ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 1, Issue 6, June 2012

# Compression and Query Evaluation over XML Databases

Vijay S. Gulhane, Dr. M.S. Ali

***Abstract*—The extensible Markup Language (XML) is a World Wide Web Consortium (W3C) recommendation which has widely been used in both commerce and research. In recent years, we have witnessed a dramatic increase in the volume of XML digital information that is either created directly as an XML document or converted from another type of data representation. The importance of XML is mainly due to its ability to represent different data types within one document, solving the problem of long-term accessibility, and providing a solution to the problem of interoperability (Al-Hamadani et al., 2009). 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. The aim of this paper is to introduce the system 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.***

***Index Term:*** **- XML, Compression Ratio, Compression Time, Decompression Time**

## I. INTRODUCTION

### *XML: An Overview*

XML or extensible markup language is a markup language for documents containing structured information. Structured information could be content and some indication of what role that content plays. XML specifies the structure and content of a document. A markup language is a mechanism to identify structures in a document. XML has a simple, flexible text format. It is a simplified subset of SGML. Here is an XML snippet:

```
<Student SID = "S1234">
<LastName>Smith</LastName>
<FirstName>Joe</FirstName>
<MInitial>M</MInitial>
<Address>620 12th Ave, Coralville</Address>
</Student>
```

***XML example***

Before the rise of the internet, 1980s witnessed the invention of Standard Generalized Markup Language (SGML) as a way to display information dynamically. Later, in 1995, W3C recommended SGML to be used for the internet. Problems occurred when using SGML included the lack of widely supported style sheets, complexity and instability in the software that were using it, and the difficulties in interchanging SGML data due to its varying levels among SGML software packages. In 1996, the first XML working draft was intended to be a powerful substitute to SGML. It was first recommended by the World Wide Web Consortium (W3C) in 1998 to be used as a mark-up language for storing and exchanging data through the web. The most recent recommendation was published in 2008, which is the fifth edition of the XML (W3C, 2008). In a very short period of time, XML has become the basis for data exchange through the Internet. This is due to its several features such as the following (NG et al., 2006; Groppe, 2008),

1. Readability: XML is readable by both human and machine. This means that the data represented by XML can be used by different users and by different parsing code.
2. Interoperability: This is the ability of the hardware and software to use XML documents without the need to make any changes to the software or the data itself. This means that XML data is stripped of any dependency on software and machine.
3. Long term usability: Since XML documents are represented using the Unicode; these documents are expected to stay in secure storage and usage for years (Augeri et al., 2007).
4. Extensibility: This means that there are no fixed set of tags that should be used to represent data.

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 1, Issue 6, June 2012

5. Generality: XML documents have the ability to represent different kinds of data representation such as images, sounds, videos, texts, etc.
6. Internationality: Almost all written languages can be represented in XML documents since they support Unicode (Norbert and Kai, 2004).

In spite of all these advantages, XML has also some weaknesses:

1. They have a huge amount of redundancy which makes these documents demand high storage memory to be archives, high band width to be transmitted, and high cost to be processed.
2. The huge amount of technologies surrounding it complicates the use of these documents such as schema, DTD, XSLT, SAX, DOM, XPath, XQuery. These technologies render the use of these documents somewhat difficult especially with naive users or in cases where these technologies are absent, it would be just as difficult as they are considered necessary for dealing with XML documents.
3. The problems that can occur when dealing with the document namespace should be carefully sorted out otherwise other problems and complications could occur during the processing of these XML documents.

### *A. XML Compression Systems*

XML or extensible markup language is a markup language for documents containing structured information. Structured information could be content and some indication of what role that content plays. XML specifies the structure and content of a document. A markup language is a mechanism to identify structures in a document. XML has a simple, flexible text format. XML representations are very large and can be up to ten times as large as equivalent binary representations. XML Compression should be done for the following reasons.

1. There is lot of "redundant" data in XML documents, including white space, and element and attribute names.
2. Its self-describing and document size is larger than other formats. This will affect query processing.
3. As a self-describing format, XML brings flexibility, but compromises efficiency.
4. Most XML documents are stored in file systems, so we need an efficient way to store file-based XML.
5. XML is the lingua franca of web services, thus necessitating large volumes of XML data to be sent over networks. Reducing data size helps conserve network bandwidth. There are a number of XML compression techniques available today which were developed, and tested over the last few years.

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.

### *B. Need for Compression*

As evident from the snippet, XML representations are very large and can be up to ten times as large as equivalent binary representations. Consider the following:

1. There is lot of "redundant" data in XML documents, including white space, and element and attribute names.
2. Its self-describing and document size is larger than other formats. This will affect query processing.
3. As a self-describing format, XML brings flexibility, but compromises efficiency.
4. Most XML documents are stored in file systems, so we need an efficient way to store file-based XML.
5. XML is the lingua franca of web services, thus necessitating large volumes of XML data to be sent over networks. Reducing data size helps conserve network bandwidth.

### *C. Desirable Features of XML Compression*

Following are some desirable quality features for XML compression technology.

I. *Effective Compression.*
ii. *Expressive Query Language and Efficient Querying Engine.*
iii. *Minimal User Intervention and Auxiliary Structures*

## 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.

### *A. 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

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 1, Issue 6, June 2012

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 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>, <text area>, <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.

The following Fig. 2 shows the complete architecture of Propose implemented research methodology

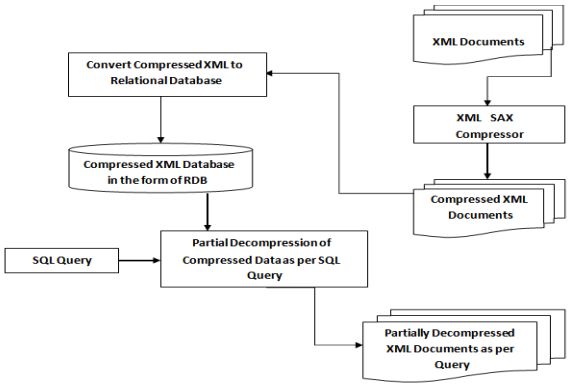



**Fig. 1: Complete Architecture of Proposed Implemented Research Methodology**

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 XMLdocuments that he/she want to compress. The compressed XML file will be created in the current working directory with name Compressed XML.xml .as per the file that has been selected by the user.

## III. 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,

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 1, Issue 6, June 2012

(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) *Swiss rot*, *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

Fig. 2 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

**Fig. 2: Compression of shakespear.xml**

Fig. 3 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.


**Fig. 3: Compression of SwissProt.xml**

Fig. 4 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.


**Fig. 4: Compression of dblp.xml**

Fig. 5 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.

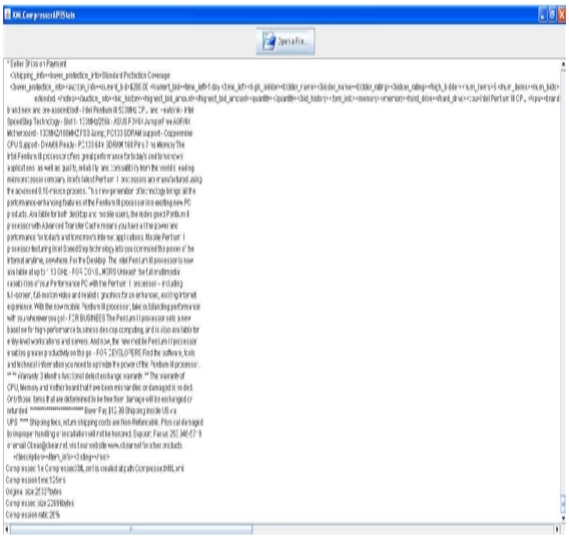


**Fig. 5: Compression of yahoo.xml**

Fig. 6 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.

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 1, Issue 6, June 2012

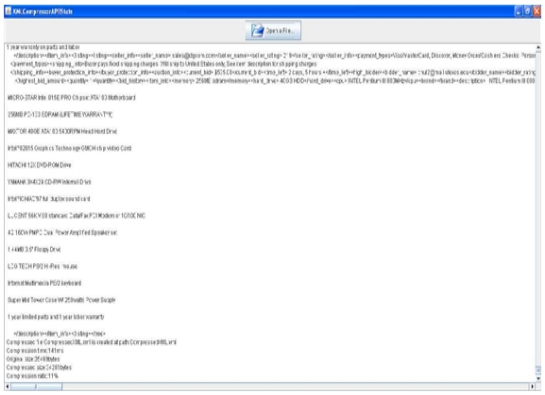
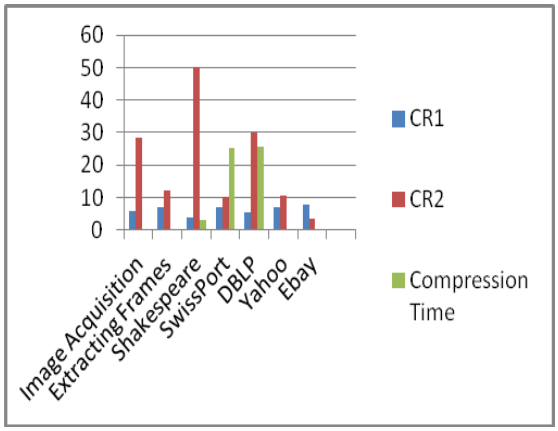


**Fig. 6: Compression of ebay.xml**

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.**

## IV. 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)

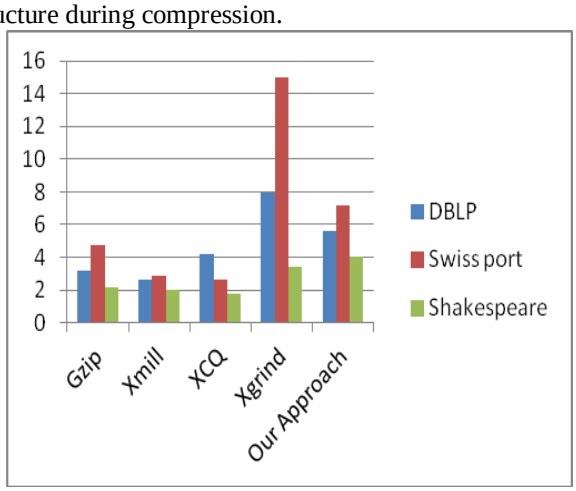
### *A. Compression Ratio:*

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.

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

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

The first compression ratio, denoted CR1, expresses the number of bits required to represent a byte. Using CR1 a better performing compressor achieves a relatively lower value. On the other hand, the second compression ratio, denotedCR2, expresses the fraction of the input document eliminated. Using CR2, 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 CR1 (bits/byte).Both XMill and XCQ consistently achieve a better compression ratio than gzip. Our approach compression ration 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.



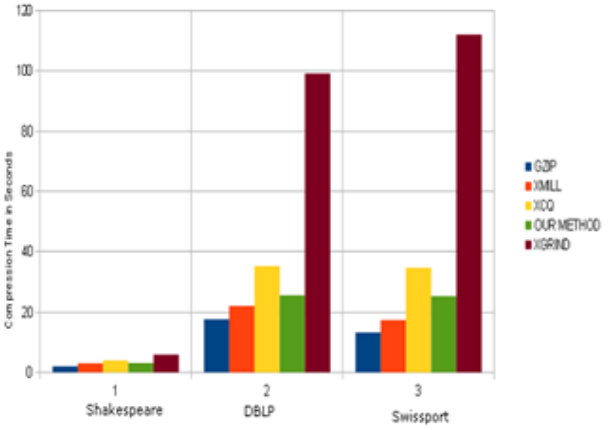

**Graph 2 Compression Ratio for different data set**

### *B. Compression Time:*

Following Graph 3shows 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 XCQand 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

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 1, Issue 6, June 2012

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.**

### *C. Implemented Research Methodology*

As discussed in previous the compressed XML file is converted into relational database. The complete architecture is shown in Fig. 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.

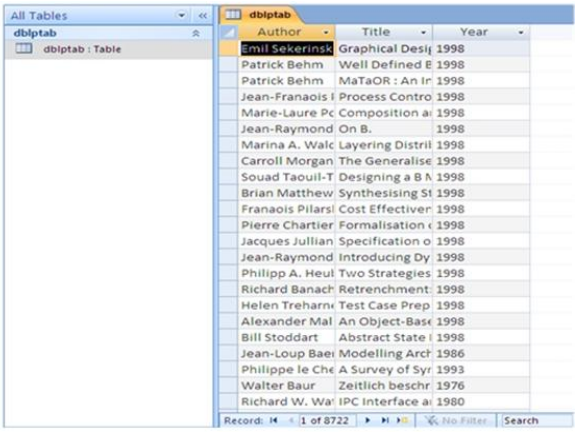



**Fig. 7: Representation Of Article Information In RDB Extracted From Compressed Dblpfile.**

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, and 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 8 shows the RDB representation of article records extracted from the compressed XML dataset.

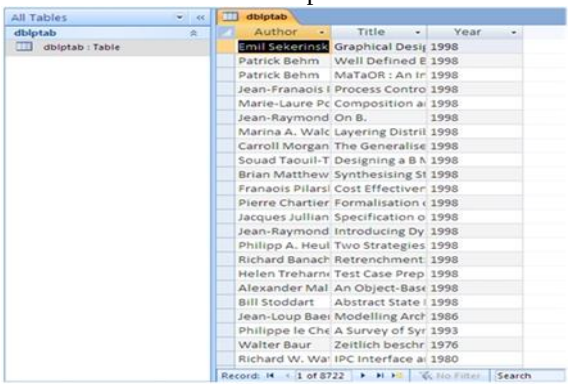



**Fig. 8: Representation Of International Proceedings Information In RDB Extracted From Compressed Dblp File.**

It can be observed that it contains the information regarding author, title, journal, and 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. Fig. 9 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 10 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
Output: RDB
Begin

1. Open an XML file.
2. Establish the connection with RDB using JDBC ODBC

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 1, Issue 6, June 2012

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

### *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 Fig. 11 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

**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 | QRT xGrind/Na | QSF Propxg | QSF Prop/Na |
|---|---|---|---|---|---|---|
| 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 | . | . | . | . | . |

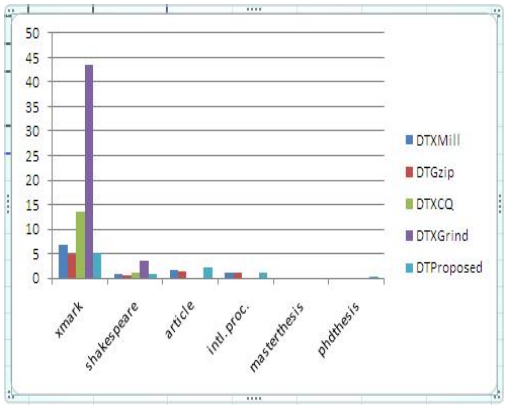
I

**Query performance**

It can be observed from the above table great speedup factor is achieved by using the proposed implemented methodology.

**DECOMPRESSION TIME:**
The XML documents used in study cover a variety of sizes, document characteristics and application domains, and are listed below:
**Xmark:**
This document was generated from Xmark, the xml-benchmark project, using their xmlgen data generator. It models an auction database and is deeply-nested with a large number of elements and attributes. Many of the element values are long textual passages.
**Article, intl. proc, master thesis and PhD thesis :**
These documents represent conference and journal entries, respectively, from the DBLP archive [27].
**Shakespeare:**
This document is the publicly available XML version of the plays of Shakespeare. Similar to xmark above, man y of the element values are long textual passages. Following table 2 shows the comparison of proposed methodology withXmill, Gzip, XCQ and XGrind on the basis decompression time. DT indicates the decompression time.

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 1, Issue 6, June 2012




**Graph 4: Decompression Times**

Graph 4 shows that the decompression required by different de-compressors. One observation from above graph is that, in general, Gzip outperforms the other compressors in de-compression and XMill runs faster than XCQ and XGrind. Other observation is that XGrind requires a much longer de-compression time that the other de-compressors. From the above graph 4 observation it is observed that decompression time for our proposed implemented methodology is better than decompression time for XCQ and XGrind and is comparable with XMill.

## V. CONCLUSION AND FUTURE WORK

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, compression ratio and decompression time, we tried to address compression time, ratio issue and effective query evaluation which is having comparatively better result. With the experimental evaluation we come to the conclusion that our compression time and decompression time for compressed XML datais better with some of the querible XML compressor and compression ratio is comparable with exiting querible 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 by designing a natural advance compressor which can help for updating operation over compressed data.

There are still rooms for further improvement in terms of reducing the resource overheads, such as an indexing scheme, to aid querying compressed XML databases.

## REFERENCES

[1] 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 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] AlHamadani, 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. &Lemon 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)

[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,ZuglLübeck University, Berlin.

[9] 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).

[10] 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.

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

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

[13] J. Gailly and M. Adler. Gzip 1.2.4. http://www.gzip.org/.

[14] 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).

[15] J.M.Martinez.MPEG-7Overview (version9).**Error! Hyperlink reference not valid.**.

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

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

[18] 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

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

[20] 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/.

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 1, Issue 6, June 2012

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

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

[23] 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/.

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

[25] 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).

[26] WinZip. http://www.winzip.com/.

[27] http://www.informatik.uni-trier.de/ ley/db.

## AUTHOR BIOGRAPHY

**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.

Dr. M.S.Ali M.Tech., Ph.D., Principal PRMCOE&M,, Bandera, Recognized research guide with 80+publication in international journal ,Ex chairman IETE Amravati Centre, Ex- Chairman BOS CSE, SGB Amravati University, Amravati with 25+ years experience in Academics.