International Journal of Computer Sciences and Engineering Systems
Vol. 9 No. 1 (June, 2015)

A Review on XML Compressors and Future Trends

Ashutosh GUPTA[1] and Suneeta AGARWAL[2]

[1]Computer Science and Engineering Department, Institute of Engineering and Rural Technology
Allahabad-211002, (UP), India, *E-mail: ashutosh333@rediffmail.com*

[2]Computer Science and Engineering Department, Motilal Nehru National Institute of Technology
Allahabad-211002, (UP), India, *E-mail: suneeta@mnnit.ac.in*

Abstract: *XML is a standardized language that describes a class of data objects called XML documents and partially describes the behavior of computer programs which process them. XML (Extensible Markup language) is a universal format for structured documents and data on web. The storage, exchange and manipulation of semi-structured data are spreading across all kinds of applications, ranging from web-services and electronic commerce to text database and digital libraries. XML language has now become the de facto standard for data exchange and storage, especially on the internet. An important task in XML document is to perform an XML query that can be considered as any inquiry about the content of a document. It can also consider as method that returns information about the document. In this paper we analyze the most practical techniques for compressing the XML document like XGrind, XPRESS, XCpaqs etc. Some of them support the query operations on the compressed XML documents. We also show the relative performance for the compressors and finally we discuss the future trends for queriable XML compressors.*

Keywords: Text Compression, XML, XML Compression., Query

1. INTRODUCTION

According to [7] Extensible Markup language is a standardized language that "describes a class of data objects called XML documents and partially describes the behavior of computer programs which process them". The universal format for structured documents and data on web is XML (Extensible Markup Language; *www.w3c.org/XML/*) [6] as opposed to the HTML which is a markup language for a specific kind of hypertext. HTML is a specific Markup language but XML is a meta-language for describing markup languages together with a strong standard for creating and parsing documents. XML language [1] has now become the de facto standard for data exchange and storage, especially on the internet, due to its self-describing and textual nature. The storage, exchange, and manipulation of semi-structured data are spreading across all kinds of applications, ranging from web-services and electronic commerce to text database and digital libraries. The XML language is look like a HTML language (see Figure 1). XML documents contain *element tags*, including start tags like <name> and end tags like </name>. Elements can contain other elements nested inside them, forming a tree structure. Elements can also contain plain text, comments, and special instructions for XML processor. Starting element tags can contain *attributes* with *values*, such as name in <Author name="Mahesh">. XML data is irregularly structured and self-describing like semi-structured data. Using tags, XML separates contents and structure in XML documents. XML database can represent more complex semantics than traditional database because of its extensibility and ability of representation of semi-structured data. The existing storage structure of XML suffers from the problem of redundancy in XML. We can classify an XML document in two parts: one is the context of data it represents and the other is structure of XML which is formed by tags. Structure information is important in XML document for semantics. But there is a lot of redundancy due to context that XML document represent and repetition of tags. As the XML traffic grows so there will be a demand for compression algorithms which exploits the XML structure to increase the compression ratio.

The XML documents have high redundancy due to the repeated nature of tags which lend themselves naturally to compression. After getting the compressed document, one still require query in compressed domain. There are several advantages of performing query under compressed domain. First, access to compressed data lead to less disk I/Os and reduce the query processing time. Second; the memory requirement in processing compressed data is lower than those for uncompressed data, finally, the result of compressed query uses less bandwidth in network while sending the result to remote machine.

Recently, many XML compressors have been proposed to solve this redundancy problem. According to [28], there are two types of compression: unqueriable compression and

queriable compression. XMill [2] is an example of unqueriable compression. XMill compress the structure and context separately and store them respectively. XMill stores data with same type as a unit and compress each unit using a corresponding compression method so that a good compression ratio is always guaranteed. But the approach used by XMill does not directly use the compressed data; the complete data must be first decompressed in order to process the required queries. The queriable compressors like XGrind [4] and XPRESS[3] encodes each of the XML data items individually so that the compressed data item can be accessed directly without a full de-compression of the entire file.

In this paper, we focus our analysis on existing compression algorithms for XML. Some of the algorithms are considered as queriable compressor algorithms while some of them are comes under the category of unqueriable compressor algorithm.

This paper is organized as follows. First, we will introduce some basic background for XML document. Then, we will present the existing compression algorithms for XML data, some of which supports query and some of them do not support query operation. Finally, we will discuss the relative performance of the algorithms and future trends for queriable XML compressors.

This paper aims at introducing the reader in the field, not at being exhaustive. We will give references for further reading at the appropriate points. Some good general references are [1,6,7]. Also some free softwarepackages are XMill (*http://sourceforge.net/projects/xmill*), XGrind (*http://cvs.sourceforge.net/viewcvs.py/xmill/XGrind*), XMLPPM (*http://sourceforge.net/projects/xmlppm*)

2. XML BACKGROUND

XML is a way of describing semi-structured data. Figure 1 shows a small example document conforming to the XML standard. We will explain some properties and terminology related to XML. The XML document is consisting of element tags including start tags like <STUDENT> and end tags like </STUDENT>.

```
<?xml version="1.0" ?>
<!DOCTYPE student SYSTEM "student.dtd">
<! - this file contains information about a student—>
<STUDENT ROLLNO="200053493">
<NAME>
      <SURNAME>Singh</SURNAME>
      <FIRSTNAME>Gaurav</FIRSTNAME>
</NAME>
<COURSE>Information Technology
      Systems</COURSE>
<DEGREE>Master of Technology</DEGREE>
</STUDENT>
```

Figure 1: The Student Example XML Document

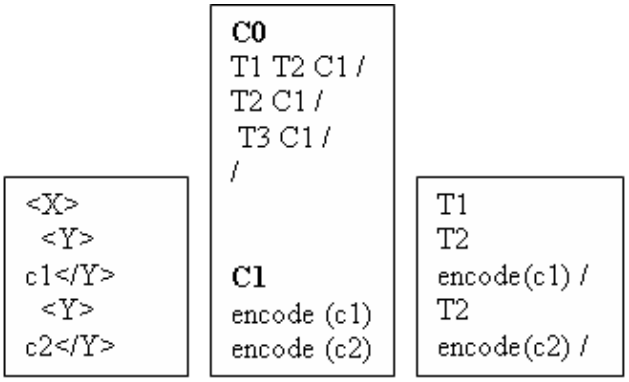
Elements can contain other elements nested inside them, forming a tree structure. Elements can also contain plain text, comments, and special instructions for XML processors. Well-formed documents obey this structure with all of their individual elements satisfying the constraints as defined in the XML recommendation [7]. Additionally, valid XML documents satisfy some further constraints. These additional constraints are defined by a Document Type Definition (DTD) or an XML schema. The <!DOCTYPE> tag is used to declare where the DTD used to validate this XML document can be found. Elements can have associated attributes with values, like the attribute ROLLNO in <STUDENT ROLLNO = "200053493">. They are enclosed in the opening tags of the associated elements and their values need to be enclosed in quotes. All data in an XML document must be stored in textual form. The literal string "Gaurav" is an example of a text element. The W3C and web community develop many additional tools, standards and technologies which are not relevant in this paper. Readers are advised to see the W3C web site's XML page, http://www.w3c.org/XML. Presently, research in XML area has focused on issues related to XML storage [9], retrieval [10,11], path indexes [5, 12] and publishing [13, 14]. Because of the irregular and verbosity nature of XM data, the research on compressors for XML data has been conducted [4].

3. TEXT COMPRESSION

Text compression is usually categorized into three groups: First group belongs to static compression which uses a fixed statistics or does not use any statistics for compression. An approach used in [32] is well suited for static compression. The authors in [32] use an encryption approach for improving the redundancy in the text to improve compression ratio. Examples of the static encodings methods are dictionary encoding, binary encoding and differential encoding. Second category belongs to Statistical compression. This category belongs to estimating source symbol probabilities and assigning them codes according to the probability. Examples of this category are: Huffman coding [15], Arithmetic coding [16] etc. Third category is dictionary methods which are consist in replacing text substring by identifiers, so as to exploit repetitions in the text. Recent advances in this category uses a transformation mechanism [33, 34] to improve compression ratio in natural language text [31]. Examples for this second category are: adaptive Huffman encoding, adaptive arithmetic encoding, LZ77[17], LZ78[18], LZW[19].

The homomorphic compression technique preserves the structure of the original XML data on compressed XML data. The homomorphic compression allows a user to evaluate queries and extract XML fragment which satisfy given query conditions efficiently. As shown in figure 2 (b), some XML compressors physically separate structures (i.e. tags) and data (i.e. value) e.g. XMill. Here the tags X, Y and Z are encoded

as T1, T2 and T3 respectively, and the end tags are replaced by '/'. This technique poses some problems during the query processing. The compressors which follow the homomorphism are XGrind and XPRESS etc. In this paper, we focus on few approaches of compression algorithms for XML text, taking advantage of its structure.



(a) XML document (b) Non-homomorphic (c) Homomorphic

Figure 2: Homomorphism Example

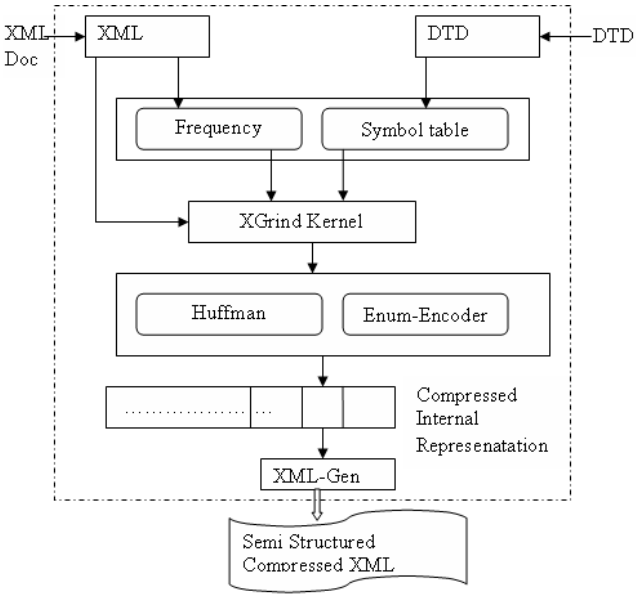


Figure 3: Architecture of XGrind Compressor

3.1 XGrind

XGrind [4] compressor has the ability that it supports queries directly over the compressed XML data and adopts homomorphic transformation to preserve the structure of the XML data. XGrind does not separate data from structure. An XML data compressed with XGrind retains the structure of the original XML document. Structure tags are having been dictionary-encoded and whose data nodes have been compressed using the Huffman [15] encoding. XGrind uses an extended SAX parser as a query processor which can handle exact match and prefix match queries over the compressed values and partial-match and range-queries on decompressed values. The query processor parses and traverses compressed XML data to evaluate a path expression. Whenever a new attribute (or element) is visited by the query processor, it finds the simple path of the visited attribute (or element) and checks whether the incoming path satisfies the given path expression. The XGrind was not able to execute a large set of XML queries like joins, inequality predicates, aggregates, nested queries etc. without decompressing the intermediate results. The architecture of the XGrind compressor, along with the information flow, is shown in Figure 3. The heart of the compressor is XGrind Kernel. The XGrind Kernel first invokes the DTD Parser which parses the DTD of XML document, initializes the frequency tables for each element or non-enumerated attribute, and populates a symbol table for attributes having enumerated type values. The kernel then invokes the XML parser.

The parser scans the XML document and populates the set of frequency tables containing statistics for each element and non-enumerated attribute. The kernel again invokes the XML parser to construct a tokenized form–tag, attribute, or data value–of the XML document. These tokens are supplied to the kernel which calls for each token, the Enum-Encoder or Huffman Compressor, based on its type. The compressed output of the encoders, along with the various frequency and symbol tables, is called the Compressed Internal Representation (CIR) of the compressor and is fed to XML-Gen, which converts the CIR into a semi structured compressed XML document. The query processing engine for compressed domain is consists of a lexical analyzer that emits tokens for encoded tags, attributes, and data values, and a parser built on top of this lexical analyzer that does the matching and generates the matched records. The parser makes use of depth first search traversal of the XML document, maintains information about its current path in the XML document and the contents of the set of XML nodes that it is currently processing.

3.2 XPRESS

XPRESS [3] is also a queriable compressor which supports direct queries on the compressed XML data. XPRESS separates the context and tag. Both are coded respectively. Then the two parts are assembled after encoding. Like XGrind, XPRESS also maintains the homomorphic nature between the compressed XML document and original document i.e. it preserves the document structure. XPRESS uses a novel method called reverse arithmetic encoding, which maps the entire path expression to intervals. XPRESS also uses a simple mechanism to infer the type (which helps for choosing an appropriate compression method) of each elementary data item. The architecture of XPRESS is shown in figure 4.

The core functions of XPRESS are XML Analyzer and XML Encoder. The compression scheme adopted in

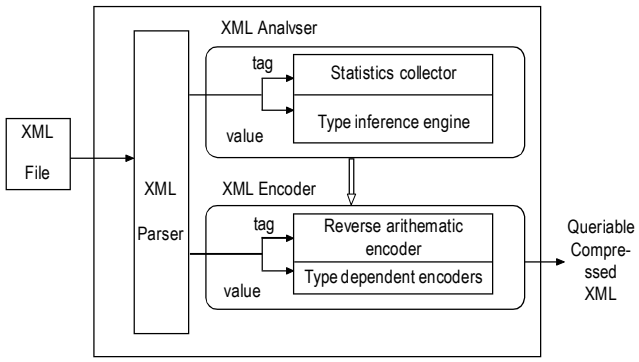


Figure 4: The Architecture of XPRESS

XPRESS is semi-adaptive compression. During the first scan of XML data, XML Analyzer is invoked. XML Analyzer gathers the information used by XML Encoder which generates queriable compressed XML data. XML Analyzer consists of two sub-modules: the statistics collector and the type inference engine. The statistics collector computes the adjusted frequency of each distinct element. The adjusted frequencies of elements are used as inputs to the reverse arithmetic encoder. The type inference engine infers the type of data values of each distinct element and produces the statistics for the type dependent encoders in XML Encoder. There are six encoders for data values in XML Encoder module given in Table 1.

The encoder for tag uses reverse arithmetic encoding scheme using simple paths. Start tags of individual elements are encoded by reverse arithmetic encoding. In [3], the authors implemented an approximated encoder, called the approximated reverse arithmetic encoder (ARAE), to improve the compression ratio and to parse compressed XML data without ambiguity.

A query processor is designed in XPRESS to evaluate queries on compressed XML data. The query processor partitions a long label path expression into short label path expressions whose corresponding interval sizes are greater than 2^{-15} using ARAE. Now, a label path expression is transformed into a sequence of intervals. By using the sequence of intervals, the query executor tests elements in compressed XML data whether their encoded values are in an interval of the sequence or not.

Table 1
Data Encoders

Encoder	*Description*
u8	Encoder for integers where max-min $<2^7$
u16	Encoder for integers where $2^7 +1 <$ max-min $< 2^{15}$
u32	Encoder for integers where $2^{15} +1 <$ max-min $< 2^{31}$
f32	Encoder for floating values
dict8	Dictionary encoder for enumerated typed-data
huff	Huffman encoder of textual data

3.3 XQueC

[XQue]ry processor and [C]ompressor [8]. XQueC is also an XML compressor which supports query. XQueC compresses each data item individually. This results in degradation in compression ratio as compared to XMill. One of the important features of XQueC is that it supports efficient XQuery evaluation. This can be done by a variety of structure information like structure trees, dataguides [5]. But, these structures would incur huge amount of space overhead along with the pointers pointing to the individually compressed data items. XQueC take the advantage of a query workload to choose the compression algorithms, and group the compressed data units according to their common properties. According to [8] , there is a trade off between compression ratio and query capability for several real cases, covered by an XML benchmark. The architecture of XQueC is depicted in figure 5.

The XQueC compressor is consist of following modules:

(i) The XQueC loader and compressor convert XML documents in a compressed queriable format.

(ii) The compressed repository stores the compressed documents and provides compressed data access methods, and a set of compression-specific utilities that enable the comparison of two compressed values.

(iii) The query processor evaluates XQuery queries over compressed documents. The complete set of operators of XQuery allows for efficient evaluation over the compressed repository.

3.4 XCpaqs

XML compressor with path query support **(**XCpaqs in short) [21] is a hybrid compressor which separates structure and

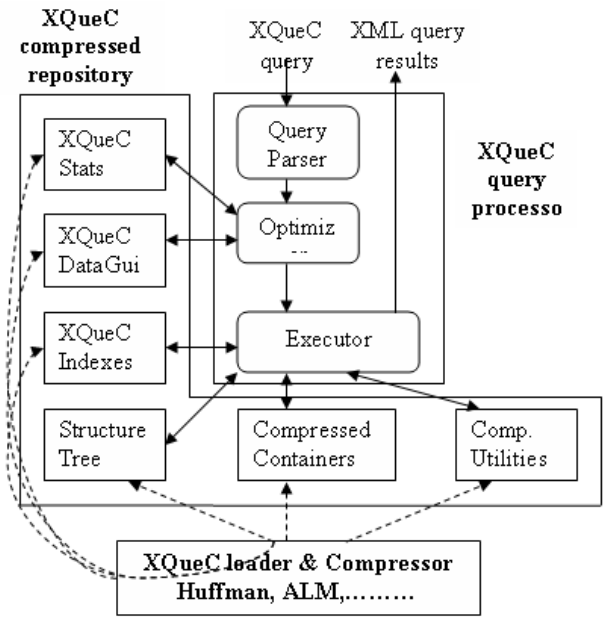


Figure 5: The Architecture of XQueC

context information from XML document. It also keeps homomorphism relation between compressed and original XML document. XCpaqs encodes path and tag respectively. The primary path of Xpath query can be processed in main memory. XCpaqs compressor recognizes different data types and uses different encoding methods to compress data with different type. This feature of XCpaqs makes it to support XML documents without schema information. With compression of both structure and context, XCpaqs gains large compression ratio and supports fast query process on compressed XML document. XCpaqs support complex and long XPath query but more complex operators such as aggregation and join across objects on XML document are not supported. The architecture of XCpaqs is shown in figure 6.

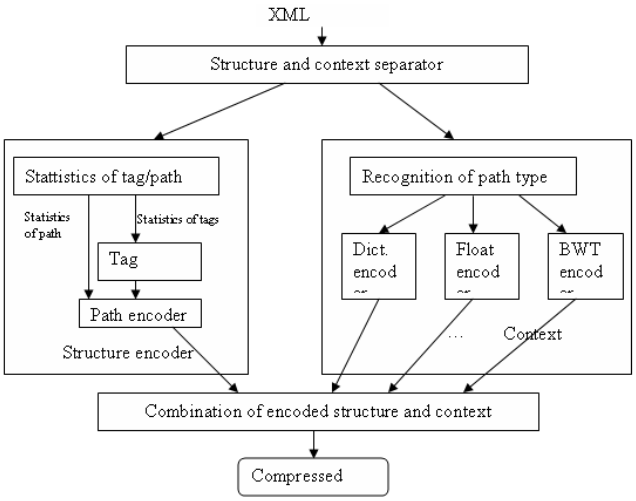


Figure 6: The Architecture of XCpaqs

At first, the structure and context are separated by scanning XML document. The statistics of tag and path and recognition of path type are performed at the same time. The connection between structure and context is the order of path in original document. Recognition of path type is implemented by estimating the data type and range of value of the values of special path class. The tags are encoded at first based on statistics and then paths formed by tags are coded in structure encoder. Context encoder encodes data with different path type with different code method. The output of structure encoder and context encoder are combined at last. The final structure is in 2-ary as (path code, context code). The tags in query are translated into corresponding code based on tag code table before query processing. After query preprocessing, query is spilt into path query, context process plan and construction. These three parts of query plans are processed one by one. The XCpaqs only designs the process techniques of query operations related to XPath. More complex operators on XCpaqs such as aggregation and join across objects on XML document are not considered.

3.5 XQzip

XQzip [28] is an XML compressor which constructs an index structure to support querying on compressed XML data. This index structure is called Structure Index Tree (SIT), on XML data. XQzip architecture consists of four modules: Compressor module, Index Constructor module, Query Processor and the Repository module. The parsing of input XML document is performed by SAX parser which distributes the XML data items (element and attribute values) to the compressor and XML structure to the Index Constructor. The data blocks are compressed by compressor which can be accessed by HashTable where element/attribute names are stored. The Structure Index Tree (SIT) for XML structure is constructed by Index Constructer. The input query is parsed by Query processor and then Query Executor uses the index to evaluate the query. The Executor checks with the Buffer Manager, which applies the LRU rule to manage the Buffer Pool for the decompressed data blocks. If the data is already in the Buffer Pool, Executor retrieves it directly without decompression. Otherwise, the Executor communicates with the Hashtable to retrieve the data from the compressed file. In addition, XPath queries such as multiple, deeply nested predicates and aggregation are supported by the XQzip. The architecture of XQzip is shown in figure 7.

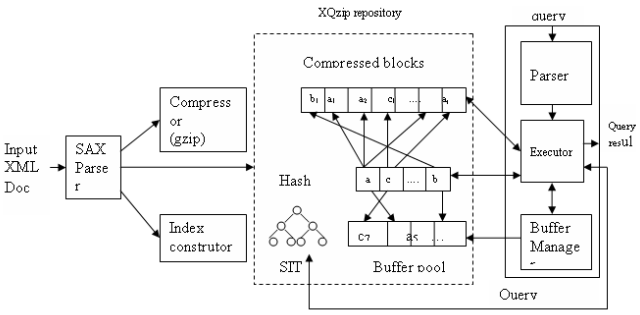


Figure 7: Architecture of XQzip

3.6. LZCS

LZCS [22, 23] is a compression method based on Lempel-Ziv for compressing highly structured data. LZCS takes the advantage of redundant information that can appear in the structure. The idea behind LZCS is to replace the frequently repeated sub-trees by a backward reference to their first occurrence. The LZCS permits random access, visualization and navigation of compressed collections. The LZCS can be integrated into a structured text retrieval system for the search or visualization of results without loss of efficiency. The main idea behind LZCS is based on Lempel-Ziv approach where the repeating substructures and text blocks are replaced by a backward reference to their first occurrence in the processed document. The result is a valid structured text with additional special tags, called backward reference tags, which can be further transmitted, handled or visualized

in a convenient way, or further compressed using some existing compressor.

3.7 SCMHuff

The SCMHuff [29] is a specific compressor obtained with a word-based byte-oriented Huffman coder [24]. The word-based Huffman method is standard for compressing large natural language text databases, because random access, partial decompression, and direct search of the compressed collection is possible. The general approach used in this compressor is SCM (Structural Contexts Model). This compressor retains all the desirable features of semi-static word-based Huffman compression and takes the advantage of structure. The text that lies inside different tags can be compressed by using separate semistatic model. For Example:, in an email archive, a different model can be used for each of the fields From;, Subject:, Date:, Body:, etc. This comes from the fact that the text under similar structural elements should follow a similar distribution, different from other texts.

3.8 XCQ

XCQ (XML Compression and Querying System) [26] is a compression method which separates structure from data. The system is based on technique called DTD Tree and SAX Event Stream Parsing (DSP). The tree shape is compressed using the DTD information and the text is compressed using a standard Lempel-Ziv method like *gzip*. It also supports querying compressed documents without fully decompressing them. The XCQ system is consist of two major parts. They are the Compression Engine and the Querying Engine. The architecture of the Compression Engine is shown in Figure 8.

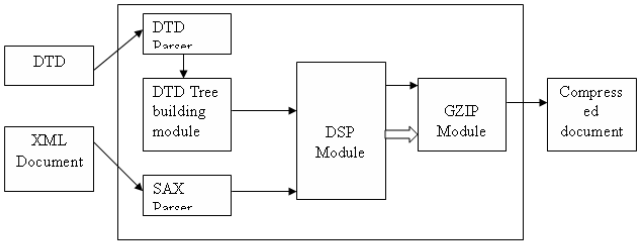


Figure 8: The Compression Engine Architecture

The idea used in DSP given in figure 8 is to extract the structural information from the input XML document that can not be inferred from a given DTD during the parsing process and to group the data elements in the document based on their corresponding tree paths in the DTD tree. The DSP module outputs the extracted structural information and data elements to its corresponding Structural Stream and data Stream respectively. This grouping approach helps a text compressor to achieve a higher compression ratio. The structure stream and the blocks in the data Stream are then compressed individually using a text compressor, such as gzip [30]. These compressed components are then packed in a single file as output by the Compression Engine. XCQ supports querying compressed document by only partially decompressing them. Queries involving only the structure of the document can be answered without decompressing the data Stream.

4. DISCUSSION AND FUTURE TRENDS

We have seen that by considering the structure, we have obtained much more significant gains in compression ratio. One has to also study the relationship between type and density of structuring to improve the compression performance for other text collections. The storage model of XQzip always requires a partial decompression for the matching of string conditions. But this is true for exact-match and numeric range-match predicates. The decompression is also required in XGrind and XPRESS for any other value-based predicates such as string range-match, start-with and substring matches. For evaluating these predicates, the block model of XQzip is much more efficient, since decompressing 'y' blocks is far less costly than decompressing the corresponding thousand of individually compressed data units. The compressors like XGrind and XPRESS make use of the homomorphic transformation to preserve the XML structure. This helps for query to be evaluated on the structure. Since the preserved structure is too large, it is inefficient to search this large structure space, even for simple path queries. The compression performance for XQueC is degraded usually as it compressed each data item individually. The more complex operators like aggregation and join across objects in XML document of XCpaqs are not considered. The implementation of thee operators is also a future work for researchers. As the size of XML document becomes large, there is a need for developing efficient, scalable native XML databases and query engines for large set of XML documents.

More research is needed in order to accommodate insertion, deletions and modifications of documents in the compressed domain. Another interesting research area is to design indexing schemes of compressed domain for fast searching of documents containing some given substructure and words.

To summarize, querying compressed XML is still in its infancy, however XML compression has received significant attention. Currently, XML compression and querying systems do not come anywhere near to efficiently executing complex queries.

REFERENCES

[1] T. Bray, *et al.* "Extensible Markup Language (XML) 1.0", October 2000, *http://www.w3.org/TR/REC-xml.*

[2] H. Liefke and D. Suciu. XMill: An Efficient Compressor for XML Data. In Proc. Of SIGMOD 2000.

[3] J. K. Min, M. J. Park, C. W. Chung. XPRESS: A Queriable Compression for XML Data. In Proceedings of SIGMOD, 2003.

[4] P. M. Tolani and J. R. Haritsa. XGRIND; A Query-friendly XML Compressor. *In Proceedings of ICDE*, 2002.

[5] R. Goldman and J. Widom. Dataguides: Enabling Query Formulation and Opeimization in Semistructured databases. *In Proceedings of VLDB*, 1997.

[6] Charles F. Goldfarb and Paul Prescod. The XML Handbook. Prentice-Hall, third edition, 2001.

[7] World Wide Web Consortium. Extensible Markup Language (XML) 1.0 (Seocnd Edition). *http://www.w3.org/TR/2000/REC-xml-20001006*.

[8] A. Arion, A. Bonifati, G. Costa, S. DAguanno, I. Manolescu, A. Pugliese. XQueC: Pushing Queries to Compresed XMl Data. *In proceedings of VLDB*, 2003.

[9] D. Florescu and D. Kossman. Storing and Querying XML Data using an RDBMS. *IEEE Data Engineering Bulletin*, **22**(3): 27-34, 1999.

[10] M. f. Fernandez and D. Suciu. Optimizing Regular Path Expressions Using Graph Schemas. In Proceedings of the 14th International Conference on Data Engineering, 14-23, 1998.

[11] C. W. Park, J. K. Min and C. W. Chung. Structural Function Inlining Technique for Structurally Recursive XML Queries. In Proceedings of 28th International Conference on Very Large Data Bases, 83-94, 2002.

[12] C. W. Chung, J. K. min and K. Shim. APEX: An Adaptive Path Index for XML Data. In Proceedings of the 2002 ACM SIGMOD *International Conference on Management of Data*, 121-132, 2002.

[13] M. F. Fernadez, W. C. Tan, and D. Suciu. SilkRoute: trading between relations and XML. *WWW9/Coputer Networks*, **33**(1-6): 723-745, 2000.

[14] J. Shanmugasundaram, E. J. Shekita, R. Barr, M. J. Carey, B. G. Lindsay, H. Pirahesh, and B. Reinwald. Efficiently Publishing Relational Data as XML Documents. In Proceedings of 26th International Conference on Very Large Data Bases, 65-76, 2000.

[15] D. A. Huffman. A Method for the Construction of Minimum Redundancy Codes. In Proceedings of the Institute of Radio Engineers 40, 1098-1101, 1952.

[16] I. H. Witten, R. M. Neal, and J. G. Cleary. Arithmetic Coding for Data Compression. Communications of the ACM, **30**(6): 520-540, 1987.

[17] J. Ziv and A. Lempel. An Universal Algorithm for Sequential Data Compression. *IEEE Trans. On Information Theory*, **23**(3): 337-343, 1977.

[18] J. Ziv and A. Lempel. Compression of Individual Sequences via Variable-rate Coding. *IEEE Trans. On Information Theory*, IT-24(5): 530-536, 1978.

[19] Terry A. Welch. A Technique for High Performance Data Compression. *IEEE Computer*, **17**(6): 8-19, 1984.

[20] J. Adiego, Pablo de la fuente and G. Navarro. Combining Structural and Textual Contexts for Compressing Semistructured Databases. In Proceedings of the 2005 IEEE Sixth Mexican International Conference on Computer Science (ENC'05).

[21] H. Wang, J. Li, J. Luo, and Z. He. XCpaqs: Compression of XML Document with XPath Query Support. In proceedings of the 2004 IEEE International Conference on Information Technology: Coding and Computing (ITCC'04).

[22] J. Adiego, G. Navarro and P. de la Fuente. Lempel-Ziv Compression of Structured Text. In Proc. 14th IEEE Data Compression Conference (DCC'04), 112-121.

[23] J. Adiego, G. Navarro and P. de la Fuente. Lempel-Ziv compression of highly structured documents. *Journal of the American Society for Information Science and Technology* (JASIST). To appear.

[24] A. Moffat. Word-based text compression. Software-Practice and Experience, **19**(2): 185-198.

[25] J. Cheney. Compressing XML with Multiplexed Hierarchical PPM Models. *In Proc. Data Compression Conference* (DCC 2001), 163-, 2001.

[26] W. Lam, P. Wood, and M. Levene. XCQ: XML comprssion and querying system. In Proc. 12th Intl. Conf. on the World Wide Web (WWW'03). Poster.

[27] James Cheng and Wilfred Ng. XQzip: Querying Compressed XML Using Structural Indexing. In Proceedings of the 2004 International Conference on Extending Database Technology, pages 219-236

[28] James Cheng and Wilfred Ng. XQzip: Querying Compressed XML Using Structural Indexing. International Conference on Extending Database Technology EDBT 2004, Lecture Notes of Computer Science Vol.2992, Heraklion, Crete, Greece, 219-236, 2004.

[29] Joaquín Adiego, Gonzalo Navarro, and Pablo de la Fuente. Using Structural Contexts to Compress Semistructured Text Collections. *Information Processing and Management (IPM)* 43: 769-790, 2007.

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

[31] Ashutosh Gupta, Suneeta Agarwal. New Transform for Improving Compression Performance in Natural Language Text. In Proc. of International MultiConference of Engineers and Computer Scientists (IMECS), HongKong, **1**, 564-567, 21-23, 2007.

[32] Ashutosh Gupta, Suneeta Agarwal. Word based Text Compression Using Encryption. In Proc. of International MultiConference of Engineers and Computer Scientists (IMECS), HongKong, **1**, 471-473, 21-23, 2007.

[33] F. S. Awan and A. Mukherjee. LIPT A Lossless Text Transform to Improve Compression. In Proceedings of International Conference on Information and Theory: Coding and Computing, Las Vegas, Nevada, 2001 IEEE Computer Society.

[34] R. Franceschini and A. Mukherjee. Data Compression using Encrypted Text. In Proceedings of the third Forum on Research and Technology, Advances on Digital Libraries, 130-138. ADL, 1996.