

USING XML COMPRESSION FOR WWW COMMUNICATION

Tomasz Müldner, Gregory Leighton, Jim Diamond

*Jodrey School of Computer Science, Acadia University,
Wolfville, NS, Canada B4P 2R6*

Email: {tomasz.muldner,005985L, [jim.diamond](mailto:jim.diamond@acadiau.ca)}@acadiau.ca

ABSTRACT

XML is a popular meta-language that facilitates the interchange and access of data. In recent years, it has seen widespread adoption as a data representation format in Web applications serving a variety of application domains. However, XML's verbose nature may increase the size of a data set as much as ten-fold. Therefore, XML-based Web applications will often experience a performance benefit from the introduction of a strategy for compressing XML. In this paper, we present an XML-conscious compression scheme that operates online and allows queries to be carried out directly on compressed data.

KEYWORDS

XML, compression, WWW communication

1. INTRODUCTION

The eXtensible Markup Language (XML) [7] is a World Wide Web Consortium (W3C) endorsed standard for semi-structured data. XML data are surrounded by textual markup that serves to describe the semantics of the data. The markup results in an increased verbosity; indeed it is not uncommon for the XML representation of a set of data to be as much as ten times as large as alternative representations of the same data (e.g. data in comma-separated value format). This overhead is of particular concern for XML-based communication over the World Wide Web (WWW), where transmitting XML messages between participating systems may result in significant delays.

Another challenge is represented by the increasing number of memory-constrained devices participating in WWW applications, such as smart phones and personal digital assistants (PDAs). While it is desirable to integrate such devices into existing XML-based web applications, their inability to process lengthy XML messages has become a serious concern.

Many WWW applications employ XML messaging. SOAP [16] is a platform- and language-independent XML-based protocol for sending and receiving messages representing remote procedure calls; it can be implemented over a variety of communication protocols, such as HTTP and SMTP. Using Web Services [19], network services can be discovered, described, and invoked in a platform- and implementation-independent way by exchanging XML messages. As a further example, Atom [2] is an XML-based protocol used for syndicating and publishing episodic content for Weblogs ("blogs") and Wikis.

There have been previous attempts to develop strategies for compressing XML messages (for more details, see Section 2). However, these attempts have typically suffered from a number of problems. First, for a compression used over the WWW to be efficient, it is essential that it operate in an *online* fashion. Here, online compression means that the decompressor does not have to wait until the entire compressed document has been received; instead it can start decompressing the document as soon as it receives the beginning of the compressed file. Clearly, for (soft or hard) real-time network applications an online compression is more efficient than an offline compression, where the decompression must be delayed until the entire compressed document has been received. Second, some attempts do not incorporate prior knowledge of XML syntax; in other words they are not XML-aware and instead have centered their compression routines on general text compressors, such as gzip [9]. Therefore, the resulting compression rate is generally inferior when compared

to XML-aware compression strategies. Finally, in nearly all of these attempts, querying of the resulting compressed data can be performed only after this data has been completely decompressed.

In this paper, we describe the application of XML compression to WWW communication. Our application is based on TREECHOP [11], an XML-conscious compression scheme which addresses all of shortcomings described in the previous paragraph. On the sending end, input data are read by the TREECHOP compressor and sent over the Internet; on the receiving end, the data are read by the decompressor and stored, or further piped into any WWW application, such as a SOAP processor. The receiving end can also store compressed data, which can be subsequently queried without having to decompress it. It is also possible to query a compressed document, without having to send it over the Internet. In this case, the user sends a query, the receiving site applies this query to the compressed document, and sends back the result of this application to the user.

This paper is organized as follows. Section 2 presents related work, and Section 3 briefly describes the functionality and implementation of TREECHOP. Section 4 shows applications of TREECHOP to WWW communication, and finally Section 5 provides conclusions and describes future work.

2. RELATED WORK

There have been several previously-proposed approaches for XML compression. XMill [12] represents the pioneering work in the area of XML-conscious compression. Although XMill often outperforms gzip on XML data, the original structure of the document is disrupted during the compression process, which precludes online processing. This serves to limit the usefulness of XMill for XML messaging applications. In addition, the XMill encoding format does not allow querying of compressed data.

XMLPPM [5] achieves a highest degree of compression (among published techniques) via the use of multiplexed hierarchical models and the PPM [6] text compression method. As with XMill, compressed data cannot be queried.

XGRIND [17] was the first XML-conscious compression scheme to support querying without full decompression. Element and attribute names are encoded using a byte-based scheme, and character data is compressed using semi-adaptive Huffman coding [10]. Use of the latter technique significantly slows down the compression process, since two passes over the original document are required (first to gather probability data for the compression model, and a second time to perform the encoding according to the generated model).

XPRESS [13] also supports querying of compressed data and claims to achieve better compression than XGRIND. However, it uses a semi-adaptive form of arithmetic coding which also necessitates two passes over the original XML document.

XQueC [1] provides a compressed storage for XML data that can be tailored to a specified query workload, offering querying support for a subset of the XQuery [21] standard over compressed data. However, the scheme used in XQueC relies on custom structures for storage and indexing, an approach which is not readily applicable to efficient streaming of XML documents over a network.

It should be noted that one alternative to using XML compression is to adopt a binary XML format. One attempt to achieve this goal is to use ASN.1 [3], which is language- and platform-independent notation for describing data transmitted by telecommunications protocols. While ASN predates XML, it provides XML Encoding Rules to define binary encodings. Universal Business Language (UBL), [18] uses the ASN.1 approach to XML data compression.

3. DESCRIPTION OF TREECHOP

3.1 Introduction

TREECHOP's design goal is to provide good compression of XML documents while supporting "online" usage. In this context, "online" usage means: (a) only one pass through the document is required to compress it, (b) compressed data is sent to the output stream incrementally as the document is read, and (c) decompression can begin as soon as compressed data is available to the decompressor. Thus transmission of a document over a network can begin as soon as the compressor produces its first output, and, consequently, the decompressor can start decompression shortly thereafter, resulting in a compression scheme that is well-suited for transmission of XML documents over a wide-area network.

This online capability allows the receiver to query the data as it is being received, which could improve the overall response time of a system which requires only a portion of the XML document. Such queries are formulated using XPath expressions [20]. These queries can be "exact" queries, where some "record" containing a specific value of some parameter is desired; alternatively, range queries, which request any "records" where the parameter of interest falls in some range, are also supported. For example, in a document containing employee records, an exact query might request the employee whose employee ID is 1000, whereas a range query might request all employees hired between January 1 and June 30 in 2003.

Normally, "lossless" compression schemes are those where the decompressor outputs an exact copy of the input data. In the case of XML documents, one can define "semantically lossless" compression (i.e. to a XML 1.0 compliant parser, the decompressed document is the same as the original document). TREECHOP compression/decompression is semantically lossless; in particular quotes and ampersands are changed; e.g. '&' to '&'. Note that TREECHOP can preserve the white space of the original document, and it can also generate a canonicalized [4] XML document.

3.2 Implementation

In this section, we briefly describe the implementation of TREECHOP; see [11] for a more detailed description.

3.2.1 Compression Strategy

The compression process in TREECHOP begins by conducting a SAX-based [15] parsing of the XML document. As tokens are returned by the parser, they are re-interpreted as tree nodes and then written out to the compression stream in depth-first order. This approach avoids building an in-memory representation of the entire document tree. As node information is added to the compression stream, it is compressed using gzip. Each non-leaf node is assigned a binary codeword, based on the path of the node (as traced from the document's root node). If there are multiple nodes with the same path value, each occurrence will receive the same codeword. The codeword $C(v)$ assigned to a non-leaf node v with parent node p is formed by the concatenation of three codes $C(p)$, $G(v)$, and $T(v)$, where $C(p)$ is the codeword assigned to p , $G(v)$ is a Golomb code [8] assigned to v based on its ordering relative to p , and $T(v)$ is a sequence of 3 bits used to indicate the node type (for example, each element node receives 000). This encoding scheme has three important properties: (1) the codeword for each node is prefixed by its parent's codeword; (2) two nodes share the same codeword if and only if they have the same absolute path, traced from the tree root downwards; and (3) the structure of the original XML document is maintained by the encoding scheme. Therefore, the decoder can reconstruct the original document from its encoded representation.

Information about the node is written to the encoding stream immediately after being assigned a codeword. This allows the decoder to set about decoding the node without waiting for the entire codeword assignment procedure to complete on the encoder side.

The encoding information for each tree node is written to the encoding stream in an adaptive fashion. Each non-leaf node is encoded as a 3-tuple (L, C, D) , where L is a byte indicating the bit length of the codeword; C is the codeword assigned to the node, consisting of a sequence of $\lceil L / 8 \rceil$ bytes; and D is the textual data stored in the node. A reserved byte value is used to indicate to the decoder that raw character data is forthcoming in the encoding stream; once D has been transmitted, a second reserved byte value is

used to signal the end of the character data sequence. For the second and subsequent occurrences of a particular codeword, only the 2-tuple (L, C) is transmitted since the decompressor is able to infer the value of D at that point.

Leaf nodes are transmitted in the same manner as D , described above.

3.2.2 Decompression Strategy

Since tree node encodings are written to the compression stream in depth-first order, it is possible for the decompressor to regenerate the original XML document incrementally. A code table is used to store $(L, C) \rightarrow D$ mappings for previously-encountered tree nodes. In addition, a stack is employed to maintain proper nesting of elements during the decompression process.

As each non-leaf tree node is encountered in the compression stream, the decompressor determines the node type by examining the final three bits in the codeword. The type information is then used to surround the D value for this node with the appropriate XML syntax before emitting it to the decompression stream.

3.2.3 Querying Strategy

Exact-match queries can be carried out via a single scan through the compression stream. The query processor employs a stack to keep track of the current path; when the query predicate path is first encountered, the associated codeword C is recorded and the next occurring D value is extracted from the compression stream as a query match. Subsequently, the remainder of the stream is scanned for future occurrences of C . With each match, the associated D value is extracted from the stream.

Range queries are handled in a similar manner, except that each query match additionally requires that the corresponding D value be converted into a numeric value and tested to see if it falls within the query range before it is returned as a search result.

4. APPLICATIONS OF TREECHOP TO WWW COMMUNICATION

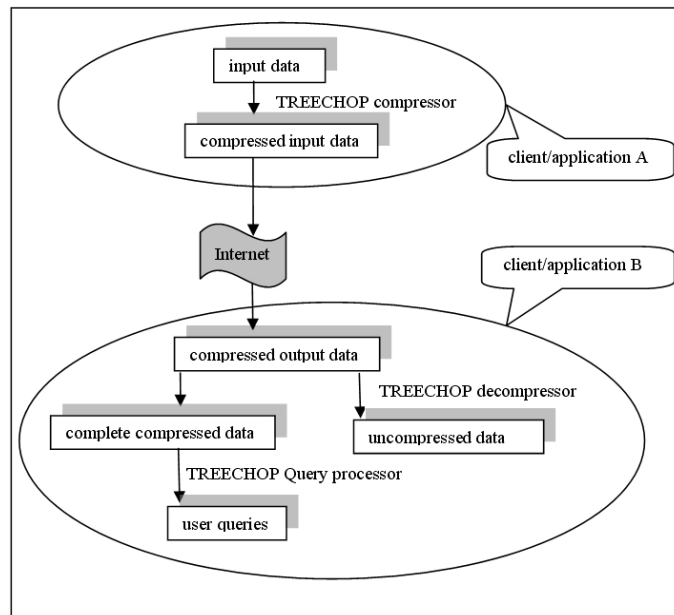


Figure 1. Functionality of TREECHOP

4.1 Introduction

Figure 1 shows the architecture of our system. Input data are piped into the TREECHOP compressor and sent over the Internet. On the receiving end, the data are piped into two programs; one that collects the entire compressed document, and a second program which performs on-the-fly decompression. The decompressed data can be piped into any WWW application, such as a SOAP processor. On the other hand, the complete compressed data can be stored, and queried without having to decompress it. It is also possible to query a compressed document, without having to send it over the Internet. In this case, the user sends a query, the receiving site applies this query to the compressed document, and sends back the result of this application to the user.

4.2 Experimental results

This section presents the results of experiments that were carried out to evaluate the effectiveness of using TREECHOP to compress XML data before sending it over a network.

Listing 1 shows an example XML/EDI message, a modification of an example borrowed from [14], in which multiple products have been added (for the sake of brevity, not all these products are shown). In this example, the EDI message was converted to XML, producing an XML file eight times larger than the size of the original message.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<PO>
  <POHeader>
    <Transaction X12.ID="850">
      <ID>850</ID>
      <ControlNo>12345</ControlNo>
    </Transaction>
  </POHeader>
  <PODetail>
    <NameInfo>
      <Name>
        <EntityId>Buying Party</EntityId>
        <IdCode>RET8999</IdCode>
      </Name>
      <Name>
        <EntityId>Ship To</EntityId>
        <IdCode>RET8999</IdCode>
      </Name>
      <Address>123 Via Way</Address>
      <Location>
        <City>Milwaukee</City>
        <State>WI</State>
        <Zip>53202</Zip>
      </Location>
    </NameInfo>
    <Product id="CO633">
      <QuantityOrdered>100</QuantityOrdered>
      <UnitPrice>1.23</UnitPrice>
    </Product>
    <!--more products removed -->
  </PODetail>
</PO>
```

Listing 1. Sample XML document

We shall first discuss the compression rate; Table 1 provides the results from our tests (measured in bits-per-character), performed on the above example, as well as four XML files: player statistics from the 1998 Major League Baseball season, Shakespeare's play *Macbeth*, a highly-structured document containing 150

employee records, and a similar document with 100,000 employee records. Note that XGrind failed to process the first file.

Table 1. Comparison of compression rates

Example	Original size	gzip	bzip2	XGrind	TREECHOP
EDI/XML	4143	1.21	1.22	fails	1.21
1998statistics	806400	0.63	0.27	2.07	0.61
employees150	25910	1.11	0.82	3.79	1.08
employees100000	17234394	0.86	0.46	2.51	0.81
macbeth	179168	2.11	1.56	3.81	2.10

The results indicate that TREECHOP outperforms all tested compressors, except bzip2. However, bzip2 is not XML-conscious (it can't differentiate whether a sequence of characters forms part of an element tag or a data value) so one couldn't use bzip2 "out of the box" in order to support querying using XPath-type predicates (where instances of particular elements and/or attributes are to be extracted).

Next, we discuss TREECHOP's operating speed. Two experiments were carried out; both compressed and transmitted each of the five sample XML files over a TCP connection to a server, where the files were decompressed to regain the original documents. In the first experiment, the client was located approximately 7,000 km from the server, and in the second the client was on the same LAN as the server. (The server was the same computer in both experiments.) Table 2 shows these results (in milliseconds); "server time" includes only the time required to perform decompression, and "server c-time" additionally includes the time required to establish the socket connection, write the decompressed document to a local file, and close the socket connection. These results show that compression times over a long distance are not much worse than those over a short distance.

Table 2. Compression times (in ms) for online compression

Example	client1 time	server1 time	server1 c-time	client2 time	server2 time	server2 c-time
EDI/XML	78	26	1203	78	12	382
1998statistics	2703	5823	7091	2461	2578	4231
employees150	250	204	755	274	115	682
employees100000	127609	131562	132682	56128	60115	61670
macbeth	1469	1629	2473	711	539	1380

Assuming that TREECHOP would be mainly used on relatively small XML documents (like SOAP documents), the compression gained by substituting bzip2 would likely be outweighed by the extra compression/decompression times.

To illustrate the operation of querying using TREECHOP, consider the XML document shown in Listing 2, containing a collection of ten employee records. In this experiment, we wish to compress and transmit the document over a TCP socket connection to a remote server, where a query handler will be registered to extract the last name of each employee record from the compression stream and display it on the console (i.e. the query handler is mapped to the path value `/employees/employee/lname`). While this particular example is trivial, it serves as an illustration of the fundamental operation of TREECHOP querying.

The client-side operation consists of specifying on the command-line the name of the document to be compressed and transmitted, along with the hostname/ip address and port information for the destination system. Figure 2 displays a screenshot of the client-side.

Figure 3 shows the output generated on the server side. Note that the value of the last name field for each of the ten employee records contained in the transmitted XML document has been extracted from the compression stream and printed to the console on a separate line.

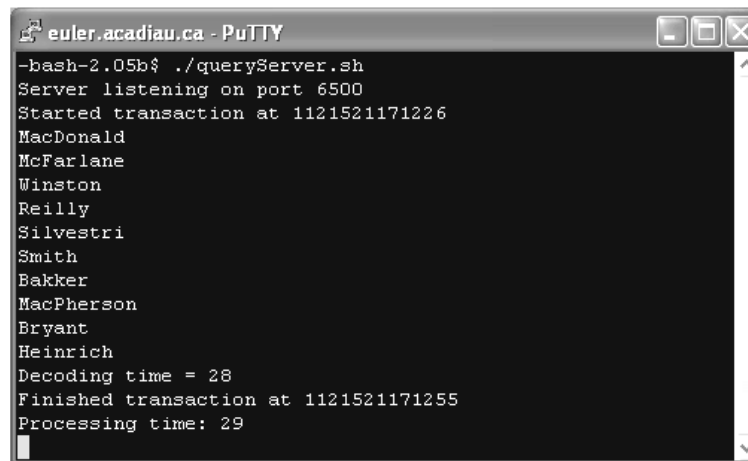
Listing 2. XML document used in the querying experiment

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<employees>
  <employee id="177015">
    <fname>Sandra</fname>
    <lname>MacDonald</lname>
    <age>49</age>
    <phone>350-9186</phone>
  </employee>
  <employee id="155489">
    <fname>Michael</fname>
    <lname>McFarlane</lname>
    <age>51</age>
    <phone>350-1184</phone>
  </employee>
  <!-- 8 additional employee records removed -->
</employees>
```



```
Command Prompt
E:\TREECHOP\dist\bin>queryclient.bat 10emp.xml
E:\TREECHOP\dist\bin>java -cp .;../lib/xercesImpl.jar;../lib/treechop.jar treechop.net.TestClient 10emp.xml euler.acadiau.ca 6500
Finished creating the document tree: 1121521157873
Tree building took 20
```

Figure 2. Screenshot of client in querying example



```
euler.acadiau.ca - PuTTY
-bash-2.05b$ ./queryServer.sh
Server listening on port 6500
Started transaction at 1121521171226
MacDonald
McFarlane
Winston
Reilly
Silvestri
Smith
Bakker
MacPherson
Bryant
Heinrich
Decoding time = 28
Finished transaction at 1121521171255
Processing time: 29
```

Figure 3. Screenshot of server in querying example

2. CONCLUSIONS AND FUTURE WORK

This paper described the application of XML compression to WWW communication. Our application is based on TREECHOP, an XML-aware online compressor of XML data over wide-area networks. Input data are piped into the TREECHOP compressor and sent over the Internet; on the receiving end, the data are piped into the decompressor (the decompressed data can be further piped into any WWW application, such as a

SOAP processor).

The complete compressed data received over the Internet can be stored and subsequently queried without having to decompress it. It is also possible to query a compressed document, without having to send it over the Internet. In this case, the user sends a query, the receiving site applies this query to the compressed document, and sends back the result of this application to the user.

While the results (in terms of both compression rate and compression speed) of the current version of TREECHOP make it an attractive option for systems transmitting XML documents over wide-area networks, there are optimizations that should improve both compression rate and speed. The authors intend to investigate these in the near future to see how much the current performance can be improved.

The current implementation of TREECHOP supports only exact queries; range queries will be added in the near future.

REFERENCES

- [1] Arion, A., Bonifati, A., Costa, G. et al. "Efficient query evaluation over compressed XML data," in 2004 Proc. Of EDBT, pp. 200-218.
- [2] Atom Publishing Format and Protocol Charter. <http://ietf.org/html.charters/atompub-charter.html>.
- [3] ASN, <http://asn1.elibel.tm.fr/en/introduction/index.htm>
- [4] Canonical XML, <http://www.w3.org/TR/2001/REC-xml-c14n-20010315>
- [5] Cheney, J. "Compressing XML with multiplexed hierarchical PPM models," in 2001 Proc. IEEE Data Compression Conference, pp. 163-172.
- [6] Cleary J.G. and Witten I.H., "Data compression using adaptive coding and partial string matching," IEEE Trans. Comm., vol. 32, no. 4, pp. 396-402, Apr. 1984.
- [7] Extensible Markup Language (XML) 1.0 (3rd ed.). <http://www.w3.org/TR/REC-xml>.
- [8] Golomb S.W., "Run-length encodings," IEEE Trans. on Info. Theory IT-12(3), pp. 399-401, July 1966.
- [9] gzip, <http://www.gzip.org>.
- [10] Huffman D., "A method for the construction of minimum redundancy codes," in Proc. of the IRE., vol. 40, no. 9, pp. 1098-1101.
- [11] Leighton G., Müldner T., Diamond J. "TREECHOP: A Tree-based Query-able Compressor for XML," Jodrey School of Computer Science Technical Report 2005-005, <http://cs.acadiau.ca/technicalReports/files/tr-2005-005.pdf>.
- [12] Liefke H. and Suciu D., "XMill: an efficient compressor for XML data," in 2000 Proc. ACM SIGMOD Int'l Conf. on Management of Data, pp. 153-64.
- [13] Min J-K., Park M-J. and Chung C-W., "XPRESS: a queryable compression for XML data," in Proc. 2003 ACM SIGMOD Int'l Conf. on Management of Data, pp. 122-33.
- [14] Ogbuji U., Tip: Compress XML files for efficient transmission. <http://www-106.ibm.com/developerworks/xml/library/x-tipcomp.html>.
- [15] Simple API for XML (SAX), <http://www.saxproject.org>.
- [16] SOAP <http://www.w3.org/TR/soap/>.
- [17] Tolani P.M. and Haritsa J.R., "XGRIND: a query-friendly XML compressor," in Proc. 2002 Int'l Conf. on Database Eng., pp. 225-34.
- [18] Universal Business Language, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ubl.
- [19] Web services <http://www.w3.org/2002/ws/>.
- [20] XML Path Language (XPath), <http://www.w3.org/TR/xpath>.
- [21] XQuery 1.0: an XML query language. <http://www.w3.org/TR/xquery>.