Investigate state-of-the-art XML compression techniques

Review data compressors that recognize XML (or not) and that you can query (or not)

Sherif Sakr

July 19, 2011

XML is considered a standard for data representation and exchange over the World Wide Web. XML has great flexibility and wide acceptance, but one drawback is the large size of XML documents. This size means that the amount of information that you transmit, process, store, and query is often larger than that of other data formats. You can choose from several XML compression techniques to deal with these problems. This article provides an overview of current XML compression techniques.

Introduction

XML is one of the most useful and important technologies that has emerged as a result of the immense popularity of HTML and the World Wide Web. XML solves numerous problems as it provides neutral data representation between different architectures, bridges the gap between software systems with minimal effort, and stores large volumes of semi-structured data.

XML is often referred to as self-describing data because it is designed so the schema repeats for each record in the document. This self-describing feature provides XML with immense flexibility, but it also introduces the problem of verbose XML documents, which results in huge document sizes. Because XML use continues to grow and because large repositories of XML documents are currently pervasive, the demand for efficient XML compression tools is great.

Frequently used acronyms

- CDATA: Character data
- DTD: Document Type Definition
- GPS: Global positioning system
- HTML: HyperText Markup Language
- PPM: Prediction by Partial Match
- SAX: Simple API for XML
- W3C: World Wide Web Consortium
Figure 1 illustrates the advantage of using XML compressors to reduce the cost of transmitting XML data over the network. To tackle the size problem of large XML documents, several XML-conscious compressors exploit the well-known structure of XML documents to achieve compression ratios that are better than those of general text compressors. The many advantages of XML compression tools include reduced network bandwidth required for data exchange, reduced disk space required for storage, and minimized main memory requirements of processing and querying XML documents.

Figure 1. An example of the advantage of using XML compressors in transmitting XML data over the network

In principle, XML compressors can be classified with respect to two main characteristics. Figure 2 illustrates the first classification, which is based on their awareness of the structure of the XML documents. According to this classification, compressors are divided into two main groups:

- **General text compressors.** Because XML data is stored as text files, the first logical approach for compressing XML documents was to use the traditional, general-purpose text compression tools (for example, gzip, bzip2). This group of XML compressors is XML-blind, that is, they treat XML documents as usual plain text documents and thus apply the traditional text compression techniques.

- **XML-conscious compressors.** This group of compressors is designed to take advantage of the awareness of the XML document structure to achieve better compression ratios over general text compressors. This group of compressors can be further classified according to their dependence on the availability of the schema information of the XML documents as follows:
  - **Schema-dependent compressors.** Both the encoder and decoder must have access to the document schema information to achieve the compression process (for example, rngzip).
  - **Schema-independent compressors.** The availability of the schema information is not required to achieve the encoding and decoding processes (for example, XMill, SCMPPM).
Figure 2. Classification of XML compressors according to their awareness of the structure of XML documents

![XML Compressors Diagram]

Figure 3 illustrates the second classification of XML compressors, which is based on their ability to support queries as follows:

- **Non-queriable (archival) XML compressors.** This group of the XML compressors does not allow any queries to be processed over the compressed format (for example, gzip, bzip2, XMill). The main focus of this group is to achieve the highest compression ratio. By default, general-purpose text compressors belong to the non-queriable group of compressors.

- **Queriable XML compressors.** This group of the XML compressors allows 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, however, is to avoid full document decompression during query execution. In fact, the ability to perform direct queries on compressed XML formats is important for many applications that are hosted on resource-limited computing devices, such as mobile devices and GPS systems. By default, all queriable compressors are XML-conscious compressors as well. You can further classify this group of compressors by how they encode the structural and data parts of the XML documents:
  - **Homomorphic compressors.** The original structure of the XML document is retained, and you can access and parse the compressed format in the same way as the original format (for example, XGrind).
  - **Non-homomorphic compressors.** The encoding process of the XML document separates the structural part from the data part (for example, XQueC). Therefore, the structure of the compressed format is different from the structure of the original XML document.
Investigate state-of-the-art XML compression techniques

Figure 3. Classification of XML compressors according to their support for executing queries

General text compressors

XML is a text representation for a tree-structured data. A straightforward logical approach for compressing XML documents is to use the traditional, general-purpose text compression tools. Numerous algorithms devised over the past decades efficiently compress text data. The most popular and efficient representatives of this group are gzip, bzip2, and PPM compressors.

The gzip compressor is based on the DEFLATE lossless data compression algorithm, which uses a combination of the LZ77 algorithm and Huffman coding. To compress data, the LZ77 algorithm replaces portions of the data with references to matching data that has already passed through both the encoder and the decoder. Huffman coding uses a specific method for choosing the representation for each symbol where the most common characters using shorter strings of bits then are used for less common source symbols.

The bzip2 compressor uses the Burrows-Wheeler transform to convert frequently recurring character sequences into strings of identical letters and then applies a move-to-front transform and finally Huffman coding. The Burrows-Wheeler transform permutes the order of the characters in a way that if the original string had several substrings that occurred often, then the transformed string has several places where a single character is repeated multiple times in a row. This approach is useful for compression because it tends to be easy to compress a string that has runs of repeated characters. In practice, the bzip2 compression compresses files at a higher compression ratio than those compressed using gzip, but it also has slower performance.

PPM is an adaptive statistical data compression technique based on context modeling and prediction. It uses a finite context statistical modeling technique that blends together several fixed-order context models to predict the next character in the input sequence. Prediction probabilities for each context in the model are calculated from frequency counts, which are updated adaptively. The symbol that actually occurs is encoded relative to its predicted distribution using arithmetic coding. Although PPM is simple and is the most efficient of the compressors presented so far, it is also computationally the most expensive.
In practice, general text compressors are used either for archiving purposes or for reducing the network bandwidth during the data exchange process. In general, there is a trade-off between these compressors in terms of two main metrics: compression ratio and the compression/decompression time. On one side, PPM compression achieves the best compression ratio while gzip achieves the least compression ratio. On the other side, gzip has best performance in terms of compression/decompression time and while the compression/decompression time of PPM is much slower. bzip2 sits in the middle for both metrics. Thus, the user selection of the compressor to use mainly depends on the requirements of the user's scenario in terms of these two metrics.

**Non-queriable (archival) XML compressors**

This group of XML compressors does not allow any queries to be processed over the compressed format as their main focus is to achieve the highest compression ratio. This section discusses representatives of the two main classes of this group:

- Schema-independent compressors
- Schema-dependent compressors

**Schema-independent compression schemes**

This class of compression schemes does not require the availability of the schema information for encoding and decoding processes. XMill 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 homogenous containers based on their relative paths in the tree and their data types.

In XMill, both the structural and data value parts of the source XML document are collected and compressed separately. In the structure part, XML tags and attributes are encoded in a dictionary-based fashion before passing the encoding to a back-end general text compression scheme. The structural encoding scheme of XMill assigns each distinct element and attribute name an integer code, which serves as the key into the element and attribute name dictionaries. In the data part, data values are grouped into homogenous and semantically related containers according to their path and data type. Each container is then compressed separately using a specialized compressor that is ideal for the data type of this container. This grouping operation serves to localize repetitions and therefore enhance the degree of compression.

In the latest versions of the XMill source distribution, the intermediate binaries of the compressed format can be passed to one of three alternative back-end general-purpose compressors: gzip, bzip2, and PPM. Figure 4 illustrates an overview of the general architecture of the XMill compressor with the XML parser, structure and data containers, one or more compression schemes, and the compressed XML file (that contains compressed structure and compressed data).
Investigate state-of-the-art XML compression techniques

Figure 4. General architecture of the XMill compressor

Figure 5 illustrates an example of splitting an XML file into the structural and data containers. The elements and attributes tables store the structural containers of the XML document. The values of each unique path (element or attribute) are stored in separate tables (containers). Thus, the values in each container become homogenous and can be compressed more efficiently. This example includes separate elements (customers, customers/customer, customers/customer/firstName, customers/customer/lastName, customers/customer/invoice, customers/customer/invoice/items, and customers/customer/invoice/items/item) and attributes (customers/customer/@id, and customers/customer/invoice/@total) tables.

Figure 5. An example of splitting an XML file into structural and data containers

```
<customers>
  <customer id="c1">
    <firstName>John</firstName>
    <lastName>Smith</lastName>
    <invoice total="300">
      <items>
        <item>item1</item>
        <item>item2</item>
      </items>
    </invoice>
  </customer>
  <customer id="c2">
    <firstName>Bill</firstName>
    <lastName>Luis</lastName>
  </customer>
</customers>
```

Elements table

<table>
<thead>
<tr>
<th>No.</th>
<th>Element Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>/customers</td>
</tr>
<tr>
<td>2</td>
<td>/customers/customer</td>
</tr>
<tr>
<td>3</td>
<td>/customers/customer/firstName</td>
</tr>
<tr>
<td>4</td>
<td>/customers/customer/lastName</td>
</tr>
<tr>
<td>5</td>
<td>/customers/customer/invoice</td>
</tr>
<tr>
<td>6</td>
<td>/customers/customer/invoice/items</td>
</tr>
<tr>
<td>7</td>
<td>/customers/customer/invoice/items/item</td>
</tr>
</tbody>
</table>

Attributes table

<table>
<thead>
<tr>
<th>No.</th>
<th>Attribute Path</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>/customers/customer/@id</td>
<td>John</td>
</tr>
<tr>
<td>101</td>
<td>/customers/customer/invoice/@total</td>
<td>Smith</td>
</tr>
<tr>
<td></td>
<td>/customers/customer/firstName</td>
<td>Bill</td>
</tr>
<tr>
<td></td>
<td>/customers/customer/lastName</td>
<td>Luis</td>
</tr>
<tr>
<td></td>
<td>/customers/customer/invoice/items/item</td>
<td>Item1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Item2</td>
</tr>
<tr>
<td></td>
<td>/customers/customer/@id</td>
<td>C1</td>
</tr>
<tr>
<td></td>
<td>/customers/customer/invoice/@total</td>
<td>300</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C2</td>
</tr>
</tbody>
</table>

Figure 6 illustrates the command-line options of the XMill compressor.
Investigate state-of-the-art XML compression techniques

Figure 6. Command-line options of the XMill compressor

Figure 7 illustrates the effect of compressing an XML document (tpc.xml, size of 282 KB) using the XMill compressor where the output compressed file (tpc.xmi, size of 41 KB) size represents 15 percent of the size of the original XML file.

Figure 7. The output of compressing an XML file using the XMill compressor

XMLPPM is a streaming XML compressor that uses a multiplexed hierarchical PPM model called MHM. XMLPPM is considered an adaptation of the general-purpose prediction by partial matching compression scheme (PPM). In XMLPPM, an XML file is first parsed using
an SAX parser to generate a stream of SAX events. Each event is encoded using a bytecode representation called ESAX (encoded SAX). The ESAX bytecodes are encoded using one of several multiplexed PPM models based on its syntactic structure (elements, characters, attributes, and miscellaneous symbols). The SCMPPM compressor has been proposed as a variant of the XMLPPM compressor. SCMPPM combines a technique called structure context modeling (SCM) with the PPM compression scheme. It uses a bigger set of PPM models than XMLPPM as it uses a separate model to compress the text content under each element symbol.

The compression ratio and compression/decompression time of XMill and XMLPPM are very dependent on and related to their back-end, general-purpose compressors (gzip, bzip2, or PPM). Thus, they have the same trade-off as their general-purpose, back-end compressors.

**Schema-dependent compression schemes**

This class of compressors requires the availability of the schema information of the XML document during their encoding and decoding processes. For example, the XAUST XML compressor converts the schema information of the DTD into a set of deterministic finite automata (DFA), one for each element in the DTD. Then, each transition is labeled by an element, and the action associated with a transition is a call to a simulator for the DFA for the element labeling that transition. XAUST groups all data for the same element into a single container, which is compressed incrementally using a single model for an arithmetic order-4 compressor. Using the DTD schema information, XAUST is able to track the structure of the document and is able to make accurate predictions of the expected symbols. Whenever the predicted symbol is unique, there is no need to encode it as the decoder generates the same model from the DTD and thus can generate the unique expected symbol.

The RNGzip XML tool compresses XML documents that conform to a given Relax NG schema. In RNGzip, the sender and receiver must agree in advance on precisely the same schema. In this sense, the schema is like a shared key for encryption and decryption. RNGzip uses a Relax NG schema validator to build a deterministic tree automaton from the specified schema. Then, given an XML document, it checks whether the XML is accepted by the automaton. Given this automaton, a receiver can reconstruct an entire XML document by transmitting very little information. If there is a choice point in the automaton, RNGzip just transmits which transition was taken, and if it encounters the text transition, then the matching text is transmitted.

Theoretically, schema-dependent compressors might achieve slightly higher compression ratios than the schema-independent compressors. But they are not preferable or commonly used as the schema information of the XML documents might not always be available, thus losing the advantage of XML’s flexibility to represent semi-structured data. This type of compressor can be effective only when it is used to compress XML documents with predefined schemas.

**Queriable XML compressors**

The main focus of queriable XML compressors is to allow queries to be directly evaluated over their compressed formats without decompressing the whole document. The compression ratio of this group is usually worse than that of the archival XML compressors. This type of compressor is very important for many applications that are hosted on resource-limited computing devices.
such as mobile devices and GPS systems. This section discusses representatives of the two main classes of this group:

- Homomorphic compressors
- Non-homomorphic compressors

**Homomorphic queriable XML compressors**

This class of compressors retains the original structure of the XML document in the compressed format so it can be accessed and parsed in the same way as the original format. XGrind is the first XML-conscious compression scheme to support querying without the need for a full decompression of the compressed XML document. XGrind does not separate data from structure. It retains the original structure of the XML document.

The homomorphic nature of the XGrind compressed format grants XGrind many interesting features:

- The compressed XML document can be viewed as the original XML document with its tags and element/attribute values replaced by their corresponding encodings. Hence, XGrind can be considered an extended SAX parser.
- XML indexing techniques can be built on the compressed document in a similar manner to those that can be built on regular XML documents. In XGrind, element and attribute names are encoded using a dictionary-based encoding scheme, and character data is compressed using semi-adaptive Huffman coding. The query processor of XGrind can handle only exact-match and prefix-match queries on compressed values and partial-match and range queries on decompressed values. Several operations are not supported by XGrind, though, for example, non-equality selections in the compressed domain. Therefore, XGrind cannot perform any join, aggregation, nested queries, or construct operations.

XPress is another homomorphic queriable XML compressor that uses a combination of the characteristics to compress and retrieve XML data efficiently. To encode the labels and paths of XML documents, it uses a reverse arithmetic encoding method over distinct intervals. Using the containment relationships among the intervals, path expressions are evaluated on compressed XML data. The compression scheme of XPress is a semi-adaptive one that uses a preliminary scan of the input file to gather statistical information, and the encoding rules for data are independent of the locations of data. It also uses proper encoders for data values based on their automatic inferred type information.

**Non-homomorphic queriable XML compressors**

This class of compressors separates the structural part from the data part during the encoding process of the XML document. Therefore, different from the homomorphic class, the structure of the compressed format is different from the structure of the original XML document and needs to be parsed in a different way during the decompression process. They, however, can achieve a better compression ratio. For example, XSeq is a grammar-based queriable XML compression scheme, which is considered an adaptation of the famous grammar-based text strings compression algorithm, Sequitur.
In XSeq, the tokens of the input XML file are separated into a set of containers, each of which is then compressed using Sequitur. The Sequitur compression algorithm is a linear-time online algorithm that forms a context-free grammar for a given string input. Using the defined context-free grammars, XSeq avoids the sequential scan of irrelevant compressed data and processes only data values that are to be matched by the given query. In addition, context-free grammars allow XSeq to process queries directly over the compressed file without full or partial decompression. To correlate the data values stored in different containers and accelerate the query evaluation time, XSeq uses a set of indices that are stored within the compressed file and are loaded into the memory before processing the rule contents. For example, it uses a structural index through which each data value can be quickly located in the container without decompression while the header index contains a list of pointers to the entrance of each container in the file.

The TREECHOP XML compressor is another queriable XML compressor where the compression process begins by conducting an SAX-based parsing of the XML document and the parsed tokens are written out to the compression stream in depth-first order. The codeword for each node is prefixed by the codeword of its parent, and two nodes in the XML document tree share the same codeword if they have the same path. Each CDATA section, comment, processing instruction, and non-leaf node is assigned a binary codeword. This codeword is uniquely assigned based on the path of the tree node. Because 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 using the adaptive encoding information incrementally. In TREECHOP, exact-match and range queries can be carried out using a single scan through the compression stream.

The XQueC system uses a fragmentation and storage model for the compressed XML documents based on the separation structure and content within an XML document. In addition, it depends on a proper choice of how to group containers to ensure that containers belonging to the same group also appear together in query predicates. To perform the evaluation of a predicate within the compressed domain, it ensures that containers involved in the predicate belong to the same group and are compressed with an algorithm supporting that predicate in the compressed domain. The information about predicates is inferred using the available query workloads. XQueC exploits the query workload information to partition the containers into sets according to the source model and to properly assign the most suitable compression algorithm to each set. XQueC has also designed an algebra for evaluating XML queries. This algebra is exploited by a cost-based optimizer that can freely mix regular and compression-aware operators.

**Conclusion**

In this overview article, you looked at state-of-the-art XML compression techniques. The primary innovation in the XML compression mechanisms was presented in the first implementation in this domain by XMill. It introduced the idea of separating the structural part of the XML document from the data part to then group the related data items into homogenous containers that can be compressed separately. This separation improves the further steps of compressing these homogenous containers with general-purpose compressors or other compression mechanisms because they can detect the redundant data more easily.
Most of the remaining XML compressors simulated this idea in different ways. The compression time and decompression time metrics play a crucial role in differentiating between the different XML compression techniques. In principle, schema-dependent XML compressors are not preferable or commonly used in practice because the schema information of the XML documents is not always available and in the required format (DTD, XML Schema, RELaxNG). While queriable XML compressors are very important for many applications, no solid implementations for grammar-based XML compression techniques and queriable XML compressors are publicly available. These two areas provide many interesting avenues for further research and development.
Related topics

- **XML 1.0 Specification, W3C**: Visit this source for specific details about XML features.
- **The XML FAQ (Editor, Peter Flynn)**: Find answers to many common questions about XML in another excellent source of XML information.
- **Merging prediction by partial matching with structural contexts model. Proceedings of the Data Compression Conference** [Joaquin Adiego, Pablo de la Fuente, and Gonzalo Navarro; (DCC), Washington, DC, USA, 2004]: Consider the text structure of compressed structured documents and learn to improve compression ratios of the basic SCM technique.
- **XGRIND: A query-friendly XML compressor**, in: [Pankaj M. Tolani and Jayant R. Haritsa, Proceedings of the 18th International Conference on Data Engineering (ICDE), Washington, DC, USA, 2002]: Learn more about a compression tool that supports queries and reuse of the standard XML techniques.
- **Supporting efficient query processing on compressed XML files** [Yongjing Lin, Youtao Zhang, Quanzhong Li, and Jun Yang; Proceedings of the ACM Symposium on Applied Computing (SAC), New York, USA, 2005]: Read about an XML compression scheme based on the Seqitur compression algorithm.
- **XPRESS: A queriable compression for XML data** (Jun-Ki Min, Myung-Jae Park, and Chin-Wan Chung; Proceedings of the ACM SIGMOD International Conference on Management of Data, California, USA, 2003): Check out an XML compressor that supports direct and efficient evaluations of queries on compressed XML data based on reverse arithmetic encoding.
- **Compression and explanation using hierarchical grammars** [Craig G. Nevill-Manning and Ian H. Witten, Computer Journal 40(2), 1997]: Learn about hierarchical structure in sequences of discrete symbols and how SEQUITUR uses that information for data compression.
- **XQueC: Pushing queries to compressed XML data** [Andrei Arion, Angela Bonifati, Gianni Costa, Sandra D'Aguanno, Ioana Manolescu, and Andrea Pugliese; Proceedings of 29th International Conference on Very Large Data Bases (VLDB), Berlin, Germany, 2003]: Learn more about the XQueC system that compresses XML data and queries it as much as possible under its compressed form.
- The **gzip website**: Download and try gzip (GNU zip), a compression utility designed to replace compress.
- The **bzip2 website**: Download and try this freely available, high-quality data compressor.
- The **XMill project website**: Download and try a user-configurable XML compressor that separates structure, layout and data and distributes data elements into separate data streams.
- The **XMLPPM project website**: Download and try a data compression program that combines the PPM algorithm for text compression and tree-structured MHM data.
- The **RNgzip project website**: Download and try a compression technique based on the document type, expressed as a Relax NG schema.
• **XML area on developerWorks**: Find the resources you need to advance your skills in the XML arena, including DTDs, schemas, and XSLT. See the XML technical library for a wide range of technical articles and tips, tutorials, standards, and IBM Redbooks.

• **IBM trial software**: Build your next development project with software, available for download directly from developerWorks.

© Copyright IBM Corporation 2011

**Trademarks**
(www.ibm.com/developerworks/ibm/trademarks/)