XML Compression Techniques: A Survey

Smitha S. Nair

Department of Computer Science, University of Iowa

Iowa City, Iowa, USA

Abstract

Over the past few years, XML has become the de facto standard for communicating between heterogeneous systems. XML is used to standardize information exchange in various fields ranging from business reporting (XBRL) to sports (SportsML). The power of XML lies in its self-describing abilities. This same ability makes XML verbose thus introducing significant amount of redundancy that adds no particular value. In addition, the increased size affects both query processing and data exchange. XML files require a lot more storage space and network bandwidth. Therefore, it is particularly important to explore effective compression techniques that will help reduce the usage of these resources yet not compromise on speed and flexibility. This paper presents an analysis of the various XML compression techniques available and how their relative strengths and weaknesses influence their effectiveness.

XML: An Overview

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

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

[Fig 1 – XML example]

Need for Compression

As evident from the snippet in Fig 1, XML representations are very large and can be up to ten times as large as equivalent binary representations [1]. Consider the following:

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

Introduction to the compression techniques

There are a number of XML compression techniques available today which were developed, and tested over the last few years. This section enumerates a few of the most promising techniques namely gZip, XMill, XGrind, Xpress, and XComp. The evaluation of these compressors considers three main aspects: compression speed, compression ratio, and flexibility of the resulting compressed document.

GZip

gZip is the most widely used commercial compressor available. Since XML is represented as a text file, an obvious choice for a compression tool would be a general-purpose compression tool such as gZip. gZip was developed by Jean-loup Gailly and Mark Adler and uses a combination of the LZ77 algorithm and Huffman coding.

XMill

XMill was developed by Hartmut Liefke and Dan Suciu in the summer of 1999. XMill was designed at AT&T Labs Research in New Jersey, USA. At present, this tool is not being developed any further by the original developers [2]. The source code has now moved into a SouceForge project. XMill is based on a regrouping strategy that takes advantage of the XML elements. XMill groups XML text strings with respect to their meaning and exploits similarities between those text strings for compression [3]. Hence, XMill typically achieves much better compression rates than conventional compressors such as gZip.

XGrind

XGrind is a compression tool for XML documents developed by Pankaj M. Tolani, and Jayant R. Haritsa of the Indian Institute of Science in July 2002. XGrind supports querying the compressed document. At the same time, this tool retains the structure of the original XML document too. This facilitates reuse of standard XML techniques for processing the compressed document [4]. These features make it very important in applications hosted on resource-limited computing devices like Palm Tops.

Xpress

Jun-Ki Min, Myung-Jae Park and Chin-Wan Chung of Korea Advanced Institute of Electrical Engineering & Computer Science (KAIST) developed XPRESS and the paper was presented at SIGMOD 2003. XPRESS is a compressor that supports direct and efficient evaluations of queries on compressed XML data. XPRESS adopts a novel encoding method, called reverse arithmetic encoding. XPRESS achieves significant improvements on query performance for compressed XML data and reasonable compression ratios [5].

XComp

An alternative XML-specific compression tool presented in this paper is XComp that was developed as a thesis paper by Weimin Li in the University of Waterloo. Similar to XMill, XComp takes advantage of the XML's self-describing feature to separate structure from data based on the semantics.

Features and Principles

gZip

This general-purpose compression utility is relatively very popular and is used in many commercial implementations. gZip has its own pros and cons. The benefits of using such a tool would be [1]:

- ❑ This tool is widely available in both open-source and commercial implementations
- ❑ gZip provides better compression rate (40-50%) and freedom from patented algorithms
- ❑ Using gZip requires no knowledge of the document-structure.
- ❑ gZip is built into http and web-servers as a standard feature.

However, the main disadvantage with using gZip to compress XML files is:

- ❑ Compression of elements/attributes may be limited due to the long-range dependencies between elements and between attributes. (Duplication is not necessarily local)

This means that generic compression algorithms are limited because they do not leverage document semantics.

XMill

XMill is a user-configurable XML compressor. The lossless compressor and decompressor are named XMill and Xdemill respectively. XMill achieves about twice the compression rate of general-purpose compressors like *gzip* at about the same speed. It does not need a document type definition (DTD) for compression and preserves the input XML file. It also allows the users to combine existing compressors in order to compress heterogeneous XML data. It is extensible with user-defined compressors for complex data types such as DNA sequences or images. XMill claims to reduce network bandwidth considerably.

Principles and architecture

There are three main principles in XMill. They are as follows:

Separating structure from data (content)

The structure is the XML tags and attributes while data is the sequence of items (strings) representing element text content and attribute values. Both structure and data are compressed separately. Start tags are assigned an integer value and end tags are replaced by the token '/'. Tags and attributes are compressed using a dictionary-encoding method.

Grouping Data values with related meaning

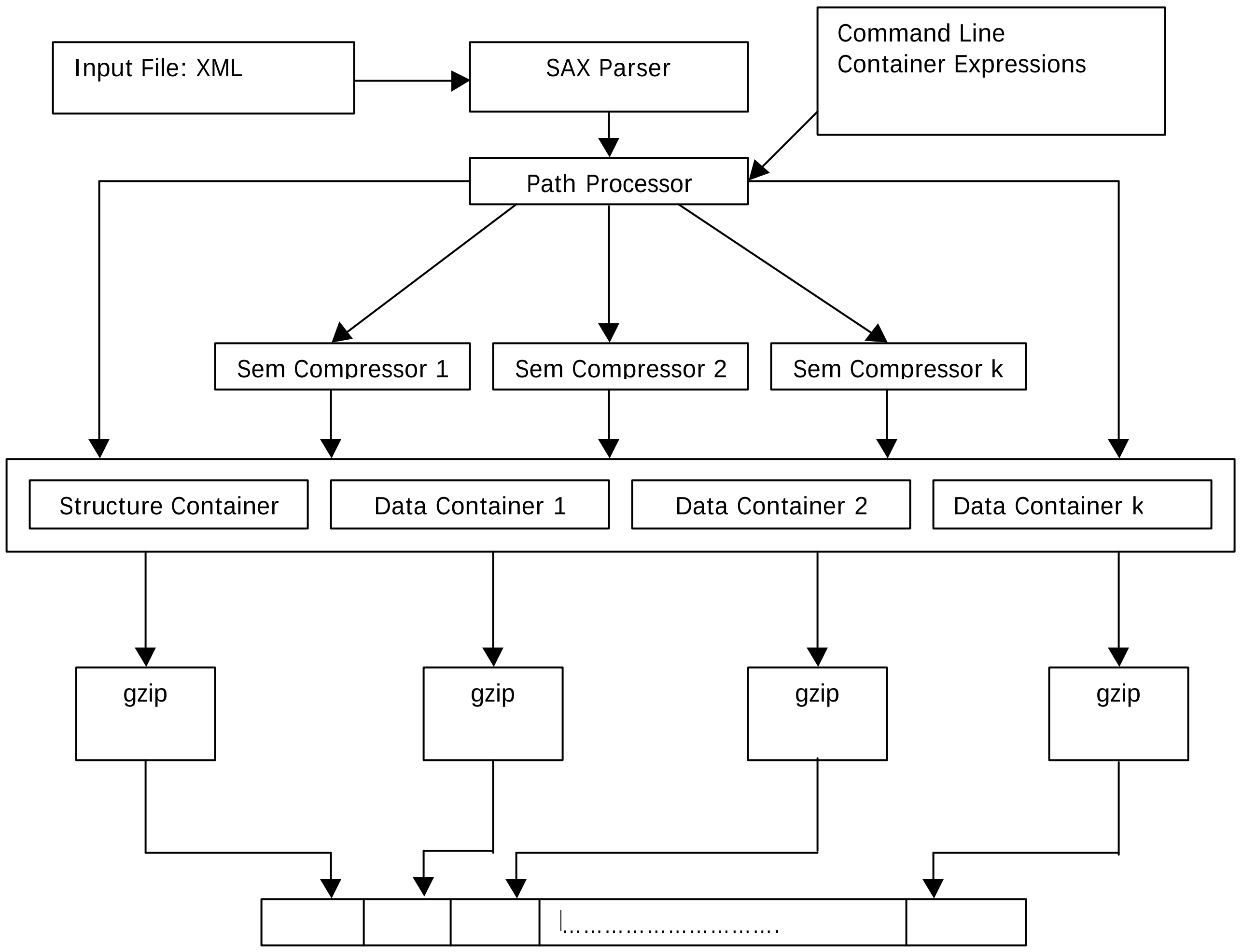
Each data value is uniquely assigned to one data container. The mapping is determined by the data's value path and the user-specified container expressions. While grouping through expression, values matching the same expression are grouped into one container.

Apply semantic compressors

XMill applies specialized compressors to different containers. There are three types of compressors available:

- Atomic compressors for the basic data types
- Combined compressors, and
- User-defined compressors

Block Diagram: Architecture



[Fig 2 – Output file: Compressed XML]

The architecture of XMill is based on the three principles discussed below [3]. A SAX8 parser parses the XML file. The parser then sends tokens to the core module of the XMill called the path processor. Every XML token is assigned to a container. Tags and attributes, which form the XML structure, will be sent to the structure container, while the data values will be sent to various data containers, according to the container expressions, and finally the containers are compressed independently. The path processor determines how to map data values to containers. The user can control this mapping by providing a series of container expressions on the command line. For each XML data value the path processor checks its path against each container expression, and determines either that the value has to be stored in an existing container, or creates a new container for that value. Containers are kept in a main memory window of fixed size (the default is 8MB). When the window is filled, all containers are gzipped, stored on disk, and the compression resumes. In effect, this splits the input file into independently compressed blocks. After loading and unzipping the containers, the decompressor (Xdemill) parses the structure container, invokes the corresponding semantic decompressor for the data items and generates the output. Optionally, XMill can preserve white spaces: in that case, it stores them in container 1. Container 0 holds the structure while container 2 holds the PIs, DTDs, and comments. The size of the compressed file typically increases only slightly when white spaces are preserved: around 4%. We observed a higher increase (30%) only for Treebank, a linguistic database (Sec. 6), because of its deeply nested structure.

Example

The first step is separating structure from content. Revisiting our example which was

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

Is converted to:

```
Student = T1, @SID = T2, LastName = T3, FirstName = T4, MInitial = T5, Address = T6
It will be changed to T1 T2 C2 T3 C3 / T4 C4 / T5 C5 / T6 C6 //
```

But in practice, all these tokens will be encoded as integers each with 1, 2, or 4 bytes. The above structure will need 16 bytes. White spaces are ignored in this case. If we choose to preserve the white spaces, it stores them in container 1. Once this is done, the matching data values are stored in separate containers; i.e., all Last Name data values will be stored in one container and similarly, all Address data values will be in a separate container. Each container will be compressed separately using specialized compressors. Users can also associate semantic compressors with containers. These containers are stored in a main memory window of size 8KB. Once this is filled, the containers will be compressed using gZip.

Experimental Evaluation

Three classes of experiments were performed – comparison of the compression ratio of XMill against *gzip* under various settings, comparison of compression/decompression times of XMill and *gzip*, and measuring the total effect of XMill in an XML data exchange application over the network.

The results were shown as follows in another discussion that is listed in [8].

For data sets that had more data and less text, XMill compressed under the default setting to 45%-60% the size of gzip. Using semantic compressors, XMill reduced the size to 35%-47% of *gzip*'s. For the more text-like data sets, XMill performs only slightly better than gzip. Xdemill's speed is comparable to gunzip. Here the price is paid to merge data from different containers. For queries that produce a lot of XML data (more than a megabyte), XMill compressed XML messages are smaller than the Zip compressed Binary messages. While this improves throughput of a networked environment with a fixed bandwidth, XMill compressed messages are at times twice slower than Zip compressed Binary messages. The processor speed has a significant impact on the observed response times.

Benefits

XMill achieves better compression rate compared to *gzip* (by a factor of 2, for data-like XML documents) without sacrificing speed. This owes to the fact than it separates structure from content. This makes it a clear winner for applications like data archiving since these applications require lesser disk space. At the same time, it reduces network bandwidth. XMill is moderately faster than gzip in XML publishing. However, relative advantage of XMill depends on the application it is used.

Disadvantages

The main disadvantages of XMill are as follows:

- Compressed output of XMill is not queryable. To be queried, the document has to be decompressed.
- If the size of the input document is less than 20KB, XMill will not exhibit any significant advantage over gzip.

- Here the compression is targeted for applications like data exchanging, data archiving etc, but not for deriving a meaningful view of the input document as is the case of compressing images or video sequences [7].
- To apply specialized compressors to containers, human intervention is required to specify the required container. Path processor is configured by user commands to map values. This is inconvenient.
- XMill precludes incremental processing of compressed documents; it actually hinders compressors other than gzip, and requires user assistance to achieve the best compression [6].

If the network bandwidth is scarce then it is useful to employ a compression scheme. If the message size is considerably large (1MB), XMill should be used.

XGrind

XGrind is a queryable XML compressor, i.e., we can execute queries on XML documents compressed using XGrind. The major reason for this is that the compressed document retains the structure of the original XML document. This is very important for resource limited computing devices like palmtops. Even though resources are available, querying on compressed documents reduces query response time. Here the disk seek times are highly reduced and at the same time disk bandwidth is increased. XGrind does compression by separating data from structure, but at the same time maintains the document structure of the input document.

Techniques

The two main techniques involved in this compression are:

- Meta-Data Compression: This is similar to that of XMill.
- Enum-type attributed value compression: This is captured in the DTD itself. This is accomplished by XGrind by examining the DTD of the document and encoding values using a simple encoding scheme to represent an enumerated domain of values.

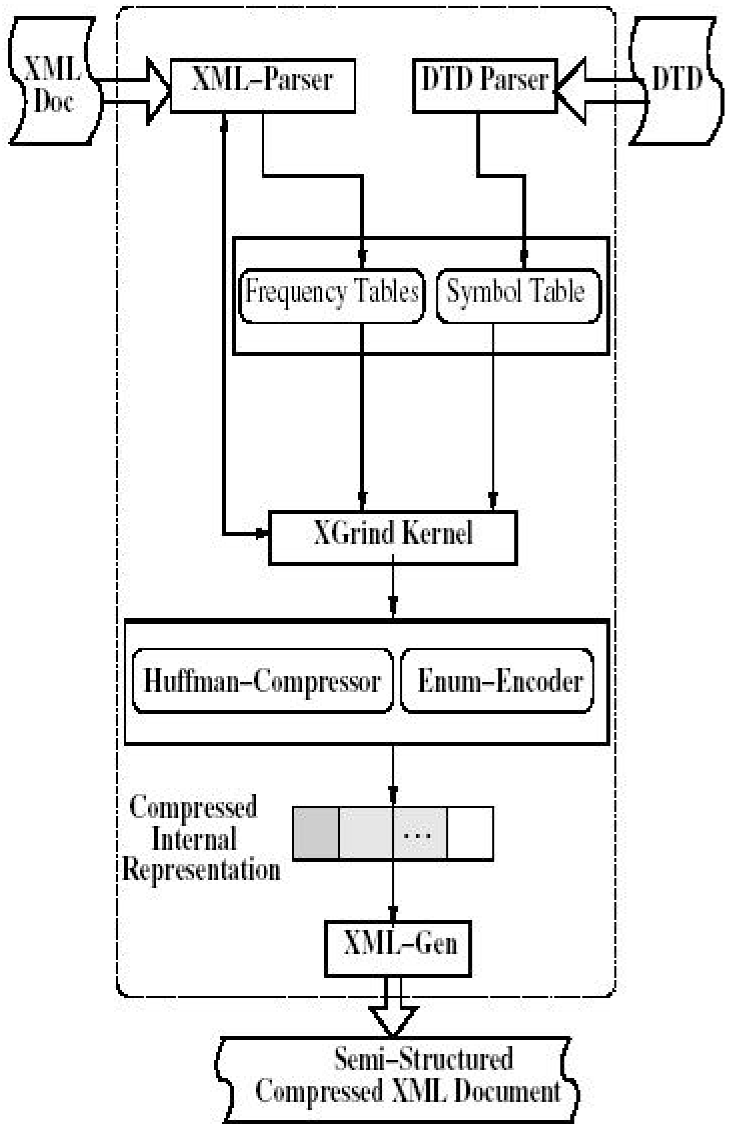
Operation Flow

A DTD parser is employed to produce the required tables for the enum data types. The DTD of the input document is used to parse the input file. An XML parser is used to tabulate the frequency table of the occurrences of each element in the XML document. It also constructs the tokens from the given document that are the elements, attributes and the tags. The XGrind kernel checks the token type while the XML generator combines tables and the output to give the final compressed XML document. This exhibits homomorphic compression.
A path expression is evaluated by scanning the compressed file and whenever a new tag is found, the two path expressions are compared and decided. To evaluate range queries, partial decompression of data values is always required.

The main advantages of employing homomorphic compression can be listed as follows [4]:

- Efficient parsing techniques
- Ability to construct indexes on the compressed document itself
- Directly executable updates on the compressed document
- Document validity can be checked against the compressed DTD

Block Diagram



[Fig 3 – Architecture of XGrind Compressor]

Example

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

```
<!- DTD for the Student database -->
<!ELEMENT STUDENT (LastName, FirstName, MInitial, Address)>
<!ATTLIST STUDENT SID CDATA #REQUIRED>
<!ELEMENT LastName (#PCDATA)>
<!ELEMENT FirstName (#PCDATA)>
<!ELEMENT MInitial (#PCDATA)>
<!ELEMENT Address (#PCDATA)>
```

will be converted as follows:

```
T0 A0 nahuff (S1234)
T1 nahuff (Smith)/
T2 nahuff (Joe)/
T3 nahuff (M)/
T4 nahuff (620 12th Ave, Coralville)/
/
```

Abstract View of the document

Performance Results

XGRIND has a lower compression ratio than XMILL. Results indicate that the compression ratio for XGrind improves with the increase in the number of enumerated attributes.

Benefits

- Considerable improvements in query response time
- Disk bandwidth is effectively increased as increased information density
- Memory hit buffer ratio increases

- Compresses at the granularity of element/attribute value using context-free compression scheme
- Range and Partial match queries have on the fly decompression of only those elements that feature in the query predicates

Disadvantages

XGrind does not support several operations like non-equality selections. In addition, it cannot perform any join, aggregation, and nested queries or construct operations.
Compression of documents is a one-time operation; at the same time querying, could be a repeated occurrence.
The statistics could change if there are many updates made to the compressed XML document.
Lastly, XGrind uses a fixed root-to-leaf navigation strategy, which is insufficient to provide alternative evaluation strategies.

XGrind vs. Xmill

- Compression Time: Factor of 2 of that of XMill
- Uses element/attribute granularity than document granularity
- Simple character-based Huffman coding scheme rather than a pattern based approach
- Makes 2 passes over the original document to provide context-free compression

Xpress

Xpress also allows queries on compressed data. XPress works only on XML trees. It cannot handle ID/IDREF tags. It creates bisimilar partitions of elements in the XML document. Then, it encodes partitions by disjoint intervals and allows query evaluation by operations on these intervals. It uses an encoding method known as the reverse arithmetic encoding. It is a combination of differential and binary encoding methods. This is an efficient path encoding method, which yields fewer overheads of partial decompression and quicker path evaluation. XPRESS provides high compression ratio.

Characteristics of XPRESS

The following are some of the primary characteristics of XPRESS.

Reverse Arithmetic Encoding

Unlike existing compressors that represent each tag by a unique identifier (making it inefficient to handle path expressions), XPRESS adopts reverse arithmetic encoding that encodes a label path as a distinct interval in [0.0,1.0]

Automatic Type Inference

A Type Inference Engine is devised here thus avoiding any human intervention in determining the types of data values.

Apply Diverse Encoding Methods to Different Types

Depending on the inferred data type received from the type inference engine, we can apply proper encoding methods to these data values. Hence, we can achieve a high compression ratio and minimal partial decompression overhead as mentioned above.

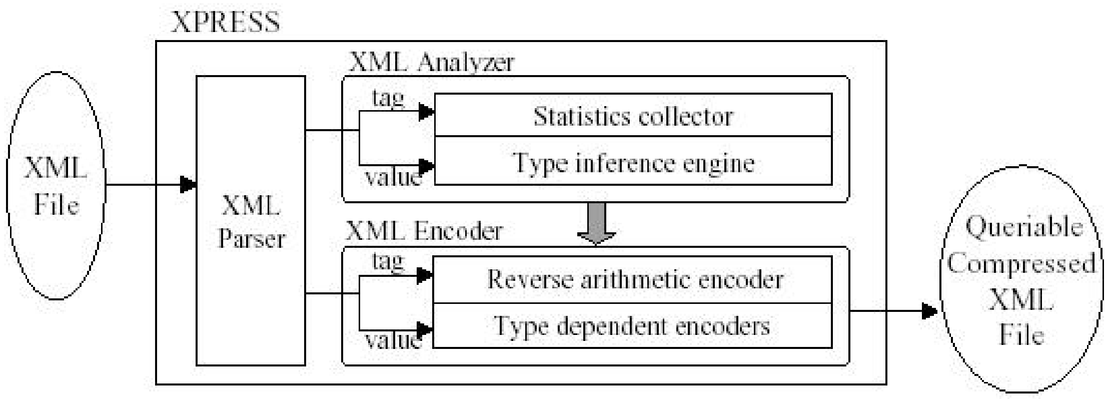
Semi-adaptive Approach

XPRESS employs a semi-adaptive approach. Unlike XGrind, the statistics is not changed during compression. At the same time, encoding rules are independent to the location.

Homomorphic Compression

This preserves the structure of the input XML document. Thus, XML segmentation that satisfies given query conditions is efficiently extracted.

Architecture



[Fig 4 – Architecture of XPress Compressor]

The core modules in the architecture of XPress are an XML analyzer and an XML encoder. The compression method used here is the semi-adaptive compression. The XML analyzer parses each token in the input file but at the same time keeps track of the path trace. If it encounters a tag, the statistics are collected. On the other hand, if it is a data value, the appropriate type inference is applied to the value. XML analyzer in turn consists of two modules that accomplish these functions – the statistics collector that calculates the adjusted frequency of each distinct element and the type inference engine that infers the type of data values of each element inductively and produces the required statistics. The type inference engine keeps track of the inferred type of the data value and related information. These types are susceptible to changes from one type to another amidst the process. The XML encoder on the other hand calculates the interval and chooses a proper encoding method.

Query Processing in XPRESS

A long label path expression is first partitioned into a short one. This is then transformed into a sequence of intervals. Generally, the length of sequence is one since a label path expression is usually short. With the help of these 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.

The range query for numeric typed element is encoded by the data value encoder for the element. Then without the decompression of encoded values, the query is evaluated. For text typed element, partial decompression for range query is required [5].

Example

Suppose that the frequencies of elements Student, LastName, FirstName, MInitial and Address were {0.1,0.1, 0.1, 0.3, 0.3}, respectively. Then, based on the cumulative frequency, the entire interval [0.0, 1.0) is partitioned as follows:

Element	Frequency	Cumulative Frequency	Interval T
Student	0.1	0.1	[0.0, 0.1)
LastName	0.1	0.2	[0.1, 0.2)
FirstName	0.1	0.3	[0.2, 0.3)
Minitial	0.3	0.6	[0.3, 0.6)
Address	0.4	1.0	[0.6, 1.0)

Features

Xpress query time is 2.83 times better than that of XGrind [7].

Xpress compression results are 80% better than XMill [11] and 3.14 times zip [12].

XML-Xpress was also faster, running 3% faster than Zip and 55% faster than XMill [12].

XComp

As mentioned above, the principle behind XComp is similar to Xmill. The structure is encoded as a sequence of integers, while the data grouping is based on XML tags/attributes and their levels in the document tree [10]. Also shown in the paper is the performance evaluation of XComp. It is shown that XComp outruns both XMill and XGrind in terms of compression.

The two main principles employed in XComp are (1) Separating document structure from data, and (2) grouping content data based on semantics. Three basic components of XML structure, namely tags, attributes, and data items play major role in the development of XComp. This feature is used mainly in order to use specific methods to process structure so that the compression ratio could be improved significantly. Related data will be compressed together after reorganization.

Compression Principles

The various elements of the input XML document are treated separately in XComp compression. Certain strings like “xml-stylesheet href= “mystyle.css” type= “text/css” ”, are not parsed, instead it will be stored as a whole data item. Similarly, special markups are treated in a different way.

Separating structure from data

In XComp, the document structure is extracted from the data, and a sequence of integers is used to record the markup and data items’ positions. The data are then reorganized to compress related data together. Each distinct tag or attribute is assigned a positive sequence id. The table used is:

Code	**Meaning**
0	End tag
1	Data item position
2	‘=’ position
3	‘>’ position
4	White space position

5	Position of any characters before XML declaration
6	PI
7	DTD
8	Comment
9	CDATA Section

Consider the input XML below:

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

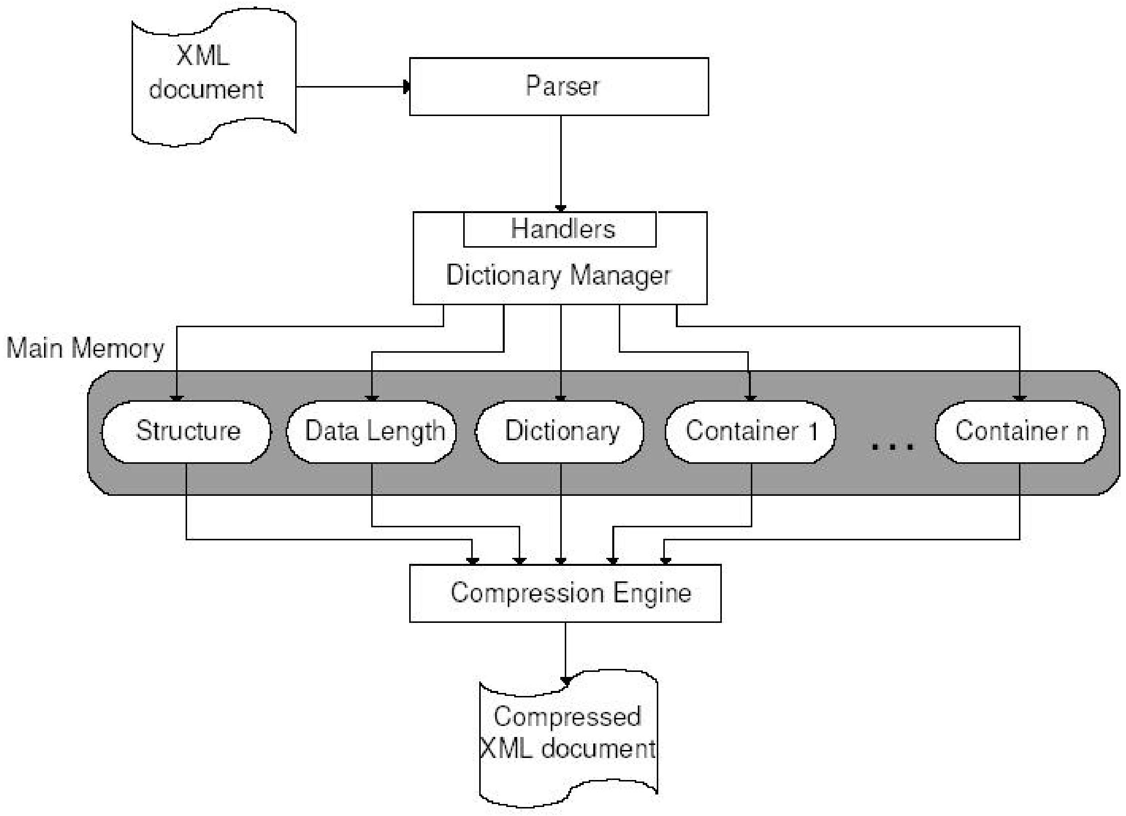
Using the above table, an input document is encoded as follows:

```
10,4,11,4,2,3,4,1,3,4,12,3,1,0,13,3,1,0,14,3,1,0,15,3,1,0,0.
```

Grouping Data

Grouping data exploits the self-descriptiveness of XML documents. The data items with the same tag or attribute name usually have a semantic relationship between them. They are put together under the assumption that they can be compressed better if put together.

System Architecture



[Fig 5 – Architecture of XComp Compressor]

The four major modules implemented in XComp are the XML parser, the dictionary manager, the containers and the compression engine. The parser employed here is a SAX parser. A SAX parser, unlike a DOM parser does not load the whole document in the main memory in order to process it. SAX is an event-based API. SAX sends an event for each element that it encounters in the document. The dictionary manager is responsible for encoding structure and distributing data items. This module contains the event handler routines and the management of the dictionary table. A handler algorithm is devised for event handling. The container is an individual memory space that contains related data. The containers are divided into four different kinds depending on the type of values stored in each. They are:

1. Structure Container: Stores the structure sequence
2. Data Length Container: Stores the data length sequence
3. Dictionary Container: Stores the tag and attribute name values

4. Data Containers: Stores the related data items based on the tag/attribute name and its type and level

Each container's size can be increased when it is full. However, the restriction is that it cannot exceed the size of the memory window.

The final module is a compression engine that can be a zlib compression engine or a Huffman code compression engine. Each container has its own alphabet frequency table since they are compressed individually.

Experimental Evaluation

Various XML compression tools were tested using four different XML documents, namely, dblp.xml, auction.xml, and nwind.xml.

Statistics can be tabulated as follows:

	Original Size (in bytes)	XMill (in bytes)	ICT Xpress (in bytes)	Zip (in bytes)
dblp.xml	140,428,497	21,479,242	24,399,612	26,110,276
auction.xml	116,524,435	33,624,883	16,184,462	38,525,120
nwind.xml	728,558	43,956	48,604	70,209

Although the source code for XGrind was available, it was not possible to build the sources due to missing dependencies. Hence, the table below is a reproduction of the author's own experimental analysis. Since the source code for XComp was not available, the statistics reflect the experimental evaluation performed by the creator of XComp.

Compression Time (in seconds)

	XMill	ICT Xpress	Zip
dblp.xml	32.216	46.09	29.152
auction.xml	40.909	47.23	42.140
nwind.xml	0.2	0.19	1.91

Compression time for XGrind

Document	CT (For XGrind)	CT (For XMill)	CTF
Xmark	1246	878	1.41
Conferences	442	222	1.99
Journals	344	170	2.02
Shakespeare	183	125	1.46
Ham-radio	353	182	1.93
Student1	978	471	2.07
Student4	1328	647	2.05

In the above figure CRF, CT and CTF refer to the Compression Ratio Factor, Compression Time and Compression Time Factor respectively.

The following table is the comparison of Compression Ratio of XMill and XGrind.

Document	CR (For XGrind)	CR (For Xmill)
Xmark	55.03	70.95
Conferences	57.44	84.61
Journals	57.85	85.59
Shakespeare	54.96	74.12
Ham-radio	76.85	93.54
Student1	77.13	91.74
Student4	82.12	93.87

The author has shown an analysis of transmission time for the XML documents over the Internet, which is shown below.

	XComp(Zlib)	XComp(Huffman)	XMill	XGrind	Zlib	No Compressoin
titles	1.158	2.025	1.209	1.987	1.777	3.488
DC10	6.629	8.113	6.590	7.943	7.235	25.595
DC100	68.897	80.145	69.594	77.212	81.093	253.712
TC10	13.097	15.970	13.309	16.180	14.160	25.780
TC100	135.415	162.097	138.129	252.436	140.128	252.436

Compression Time for XComp

	XComp(Zlib)	XComp(Huffman)	XMill	XGrind	Zlib
provinces	0.011	0.005	0.011	0.010	0.010
emails	0.011	0.011	0.011	0.011	0.011
countriesDC	0.011	0.010	0.011	0.011	0.013
lnames	0.071	0.039	0.073	0.058	0.055
words	0.116	0.068	0.115	0.077	0.093
titles	0.532	0.229	0.115	0.326	0.526
DC_10	2.361	1.310	2.389	1.264	2.100
DC_100	23.557	13.000	23.479	13.069	23.299
DC_1000	259.081	150.073	260.195	140.321	252.161
TC_10	4.042	2.016	4.039	2.035	3.795
TC_100	39.086	19.560	39.771	19.085	37.043
TC_1000	431.753	227.800	432.791	221.589	397.310

Conclusion

In this paper, one general purpose and four XML specific compressors were examined. It can be concluded that specific application types determine the choice of the compressor. In a local distributed system, where the size of XML documents that need to be transferred will usually be small (<20K), XMill is not advantageous since the cut-off size for XMill is over 20K. In such a case, it would be best to use a general-purpose compressor such as gZip or WinZip. On the other hand, if the XML data were small, XComp would also be a better choice compared to XMill since the boundary for XComp is 4K which is best suitable for applications involving compression of emails. Also, in wireless applications like PDAs, palmtops etc, neither XMill nor gZip will be advantageous since querying of data is very important and it is not a good choice to decompress each time for a query, the best choice would be to use XGrind.

There is also a possible argument that since bandwidth is very cheap today, there is really no need for compression. However, with the advent of wireless devices such as phones, PDAs etc, bandwidth is at a premium. Therefore, compressing data to transmit to such devices becomes essential.

Quick comparison of the compressors studied

	GZip	XMill	XGrind	ICT Xpress	XComp
Queryable	**X**	**X**	Ö	Ö	N/A
Human Intervention	**X**	Ö	**X**	**X**	
Separation of data & structure	**X**	Ö	Ö	Ö	Ö
Ease of use	Ö	Ö		Ö	
Utilization of DTD	**X**	**X**	Ö	**X**	**X**

References

[1] http://www.flatironssolutions.com/white_papers/XML_Compression.pdf

[2] http://www.oledo.com/hartmut/XMill/XMill.html

[3] H. Liefke and D. Suciu. XMill: An Efficient Compressor for XML Data

[4] P. Tolani and J. Harista. XGrind: A Query-friendly XML Compressor

[5] J. Min, M. Park and C. Chung. XPRESS: A Queriable Compression for XML Data

[6] J. Cheney. Compressing XML with Multiplexed Hierarchical PPM Models

[7] http://www.cse.ogi.edu/class/cse582/Lectures/Lecture15/XML_compression_discussant.ppt

[8] M. Cai, S. Ghandeharizadeh, R. Schmidt and S. Song: A Comparison of Alternative Encoding Mechanisms for Web Services

[9] http://www.cmpe.boun.edu.tr/courses/cmpe521/fall2003/cetin-durusut.ppt

[10] http://darwell.uwaterloo.ca/~ddbms/publications/distdb/Weimin.pdf

[11] http://www.cs.uu.nl/~johanj/publications/gp4xml.pdf

[12] http://www.ictcompress.com/PDF/XML_WhitePaper.pdf