An Empirical Study of Schema Independent Homomorphic Queriable XML Compression Techniques

Dissertation

Submitted To:

Central Department of Computer Science & Information Technology

Tribhuvan University

Kirtipur, Kathmandu

Nepal

In partial Fulfillment of the requirements for Master's Degree in Computer Science & Information Technology

Submitted by:

Dinesh Tuitui

December, 2015

An Empirical Study of Schema Independent Homomorphic Queriable XML Compression Techniques

Dissertation

Submitted To:

Central Department of Computer Science & Information Technology

Tribhuvan University

Kirtipur, Kathmandu

Nepal

In partial Fulfillment of the requirements for Master's Degree in Computer Science & Information Technology

Submitted by:

Dinesh Tuitui

December, 2015

Supervisor

Prof. Shashidhar Ram Joshi (Ph.D.)

Co-supervisor

Mr. Jagdish Bhatta

Tribhuvan University

Institute of Science and Technology

Central Department of Computer Science and Information Technology

Supervisor's Recommendation

I hereby recommend that the dissertation prepared under my supervision by **Mr. Dinesh Tuitui** entitled "**An Empirical Study of Schema Independent Homomorphic Queriable XML Compression Techniques**" be accepted as in fulfilling partial requirement for the completion of Master's Degree of Science in Computer Science & Information Technology.

Prof. Shashidhar Ram Joshi (Ph.D.)
Department of Electronics & Computer Engineering,
Institute of Engineering,
Pulchowk, Nepal
Date:

Tribhuvan University

Institute of Science and Technology

Central Department of Computer Science and Information Technology

LETTER OF APPROVAL

We certify that we have read this dissertation work and in our opinion it is appreciable for the scope and quality as a dissertation in the partial fulfillment of the requirements of Master's Degree of Science in Computer Science & Information Technology.

Evaluation Committee

Mr. Nawaraj Paudel
Head of Department
Central Department of Computer Science
& Information Technology
Tribhuvan University
Kirtipur

Prof. Dr. Shashidhar Ram Joshi
(Supervisor)
Department of Electronics & Computer
Engineering,
Institute of Engineering,
Pulchowk, Nepal

(External Examiner)

(Internal Examiner)

Date:

Tribhuvan University

Institute of Science and Technology

Central Department of Computer Science and Information Technology

Student's Declaration

I hereby declare that I am the only author of this work and that no sources other than the listed here have been used in this work.

… … … … … … …

Dinesh Tuitui

Date:

ACKNOWLEDGEMENT

I would like to thanks all the people who help me directly and indirectly to complete this dissertation.

First, my big thanks goes to my supervisor **Prof. Shashidhar Ram Joshi (Ph.D.)**, Institute of Engineering, Pulchowk Campus, for his continuous support and guidance. I thank him for his supervision and time spent for this work. Also, the credit of the success of this dissertation work goes to my co-supervisor **Mr. Jagdish Bhatta**. I appreciate for his supervision and guidance.

Most importantly I would like to express my gratitude to the respected Head of Department of Central Department of Computer Science and Information Technology, **Mr. Nawaraj Paudel** for his kind support, help and constructive suggestions. I am very much grateful and thankful to all the respected teachers Prof. Dr. Subarna Sakya, Mr. Dheeraj Kedar Pandey, Mr. Sarbin Sayami, Mrs. Lalita Sthapit, Mr. Arjun Singh Saud, and Mr. Bikash Balami for providing me such a broad knowledge and inspirations. I would also like to thanks all my classmates for the wonderful friendship and support during the college period.

Special thanks to my family for their love, care and support without that I am not be able to gain education up to this level.

I have done my best to complete this research work. Suggestions from the readers are always welcome, which will improve this work.

ABSTRACT

With the rise of computer or digital world, data transfer also increase drastically. XML document standard became the de-facto standard for data transfer since from its inception. To minimize the amount of data volume many compressors came into existence. XML Compressors are proposed to minimize the data size of the XML file. The verbose and repetitive nature of XML is the major cause of large file size. But the structure of XML file is very simple because of this, XML become so popular in data representation and data transfer despite of its large size than any other data representation. Homomorphic XML compressor hence save the structure of XML structure. This study compares three schema-independent homomorphic queriable compressors: XGRIND, XPRESS and XQPoint. The behavior of these three XML compressors are compared using four set of XML data. The study shows the performance of the evaluated XML compressors with change in properties of XML document like change in node count, element count and depth. Also, the memory consumption during compression and decompression is evaluated. Finally, better XML compressor is proposed among the evaluated compressors.

Keywords: *Compression, XML, Homomorphic, Schema-independent, XGRIND, XPRESS, XQPoint*

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ARAE	Approximated Reverse Arithmetic Encoding
CAS	Content And Structure
CIT	Compressed Internal Representation
CR	Compression Ratio
DTD	Document Type Definition
FPNRT	Fixed-Point Number Representation
GPS	Global Positioning System
HTML	Hypertext Markup Language
IDE	Integrated Development Environment
ISO	International Organization for Standardization
KB	Kilobyte
MB	Megabyte
MSB	Most Significant Bit
RAM	Random Access Memory
Sec	Second
SGML	Standard Generalized Markup Language
XML	Extended Markup Language

CHAPTER 1

INTRODUCTION

1.1 Introduction

Now-a-days, the extensive use of computer systems in real world is also increasing the computer data and exchange information between the systems. One of the most challenging tasks for the developers was to exchange the information between two incompatible systems. The data exchange problem is more or less solved after XML document standard that is maintained by the W3C, came into existence. Extended markup language (XML) is the de-facto standard for data representation and exchange. XML is markup language just like HTML. Both XML and HTML contain markup symbols to describe the document contents. In XML, the structure of data is embedded with the data. So, the XML document is self-describing. Actually, XML is simple, flexible text format derived from Standard Generalized Markup Language (SGML) (ISO 8879)[1]. XML is platform independent therefore it provides interoperability between different applications. Use of XML is increasing because of the ability of XML to represent different data types in one document with the solutions to interoperability problem.

The basic building block of an XML document is an element, defined by tags. Elements can be nested forming the tree like structure. All elements in an XML document are contained in an outermost element known as the root element. Below is an example of XML structure:

```
<Message>
<To> John</To>
<From> Harry</From>
<Heading>Reminder</Hedading>
<Body>You have a meeting at 11AM today. </Body>
</Message>
```

XML's power is its simplicity, readability and flexibility. It can take large chunks of information and consolidate them into an XML document. The simplicity, readability and flexibility are maintained at the cost of increased verbosity. The verbosity of the XML file is

[1] http://www.w3.org/XML/

the major cause of large xml files. It is common for an XML representation of a data set to be several times as large as alternative representation of same data set using other data encoding formats.

Many systems that extensively use XML data formats for data storage and exchange have limited resources. Mobile and GPS devices or many devices with embedded systems have limited storage capacity and physical memory. Also networks limitation is another obstacle for data exchange. Reducing the size of XML file can improve the performance and data transfer between these resources constraint devices. Working with lower amount of data also decreases the power consumption of handheld devices. Decreased data size is also beneficial for data exchange over worldwide web and data streaming from many satellites.

Many data compression techniques are so far developed for minimizing the size for data exchange and archiving [1]. Generic compressions take XML documents as regular text file. Gzip, BZip2. Many XML specific compression techniques have been proposed so far to solve the data inflation problem in XML file. XML conscious compression techniques take advantage of knowledge of XML document structure to improve the compression of the document. Schema dependent XML compressor requires schema information of the XML document for the compression and decompression. Those compressors that does not require schema information is schema independent XML compressor. Query support on the compressed XML file classifies the compressor techniques into queriable or non-queriable XML compressor. If the structure of the XML file is maintained in the compressed file, then the tool is grouped in homomorphic XML compressor. Conserving the structure of the XML in compressed document also adds the advantages of original XML document like indexing and querying technique. Parser for the compressed document can be developed same way as the existing XML parser.

1.2 Problem definition

Despite of being the de-facto standard for data encoding, XML standard suffers with some of the drawbacks that are hindering it from gaining widespread use since its inception. Among them large file size is the major drawback of XML. To overcome this, many XML compression tools are proposed to reduce the size of XML document for better exchange of data and reduction of archive size. Since homomorphic XML compression technique maintains the structure of the document, many features developed for XML document so far can be developed for the compressed document. Although, non-homomorphic compressor with better

compression performance are proposed, the loss of the XML structure in compressed document prevents the use of existing technique of parsing and indexing.

Only compression ratio and compression time cannot determine which tool is better than other. Change in compression performance with the change in structure and size of XML document is also required. Resources consumption is other factor to be analyzed. Compression tools for the resource limited devices like mobile and GPS devices should use resources like memory wisely with better compression ratio and compression time as well. As high memory consumption drains charges quickly, decrease in memory use will also reduce the use of battery charge which is also limited in handheld device. Some techniques may be better for one type of XML document while worst for other type. Some may operate better in low resource while other may not. However, choosing best tool among the existing XML compressors for different devices and different datasets is a challenging task.

1.3 Objectives

The objectives of this study are

1. To implement schema independent homomorphic queriable XML compression techniques.
2. To compare performance of the techniques on the basis of compression ratio, compression time and memory consumption on the test datasets.

1.4 Motivation

Since its first introduction in 1998, the use of XML is constantly increasing mainly due to its sustainability for the exchange on the World Wide Web. Decrease in the XML file size will largely reduce the data exchange volume worldwide. To address this issues, different XML compressor are proposed. Queriable XML compressor supports direct query on the compressed document. Best for the resource limited devices, these type of compressor avoids full decompression for the query evaluation. Only the part that is the result of the query is decompressed and presented. Although, full query supporting XML compressor is not proposed till date, it's better to have some basic query support than non- queriable compression where frequent query over the document is required.

Another aspect is the conservation of original XML structure in the compressed document. Homomorphism has many advantages over non-homomorphism. Compressed document can be verified over the compressed format of DTD of the original XML document. Indexing and query parser can be built in similar fashion as in original document.

Hence, queriable homomorphic compression technique is an innovative technique which needs to be researched more. Comparison if XML Compressor is done in the basis of compression ratio, compression time and query response time. These metrics are tested with only with few XML files of various size and data contents. Neither there is comparison on variation of these metrics with the variation in node counts, depth of the XML structure, and distinct tag counts nor their detail analysis of resource consumption of these compression techniques. The analysis of the XML compressor in these aspects is to view the XML compression in different perspective.

1.5 Thesis Organization

The organization of this document is as follows:

- Chapter 1 is the introductory part of the dissertation work. It focuses on the background and introduction of XML document and its compression. It also briefs about the problem definition and objective and motivation of this study.
- Chapter 2 deals with the literature review where works on XML compression are discussed. Here, the examined XML compressors are also briefly described.
- Chapter 3 deals with the implementation part. Here, the tools and environment of development is discussed.
- Chapter 4 deals with data preparation and result analysis. Here the output result of execution of implemented application is analyzed for and compared.
- Chapter 5 consists of conclusion of this dissertation work and the future work which provide guidelines for future studies and research.

CHAPTER 2

LITERATURE REVIEW AND METHODOLOGY

2.1 Literature review

Morse code, invented in 1838 for use in telegraphy, is an early example of data compression based on using shorter code words for letters such as ‘e’ and ‘t’ that are more common in English. Since then data compression has initiated. With the development of Information theory in 1940s, modern work on data compression began [2]. Many compression techniques are proposed since then.

Since XML uses are continuing to grow, a great demand for efficient XML compression tools has been exist. XML data are stored as text files, the first logical approach for compressing XML documents was to use the traditional general purpose text compression tools like BZip2, GZip. These types of compressors are XML-Blind, i.e. they treat XML documents as usual plain text documents. XML conscious compressors are designed to take the advantages of the awareness of XML document structure in order to achieve better compression ratios over the general text compressors. This group can be further classified based on the supporting direct queries on compressed document:

2.1.1 Non-queriable (archival) XML compressor

Non-queriable XML compressor does not support any queries to be processed over the compressed format. These compressors focus on achieving highest compression ratio. All general XML compressors are by default non-queriable compressors. This section can be further classified into two class:

2.1.1.1 Schema-independent compressors

This class of compression schemes does not require the availability of the schema information for encoding and decoding processes. XMill [3] is the first implementation of an XML-conscious compressor that introduced the novel idea of separating the structure of the XML document from data and the grouping of the data values into containers based on their relative paths in the tree and data types. Then the structure and the containers are compressed separately. Both compression and decompression in XMill do not require schema information. XMLPPM, SCMPPM, EXalt are some of the XML compressor in this category [1].

2.1.1.2 Schema-dependent compressors

This class of compressors requires the availability of the schema information of the original XML document during their encoding and decoding processes. Millau [4] is the first XML schema dependent compressor. Document Type Definition (DTD) can be used to build and optimize the token dictionaries in advance. XAUST presented by Subramanian and Shankar converts the schema information of the DTD into a set of Deterministic Finite Automata (DFA) one for each element in the DTD [1]. RNGzip [5] is another schema-dependent compressor which build a deterministic tree automation from the specified schema. Although schema dependent compressors may be able to achieve slightly higher compression ratios, they are not preferable or commonly used in practice because there is no guarantee that the schema information of the XML documents is always available.

2.1.2 Queriable XML compressor

Queriable XML compressors are the compressors which allow queries to be processed over their compressed formats. The compression ratio of this group is usually worse than that of the archival XML compressors. The main focus of this group is to avoid full document decompression during query execution. The ability to perform direct queries on compressed XML formats is important for many applications that are hosted on resources-limited computing devices, such as mobile devices and GPS systems. This can further classified by how they encode the structural and data parts of the XML documents.

2.1.2.1 Homomorphic Queriable XML Compressor

Homomorphic compressors retain the original structure of the XML document, and you can access and parse the compressed format in the same way as the original format. XGRIND was the first homomorphic compressor proposed by Tolani and Haritsa in 2002 [6]. The compressed file has the same structure as the original file. Many features like indexing and parsing can be done similar to the original XML files. XPRESS [7], QXT [8] are other homomorphic compression proposed in 2003 and 2007 respectively.

2.1.2.2 Non-Homomorphic Queriable XML Compressor

Non- homomorphic compressors separates the structural part from the data part while encoding. Therefore the compressed format is different from the structure of the original XML document. XSeq proposed by Lin et al. is a grammar-based queriable

XML compression scheme. In XSeq, tokens of the input XML file are separated into a set of containers each of which is then compressed using Sequitur, a grammar-based text strings compression algorithm [1]. Another non-homomorphic compressor XCQ in proposed by Wilfred Ng et al. in 2006 which exploit the information provided by document type definition (DTD) associated with an XML document to achieve better compressions as well as generate more usable compressed data to support querying.

2.2 Studied Homomorphic Queriable Compression

2.2.1 XGRIND

XGRIND [6] is the first queriable compressor proposed by Tolani and Haritsa in 2002. XGRIND is homomorphic that means it retains the structure of the original XML document in the compressed format also. Since it is also queriable, it supports direct query over compressed document. Further, updates to the XML document can be directly executed on the compressed version. Because of the homomorphic property, the compressed document can be checked for validity against the compressed version of its DTD.

XGRIND is schema-independent but if DTD is available, it attempts to utilize the information in the DTD to enhance the compression ratio. Enumerated type can be recognized from the DTD and are encoded differently from the attribute values.

2.2.1.1 Compression technique

XGRIND uses different technique for compressing structure and data values of XML documents. Those techniques are described below:

a. Meta-Data Compression

   Structure of XML is compressed using the dictionary encoding. Each start-tag of an element is encoded by a 'T' followed by a uniquely assigned element-ID. All end-tags are encoded by '/'s. attribute names are similarly encoded by the character 'A' followed by a uniquely assigned attribute-ID.

b. Enumerated-type Attribute Value Compression

   District, country, department of college are the example of enumerated-type attribute value. XGRIND identifies enumerated-type from the DTD if provided. Enumerated data types are encoded using a simple $\log_2 K$ encoding scheme to represent an enumerated domain of K values.

c. General Element/Attribute Value Compression

Since XGRIND's goal is to support direct query over compressed document, a context-free compression scheme is required. General element/attribute values are compressed using non-adaptive Huffman encoding [9, p. 74]. In this compression scheme, the code assigned to a string in the document is independent of its location in the document. To support the non-adaptive feature, two passes have to be made over the XML document: the first to collect the statistics and the second to do the actual encoding.

2.2.1.2 System Architecture

The architecture of XGRIND compressor is shown in the Fig 2.1. The XGRIND Kernel is the heart of the compressor that controls other compressor. DTD parser parses the DTD if available and initializes frequency tables for each element or non-enumerated attribute, and populates a symbol table for attributes having enumerated-type values. XML Parser scan the XML document twice. In first scan, the parser populates the frequency tables which stores the frequencies of characters and dictionaries for elements and attributes tag. Second scan is done to tokenize the document into tags, attributes or data values of the XML document. Then these tokens are passed sequentially to the respective encoder. Dictionary encoder encode the tags, attributes to the compressed code. Enum-Encoder encodes the enumerated-type attribute values using the symbol table information. Huffman encoder encodes all non-enumerated data items. This module implements the non-adaptive Huffman coding compression scheme.

The compressed output of the encoder and all the frequency and symbol tables is called the Compressed Internal Representation (CIT) of the original XML document. CIT is then fed to XML-Gen, which converts the CIR into a semi-structured compressed XML document.

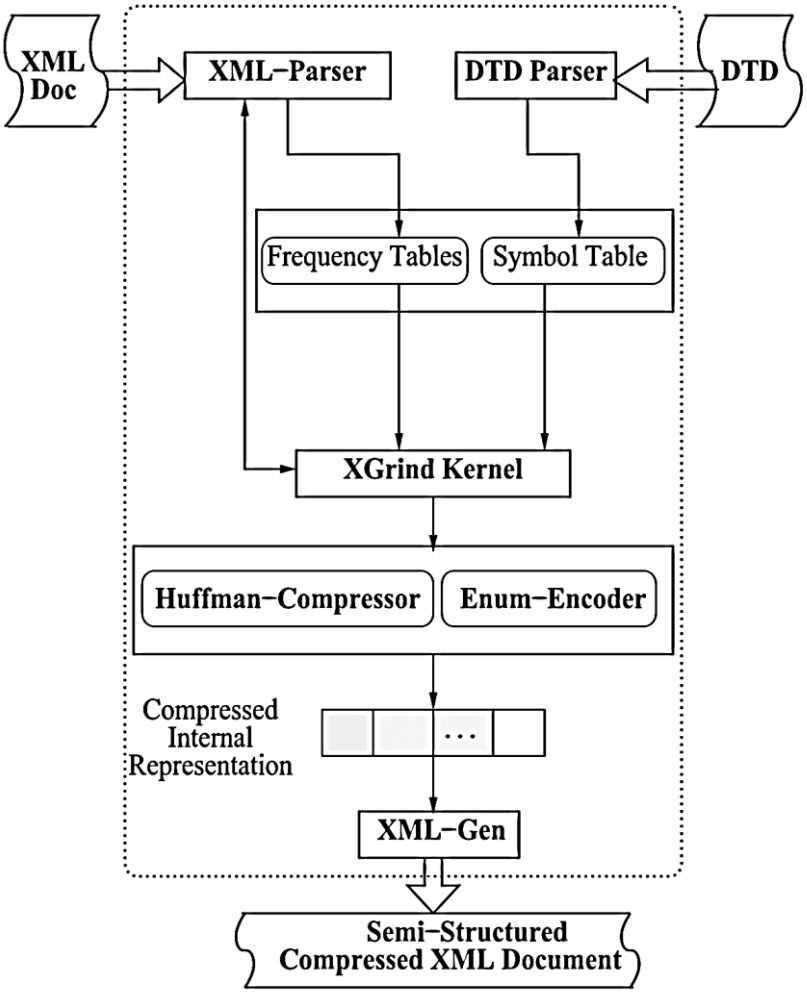

Fig 2.1Architecture of XGRIND Compressor

2.2.2 XPRESS

XPRESS [7] is the queriable homomorphic compressor proposed by Min at al. in 2003. It is the dictionary based XML compressor like XGRIND but claimed to perform better compression and query than XGRIND. XPRESS use reverse arithmetic encoding to encode the label paths of XML. The XPRESS also implements diverse encoding methods depending on the types of data values. It has built-in type-inference engine to infer the types of data. The semi-adaptive approach of compression scheme is applied which scan the XML document twice: first to collect the statistics and second for actual compression.

2.2.2.1 Compression Techniques

The compression technique of XPRESS is homomorphic, hence it preserve the structure of the XML document. The compressor uses different encoding for elements and data values. XPRESS uses following techniques to compress and retrieve XML data efficiently.

a. Reverse Arithmetic Encoding

XPRESS incorporates the reverse arithmetic encoding method that encodes the label path as a distinct interval in [0.0, 1.0). Since it encodes the label path of the XML document, it can handle the path expression on compressed document than previously available XML compressor which simply represent each tag by using a unique identifier.

First, reverse arithmetic encoding partitions the entire interval [0.0, 1.0) into subintervals one for each element. The size of interval of tag 'T' represented as $Interval_T$ is proportional to the frequency of tag T. then, the reverse arithmetic encoding encodes the simple path $P= P_1, \ldots\ldots, P_n$ of an element 'e' into interval $[min_e, max_e)$ using the algorithm in Fig 2.2.

```
Function reverse_arithmetic_encoding(P= P1, ……., Pn)
begin
        [mine, maxe) := Intervalpn
        If (n=1) return [mine, maxe)
        Length := maxe - mine
        [qmin, qmax) := reverse_arithmeti_encoding(P1, ……., Pn-1)
        mine := mine + length * qmin
        maxe := maxe + length * qmax
        retrun [mine, maxe)
end
```

Fig 2.2An algorithm of reverse arithmetic encoding

b. Automatic Type Inference

Many existing XML compressors blindly use predefined encoding methods or apply some encoding methods manually. XPRESS applies different types of encoding for different types of data values. For identifying the type of data, XPRESS have the type-inference engine. At the preliminary scan, XPRESS infers the type of data values of each distinct element. Algorithm for type-inference engine is show in Fig 2.3. The algorithm in Fig 2.3, there only type inference for integer, string and enumeration for simplicity. But it can be extended for floating point values and integer values can be differentiate into different integer encoding like u8, u16, u32 and so on.

```
Procedure Type Inferencing(Token, Pathstack, Elemhash)
begin
        Tag := Pathstack.top()
        eleminfo := Elemhash.hash(Tag)
        type := Infer Type(Token)
        switch(eleminfo.inferred type) {
                case undefined :
                case integer :
                        if(type = integer){
                                eleminfo.inferred type := integer
                                intvalue := get IntValue(Token)
                                eleminfo.min := MIN(eleminfo.min, intvalue)
                                eleminfo.max := MAX(eleminfo.man, intvalue)
                                eleminfo.symhash.insert(Token)
                                eleminfo.accumulate chars frq(Token)
                        }
                        else { // string
                                eleminfo.symhash.insert(Token)
                                if(the number of entries in eleminfo.symhash < 128) {
                                eleminfo.inferred type := enumeration
                                }else eleminfo.inferred type := string
                                eleminfo.accumlate chars frq(Token)
                        }
                        break
                case enumeration :
                        eleminfo.symhash.insert(Token)
                        if(the number of entries in eleminfo.symhash < 128) {
                        eleminfo.inferred type := enumeration
                        }else eleminfo.inferred type := string
                        eleminfo.accumlate chars frq(Token)
                        break
                case string :
                        eleminfo.accumlate chars frq(Token)
                break
                }
end
```

Fig 2.3The algorithm of the type inference engine

c. Diverse Encoding Method

From the type-inference engine, data values are classified into different data types. XPRESS implies diverse encoding scheme for diverse data types. This technique ensures the high compression ratio and minimize the overhead of partial decompression in the query processing phase. Shows the diverse encoders for different types of data values inferred by type-inference engine.

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 enumeration typed data
huff	Huffman encoder of textual data

Table 2.1 Data encoders

d. Semi-adaptive approach

XPRESS scan the XML document twice, first scan for collecting statistics and second for actual compression. This compression scheme is the semi-adaptive approach. So the statistics do not change during the compression, so the encoding is independent to the location of data. This context insensitive encoding scheme supports the direct querying on compressed domain because we do not need to decompress from the beginning.

2.2.2.2 System Architecture

Fig 2.6 shows the architecture of XPRESS compression. The core module of XPRESS are XML Analyzer and XML Encoder. The XML analyzer analyze the original XML to collect the statistics and type infer the data values. XML encoder uses the statistics collected by XML analyzer and encodes the XML into queriable compressed XML data.

a. XML Analyzer

Algorithm of XML Analyzer is shown in Fig 2.5 below. From the algorithm it is clear that the frequency of each distinct element is calculated from procedure Statistics_Collection and all data values token are passed for Type_inferencing procedure. Fig 2.4 shows the procedure of statistics collection. The frequencies of the elements are used by the XML encoder to calculate the interval for each path using reverse arithmetic encoding. The algorithm for type_inferencing procedure is shown in Fig 2.1 in section 2.2.2.1 where all the data values are categorized according to their type inferred.

```
Procedure Statistics_Collection(Token, Pathstack, Elemhash)
Begin
        If(Token is START_TAG){
                Pathstack.push(Token)
                Eleminfo := Elemhash.hash(Token)
                If(taginfo := NULL){
                        Eleminfo := new ELEMINFO(Token)
                        Elemhash.insert(eleminfo)
                        For each token t in Pathstack do {
                                tempinfo := Elemhash.hash(t)
                                tempinfo.adjusted_frequency += 1
                                elemhash.total_frequency += 1
                        }
                }
        }else //token is END_TAG
                Pathstack.pop()
end
```

Fig 2.4 An algorithm of Statistics Collection

```
Function XML_Analyzer()
Begin
        Pathstack := new Stack()
        Elemhash := new Hash()
        do{
                Token := XMLParser.get_Token()
                If(Token is a tag)
                        Statistics_Collection(Token, Pathstack, Elemhash)
                Else //token is data value
                        Type_Inferencing(Token, Pathstack, Elemhash)
        }while(Token!= EOF)
        Return Elemhash)
end
```

Fig 2.5An algorithm of XML Analyzer

b. XML Encoder

This is the second phase of XML compression where the actual compression is performed. The XML parser parse the original XML document and pass token by token to the XML encoder which treats the token according to their type and encodes to the compressed domain. There are six encoder described in Table 2.1 that is used by

XPRESS to encode the data values. Each distinct element has its own encoder which is one of the six encoder. All those encoding are designed to generate 0 as the most significant bit (MSB). The tags are encoded using reverse arithmetic encoding which encode each simple path into an interval between [0.0, 1.0). In the implementation we use the approximated reverse arithmetic encoding (ARAE), to improve the compression ratio and to parse the compressed XML data without ambiguity. This method generates the code for the tag that have 1 as the MSB. In this way, it can be distinguished from the data values which have 0 as the MSB in its code.

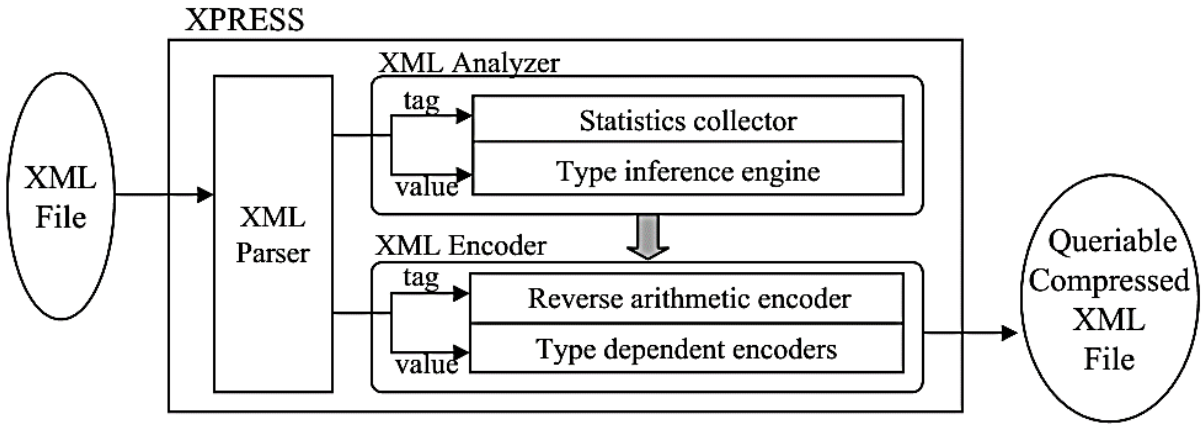

Fig 2.6The System Architecture of XPRESS

2.2.3 XQPoint

XQPoint [10] is yet another homomorphic queriable XML compressor proposed by Al-Hamadini at al. in 2009. In XQPoint, the XML elements and attributes name are compressed by Fixed Point dictionary based technique. XML data parts are classified according to the path from the root attributes and are compressed using Fixed Point technique. The compressed document preserves the structure of original XML document and also supports direct query over the compressed domain.

2.2.3.1 Compression Techniques

As already mentioned above, XQPoint maintain the structure of the original XML document. To obtain this homomorphic compression the XQPoint compressor treats the structure of an XML document in different manner than treating the data part of the document. XQPoint scans

the XML document twice: first scan for analyzing the XML document and second scan is done for actual compression.

In the analysis phase, each element and attribute names of the document is given a unique pair of numbers (ID_{pre}, ID_{post}). This pairs are called structural identifiers, where ID_{pre} represents the preorder traversing of the nodes in the tree while ID_{post} represents the post order traversing of the nodes in the tree. Fig 2.7 shows a sample documents with the nodes structural identifiers. The pairs of identifiers is then encoded into binary. For each pair requires $2^* \log_2 (N)$ and hence total bits required for N elements is $N^* 2^* \log_2(N)$ bits.

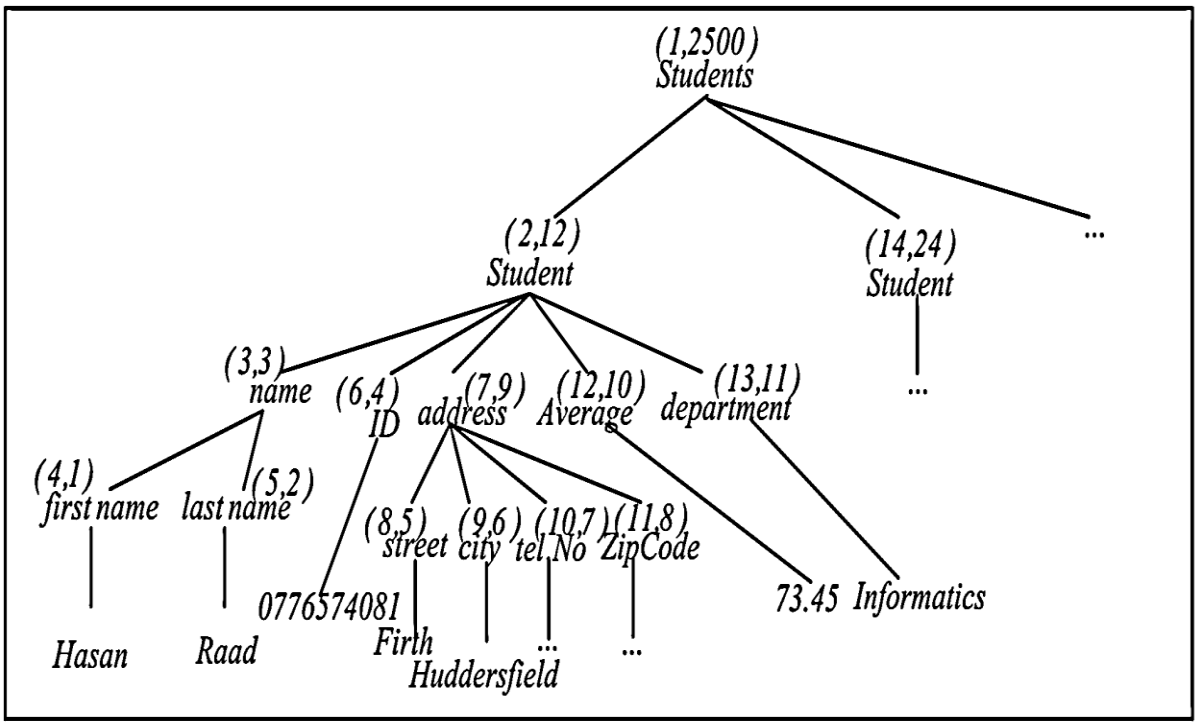

Fig 2.7 Sample of XML document with the structural identifiers of its nodes

Data parts of the XML document is compressed using different encoding technique. For this, XQPoint separated these data into different parts according to their position path from the root.

a. Integer data types:

   XQPoint uses variable-byte coding to encode integer numbers. In this encoding scheme, integer values are stored as a sequence of variable bytes where first seven bits of each byte stores the part of integers and the last bit stores if that byte is last byte of the representation of the integer value.

b. Floating-point data types:

Floating-point data types are compressed using predictive floating point compression. This technique splits the floating-point value into sign, exponent, and mantissa and then encodes using context based arithmetic coder [11].

c. Enumerated data and text data types:

Text and enumerated data types are encoded using Fixed-Point Number Representation Technique (FPNRT). XQPoint compute the numeric value of the word (f) using the following formula:

$$f = \sum_{i=0}^{n-1} (ASC - 65) * 26^i$$

Where n represents a word length, ASC is the ASCII code of any letter, i is the letter's position in the word.

2.2.3.2 System Architecture

The system architecture of the XQPoint is shown in Fig 2.8. The XML Analyzer component of the architecture is to analyze the XML document to create the dictionary for all elements and attributes with their structural identifiers as described in previous section. Also data types of all data values are also identified. All these information is passed to the XML Encoder section.

The XML Encoder section encodes the element and attributes using binary coding described in previous section. Also, data values are also encoded by the Encoder according to the types of data values. Finally, the encoded data are written into compressed file maintaining the structure of the original XML document. Query manipulator is the module for direct query support where the query are parsed and the required information is extracted from the compressed domain. XQPoint claims to support Content-and-Structure (CAS) queries, where content represent the data itself of the XML document and the structure represent tags and attributes.

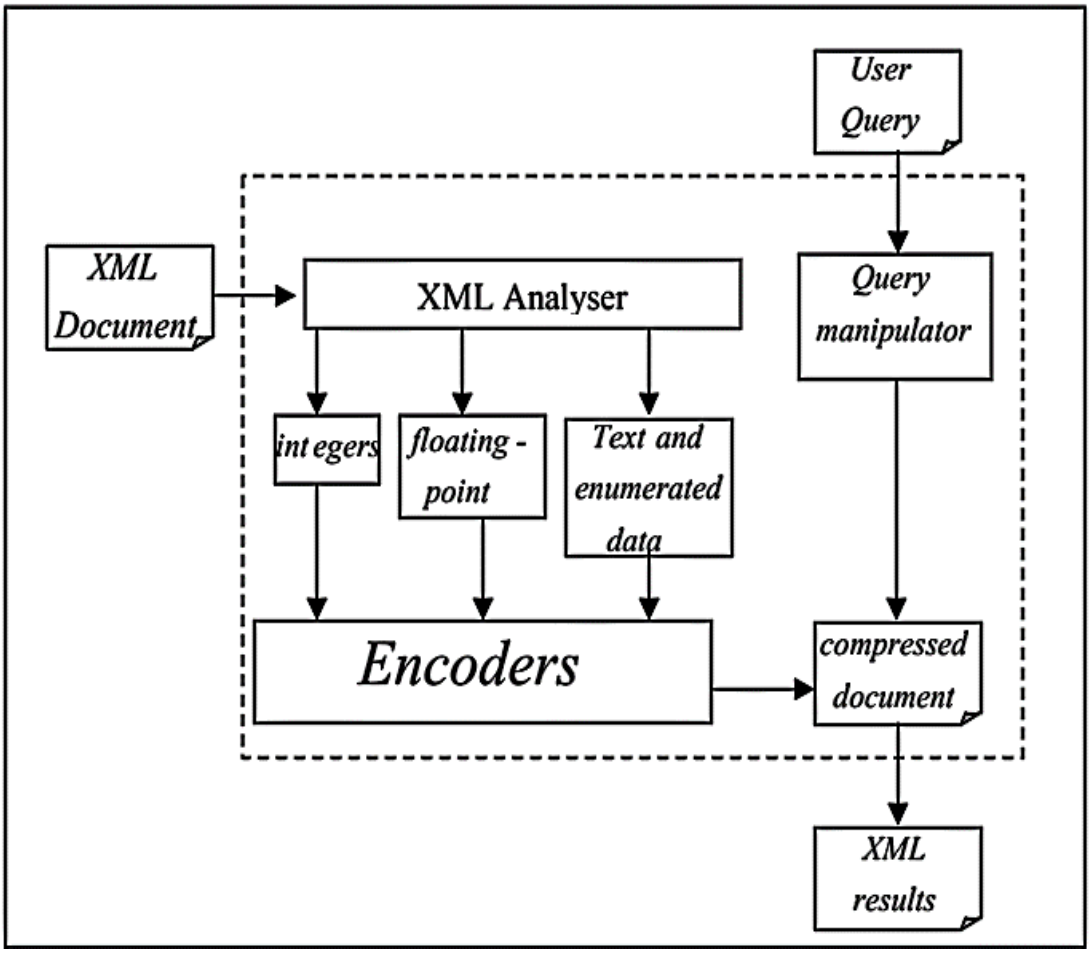

Fig 2.8The system architecture of XQPoint

2.3 Methodology

The methodology applied in this dissertation work is totally empirical approach where the conclusion is drawn based on the observation and analysis of the result from the study. The approach is based on tracing the outcomes of the XML compression over various types of randomly generated XML files. The main focus of this dissertation is to compare three homomorphic XML compressor stated above in section 2.2. The compressors are tested with the datasets and the output is collected for further analysis. The output is analyzed in quantitative approach from various perspective of data compression. And finally, conclusion is drawn based on the result of the analysis.

CHAPTER 3

IMPLEMENTATION AND TESTING ENVIRONMENT

3.1 Implementation Tools

All the implementation is done in C#.Net programming language using visual studio 2012.

3.1.1 Visual Studio 2012

Visual studio is the integrated development environment developed by Microsoft. This IDE can be used to develop windows applications, windows mobile applications, websites, and web services. Visual studio includes a code editor with intellisense as well as code refactoring. It also have the integrated debugger for debugging the .Net codes. Visual studio supports different programming languages. Built-in languages include visual C++.NET, VB.NET, Visual C#.NET, F#.NET. Also, other language services can be installed for support in visual studio.

3.1.2 C#.NET Programming Language

C#.NET is the Microsoft implementation of the C# language integrated in .NET framework. C# programming language is more like Java programming language. C# is an object oriented programming language. C# language is power full language that is used to build small to enterprise level application.

XmlReader within System.Xml namespace from .NET framework is used for parsing the XML document. Also XmlWriter within System. Xml namespace is used for writing the XML file.

3.1.3 Microsoft Excel

Microsoft excel is the spreadsheet program developed by Microsoft that is used to store the result of the XML compression testing. This is the powerful tool for data manipulation. All the tables and charts are generated from the result data in excel.

3.2 Testing Environment

The implemented XML compressors are executed in the machine with following configuration.

Machine type: Laptop

Model: Lenovo G510

Processor: Intel® Core™ i5-4200M CPU @ 2.5GHZ

Installed memory (RAM): 4GB

Operating System: Windows 8.1 Enterprise

System type: 64bit OS, X64 based processor

Hard disk: WD5000LPCX 500GB, 5400 rpm

CHAPTER 4

DATA PREPARATION AND RESULT ANALYSIS

4.1 Data Preparation

There is no universally accepted XML text file corpus available for XML compression. The analysis of XML compressor is done based on the primary datasets of random XML file generated from the XML generator application. The required XML file generator is developed. Primary data sets are generated from the developed XML generator. The generated XML file have random nodes and some scattered data of varies data types.

4.1.1 Primary Dataset 1

This dataset is collection of XML files generated from the XML generator application with different ranges of node counts with fixed distinct tag counts and fixed depth. This data set contains 10 file with node count range from 10000 to 100000 with difference of 10000. The element count and the depth is set fixed with 100 and 20 respectively. The size of file range from 4MB to 365MB.

4.1.2 Primary Dataset 2

This dataset is collection of XML files generated from the XML generator application with fixed node count with varying distinct element counts fixed depth. This dataset contains 10 files with element count from 100 to 1000 with difference of 100. The node count and the depth is set fixed with 50000 and 20 respectively. The size of file range from 92MB to 93MB.

4.1.3 Primary Dataset 3

This dataset is collection of XML files generated from the XML generator application with fixed distinct tag and node counts with varying depth. This data set contains 10 files with depth varying from 10 to 100 with difference of 10. The node count and the element count is set fixed with 50000 and 500 respectively. The size range from 26MB to 122MB.

4.1.4 Secondary Dataset 1

This dataset is not the generated XML files but the real data collected from various source. This dataset contains XML file with different natures used for XML compression tools benchmarking [12]. This same XML corpus is used in [1]. From this corpus only 10 files are selected for simplicity. The details of the XML files is shown in the table

Document Name	Size (MB)	Tags	No. of Nodes	Depth	Data Ratio
BaseBall.xml	0.65	46	57812	6	0.11
DBLP.xml	130.72	32	4718588	5	0.58
DCSD-Small.xml	10.6	50	6190628	8	0.45
EXI-Array.xml	22.18	47	1168115	10	0.68
EXI-Invoice.xml	0.93	52	78377	7	0.57
EXI-Telecomp.xml	0.65	39	651398	7	0.48
SwissProt.xml	112.13	85	13917441	5	0.6
TCSD-Small.xml	10.95	24	831393	8	0.78
XMark1.xml (Small)	11.4	74	520546	12	0.74
XMark2.xml (Medium)	113.8	74	5167121	12	0.74

Table 4.1 Properties of XML files in secondary dataset 1

4.2 Evaluation Metrics

The performance of XML compression tools is evaluated on the basis of following matrices:

4.2.1 Compression ratio

It is the ratio between the size of compressed and uncompressed XML documents. It is the ratio of space occupied by compressed document over original document. Subtracting the ratio from 1 gives the ratio of space earned after compression. The compression ratio for an XML document can be calculated as:

$$CR = 1 - \frac{size\ of\ compressed\ document}{size\ of\ original\ document}$$

4.2.2 Compression time

It is the elapsed time during compression process of XML file i.e. the period of time between the start of compression program execution on a document until all the data are written to disk. Compression time for all primary and secondary dataset is compared between different compression techniques.

4.2.3 Decompression time

It is the elapsed time during decompression of the compressed file i.e. the period of time between the start of decompression program execution on a compressed document until original XML document is delivered. Decompression time for all primary and secondary dataset is compared between different compression techniques.

4.2.4 Memory Consumption

It is the memory consumption by the compression application during the compression process of XML file. Changes in memory consumption in different dataset is compared. Memory consumed by the application is obtained using built-in function from .NET framework.

4.3 Results and Analysis

All implemented XML compression tools are executed for all the datasets and the result is imported to the excel file separately for each dataset.

4.3.1 Results

The test is executed 10 times for each dataset and the result is taken as average of all the output. The average result for each datasets is shown in the tables shown from Table 4.2to Table 4.5.

XGRIND						
Input file	**original size (KB)**	**compressed size (KB)**	**compression time (sec)**	**compression Memory(MB)**	**decompression time (sec)**	**decompression memory (MB)**
10000-100-20.xml	4155	397	1.2187841	42	0.2031435	63
20000-100-20.xml	15488	1241	1.5937351	44	0.3750031	66
30000-100-20.xml	33432	2661	1.984364	48	0.6718821	67
40000-100-20.xml	59227	4673	3.8437799	47	1.1250277	68
50000-100-20.xml	93004	6804	4.4687578	48	1.875015	68
60000-100-20.xml	131731	9288	5.2188072	48	2.0469521	67
70000-100-20.xml	179242	10814	5.0000332	49	3.5781823	68
80000-100-20.xml	234602	13935	7.3594101	63	4.031293	69
90000-100-20.xml	297356	17368	13.5938788	60	7.2656911	71
100000-100-20.xml	364312	21239	21.7033209	62	9.0313378	72
XPRESS						
Input file	**original size (KB)**	**compressed size (KB)**	**compression time (sec)**	**compression Memory(MB)**	**decompression time (sec)**	**decompression memory (MB)**
10000-100-20.xml	4155	356	0.1406568	43	0.4375067	48
20000-100-20.xml	15488	1149	0.7031329	45	0.7968799	48
30000-100-20.xml	33432	2328	0.9219112	46	0.7343803	48
40000-100-20.xml	59227	4082	1.4687394	44	1.2812783	48
50000-100-20.xml	93004	5875	2.250037	47	1.5781372	48
60000-100-20.xml	131731	7972	2.8125393	46	2.0408875	47
70000-100-20.xml	179242	9022	3.7344071	45	2.8125442	47
80000-100-20.xml	234602	11590	4.750039	48	3.4688318	43
90000-100-20.xml	297356	11422	11.312612	52	4.4688112	48
100000-100-20.xml	364312	13954	18.9064308	47	5.1122541	48
XQPOINT						
Input file	**original size (KB)**	**compressed size (KB)**	**compression time (sec)**	**compression Memory(MB)**	**decompression time (sec)**	**decompression memory (MB)**
10000-100-20.xml	4155	315	0.1250014	43	0.2812381	44
20000-100-20.xml	15488	994	0.4062707	44	0.6718329	45

30000-100-20.xml	33432	1659	0.8593995	46	0.7031296	45
40000-100-20.xml	59227	2897	0.8593645	45	1.1875272	45
50000-100-20.xml	93004	5875	1.2656357	48	1.609372	46
60000-100-20.xml	131731	6655	1.7344066	46	2.1094323	46
70000-100-20.xml	179242	7230	2.2812688	45	2.7344321	46
80000-100-20.xml	234602	9244	2.9531493	48	3.3281417	43
90000-100-20.xml	297356	5475	10.040463	45	3.556825	46
100000-100-20.xml	364312	10311	17.0314458	47	5.124862	48

Table 4.2 Output for dataset 1

XGRIND						
Input file	**original size (KB)**	**compressed size (KB)**	**compression time (sec)**	**compression Memory (MB)**	**decompression time (sec)**	**decompression memory (MB)**
50000-100-20.xml	93282	4361	4.5468897	51	2.7500049	49
50000-200-20.xml	92918	5281	4.4219125	50	2.5000242	49
50000-300-20.xml	93012	5269	4.4531332	51	2.5156369	50
50000-400-20.xml	92311	6226	4.515646	52	2.5937544	50
50000-500-20.xml	93348	6280	5.0156442	53	2.6094409	51
50000-600-20.xml	92494	7160	5.0000148	53	2.4843845	51
50000-700-20.xml	92906	7176	5.0000341	53	2.4844034	50
50000-800-20.xml	92528	8085	5.9375161	54	3.562525	52
50000-900-20.xml	92237	8068	6.5062657	54	3.5468975	53
50000-1000-20.xml	92458	8079	6.5062674	55	3.5000066	53
XPRESS						
Input file	**original size (KB)**	**compressed size (KB)**	**compression time (sec)**	**compression Memory (MB)**	**decompression time (sec)**	**decompression memory (MB)**
50000-100-20.xml	93282	4361	3.2812799	47	1.6093863	46
50000-200-20.xml	92918	3423	3.4687583	51	2.9375798	47
50000-300-20.xml	93012	5269	3.6875227	47	1.7500079	49
50000-400-20.xml	92311	5303	4.1718945	46	1.7031386	50
50000-500-20.xml	93348	4413	4.1719051	51	1.8029341	50
50000-600-20.xml	92494	7160	4.4687866	47	2.421915	50
50000-700-20.xml	92906	6247	4.4843787	46	1.7969061	51
50000-800-20.xml	92528	7159	4.5156459	51	2.5624662	51
50000-900-20.xml	92237	7146	4.5469497	47	1.6562822	50
50000-1000-20.xml	92458	6230	4.6719523	47	2.3750043	51
XQPOINT						
Input file	**original size (KB)**	**compressed size (KB)**	**compression time (sec)**	**compression Memory (MB)**	**decompression time (sec)**	**decompression memory (MB)**
50000-100-20.xml	93282	2495	1.3906527	46	2.1250572	42
50000-200-20.xml	92918	1565	1.8593912	50	1.6719067	43
50000-300-20.xml	93012	4339	1.8906223	47	1.7031399	43

50000-400-20.xml	92311	4589	2.0937827	51	1.6875313	45
50000-500-20.xml	93348	5347	2.5625106	47	1.765896	47
50000-600-20.xml	92494	5310	2.5937622	50	1.6562653	48
50000-700-20.xml	92906	6247	2.6250134	47	1.6875317	47
50000-800-20.xml	92528	4384	2.7500144	47	1.7031354	49
50000-900-20.xml	92237	6224	2.7656389	51	1.5781471	49
50000-1000-20.xml	92458	5305	3.6249973	47	1.5781565	50

Table 4.3 Output for dataset 2

XGRIND						
Input file	**original size (KB)**	**compressed size (KB)**	**compression time (sec)**	**compression Memory (MB)**	**decompression time (sec)**	**decompression memory (MB)**
50000-500-10.xml	121622	6998	1.9843705	48	3.2500361	53
50000-500-20.xml	92394	5316	1.8437823	49	2.6875149	53
50000-500-30.xml	62997	3624	1.7968991	51	2.3281544	54
50000-500-40.xml	49091	3315	1.6093966	50	2.1094032	53
50000-500-50.xml	40711	3156	1.2969045	50	1.9687402	47
50000-500-60.xml	35504	3108	1.2656349	50	1.8906539	48
50000-500-70.xml	31942	3115	1.2500127	50	1.8125024	48
50000-500-80.xml	29594	3182	1.1875091	50	1.7656077	49
50000-500-90.xml	27835	3271	1.125008	50	1.7031231	47
50000-500-100.xml	26454	3373	1.0156468	50	0.8718812	44
XPRESS						
Input file	**original size (KB)**	**compressed size (KB)**	**compression time (sec)**	**compression Memory (MB)**	**decompression time (sec)**	**decompression memory (MB)**
50000-500-10.xml	121622	4713	1.9218879	49	3.037847	48
50000-500-20.xml	92394	4504	1.3594426	48	2.618768	50
50000-500-30.xml	62997	3701	1.2812586	51	2.23849	51
50000-500-40.xml	49091	3375	1.2344017	47	2.21623	48
50000-500-50.xml	40711	3206	1.1406518	49	2.003125	50
50000-500-60.xml	35504	2796	1.0968585	46	1.0859381	52
50000-500-70.xml	31942	2834	1.0312651	49	1.0781273	46
50000-500-80.xml	29594	2922	1.00998	49	1.0552623	47
50000-500-90.xml	27835	2748	0.9687751	51	1.0423654	49
50000-500-100.xml	26454	2612	0.8593978	46	1.041	50
XQPOINT						
Input file	**original size (KB)**	**compressed size (KB)**	**compression time (sec)**	**compression Memory (MB)**	**decompression time (sec)**	**decompression memory (MB)**
50000-500-10.xml	121622	3092	1.4843738	50	2.0000718	54
50000-500-20.xml	92394	4504	1.1406215	55	1.5012791	56
50000-500-30.xml	62997	3071	1.0156517	48	1.42358	55

50000-500-40.xml	49091	2393	0.9219006	51	1.0468832	50
50000-500-50.xml	40711	1984	0.8750081	54	0.9531505	52
50000-500-60.xml	35504	2086	0.8593814	50	0.8437728	53
50000-500-70.xml	31942	1876	0.8281405	52	0.7968868	55
50000-500-80.xml	29594	2034	0.8281297	55	0.796863	56
50000-500-90.xml	27835	1913	0.67188	58	0.7500066	48
50000-500-100.xml	26454	1818	0.5781273	52	0.718757	49

Table 4.4 Output for dataset 3

XGRIND						
Input file	**original size (KB)**	**compressed size (KB)**	**compression time (sec)**	**compression Memory (MB)**	**decompression time (sec)**	**decompression memory (MB)**
BaseBall.xml	656	272	1.4531746	44	0.578153	44
EXI-Invoice.xml	956	325	1.2344142	82	0.3125011	64
EXI-Telecomp.xml	10326	3969	4.1562621	83	4.3437942	84
XBench-DCSD-Small.xml	10831	4649	5.1875363	256	4.817320889	95
XBench-TCSD-Small.xml	11319	5972	7.9063016	355	7.597677445	256
XMark1.xml	11596	3059	8.9531673	395	8.319724594	284
EXI-Array.xml	22591	6468	4.8750401	110	7.2494098	105
SwissProt.xml	115467	62442	41.0662233	455	39.79246544	350
XMark2.xml	115774	30696	41.8531202	259	38.74865026	224
DBLP.xml	134243	75188	49.9485798	449	41.95618452	375
XPRESS						
Input file	**original size (KB)**	**compressed size (KB)**	**compression time (sec)**	**compression Memory (MB)**	**decompression time (sec)**	**decompression memory (MB)**
BaseBall.xml	656	217	0.3593794	44	0.3593807	41
EXI-Invoice.xml	956	286	0.2187527	49	0.468748	42
EXI-Telecomp.xml	10326	3812	2.6719018	50	8.2188363	42
XBench-DCSD-Small.xml	10831	4421	2.4375229	65	3.398412	45
XBench-TCSD-Small.xml	11319	5910	3.9687922	74	6.478256	49
XMark1.xml	11596	2804	3.9375423	61	5.256897	52
EXI-Array.xml	22591	6514	3.9844009	50	9.0782195	49
SwissProt.xml	115467	58236	31.3284475	57	34.1245741	52
XMark2.xml	115774	28036	35.8285079	52	34.658741	48
DBLP.xml	134243	72025	38.3285346	63	38.12475482	53
XQPOINT						
Input file	**original size (KB)**	**compressed size (KB)**	**compression time (sec)**	**compression Memory (MB)**	**decompression time (sec)**	**decompression memory (MB)**
BaseBall.xml	656	205	0.1003796	44	0.234376	41
EXI-Invoice.xml	956	253	0.2112679	49	0.4375012	41

EXI-Telecomp.xml	10326	3459	1.5625264	50	8.2813537	41
XBench-DCSD-Small.xml	10831	4327	1.0375122	50	3.105896	43
XBench-TCSD-Small.xml	11319	6907	3.1688083	43	5.9235765	46
XMark1.xml	11596	2654	3.2156726	43	4.7584325	42
EXI-Array.xml	22591	6085	3.405059	50	9.2032134	41
SwissProt.xml	115467	53214	25.7815848	45	30.45879	48
XMark2.xml	115774	26045	31.2034836	43	29.4587935	47
DBLP.xml	134243	70892	33.8285251	40	32.78952	49

Table 4.5 Output for dataset 4

4.3.2 Result Analysis

4.3.2.1 Compression Ratio

Compression ratio is calculated for all four datasets with all the implemented compressors. The results are shown in charts from Fig 4.1 to Fig 4.4.

Fig 4.1 shows the compression ratio of dataset 1 where the files have same number of element count and same depth but with increasing node count. The result shows that increased in node count slightly increases the compression ratio in all three XML compressors. This result is as assumed, because with increase in node count with constant element count, the repetition of the element is increased. Thus the compressor takes the advantage of this repetition to increase the compression ratio.

Fig 4.2 shows the compression ratio for dataset 2 where the files have same number of node count and same depth but with increasing element count. The result is reverse of the result from dataset 1. The compression ratio is slightly decreased with increase in element count. The reason behind this is also the reverse of dataset 1. With increase in element count with constant node count, the repetition is decreased which is the reason for decreased compression ratio. Also, saving the dictionary of elements and its code also increases with element count.

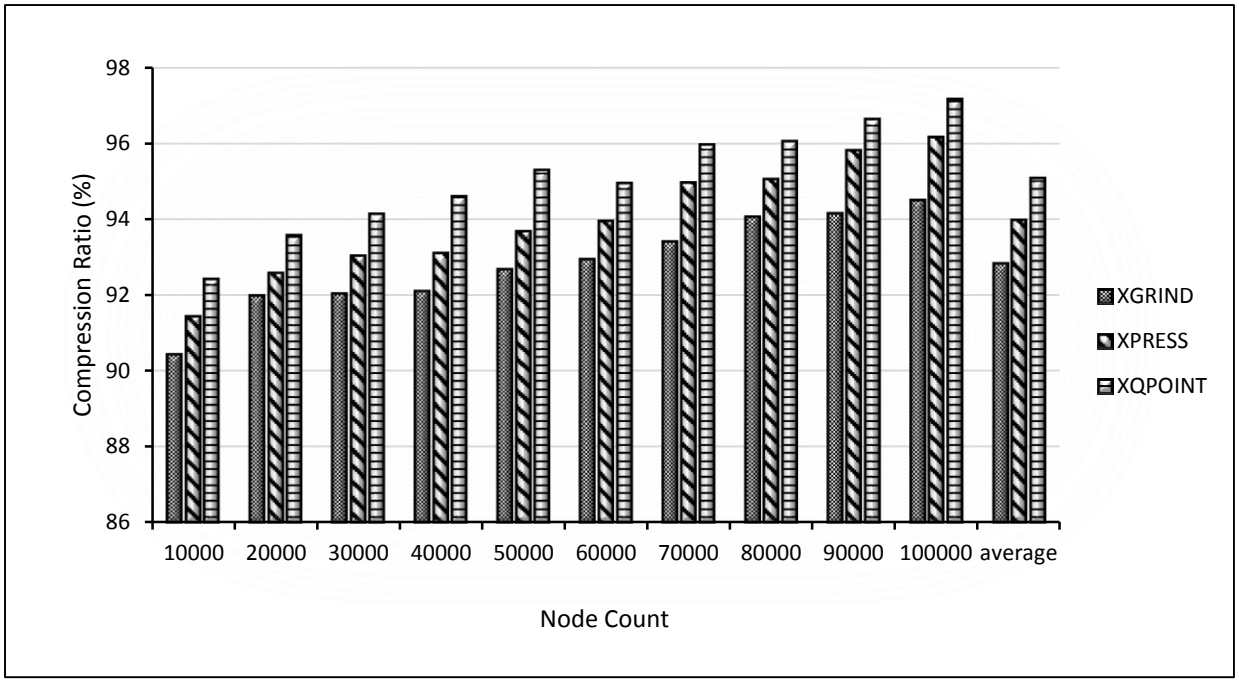

Fig 4.1 Compression ratio comparison for dataset 1

Fig 4.3 shows the compression ratio for dataset 3 where the XML files have constant number of nodes and constant number of element count but increasing depth of the XML tags. The result shows that with increase in depth, the compression ratio is decreased. The XML files with lesser depth contains more data part than the file with higher depth because in XML files with lesser depth most of the nodes are child nodes that contains data. That means increase in depth decreases the data ratio. So, files with same node count and same element count the compression is same to all. But for XML with higher data ratio, compression for data also counts in compression. Thus, this increased the compression ratio in less depth.

Fig 4.4 shows the compression ratio for dataset 4 where XML files are real world data collected from various sources. The files in this data set have different nodes count, element count and depth. Also, they have different ranges of data ratio. There are some errors while compression of these files. This may be due to invalid structure of those XML document since all implemented compressors are schema-aware XML compressors. And also, sometimes the XML generator used found the invalid tag values during compression and decompression which causes error. The files selected in this dataset is among the successful compression. So, error is not shown in the result.

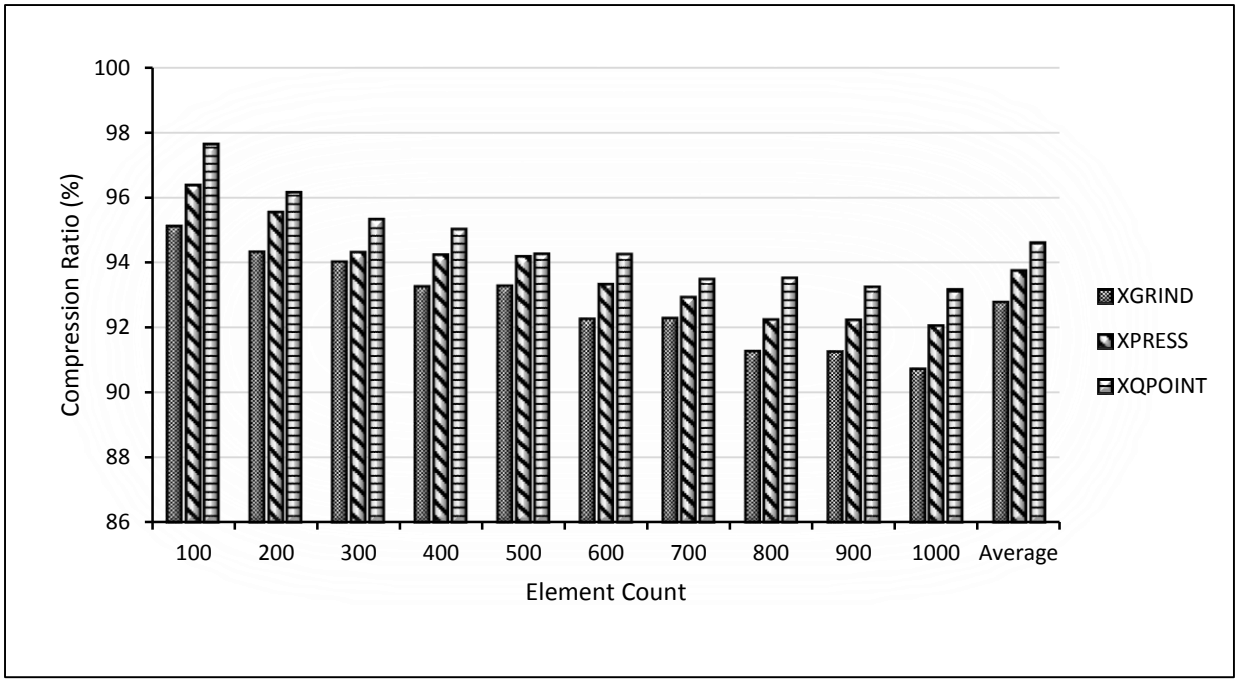

Fig 4.2 Compression ratio comparison for dataset 2

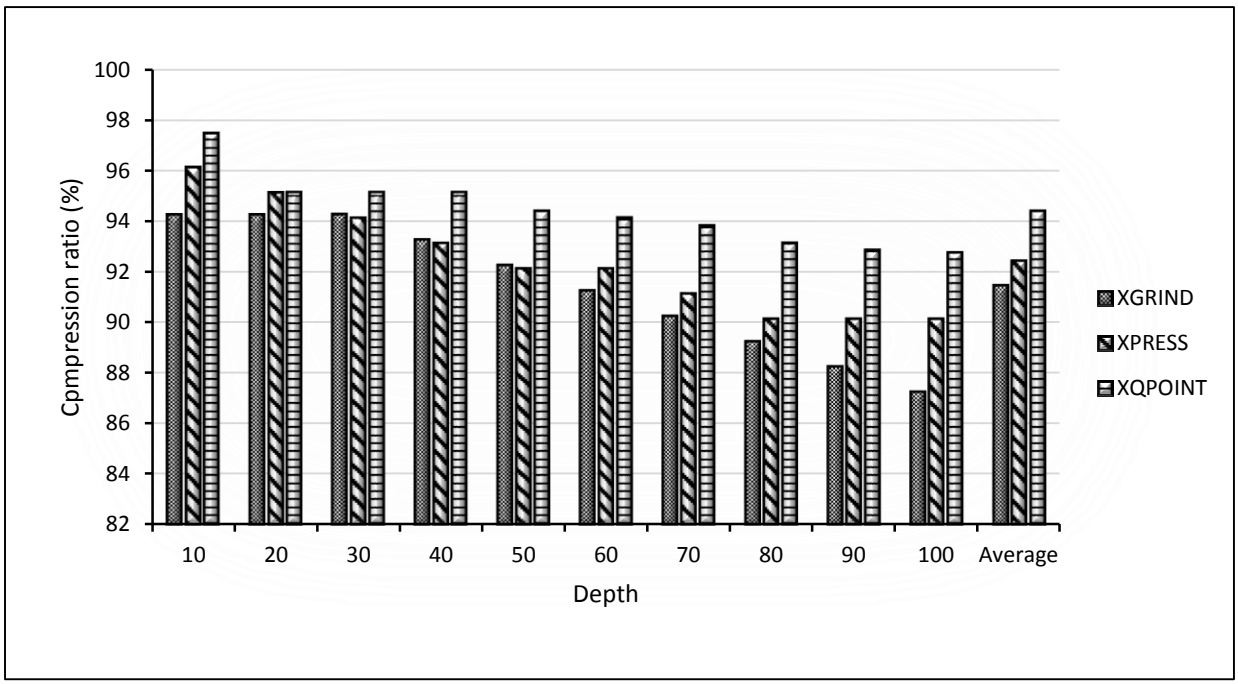

Fig 4.3 Compression ratio comparison for dataset 3

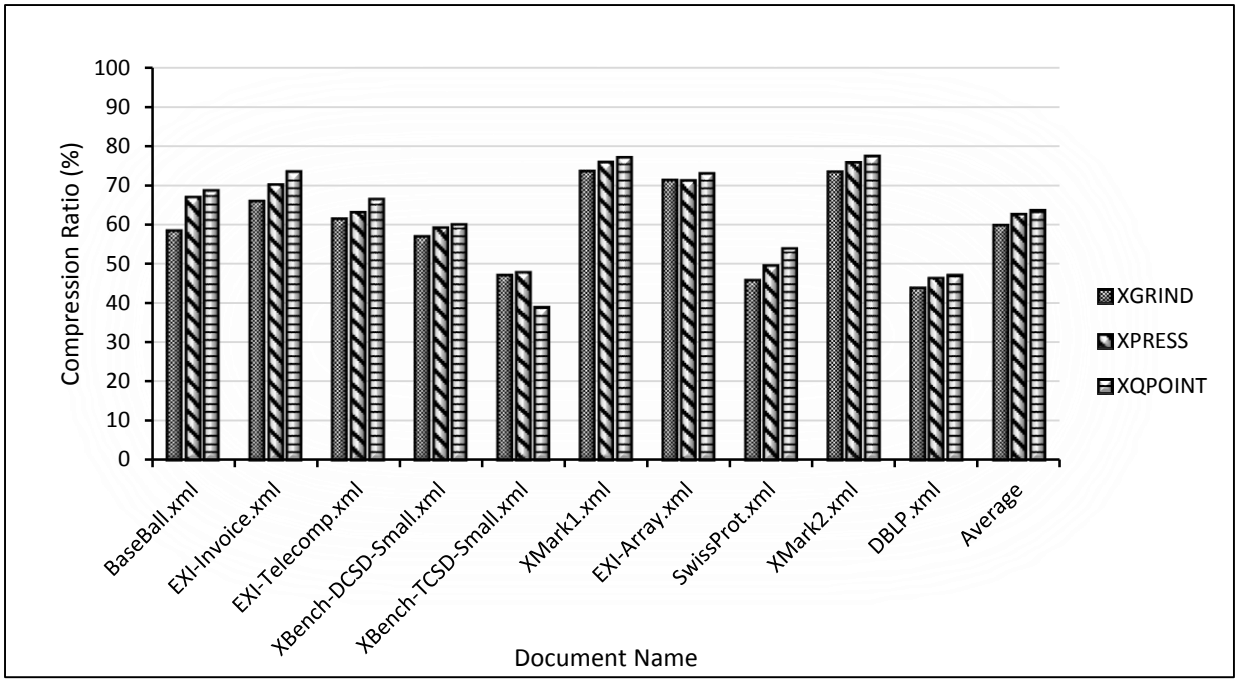

Fig 4.4 Compression ratio comparison for dataset 4

Also, significant portion of the compression ratio in shown for dataset 1, 2 and 3 is because of the whitespace used for indenting the XML tags which is eliminated in compressed document. In dataset 4, there is very less or no indenting in XML files hence have less compression ratio that other three dataset. In all four charts, the last column is the average compression ratio for each XML compressor within the datasets. In all for charts the clear winner is XQPoint with higher compression ratio. XGRIND have the less compression ratio among all three.

4.3.2.2 Compression Time

Compression time is calculated for all four dataset and compared with all the implemented compressors. The result is shown in Fig 4.5 to Fig 4.8.

Fig 4.5 shows the compression time result for dataset1. From the chart it is clear that the compression time increased with increase in node count. The increase in compression time is because of time consumption for encoding more nodes and the increased file size.

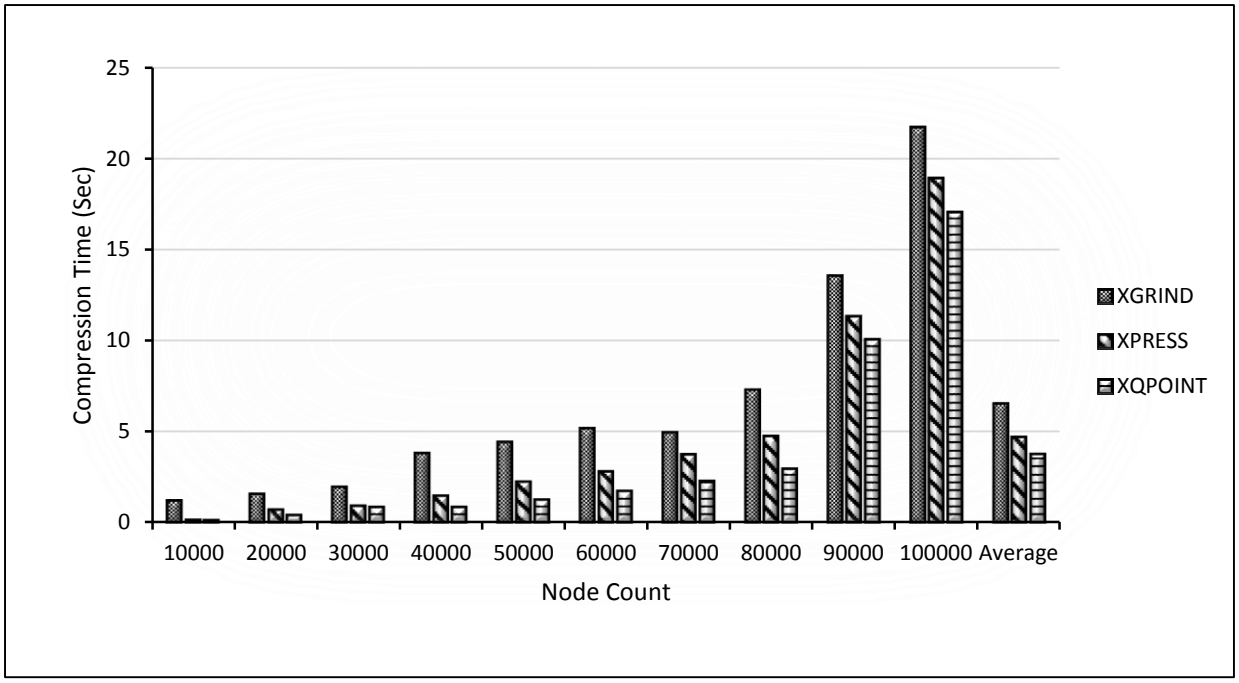

Fig 4.5 Compression time comparison for dataset 1

Fig 4.6 shows the compression time for dataset 2. There is slight increase in compression time with the increase in element count. The increased time is because of the time consumption for the encoding of elements.

Fig 4.7 shows the compression time for dataset 3. The compression time is decreased with the increase in depth. The size of XML file is indirectly proportional to the depth of the XML nodes for constant node count. This cause the decrease in compression time with increase in depth.

Fig 4.8 shows the compression time for dataset 4. Compression time for different files is also different. Larger file require more time for compression.

In all for chart the last column shows the average compression time for three XML compressors. The average compression time for XQPOINT is lowest in all the cases and XGRIND is worst of all with high compression time.

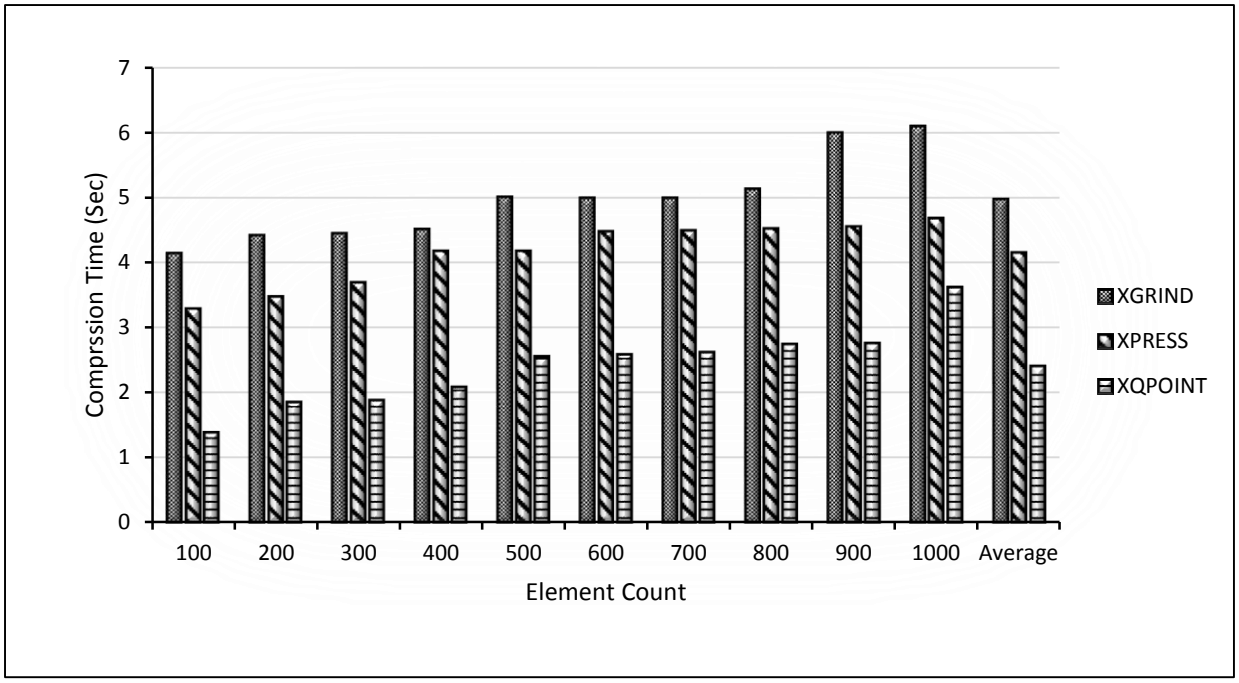

Fig 4.6 Compression time comparison for dataset 2

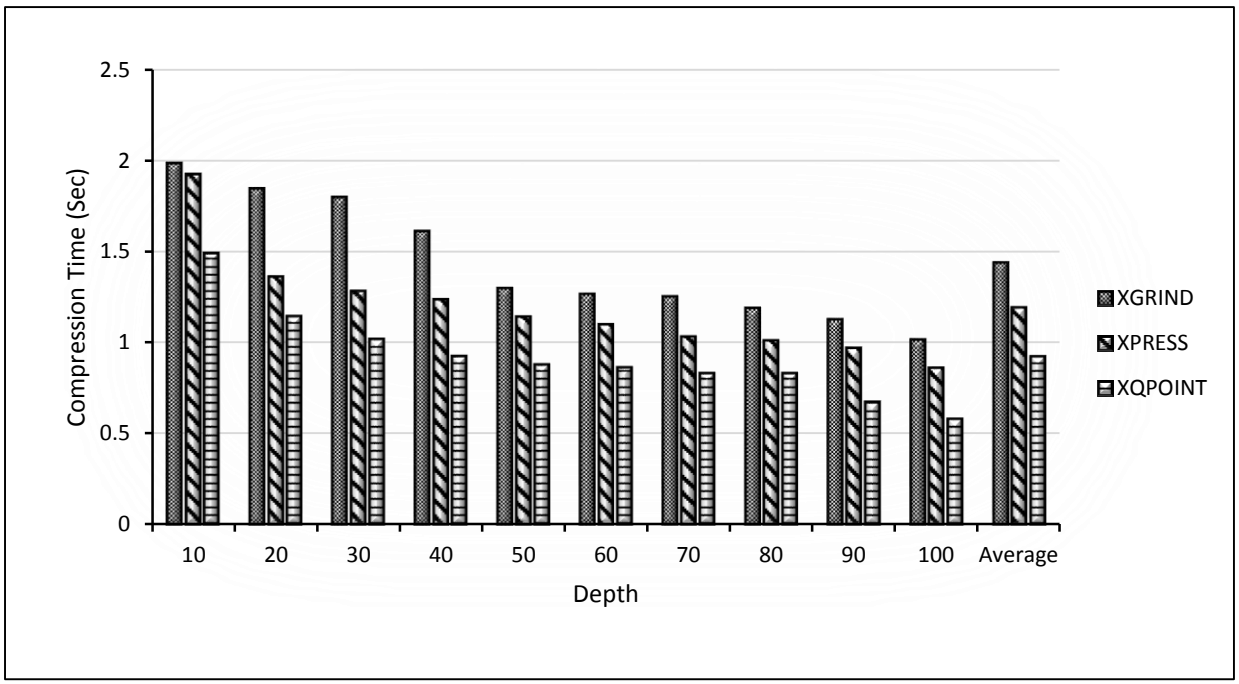

Fig 4.7 Compression time comparison for dataset 3

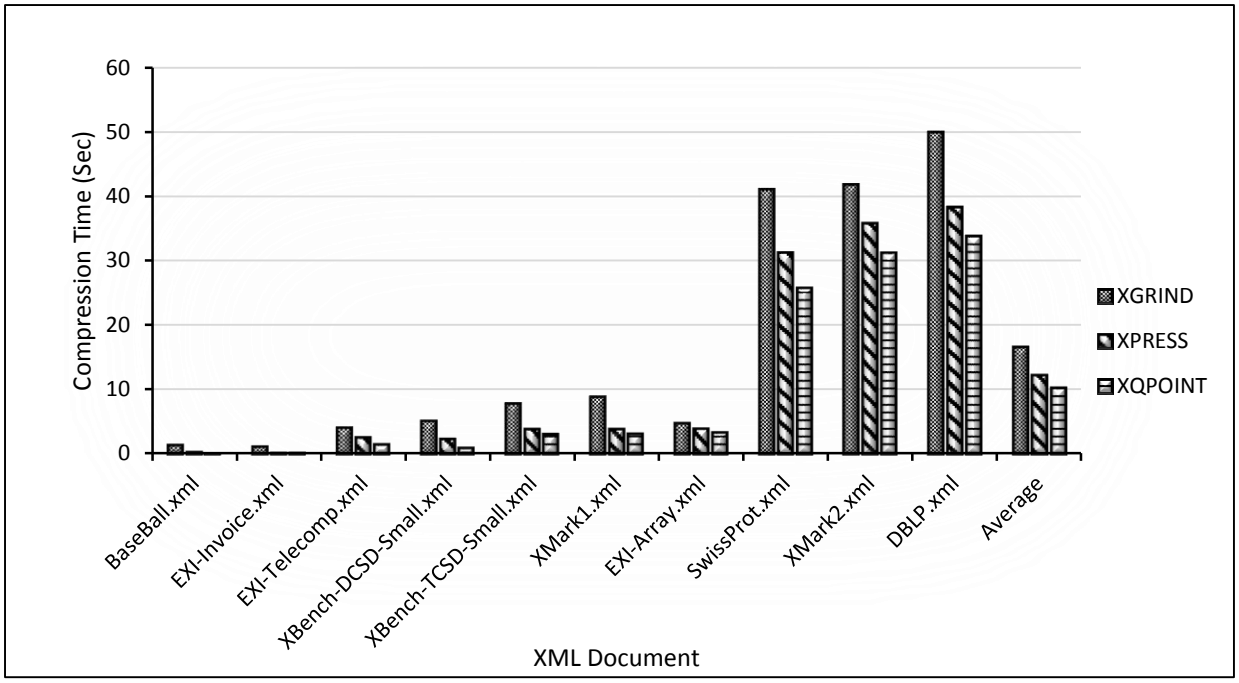

Fig 4.8 Compression time comparison for dataset 4

4.3.2.3 Compression Memory Consumption

Memory consumption is the memory occupied by the application during the compression of XML file. The memory includes the application itself and other information stored during the execution of the application. The memory consumed by the application is obtained by the method from the .NET framework for the current process. The result of memory consumption for all datasets are shown in Fig 4.9 to Fig 4.12.

Fig 4.9 shows the memory consumption for dataset 1. The chart shows that the memory is negligible increasing with increasing with node count. This shows that with increase in node count there is very less effect in memory consumption.

Fig 4.10 shows the memory consumption for dataset2. The chart shows that the memory consumption is proportional to the number of element in the document. Increase in element count, increased the dictionary size storing the code for the element thus, causing the increase in memory.

Fig 4.11 shows the memory consumption for dataset 3. With the increase in depth, the memory consumption is slightly increased.

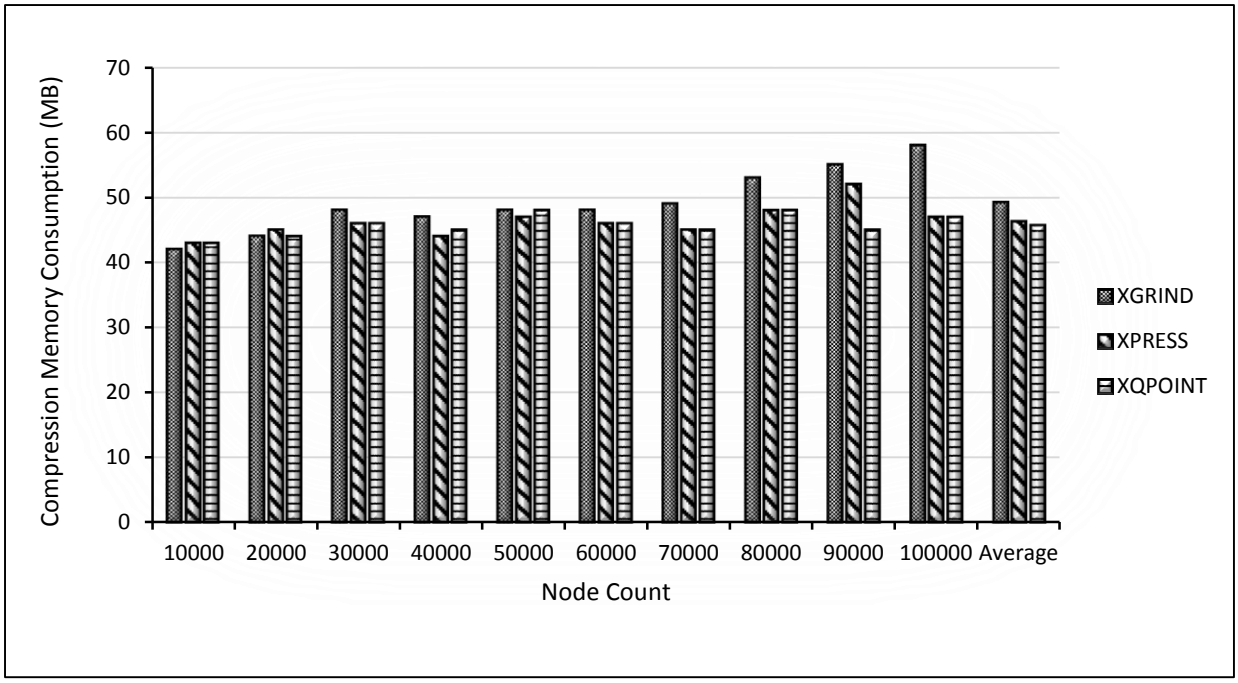

Fig 4.9 Compression memory consumption for dataset 1

Fig 4.12 shows the memory consumption for dataset 4. The memory consumption is low in some files and high in other. The memory consumption is largely dependent with the data volume and data types in the document. XGRIND uses Huffman coding for data encoding which consume more memory to store frequency table and Huffman tree. This causes excessive memory consumption than XPRESS and XQPOINT.

In all the chart, the last column shows the average value for each dataset. The memory consumption for XGRIND is comparatively higher than other two compressors. XQPoint is less in memory consumption in all for set of data.

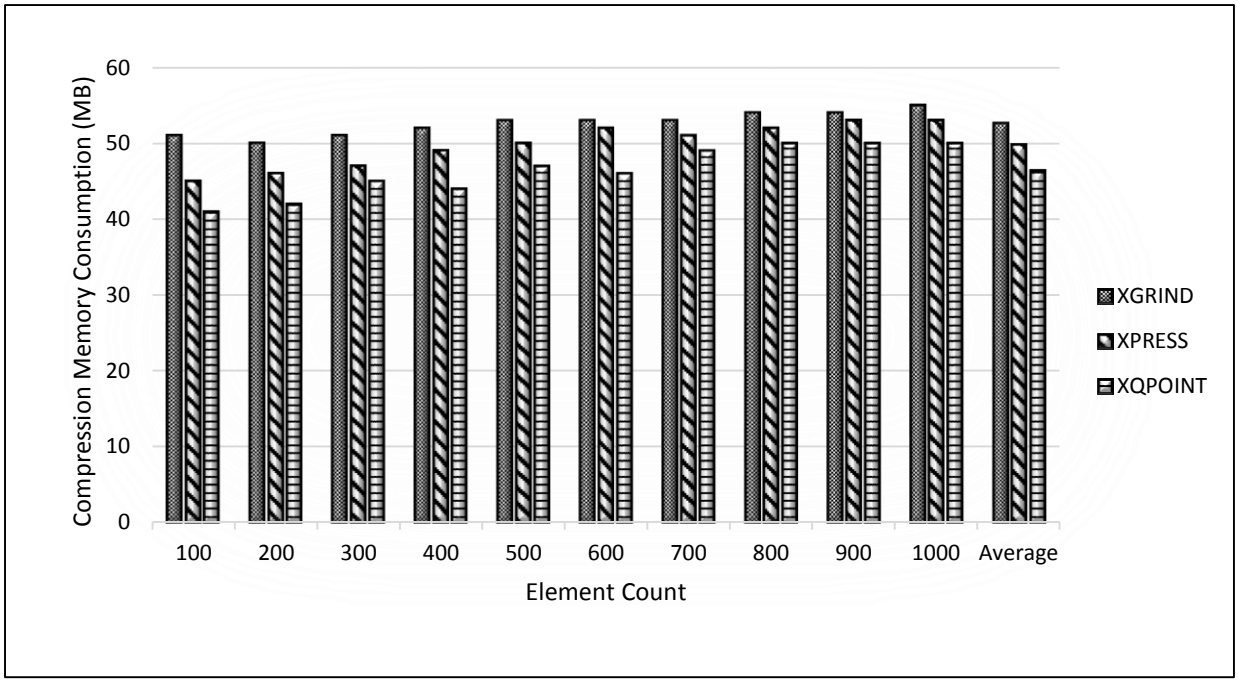

Fig 4.10 Compression memory consumption for dataset 2

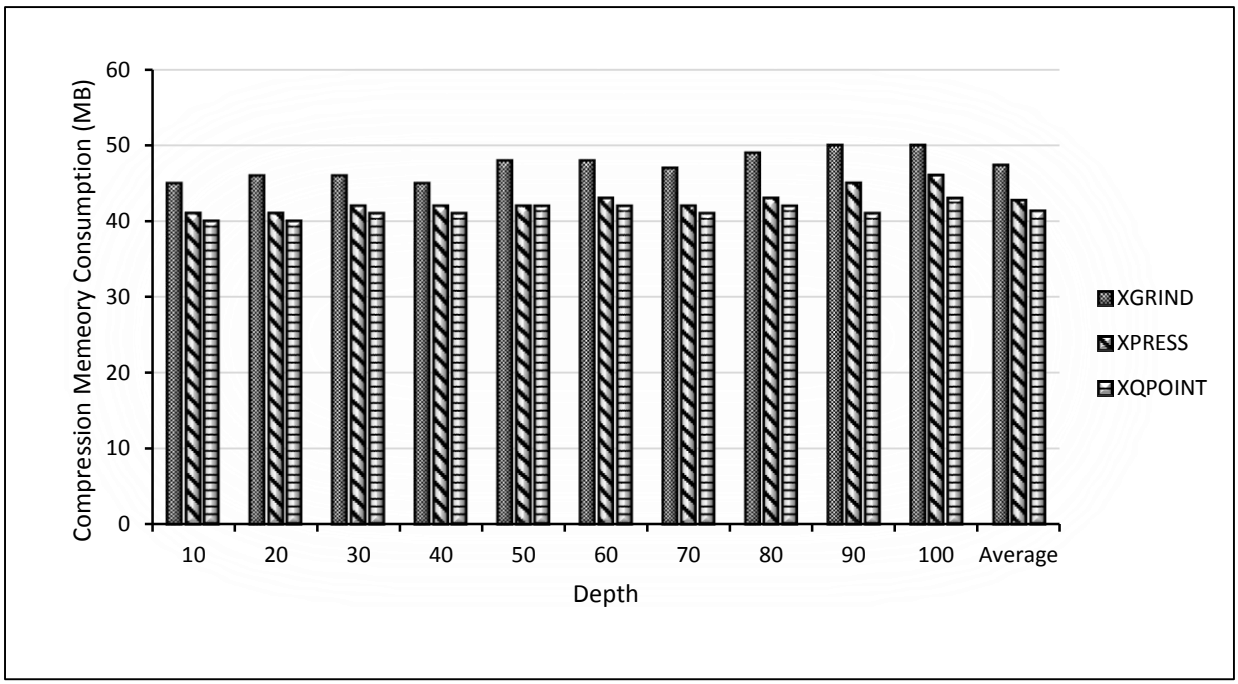

Fig 4.11 Compression memory consumption for dataset 3

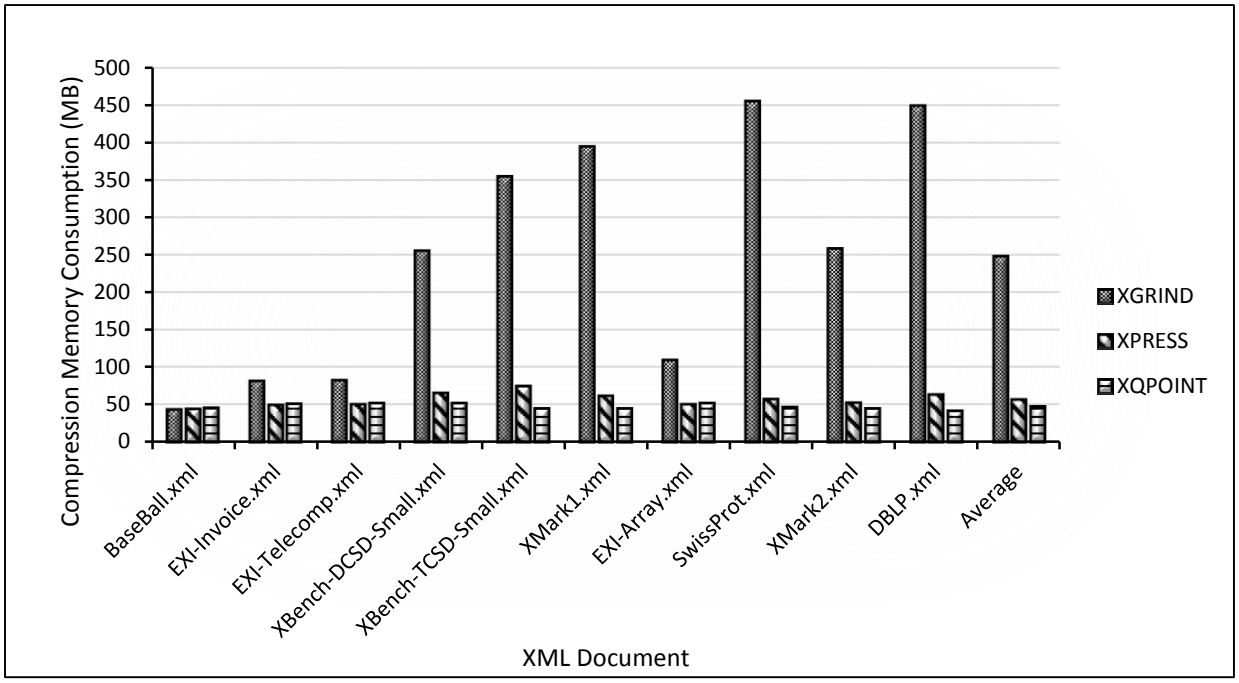

Fig 4.12 Compression memory consumption for dataset 4

4.3.2.4 Decompression Time

Decompression time is time taken by the decompressor for decoding the compressed file to the original XML document. Fig 4.13to Fig 4.16 shows chart representation of the decompression time taken for each datasets.

Fig 4.13 shows the time taken for decompression for the files in dataset 1. The chart shows the changes in time consumption with respect to node count in XML document. From the chart, it is clear that the time consumption increases with the increase in node count. This is obvious because with the increase in nodes in XML document the file size also increase and takes more time to decompress.

Fig 4.14 shows the decompression time for dataset 2. The chart shows that the decompression time is almost constant with increase in element count of the XML document. Sudden increase in time in some column may be due to the changes in data in the document. But the change is not so prominent.

Fig 4.15 shows the decompression time for dataset 3. With the increase in depth of XML document the decompression time is decreased. As the document have same number of nodes, the change in time consumption is because of data contained in the documents. XML document

with less depth have more data values than with higher depth document for same node count. This change in data causes the change in decompression time.

Fig 4.16 shows the decompression time for dataset 4. Time consumption is different for different types of XML document. Document is arranged in order of size that shows that decompression increase with the size of the document.

In all four charts the last column is the average decompression time for three compressors. XQPoint is the clear winner in decompression time consumption. XGRIND is worst of the three.

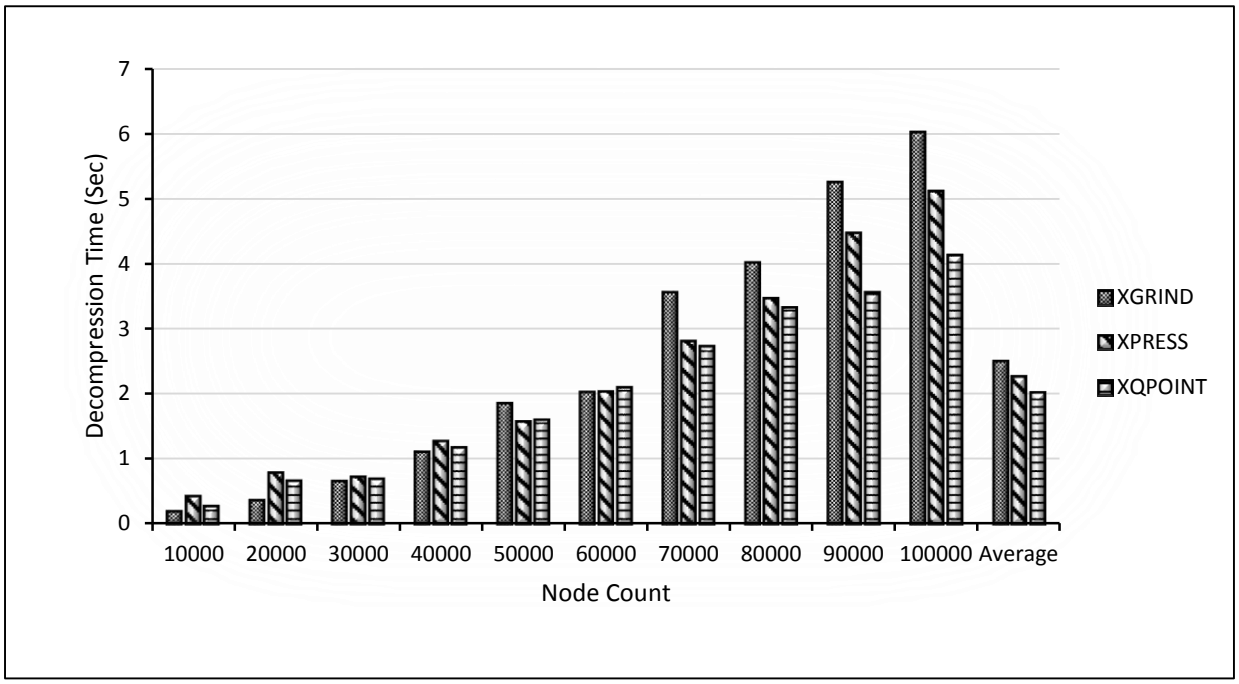

Fig 4.13 Decompression time comparison for dataset1

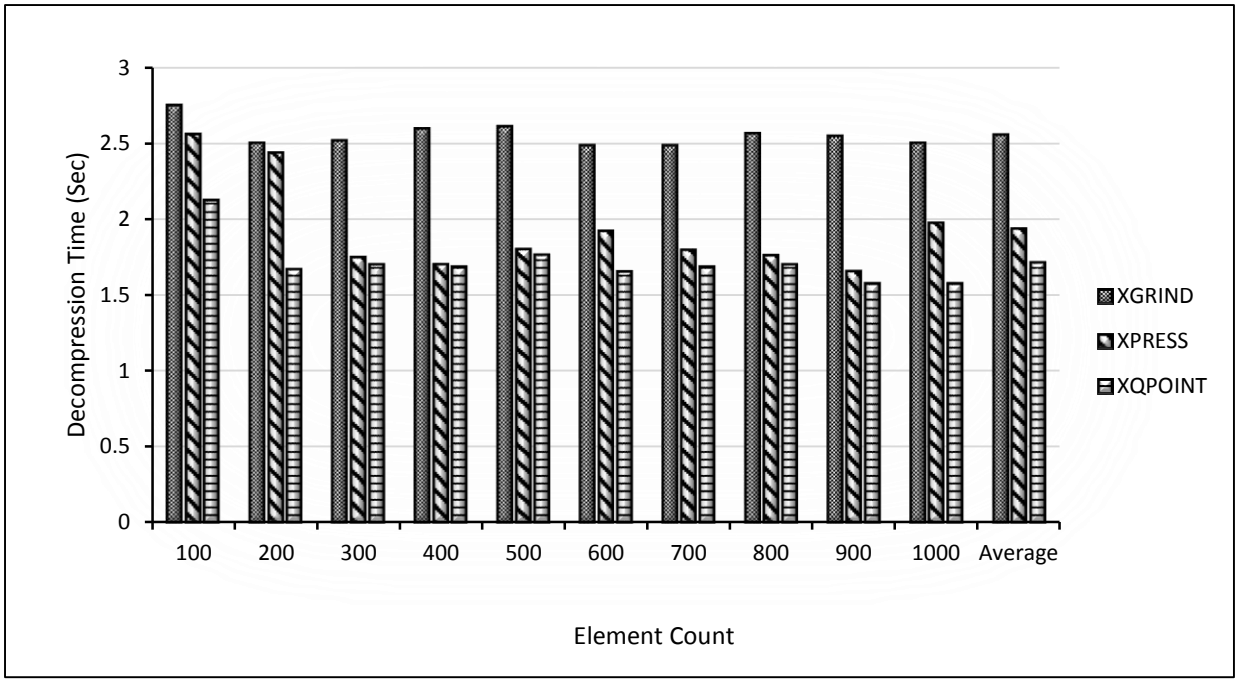

Fig 4.14 Decompression time comparison for dataset 2

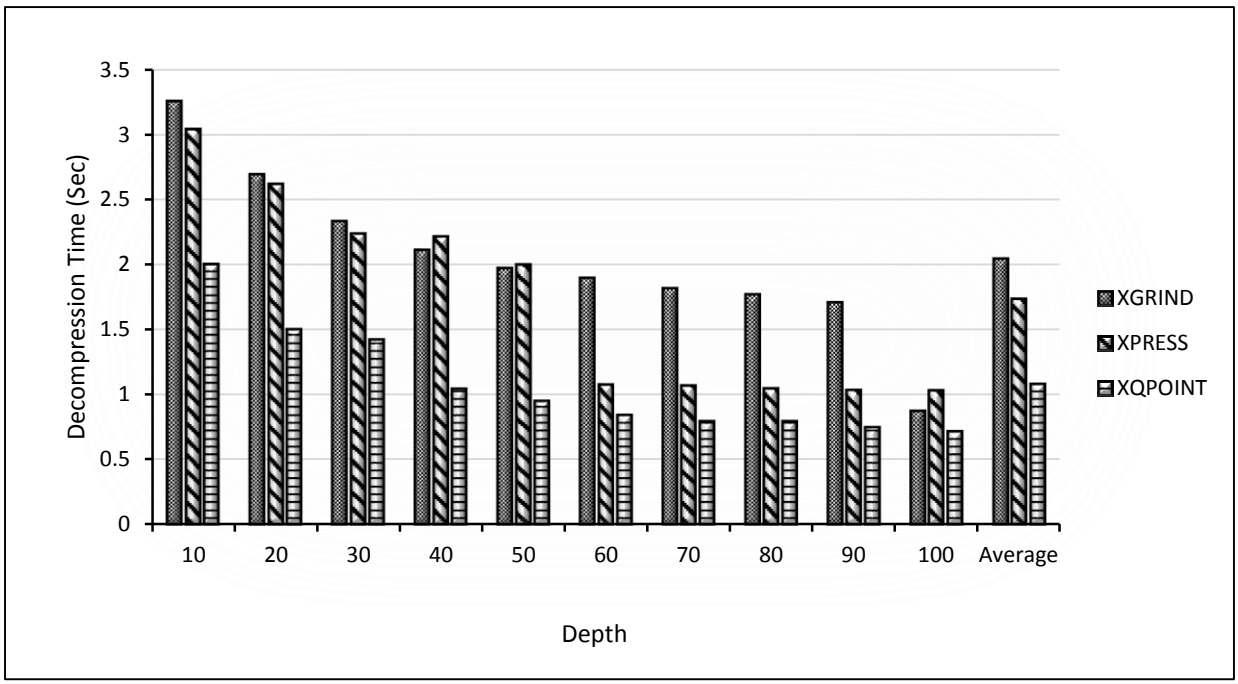

Fig 4.15 Decompression time comparison for dataset 3

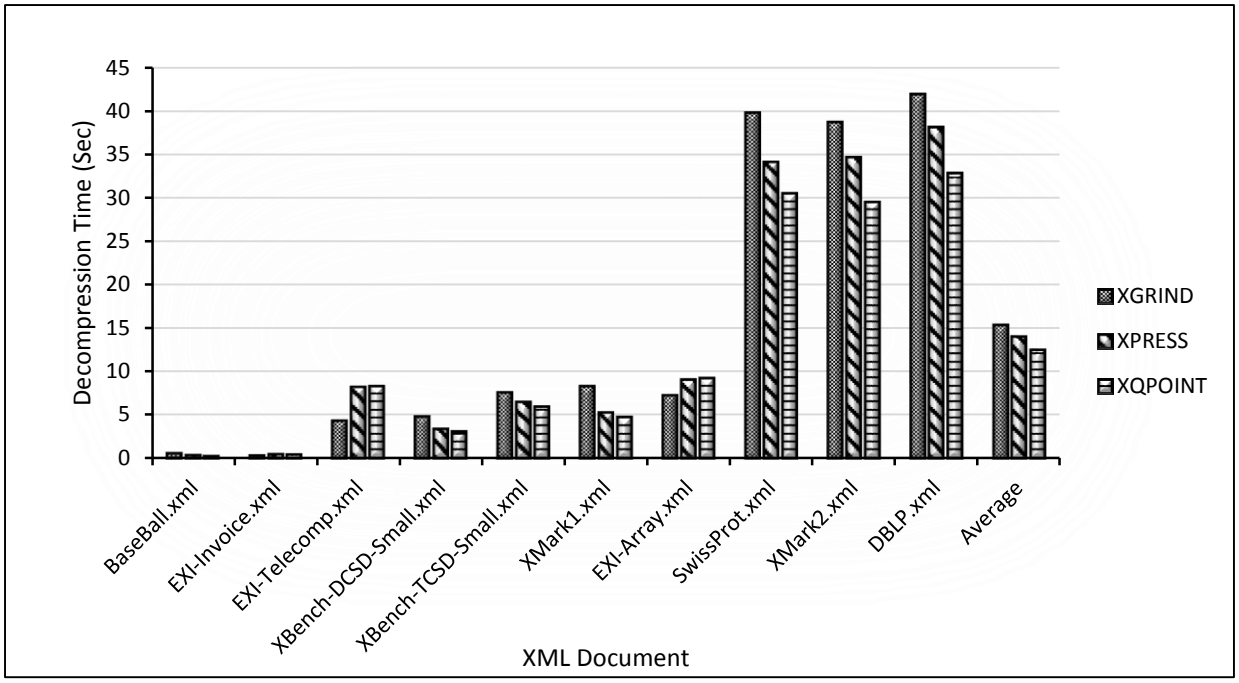

Fig 4.16 Decompression time comparison for dataset 4

4.3.2.5 Decompression Memory Consumption

Fig 4.17 to Fig 4.20 shows the memory consumption by the XML compressor during the decompression of compressed XML document.

Fig 4.17 shows the memory consumption for dataset 1. The memory consumption is slightly inclining. The increase is due to the increase in data and nodes in the file. But the change is not so prominent.

In Fig 4.18, with increase in element count, the memory increases slightly. Increase in the dictionary size with the element count causes the memory changes.

Fig 4.19 shows the memory consumption during decompression for dataset 3. The consumption of memory is decreasing with the increase in depth of the XML document. The change is due to decrease in data values with the increase in depth.

Fig 4.20 shows the memory consumption of dataset 4. There is variations in memory consumption because of different verities of document. Also the node count is very large in some of the files that cause the large memory consumption. XPRESS and XQPOINT much less memory than XGRIND.

From all four chart the overall memory consumption of XQPoint is lower than XGRIND and XPRESS.

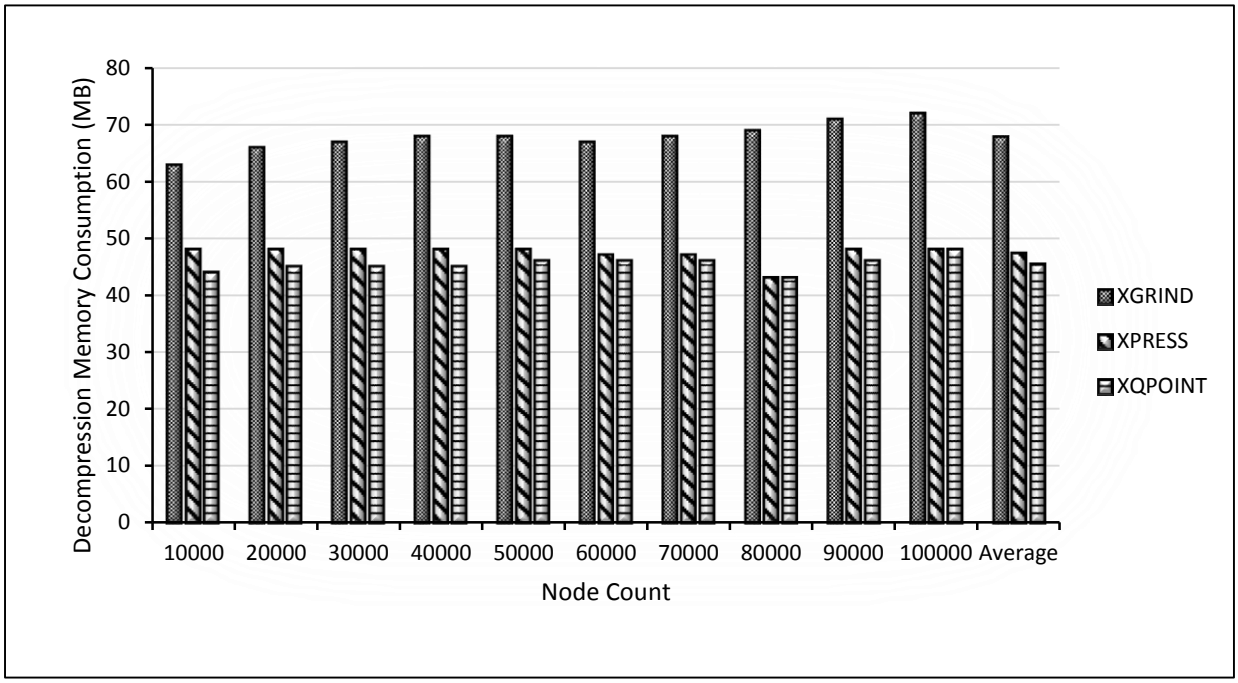

Fig 4.17 Decompression memory consumption for dataset 1

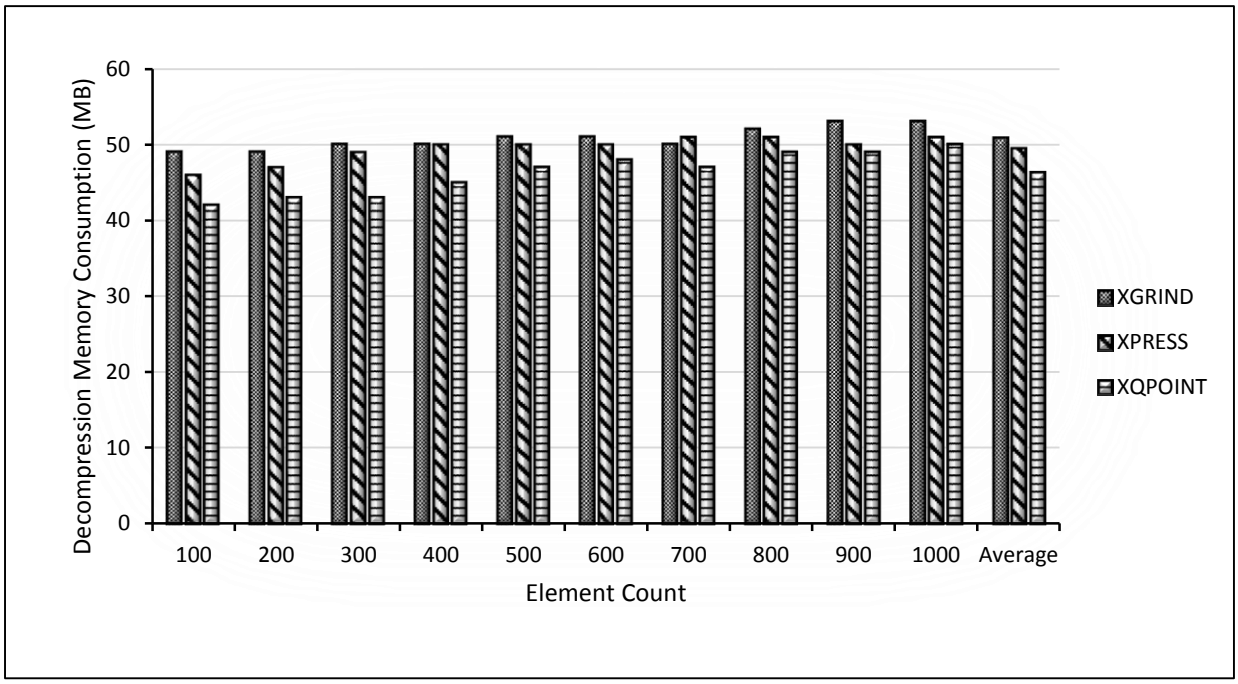

Fig 4.18 Decompression memory consumption for dataset 2

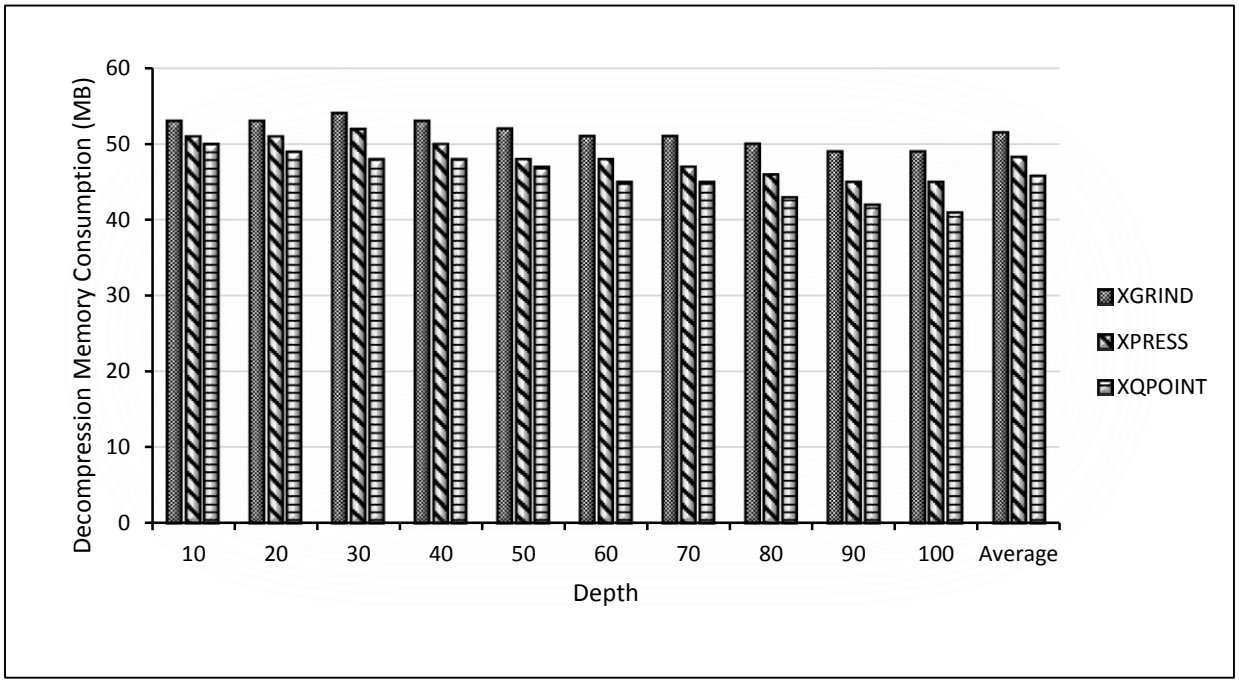

Fig 4.19 Decompression memory consumption for dataset 3

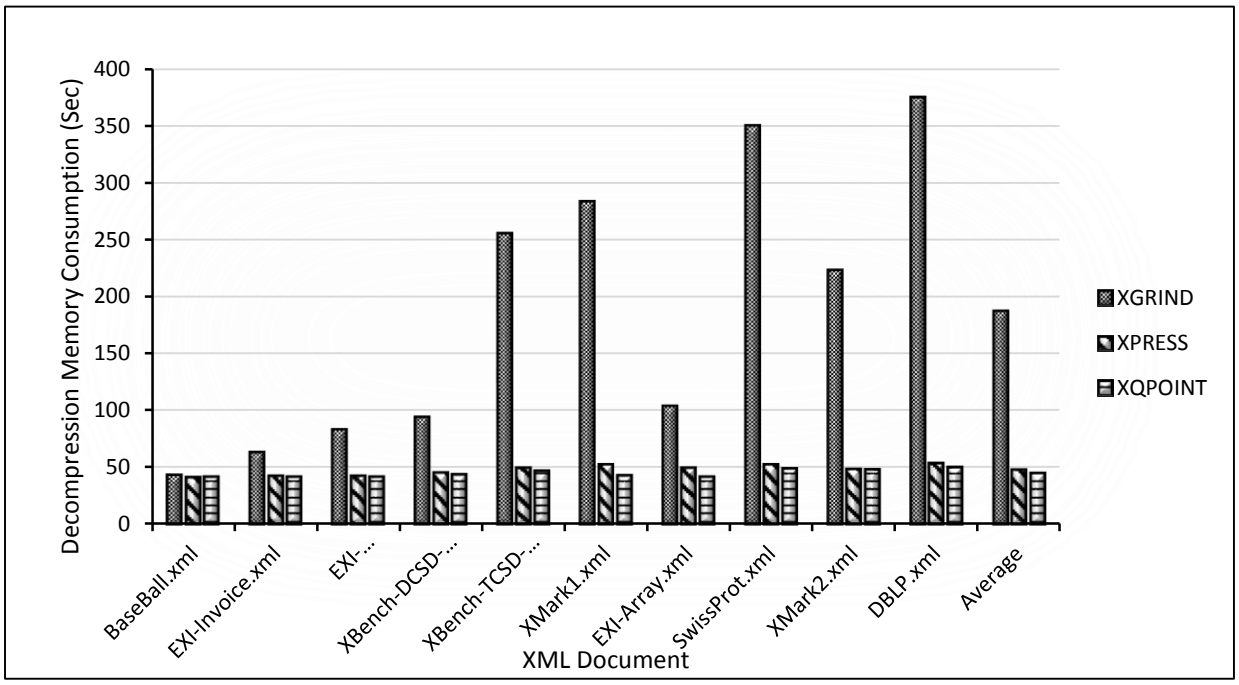

Fig 4.20 Decompression memory consumption for dataset 4

CHAPTER 5

CONCLUSION AND FUTURE WORKS

5.1 Conclusion

In this study, the three homomorphic XML compressor are tested on different metrics. Compression ratio, compression time, compression memory consumption, decompression time, and decompression memory consumption are the metrics evaluated with different perspective that determines the better XML compressor. Extensive comparison between the XML compressors are carried out. The result shows that XQPOINT has better performance among three in all metrics. It shows better performance in all the metrics that is analyzed in the study. XPRESS follows XQPoint with very small scale. Hence, we can clearly conclude that XQPOINT is better compressor than other two compressors from the outcomes.

5.2 Future Works

In this study, only three homomorphic XML compressors are studied. There are other XML compressors proposed in this category. These can be included in the analysis. Also, similar analysis can be done with homomorphic and non-homomorphic in combine. Here, query parser are not included in the study. Query support is also another metrics that can be compared. Query evaluation time can be compared for the XML compressor with different types of query combining with same perspective as in the study i.e. change in node count, element count and depth.

REFERENCES

[1] S. Sakr, "An Experimental Investigation of XML Compression Tools," CoRR, vol. abs/0806.0075, Sydney, Australia, 2008.

[2] P. Demitrov, "The History of Data Compression (Infographic)," 19 november 2014. [Online]. Available: http://techmeup.net/history-data-compression-infographic/.

[3] H. Liefke, D. Suciu, "XMill: An Efficient Compressor for XML Data," *Proc. of ACM SIGMOD,* May 2000.

[4] M. Girardot, N. Sundaresan, "Millau: An encoding format for efficient representation and exchange of XML over the Web," 2000.

[5] Christopher League, Kenjone Eng, "Schema-Based Compression of XML Data with Relax NG," *JOURNAL OF COMPUTERS,* vol. Vol 2 No 10, 2007.

[6] Pankaj M. Tolani, Jayant R. Haritsa, "XGRIND: A Query-friendly XML Compressor," in *Proccedings of 18th International Conference on Database Engineering*, 2002.

[7] Jun-Ki Min, Myung-Jae Park, Chin-Wan Chung, "XPRESS: A Queriable Compression for XML Data," 2003.

[8] Przemyslaw Skibinski, Jakub Swacha, "Combining effieient XML Compression With Query Processing," *ADBIS 2007, LNCS 4690,* pp. 330-342, 2007.

[9] D. Salomon, Data Compression: The Complete Reference, 2004.

[10] Baydaa T. Al-Hamadani,Raad F. Alwan,Joan Lu, "XQPoint: A Queriable Homomorphic XML Compressor," in *6th International Conference on Innovations in Information Technology*, 2009.

[11] Martin Isenburg, Peter Lindstrom, Jack Snoeyink, "Lossless Compression of Predicted Floating-Point Geometry," *Elsevier Science,* 2004.

[12] "Benchmark of XML Compression Tools," [Online]. Available: http://xmlcompbench.sourceforge.net/.

BIBLIOGRAPHY

[1] J. Cheney, "An Empirical Evaluation of Simple DTD-Conscious Compression Techniques," 2005.

[2] J. Glen G. Langdon, "An Introduction to Arithmetic Coding," vol. 28, pp. 135-149, March 1984.

[3] Ian H. Witten, Radford M. Neal, John G. Cleary, "Arithmetic Coding For Data Compression," vol. 30, pp. 520-540, 1987.

[4] "Introduction to Fixed Point Number Representation," 2006. [Online]. Available: http://www-inst.eecs.berkeley.edu/~cs61c/sp06/handout/fixedpt.html.

[5] Gregory Leighton, Denilson Barbosa, "Optimizing XML Compression".

[6] S. S. Nair, "XML Compression Techniques: A survey".

[7] Christopher D. Manning, Prabhakar Raghavan, Hinrich Schutze, An Introduction to Information Retrival, Cambridge: Cambridge University Press, 2009.

[8] "Extensible Markup Language (XML)," [Online]. Available: http://www.w3.org/XML/.

APPENDIX

1. Sample XML Data- Random generated data

Note: Sample data with 50 node, 10 element and depth of 5 level

```
<?xml version="1.0" encoding="Windows-1252"?>
<root>
 <TRKC>4.54223087479314E+307</TRKC>
 <GHRO QFXA="QPKPRVBIEATZPUBAVRMEOSU">1.06049455815009E+308</GHRO>
 <DE B="MNBYJM">1349319886</DE>
 <N>1640223644</N>
 <QGV>
  <DE>
   <RDCZ>
    <GHRO>
     <C>ZRLAHBOFXRKOXEHODUCC</C>
    </GHRO>
   </RDCZ>
  </DE>
  <RDCZ>UFHYCFAMSSQSMAMPDOEZISYQGWSCVREM</RDCZ>
  <N>139112099</N>
  <TRKC>1.23858955179076E+308</TRKC>
  <QGV>
   <FYQK>
    <DE MZ="JIICHRSFNYQBOEJRUKVNKMJKVTCSVOUIRMD">1045166096</DE>
    <FYQK>1941553840</FYQK>
    <C HVQ="EJJXRLHWZYGNPZKZDHJC">EQQSQLIQCADRNLFWRG</C>
    <FYQK
O="HNXOEOZMOJVETHZUWOIVFJXMZTGQCYUIQSIESXDCDOOIS">1019607202</FYQK>
    <QGV>5.43470315178352E+307</QGV>
    <TRKC>
     <TRKC>
      <N>625639220</N>
      <N PMO="YU">2043899039</N>
      <N RUO="SHRXRQCQXALNHISUTEYTGRLRXJJGLAACCRWDQM">15822504</N>
     </TRKC>
    </TRKC>
   </FYQK>
   <GHRO>
    <C>QTLZMWUNBVZAHHVFXAILPMOEHMYGKBLEPOSNABNG</C>
    <RDCZ IQLK="RPTZEWEDEHRAXOCZJVKSOHFVMCDLRGNVBFOUH">T</RDCZ>
    <N>1581709457</N>
    <N>1572499370</N>
    <DE>
     <B EKK="NBSTMYTWZSZHSXVQJZYMMZUKKWISUJADKKZYAQCKANNP">
      <TRKC Z="IFMYLNHIIQQABPSVXHYW">
       <TRKC>8.27114246639615E+307</TRKC>
      </TRKC>
     </B>
    </DE>
    <C>RTHTJAURLZLQSILUZLYXXVVCCUQIZMMBC</C>
    <QGV>
     <RDCZ PXE="INFXKLSMMNHUWXFQPKLRYMJ">
      <DE>
       <GHRO>
        <N>773000713</N>
       </GHRO>
      </DE>
     </RDCZ>
```

```
        <GHRO>
         <B HAZO="WTZCRCELNNVDRVRWGTTEHBNJQTTEOKYMVSFYOJBQE">
          <FYQK>
           <N O="AHG">1154730687</N>
           <B MXJ="XCPIIZPTHWRCATWQRFOSQMJRFLEKRZXDHEPCHJCFKVTEPEJUY">
            <C RJ="SXCXVFBIZNUVETJHDIHVS">BBK</C>
            <GHRO
R="PVEJJGFKFTLFBJOPEGQDBKNSNWBVKSZHHTFZZK">6.69252046093975E+306</GHRO>
            <C>HZZR</C>
            <GHRO>1.49423855790974E+308</GHRO>
            <N>438711425</N>
           </B>
          </FYQK>
         </B>
         <RDCZ>
          <GHRO AN="EXHEOQUFPFDJLLNWJHQKEWTAZRMNOVOXNICCRN" />
         </RDCZ>
        </GHRO>
       </QGV>
      </GHRO>
     </QGV>
    </QGV>
</root>
```

2. Sample data : from dataset 4

Note : it is part of file Baseball.xml

```
<SEASON>

 <YEAR>1998</YEAR>

 <LEAGUE>

  <LEAGUE_NAME>National</LEAGUE_NAME>

  <DIVISION>

    <DIVISION_NAME>East</DIVISION_NAME>

<TEAM><TEAM_CITY>Atlanta</TEAM_CITY><TEAM_NAME>Braves</TEAM_NAME>

<PLAYER><NUMBER>1274</NUMBER><SURNAME>Malloy</SURNAME><GIVEN_NAME>Marty
</GIVEN_NAME><POSITION>Second
Base</POSITION><GAMES>11</GAMES><GAMES_STARTED>8</GAMES_STARTED><AT_BATS>28
</AT_BATS><RUNS>3</RUNS><HITS>5</HITS><DOUBLES>1</DOUBLES><TRIPLES>0</TRIPLES>
<HOME_RUNS>1</HOME_RUNS><RBI>1</RBI><STEALS>0</STEALS><CAUGHT_STEALING>0</C
AUGHT_STEALING><SACRIFICE_HITS>0</SACRIFICE_HITS><SACRIFICE_FLIES>0</SACRIFICE_F
LIES><ERRORS>0</ERRORS><PB>0</PB><WALKS>2</WALKS><STRUCK_OUT>2</STRUCK_OUT
><HIT_BY_PITCH>0</HIT_BY_PITCH></PLAYER>

<PLAYER><NUMBER>2359</NUMBER><SURNAME>Lockhart</SURNAME><GIVEN_NAME>Keith
</GIVEN_NAME><POSITION>Second
Base</POSITION><GAMES>109</GAMES><GAMES_STARTED>89</GAMES_STARTED><AT_BATS>
366</AT_BATS><RUNS>50</RUNS><HITS>94</HITS><DOUBLES>21</DOUBLES><TRIPLES>0</TRI
PLES><HOME_RUNS>9</HOME_RUNS><RBI>37</RBI><STEALS>2</STEALS><CAUGHT_STEALIN
G>2</CAUGHT_STEALING><SACRIFICE_HITS>2</SACRIFICE_HITS><SACRIFICE_FLIES>3</SACR
IFICE_FLIES><ERRORS>6</ERRORS><PB>0</PB><WALKS>29</WALKS><STRUCK_OUT>37</STR
UCK_OUT><HIT_BY_PITCH>1</HIT_BY_PITCH></PLAYER>
```

<PLAYER><NUMBER>2844</NUMBER><SURNAME> Springer</SURNAME><GIVEN_NAME>Russ
</GIVEN_NAME><THROWS>Right</THROWS><POSITION>Relief
Pitcher</POSITION><WINS>5</WINS><LOSSES>4</LOSSES><SAVES>0</SAVES><GAMES>48</GA
MES><GAMES_STARTED>0</GAMES_STARTED><COMPLETE_GAMES>0</COMPLETE_GAMES><
SHUT_OUTS>0</SHUT_OUTS><ERA>4.1</ERA><INNINGS>52.2</INNINGS><HOME_RUNS>51</HO
ME_RUNS><RUNS>4</RUNS><EARNED_RUNS>26</EARNED_RUNS><HIT_BATTER>24</HIT_BAT
TER><WILD_PITCHES>1</WILD_PITCHES><BALK>5</BALK><WALKED_BATTER>0</WALKED_B
ATTER><STRUCK_OUT_BATTER>30</STRUCK_OUT_BATTER></PLAYER>

<PLAYER><NUMBER>2898</NUMBER><SURNAME>Guillen</SURNAME><GIVEN_NAME>Ozzie
</GIVEN_NAME><POSITION>Shortstop</POSITION><GAMES>83</GAMES><GAMES_STARTED>59
</GAMES_STARTED><AT_BATS>264</AT_BATS><RUNS>35</RUNS><HITS>73</HITS><DOUBLES
>15</DOUBLES><TRIPLES>1</TRIPLES><HOME_RUNS>1</HOME_RUNS><RBI>22</RBI><STEALS
>1</STEALS><CAUGHT_STEALING>4</CAUGHT_STEALING><SACRIFICE_HITS>4</SACRIFICE_H
ITS><SACRIFICE_FLIES>2</SACRIFICE_FLIES><ERRORS>6</ERRORS><PB>0</PB><WALKS>24</
WALKS><STRUCK_OUT>25</STRUCK_OUT><HIT_BY_PITCH>1</HIT_BY_PITCH></PLAYER>

<PLAYER><NUMBER>2954</NUMBER><SURNAME>Bautista</SURNAME><GIVEN_NAME>Danny
</GIVEN_NAME><POSITION>Outfield</POSITION><GAMES>82</GAMES><GAMES_STARTED>27</
GAMES_STARTED><AT_BATS>144</AT_BATS><RUNS>17</RUNS><HITS>36</HITS><DOUBLES>1
1</DOUBLES><TRIPLES>0</TRIPLES><HOME_RUNS>3</HOME_RUNS><RBI>17</RBI><STEALS>1
</STEALS><CAUGHT_STEALING>0</CAUGHT_STEALING><SACRIFICE_HITS>3</SACRIFICE_HIT
S><SACRIFICE_FLIES>2</SACRIFICE_FLIES><ERRORS>2</ERRORS><PB>0</PB><WALKS>7</WA
LKS><STRUCK_OUT>21</STRUCK_OUT><HIT_BY_PITCH>0</HIT_BY_PITCH></PLAYER>

<PLAYER><NUMBER>2989</NUMBER><SURNAME> Martinez</SURNAME><GIVEN_NAME>Dennis
</GIVEN_NAME><THROWS></THROWS><POSITION>Relief
Pitcher</POSITION><WINS>4</WINS><LOSSES>6</LOSSES><SAVES>2</SAVES><GAMES>53</GA
MES><GAMES_STARTED>5</GAMES_STARTED><COMPLETE_GAMES>1</COMPLETE_GAMES><
SHUT_OUTS>1</SHUT_OUTS><ERA>4.45</ERA><INNINGS>91</INNINGS><HOME_RUNS>109</HO
ME_RUNS><RUNS>8</RUNS><EARNED_RUNS>53</EARNED_RUNS><HIT_BATTER>45</HIT_BAT
TER><WILD_PITCHES>3</WILD_PITCHES><BALK>2</BALK><WALKED_BATTER>0</WALKED_B
ATTER><STRUCK_OUT_BATTER>19</STRUCK_OUT_BATTER></PLAYER>

<PLAYER><NUMBER>3143</NUMBER><SURNAME>Williams</SURNAME><GIVEN_NAME>Gerald
</GIVEN_NAME><POSITION>Outfield</POSITION><GAMES>129</GAMES><GAMES_STARTED>51
</GAMES_STARTED><AT_BATS>266</AT_BATS><RUNS>46</RUNS><HITS>81</HITS><DOUBLES
>18</DOUBLES><TRIPLES>3</TRIPLES><HOME_RUNS>10</HOME_RUNS><RBI>44</RBI><STEAL
S>11</STEALS><CAUGHT_STEALING>5</CAUGHT_STEALING><SACRIFICE_HITS>2</SACRIFICE
_HITS><SACRIFICE_FLIES>1</SACRIFICE_FLIES><ERRORS>5</ERRORS><PB>0</PB><WALKS>17
</WALKS><STRUCK_OUT>48</STRUCK_OUT><HIT_BY_PITCH>3</HIT_BY_PITCH></PLAYER>