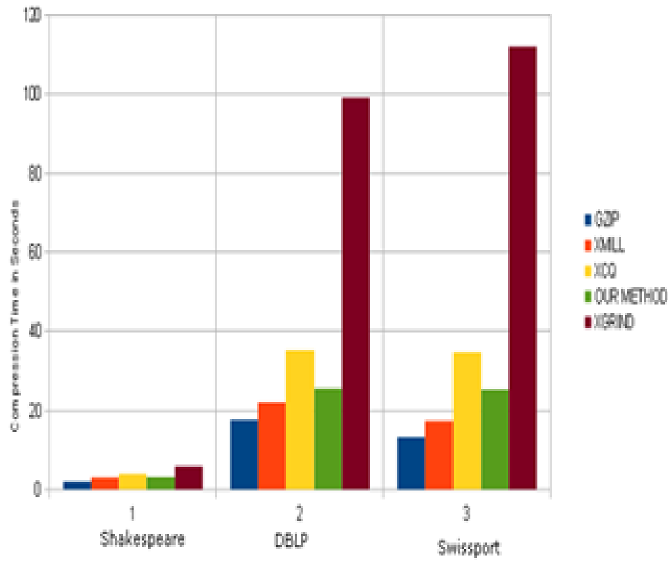
$$CR_1 = \frac{\text{Size of compressed file X 8}}{\text{Size of Original file}} \text{ bits/byte}$$

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

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.

B. Compression Time :

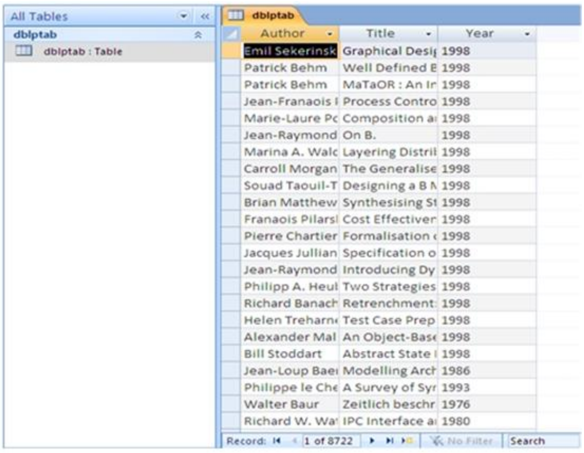
Following Graph 2 shows the compression time (expressed in seconds) required by the compressors to compress the XML documents. From the observation it is clear that for our approach, we are getting better compression time as compared to other queribale XML compressor. It is clear that gzip out performs 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 2 : Compression time for different data sets for different techniques.

C. Implemented Research Methodology

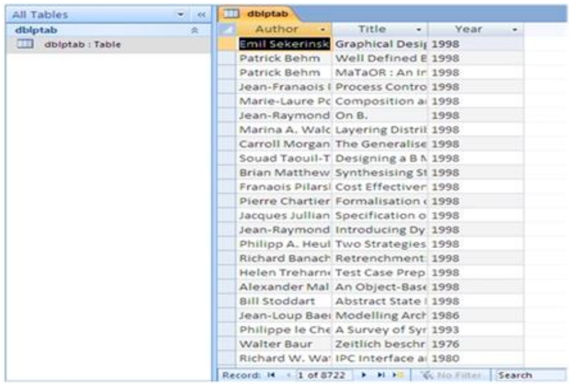
As discussed in previous the compressed XML file is converted into relational database. The complete architecture is shown in figure 2. There are some disadvantages and limitations of intermediate representation of compressed data before decompression for other existing compression techniques. In the implemented methodology as per the compressed XML file it is divided and represented by the RDB. By doing so the results are provided in faster way without any restrictions on memory and size of data. Any type of query is supported in this technique. The type of query is not restricted to aggregation queries.



Author	Title	Year
Emil Sekerinsk	Graphical Desi	1998
Patrick Behm	Well Defined E	1998
Patrick Behm	MaTaOR : An Ir	1998
Jean-Franaois I	Process Contro	1998
Marie-Laure Pc	Composition ai	1998
Jean-Raymond	On B.	1998
Marina A. Walc	Layering Distril	1998
Carroll Morgan	The Generalise	1998
Souad Taouil-T	Designing a B N	1998
Brian Matthew	Synthesising St	1998
Franaois Pilarsl	Cost Effectiver	1998
Pierre Chartier	Formalisation (	1998
Jacques Jullian	Specification o	1998
Jean-Raymond	Introducing Dy	1998
Philipp A. Heul	Two Strategies	1998
Richard Banach	Retrenchment:	1998
Helen Treharn	Test Case Prep	1998
Alexander Mal	An Object-Base	1998
Bill Stoddart	Abstract State I	1998
Jean-Loup Baei	Modelling Arch	1986
Philippe le Che	A Survey of Syr	1993
Walter Baur	Zeitlich beschr	1976
Richard W. Wa	IPC Interface ai	1980

Figure 10: Representation of article information in RDB extracted from compressed dblp file.

The queries are executed on the intermediate RDB representation and relevant results of query are returned and again represented in the form of decompressed XML. Consider the dblp dataset. It is divided into various intermediate RDB representations after compression. These RDB consists of information about articles, international proceedings, phd thesis, master thesis. This information is represented in separate RDB. This information is extracted from the compressed XML file. Because of these representations the user query evaluation will be faster. One more advantage is during decompression. It is not needed to decompress the whole dataset. The partial decompression of the compressed dataset is achieved by using this methodology. The above figure 10 shows the RDB representation of article records extracted from the compressed XML dataset.



Author	Title	Year
Emil Sekerinsk	Graphical Desi	1998
Patrick Behm	Well Defined E	1998
Patrick Behm	MaTaOR : An Ir	1998
Jean-Franaois I	Process Contro	1998
Marie-Laure Pc	Composition ai	1998
Jean-Raymond	On B.	1998
Marina A. Walc	Layering Distril	1998
Carroll Morgan	The Generalise	1998
Souad Taouil-T	Designing a B N	1998
Brian Matthew	Synthesising St	1998
Franaois Pilarsl	Cost Effectiver	1998
Pierre Chartier	Formalisation (	1998
Jacques Jullian	Specification o	1998
Jean-Raymond	Introducing Dy	1998
Philipp A. Heul	Two Strategies	1998
Richard Banach	Retrenchment:	1998
Helen Treharn	Test Case Prep	1998
Alexander Mal	An Object-Base	1998
Bill Stoddart	Abstract State I	1998
Jean-Loup Baei	Modelling Arch	1986
Philippe le Che	A Survey of Syr	1993
Walter Baur	Zeitlich beschr	1976
Richard W. Wa	IPC Interface ai	1980

Figure 11: Representation of international proceedings information in RDB extracted from compressed dblp file.

It can be observed that it contains the information regarding author, title, journal, volume and publication year of article. So any user query can be fired on this representation and the relevant results will be returned to the user. It supports any type of query it will not be restricted to only one type of query. It is also observed that this representation consist of more than 9000 records. So there will not be any restriction on total number of records as it is in earlier techniques.

Figure11 shows the representation of international proceedings information extracted from the compressed dblp dataset. As shown it consists of information regarding author, title and publication year of proceedings. It consists of more than 8000 records.

The following figure 12 shows the algorithm for XML to RDB conversion process. Initially node list is created by using SAX parser. This node list consists of nodes these are nothing but the sub tags in the main tag. Consider while retrieving the article information from the dblp dataset the article tag is the root node. The node items belonging to the article tag are nothing but its child tags. So the node list will consists of items namely author, title, journal,volume and year. After generating the node list the values of each node item will be extracted from the XML document and it will be placed in the separate field in the RDB.

Input: XML File
Outpt: RDB
Begin

1. Open an XML file.
2. Establish the connection with RDB using JDBC ODBC drivers and created DSN.
3. Generate node list for root node tag.
4. For each node item in node list repeat the following steps
   - 4.1 Extract the values of node items from the XML document.
   - 4.2 Insert these values in their respective fields in the RDB
   - 4.3 Update RDB
5. Close the connection with RDB.

End

Figure 12: Algorithm for XML to RDB conversion.

ISSN: 2278-0181
Vol. 1 Issue 4, June - 2012

D. Querying Compressed XML and Query Evaluation Proposed Methodology

As discussed in previous section, proposed XML compression methodology converts the compressed XML into its relational database representation. This approach provides an advantage of executing any type of query as well as decompression time will be less as there are multiple relational database representations for single compressed XML file. Due to these representations the query search space is reduced and relevant results are returned as a result of decompression by using the concept of partial decompression. The following figure 13 shows an algorithm used for partial decompression of compressed XML data.

Input: Compressed XML Relational Database Representations Output: Partially Decompressed XML Data
Begin
1. Create an instance of Document Builder factory.
2. Create Document Builder.
3. Create new document.
4. Create root element.
5. Establish database connectivity.
6. Retrieve the relevant records as per user query and store it in result set.
7. Retrieve the metadata from the result for knowing the total number columns in the table.
8. For each column create an element tag and enclose data within tag.
9. Parse this complete data using DOM.
10. Send the DOM data to XML file.

End

Figure 13: Algorithm for partial decompression**.**

Query Performance:

The performance of the proposed implemented methodology is measured by using various performance metrics. From the query perspective, proposed implemented methodology is compared with XGrind and Native approach on the basis of query response times. These metrics are defined below:

Query Response Time (QRT): Total time required to execute the query.

Query Speedup Factor (QSF): Normalizes the query response time of Native and XGrind with respect to proposed methodology, that is,

$$QSF_{XGrindNa} = \frac{QRT_{Native}}{QRT_{XGrind}}$$

$$QSF_{PropXG} = \frac{QRT_{XGrind}}{QRT_{Proposed}}$$

$$QSF_{PropNa} = \frac{QRT_{Native}}{QRT_{Proposed}}$$

As indicated in above algorithm the user query is executed on the compressed XML's relational database representation. The query results are provided in terms of partially decompressed XML document. For example, In case of article queries the partial decompression result is provided in 'partialart.xml' file. The implemented proposed methodology is evaluated on the basis of query response time, query speedup factor and the decompression time. The following table 1 shows the query performance measure in terms of query response time and query speedup factor.

Document	QRT proposed	QRT xGrind	QRT Native	QRTxGrindNa	QSF Propxg	QSF PropNa
Article	0.234	27	68	2.51	115.38	290.59
intl. proc.	0.188	21	53	2.52	111.70	281.91
Xmark	0.712	80	185	2.31	112.35	259.83
shakespeare	0.124	14	31	2.21	112.90	250
masterthesis	0.125	.	.	.	.	.
phdthesis	0.219	.	.	.	.	.

Table 1: Query performance

It can be observed from the above table great speedup factor is achieved by using the proposed implemented methodology. Following table 2 shows the comparison of proposed methodology with Xmill and gzip on the basis decompression time. DT indicates the decompression time.

Document	DT_{Xmill}	DT_{gzip}	$DT_{Proposed}$
article	151	145	2.218
intl. proc.	116	107	1.235
xmark	663	488	5.228
shakespeare	71	65	0.994
masterthesis	-	-	0.141
phdthesis	-	-	0.344

Table 2: Decompression Times

The time require for decompression is very less in case of the proposed implemented methodology as compared to Xmill and gzip.

VI. Conclusion

we recognize that the *size problem* already hinders the adoption of xml, since in practice, it subsequently increases the cost of data processing, data storage and data exchange over the web .We have presented here our approach for compression of XML database . As there is a tradeoff among compression time and compression ratio, we tried to address compression time issue and effective query evaluation which is having comparatively better result. With the experimental evaluation we come to the conclusion that our compression time is better with some of the querible XML compressor and query response time is better than XGrind and native . And of course there is further room for the improvement in compression ratio and compression time.

VII. 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, Pripipal 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/.

First Author **Vijay S. Gulhane,** M.E. in computer Science and Engineering , having more 30+ publication in international journal ,Area of research is XML database compression techniques and is a research scholar in SGB Amravati university , Amravati.

Second Author Dr. M.S.Ali M.Tech., Ph.D., Recognized research guide with 100+publication in international journal ,Ex chairman IETE Amravati Centre, Ex- Chairman BOS CSE, SGB Amravati University, Amravati with 25+ years experience in Academics.