Categorizing Efficient XML Compression Schemes

John N. Dyer, Department of Information Systems
College of Business Administration, Georgia Southern University,
P.O. Box 7998, Statesboro, GA 30459

Abstract

Web services are Extensible Markup Language (XML) applications mapped to programs, objects, databases, and comprehensive business functions. In essence, Web services transform XML documents into and out of information technology systems. As more businesses turn to web services data transfer, XML has become the language of web services. Unfortunately, the structure of XML results in extremely verbose documents, often 3 times larger than ordinary content files. As XML becomes more common through Web services applications, its large file sizes increasingly burden the systems that must utilize it. This paper provides a qualitative overview of existing and proposed schemes for efficient XML compression, proposes three categories for relating XML compression scheme efficiency for Web services, and makes recommendations relating to efficient XML compression based on the proposed categories of XML documents. The goal of this paper is to aid the practitioner and Web services manager in understanding the impact of XML document size on Web services, and to aid them in selecting the most appropriate schemes for applications of XML compression for Web services.

Keywords: Compression, Web services, XML

Introduction

XML is the foundation upon which Web services are built, and provides the description of data, as well as the storage and transmission format of data exchanged via Web services (Newcomer, 2002). XML is similar to Hypertext Markup Language (HTML), and well-formed XML documents can even be displayed in Web browsers. XML is gaining much acceptance by e-business and other web dependent enterprises as a method of data exchange, data sorting, and data archiving across different software applications and platforms. XML provides a home for many "niche" application areas that do not fit into the standard HTML data model (Cheney, 2001). If you go to any of several web sites exploring XML, you will probably run across an

exhaustive list of such applications, including e-business transactions, medical information, and user interface descriptions.

XML is also gaining momentum in many areas of the computer industry; for example, Microsoft has announced plans to base future software systems on XML (Bosworth, 1998). Additionally, Microsoft's XML online demo provided an early demo of how XML might eventually be implemented (Walsh, 1998). The demo profiles an art auction whereby all of the artwork, bids, and descriptions are downloaded and reside on the client. The server is then pinged by the client to see if any bids have been updated or if the end-user submits a bid for a specific picture. Concerning the popularity of XML, Bosak and Bray (1999) relate that XML is the ``next big thing'' after HTML.

The required structure of an XML document, including required tags, symbols, and attributes (all text), as well as the abundance of redundant data and white spaces, bloats the file size and hence impacts document processing and web transmission speeds. Walsh (1998) relates that Web designers are concerned with how users will react to additional XML data being sent over the Internet, while Tim Sloan (analyst at Aberdeen Group in Boston) related that there is the risk that XML will become overused, hence creating an overhead that is not valuable to the end-user. Walsh further stated, “given the current standards based on XML, there may also be a need to either heighten the compression in HTTP or add a new compression layer to address XML data.” Dodds (2000) related that he discovered that designer’s concerns over XML file size was behind several contentious design decisions and suggested that, based on XML structure, XML documents were a prime candidate for compression.

In spite of the growing popularity and use of XML, its greatest disadvantage is document size. XML's parent standard, Standardized General Markup Language (SGML), made many provisions for minimizing document size. As a result, SGML was rendered complex and difficult to implement, hence the provisions were omitted from XML. As evidence, the XML standard explicitly states that markup terseness “was not a design goal.” Consequently, XML is not a particularly efficient format for representing information since it is a text-based, human-readable, and metadata-encoded markup language that operates on the principle that the metadata that describes a message's meaning and context accompanies the content of the message.

Schmelzer (2002) related that XML document sizes can easily be ten to twenty times larger than an equivalent binary representation of the same information,

and most articles and studies (Bromberg, 2001, Mertz, 2001, 2003) related to XML document size estimate size to be at least three times larger than an ordinary text document. Regardless of the exact numbers, the point is that XML documents can be many times larger than equivalent non-standardized text or binary formats, even if compressed. Even though it is inefficient, XML's numerous advantages are increasing its use for ever broader and more mission-critical functions.

There is growing concern in the XML community, particularly for Web services applications, that inefficiency arising from document size will hinder adoption and use of XML, as well as Web services technologies. While XML's verbosity may be acceptable for situations with moderate transaction volumes, XML's processing overhead, storage requirement, and bandwidth consumption becomes quite problematic when transaction volumes are high. As a result, many companies are resorting to potentially dangerous tactics for squeezing every last drop of performance out of XML. Three common tactics include compressing XML, ignoring XML validity, and changing the parsing rules for XML (Schmelzer, 2002).

XML compression addresses some of the problems of Web services via XML by reducing the size of XML documents transferred between a server and client, thereby conserving bandwidth and reducing user perceived latency. Although there is a wide variety of potential hardware/software solutions to remedy XML's performance problems, many developers and researchers are resorting to a variety of tactics to improve the performance of XML processing and transmission. Many of these approaches simplify certain aspects of XML to reduce document size via compression, improve parser performance, and speed the mapping of XML document components to application objects (Schmelzer, 2002).

The current body of literature is quite sparse in regards to efficient XML compression. This paper provides an overview of existing and proposed schemes for compression of XML documents, proposes three categories for relating XML compression scheme efficiency (based on specific metrics), and makes recommendations relating to efficient XML compression based on the proposed categories of XML documents. The following sections more fully define XML, processing XML documents, consider related compression problems, introduce the basic concept of data compression, provide an overview of the traditional algorithms for XML compression as well as XML-conscious compression schemes, categorize compression schemes based on three proposed categories of XML documents, introduce recently proposed

compression schemes, and make recommendations as related above. The goal of this paper is to aid the practitioner and Web services manager in understanding the impact of XML document size on Web services, and to aid them in selecting the most appropriate schemes for applications of XML compression for Web services.

XML Defined

XML is a language for semi-structured data standardized by the World Wide Web Consortium (W3C), which has most likely become the de facto standard for web documents (Cannataro et al., 2001). XML is considered an "up-and-coming" standard for structured data files, drawing on considerable existing experience with HTML/CSS, but being much more general (Cheney, 2001). It is not a specific markup language like HTML, but instead is a meta-language for describing markup languages (like HTML) together with a strong standard for creating and parsing documents. XML allows building machine-readable documents that are naturally convertible in visualization formats; this is obtained by means of a complete separation among structure, content and style of documents (Cannataro et al., 2001). More specifically, XML is a standardized language that "describes a class of data objects called XML documents and partially describes the behavior of computer programs which process them" (W3C, 1998).

Superficially, XML documents looks a lot like HTML documents. XML provides the description, storage, and transmission format for data exchanged via Web services. It allows user-defined elements and attributes that independently define type and structure information for the data they carry, including the capability to model data and structure that are specific to a given software domain. The XML syntax specifies how data is generically represented, defines how and with what qualities of service the data is transmitted, and details how the services are published and discovered (Newcomer, 2002).

XML independently stores data values within descriptive element tags that are enclosed in angle brackets (< >) and have a start and an end. The end is marked with a slash (/). Elements can have one or more attributes associated with the element name, using a name/value pair for each attribute. XML documents must be well formed, resulting in XML being more restrictive than HTML. Specifically, elements must be nested, start tags must match corresponding end tags, and attribute names within start tags must be unique, among other things. XML documents can also include a document type definition (DTD). Validation, or checking that an XML document follows the rules of a DTD, ensures that only meaningful data reaches an application.

An XML schema (also an XML document) can be used to define validation, data typing, and document structure of the original XML document. Schemas separately define the types, structure, and semantic meaning to be applied to the data contained within the element tags. In essence, the schema is used to transform data into and out of XML format. XML schemas were developed to resolve some of the limitations and problems with DTDs, which themselves were developed to express a content model for XML documents, defining valid elements, attributes, and some ordering constraints. Although schemas can replace DTDs, validation of content is still often done via DTDs, especially when existing XML documents are transmitted via Web services. The interested reader is referred to other sources (Newcomer, 2002) for detailed information about XML, schemas, DTDs, parsing, processing, and transforming XML documents.

Processing XML Documents

Document Object Model (DOM) and Simple API for XML (SAX), both APIs, are models and programming libraries for parsing XML documents, either by creating an entire tree to be traversed or by reading and responding to XML elements one-by-one (Newcomer, 2002). Simply stated, DOM and SAX facilitate the parsing of XML documents.

The DOM API provides a generic object model to represent the structure of documents and a standard set of interfaces for traversing and manipulating the document. Most DOM implementations work in main memory, hence the DOM API allows multiple passes through the document. In essence, the DOM API treats the document as a memory-resident database that can be searched multiple times. With DOM, the parser itself does almost everything, including reading the XML document, creating a Java object model, converting textual XML information into a tree of nodes, and providing a reference to the Document object (Halloway, 2000). The DOM API is recommended if an XML document is a continuing source of data or is a document that will experience repeated interaction.

SAX is a standard interface for event-based XML parsing, designed to give programmers access to the information stored in XML documents using any programming language and corresponding parser (Megginson, 2000). SAX is currently one of the most popular APIs for manipulating XML documents. SAX (similar to the DOM) is designed specifically to allow programmers to access XML information without having to write a parser in their own particular programming language. When storing information in XML format, and using the SAX API, a program is free to use any parser it wishes. This is

possible because parser writers implement the SAX API using their preferred programming language. SAX (and DOM) APIs are both available for multiple languages (Java, Perl, C++, Python, etc.) (Megginson, 2000).

The SAX API works by firing callback events into the application as the document is parsed, element-by-element (Newcomer, 2002). The SAX approach uses less memory and is more efficient for messaging and transformation. The SAX approach is recommended if document parsing for only one reason, such as to map the XML document to a software program or database.

SAX can run much faster relative to DOM, for simple object models. In such cases, SAX is faster because it does not create a tree-based object model of the information (Armstrong, 2000). This speed advantage is counter-balanced by the need to write a document handler to interpret all the SAX events generated by the parser (Armstrong, 2000). The appropriateness of SAX depends largely on the nature of the underlying XML data.

XML parsers convert XML tags into nodes in different ways. The SAX parser forms elements one at a time, forgetting about a node once it is completed. The DOM parser builds a tree that stores information about the structure over time (Cagle, 2000). The SAX parser converts a tag into a node, but it does not keep the node in the memory once the node is finished, so the SAX parser requires very little memory and processor time to process content. However, the parser only knows about the current node, and its younger siblings or children, so it's essentially forward-read only (Cagle, 2000). DOM on the other hand, parses a node and stores it in a structural tree, so that the entire data structure is available while processing any node. This allows more complex processing, but requires keeping the entire structure in memory, which can be impractical for XML documents with thousands of nodes (Cagle, 2000).

The fact that SAX allows a programmer to define a custom document model means the programmer can decide to discard information that will not be needed later. This can result in reduced memory overhead, which is typically very high with DOM (a 100 megabyte XML file can take over 1 gigabyte of memory just to read the data into a libxml DOM tree (Veillard, 2000)). In summary, SAX is an event-based API, suitable for one-pass algorithms such as search tools and filters. DOM provides an interface to XML data stored in memory as trees, and is better suited to multi-pass algorithms.

Common Compression Schemes

There are three major approaches to lossless text compression: (1) Dictionary-based, (2) Block sorting-based, and (3) Symbol probability prediction-based (King, 2003). A Lossless data compression algorithm is one that, on decompression, can recreate the original data, bit-for-bit (Whatis.com, 2001). Most file compression schemes are based on dictionary algorithms of previously occurring phrases. These algorithms compress by substituting distance to the last occurrence and the length of the phrase. These algorithms are very fast, provide moderate compression ratios, and use only modest memory requirements. Block sorting-based algorithms perform block-sorting transforms wherein letters are grouped together, while the text remains the same size. The resulting transformed document is then compressed with a fast and simple coding technique that results in high compression ratios and moderate memory requirements. Probability based-prediction algorithms calculate the probability distribution for every symbol and then optimally encode them. These algorithms are typically slow and quite memory intensive.

Although there are many different lossless compression algorithms available for file compression, most are variations of two popular schemes: Huffman encoding and the Lempel-Ziv algorithm. Huffman encoding is a probability based-prediction algorithm, while Lempel-Ziv is a dictionary-based algorithm.

Huffman encoding works by assigning a binary code to each of the symbols (characters) in an input stream (file). This is accomplished by first building a binary tree of symbols based on their frequency of occurrence in a file. The assignment of binary codes to symbols is done in such a way that the most frequently occurring symbols are assigned the shortest binary codes and the least frequently occurring symbols assigned the longest codes. This in turn creates a smaller compressed file (Goebel, 2001).

The Lempel–Ziv algorithm, also known as LZ-77, exploits the redundant nature of data to provide compression. The algorithm utilizes what is referred to as a sliding window to keep track of the last *n* bytes of data seen. Each time a phrase is encountered that exists in the sliding window buffer, it is replaced with a pointer to the starting position of the previously occurring phrase in the sliding window along with the length of the phrase (Goebel, 2001).

The primary metric for data compression algorithms is the compression ratio, which refers to the ratio of the size of the original data to the size of the compressed data (Gzip, 2001). For example, if we had a 100-kilobyte file and were able to compress it down to only 20 kilobytes we would say the compression ratio is 5-to-1, or 80%. The contents of a file, particularly the redundancy and orderliness of the data, can strongly affect the compression ratio. Additionally, the speed of document compression and decompression should be considered, since speed directly impacts the overall efficiency of any given compression scheme.

XML-Conscious Compression Schemes

XML is stored in plain text files, so the most obvious approach to XML compression has been to use existing text compressors. The most commonly used compressors are Gzip (www.Gzip.org/) and Bzip2 (sourceware.cygnus.com/bzip2/index.html).

Gzip is based on Lempel-Ziv (Lempel and Ziv, 1997). Gzip works by checking if the current data has already been recorded in a temp buffer and if so, the current data can be recorded as a reference to the previous data. When similar data are compressed together, high compression ratios can be achieved. Since Gzip is a plain text compressor it cannot take advantage of the XML document structure, consequently, a compression tool that makes use of the special properties of XML documents can be expected to yield better performance. Unfortunately, none of the existing XML-conscious methods on the market outperform Gzip in all aspects. Some are slower than Gzip, while some have lower compression ratios (Gaily and Adler, 2003). Some advantages of using Gzip include the ability to adapt Gzip sources to perform in-memory compression, and Gzip is already supported in Web streaming via HTTP compression.

An alternative general compression method is Bzip2, which produces a better compression ratio than Gzip, but runs slower. Since Bzip2 is based on Huffman coding, it must determine the frequencies at which characters occur within the source text, then build a mapping between the Huffman code and characters according to the frequency of each character. This mapping is added on the top of the document and sent along to the client. Bzip2 provides a good compression ratio, but it requires knowing the statistics of the source text ahead of time, and the overhead mapping is added to the document (Seward, 2002).

Levene and Wood (2002) relate that the problems with compression using Gzip, Bzip2, and related methods are twofold: first, compression of elements or attributes may be limited by existing tools due to the long range dependencies between elements and between attributes, that is, the duplication is not necessary local, and second, to enhance compression, it may be useful to use different compression techniques on different components of XML. Levene and Wood (2002) suggest using two other XML compression systems, XMILL (Liefke and Suciu, 2000) and XMLPPM (Cheney, 2001). These systems are considered XML-conscious compression systems since the compression techniques take advantage of XML document structure. Other popular XML-conscious systems include XMLZip (XML Solutions, 2003), Millau (Sundaresan and Moussa, 2001), and XML-Xpress (ICT, 2003).

XMILL

XMILL is an XML compressor/decompressor that claims to achieve twice or better compression ratios than Gzip, with similar execution time for compression and decompression. XMILL is designed to take advantage of regularity in XML data to improve compression performance. The idea behind XMILL is to first parse the XML data with a SAX parser, then transform the XML into three components: (1) elements and attributes, (2) text, and (3) document structure, and then to pipe each of these components through existing text compressors. By this method, text within a certain set of element tags is placed into a container. Users with detailed XML knowledge can also define their own heuristics to improve performance, based on DTD conventions or XML-schema rules (Levene and Wood, 2002).

XMILL is based on the following 3 compression principles:

1. Separate structure from data, and compress them separately. Structure consists of tags and attributes that form the XML tree, and data consists of strings that make up element names and attribute values.
2. Group data items by meaning. Data is compressed according to container, to increase the likelihood of structural similarity.
3. Use different compressors for each container, since different containers may contain different types of data (names, numbers, web logs, etc.)

After the data is transformed, the result is compressed using a text-based compression program such as GZIP, and then stored in an output file.

XMLPPM
XMLPPM is a stream-oriented parser that requires setting handlers to deal with the structure that the parser discovers in the document. XMLPPM uses different text compressors with different XML components, that is, one model for element and attribute compression and another for text compression. Additionally, XMLPPM utilizes the hierarchical structure of XML documents to further compress documents.

XMLZip
XMLZip is a compressor and decompressor for XML documents written in Java and produces ordinary pkzip/WinZip zip files, based on the W3C DOM. XMLZip first parses XML data with a DOM parser, then breaks the structural tree into multiple components: a root component containing all data up to depth *d* from the root, and one component for each of the subtrees starting at depth *d*. The root component is then modified, then references to each subtree are added onto the root, and finally components are compressed using Java's built-in ZIP/DEFLATE library (Cheney, 2001).

XMLZip allows users to choose the depth at compression time, thus allowing users to select the DOM level at which to compress the XML files. This allows continued use of the DOM API without decreased performance. XMLZip only decompresses the portion of the XML tree that needs to be accessed, allowing applications to access data without uncompressing the whole file, thus reducing execution time, run-time space, and memory usage.

Millau
Millau was designed as a binary XML compression method that is schema aware, and has been shown (Sundaresan and Moussa, 2001) to exhibit superior compression over ZIP compression for XML documents less than 5k. Millau encoding is based on the Binary XML format from the Wireless Application Protocol (WAP) that losslessly reduces the size of XML documents. Millau improves on the compression performance of WBXML by using the structure and data types in XML documents, and also extends the format to make it more suitable for business-to-business applications.

Recall, one of the drawbacks of using traditional text-compression algorithms is that they perform character-based compression. A proposed XML encoding format from the WAP forum is based on a table for associating tokens with XML tags and attribute names, but does not compress character data, and lacks a method for building the association table. Millau extends further and supports compression of character data, and sets out a strategy for building the table.

Millau format also saves space tokens rather than strings (which can be arbitrarily long in XML). A custom parser for processing the format is implemented using DOM and SAX adapted to handle Millau streams, resulting in better performance. This layer of abstraction for Millau streams is significant, since it allows applications to access the data transparently through either the SAX or DOM API, making it easy to design applications based on Web standards.

XML-Xpress

XML-Xpress is a DTD/schema specific XML coder. The compression ratio of an XML file can be greatly improved when a known schema is used. When the schema is known, XML tags can be encoded very efficiency. Schemas also provide the data types of element data, thus allowing compression routines for specific data to be used, further improving the compression ratio.

XML-Xpress can parse and encode files one at a time, or if input data is known to arrive more slowly, the program can be configured to accept data as it is received (packet level compression). This prevents the performance-degrading latency caused by waiting for entire large files to arrive. XML-Xpress also supports concurrent compression, which uses the similarities between files when multiple files are compressed simultaneously.

The disadvantage of XML-Express is that it is a schema-specific encoder, and significant compression ratios are dependent on the presence of a known single schema. In the absence of such a schema, XML-Express resorts to using a general-purpose encoder, and the reported outstanding compression performance is lost.

Scheme Characteristics and Performance Evaluations

Little research has been conducted on the efficiency of XML compression and XML-conscious compression schemes, particularly geared to measuring and comparing/contrasting compression ratios, and compression/decompression times over the spectrum of schemes, as well as aptness for Web style services. Most research has focused on proposed alternatives to existing schemes (Cannataro et al. 2001, Cheney 2001, Mertz 2001, 2003, Sundaresan and Moussa 2001, Levene and Wood 2002, Neidermeier et al. 2002) typically measuring, and comparing/contrasting against Gzip, Bzip2, and XMILL. As a result, none of the existing XML-conscious methods on the market have been shown to outperform any other in all aspects. Some are slower/faster at

compression/decompression, while some have lower/higher compression ratios.

Sundarson and Moussa (2001) conducted experiments compressing various size documents with Gzip. The documents included XML formatted Web log access files and Shakespeare's play, Hamlet. The experiment purported to emulate documents such as XML database files that are rich in data redundancy. The results showed that Gzip achieved compression ratios between 96% (log document size = 244655 bytes, 42% content, 58% structure) and 72% (Hamlet document size = 288735 bytes, 60% content, 40% structure). Compression/decompression times for the documents were quite fast: 3.33-msec/3.30-msec for the log document, and 77-msec/70-msec for the Hamlet document. Gzip is generally considered an efficient compression tool but is often lacking in providing the best compression ratio since the XML document structure inhibits its performance.

Bromberg (2001) reported that he had consistently achieved 80% - 95% compression ratios using Bzip2 to compress XML documents, but compression speeds were long for tag heavy documents (4 seconds for 110K document), in spite of its more rapid decompression speed (170-msec). Cheney (2001) reported that Bzip2's compression ratio was 20% - 30% better than Gzip, and about the same as XMLPPM. Disadvantages of Bzip2 are that it exhibits slow compression speed and the compression is off-line. According to Cheney, "Off-line compression is undesirable because it forces a long wait before document parsing and processing can begin."

Cheney (Cheney, 2001) reported that XMILL compression ratios using Gzip are only about 10% better than when using Gzip alone, and that using compressors other than Gzip compress 5% - 10% worse than the original document. Liefke and Suciu (2000) reported that XMILL provides compression ratios two or more times that of Gzip, at about the same speed. There are several known limitations to XMILL. First, XMILL is not designed to work with a query processor, hence integration of XMILL's decompressor and a DB query engine. Second, XMILL only achieves greater compression ratios than traditional text-compression methods if dealing with data larger than approximately 20,000 bytes (Liefke and Suciu, 2000). Last, since XMILL does not support online encoding, it might be a disadvantage in some online transaction, data exchange applications. Cheney (2001) also related that XMILL always requires user assistance to achieve the best compression. XMILL is most efficient for large files, and allows the user to choose a compressor for data containers, but does not support query processors. The

results of this study did not find measures relating to XMILL compression/decompression speeds.

XMLZip allows users to access specific portions of files, but does not outperform traditional Gzip compression ratios. XMLZip's compression ratio is not as good as Gzip when measured over an entire document. An advantage of XMLZip is that it reduces the size of XML file while maintaining the accessibility of the DOM API. Additionally, XMLZip is capable of selective compression and decompression of the documents, allowing users to determine the DOM level at compression time. However, XMLZip can only be run on entire XML file objects, and is thus offline-only. Lastly, Sundarson and Moussa (2001) reported that the main limitation of XMLZip is that "it consumes large memory resources and runs out of memory for large documents." The results of this study did not find measures relating to XMLZip compression ratios, or compression/decompression speeds.

Girardot and Sundaresan (2001) reported that Millau's token parsing is faster than XMILL and Gzip, has the highest compression ratio for small files, but does not outperform Gzip for files over 5k. Although traditional text-compression algorithms outperform on large XML files, Millau achieves better compression for file sizes between 0-5k, which the above authors claim is the typical file size for e-Business transactions, such as orders, bill payments, etc. In experiments (Girardot and Sundaresan, 2001), Millau's compression ratio for the entire XML document was about 81% versus 87% when compressed with Gzip. When applied to the markup portions of the document, the compression ratios were approximately 82% versus 91%, and when applied to the data only, equal compression ratios of 79.5% resulted. Sundarson and Moussa (2001) conducted experiments compressing various size documents with Millau. The documents included the same as their tests on Gzip. The results showed that Millau achieved compression ratios between 96% (log document size = 244655 bytes, 42% content, 58% structure) and 75% (Hamlet document size = 288735 bytes, 60% content, 40% structure). Compression/decompression times for the documents were relatively slow when compared to Gzip: 1170-msec/1075-msec for the log document, and 1511-msec/861-msec for the Hamlet document. Further experiments show that parsing a Millau stream (employing tokens) is five times faster than parsing a straight ASCII XML stream, because the Millau stream requires only binary token comparisons, versus the string comparisons for ASCII streams. Additionally, because the structure and content are separate, only the structure stream needs to be parsed, further reducing run-time.

Intelligent Compression Technologies (2003) claims that XML-Xpress has the highest compression ratios, and the fastest execution times, but requires that files adhere to a specific schema. Otherwise the gains are erased. XML-Xpress also supports on-line encoding and concurrency compression. On average XML-Xpress achieves 81% higher compression rates than XMILL, and runs on average 55% faster than XMILL (ICT, 2003). The speed of the compression may be slower than Gzip, since this method involves added DOM parsing. However, the ratio should be higher than Gzip because tag names and attribute names are the major contributor towards the huge size of XML files, and are all replaced by tokens, resulting in significant space savings. XML-Xpress is also claimed to reduce file sizes on the order of up to 34-to-1 at throughputs up to 9 mb/sec on a test database.

XML Compression Categorized

Until recently, general-purpose text compressors were the primary tools for XML compression. Unfortunately, these tools were not designed with regard to the Web environment or Web services applications. The environment of the Web services applications (client-server models, bandwidth considerations, middleware programs, etc) requires attention not only to file size, but also to processor overhead, execution speed, transmission speed, middleware flexibility, and data flow/streaming considerations, among others. Users must carefully select an XML compression scheme depending on the type of data and applications involved, and whether file size, execution speed, or data flow flexibility is the highest priority. Unfortunately, none of the current XML compression schemes facilitate a compress-before-transmission/decompress-on-receipt framework that is transparent to users (Dodds, 2000) and expected in many Web services applications. In order to qualify an XML compression system for appropriate use, the XML document in question should be categorized into one of three categories based on the data within and the accompanying schemes/DTDs. This section attempts to provide a categorization of each of the XML compression and XML-conscious compression schemes discussed above.

The first category, *Client Priority*, is for documents that are to be compressed once (server side), and then decompressed many times by different users in different places (client side). For this type of document, the compression ratio and speed of decompression is more important for the client than the compression speed and processor overhead of the server. The second category, *Equal Priority*, is for smaller documents that are created once, sent once, and received once (such as inter-database communication documents).

This type of document requires speedy compression, decompression, and transmission, while the compression ratio is of lesser importance. The third category, *Server Priority*, is for large documents for which storage capacity is a constraining factor, while the speed of compression and decompression is secondary. This type of document requires an emphasis on the compression ratio.

Based on the findings in this study, each of the compression schemes discussed above (with the exception of XMLPPM) is categorized below (Table 1). The goal of this categorization is to provide a summary of characteristics of XML compression schemes to aid the practitioner in selecting the appropriate scheme for application. Each scheme is categorized according to characteristics relating to compression ration, compression speed, decompression speed, and processing characteristic (on-line versus off-line). Note, in Table 1 some compression schemes are relevant to more than one category.

Table 1. Compression Scheme Categories

Scheme	Ratio	Speed	Decompression Speed	Off-Line vs. Online	Compression Category
Gzip	[1]Moderate [2]High	Fast	Fast	On-line	Client Priority Server Priority
Bzip2	High	Slow	Slow	Off-line	Server Priority
XMILL	[3]Moderate [4]High	Fast	Fast	Offline	[4]Client Priority Equal Priority [4]Server Priority
XMLZip	Moderate	[7]Variable	[7]Variable	Off-line	Equal Priority
Millau	[5]Moderate [6]High	Slow	Slow	Off-line	[5]Server Priority
XML-Xpress	High	Moderate	Fast	On-line	Client Priority Equal Priority Server Priority

Notes: [1] small documents, [2] large documents, [3] documents smaller than 20k, [4] documents larger than 20k, [5] document smaller than 5k, [6] documents larger than 5k, [7]

Other Recently Proposed XML Compression Schemes

Most current research regarding XML compression relates to proposed alternatives to the traditional XML compression and existing XML-conscious compression schemes. Cheney (2001) describes two proposed alternatives: Encoded SAX (ESAX) and Multiplexed Hierarchical Modeling (MHM). Cannatoro et al. (2001) describe an alternative based on data restructuring and compression, called semantic lossy compression (SLC). Mertz (2001, 2003) described an alternative based on block-level compression. Levene and Wood (2002) describe an algorithm to compress XML documents that are valid with respect to a given DTD, using the DTD to encode the structure of the data. Similarly, Niedermeier et al. (2002) describe a scheme based on the binary format for XML data, using a context sensitive approach that builds on the knowledge of the standardized schema definition at the encoder and decoder. These proposed alternatives are briefly discussed below.

According to Cheney, the idea behind ESAX is to leverage the work a SAX parser does by encoding the sequence of certain parsing events. A decoder can decode these events, and reconstitute an XML document equivalent to the original. A single byte event encoding was used to encode element start tags, end tags, and attribute names, and to indicate events such as "begin/end characters", "begin/end comment", and so on. The encoder and decoder maintain consistent symbol tables such that when a new symbol is encountered, the encoder sends the symbol name and the decoder enters it into the table. The encoding was implemented using *Expat* XML parser, version 1.95. The author concluded, through experimentation, that ESAX speeds up and improves compression for all compressors, and compresses 2% - 4% better than Bzip2 when applied to text XML. ESAX also facilitates incremental transmission.

MHM, also resulting from the work by Cheney, is based on the SAX encoding related to ESAX and on PPM modeling. Refer to the author's study (Cheney, 2001) for details regarding PPM. The MHM technique employs two basic ideas: multiplexing several text compression models based on XML's syntactic structure (different models based on structure, attributes, etc.), and injecting hierarchical element structure symbols into the multiplexed models. The author relates that multiplexing enables more efficient hierarchical structure modeling, while model multiplexing breaks existing cross-class sequential dependencies. The idea is that if the dependencies can be restored, then prediction can be improved. A common case for these dependencies is for the enclosing element tag to be strongly correlated with enclosed data. According to Cheney, "MHM exploits this by injecting the enclosing tag symbols into the element, attribute, or

string model immediately before an element, attribute, or string is encoded." He further related that "Injecting" a symbol means "telling the model that it has been seen but not explicitly encoding or decoding it." Although several models were built, the author concluded that MHM compressed text XML data about 5% better and structured data about 10% - 25% better than the "best" existing method. Unfortunately, MHM was found to be very slow.

Cannatoro et al. (2001) relate that the idea behind SLC is to process the XML document (both data and structure) in such a way that elements can be regarded as tuples of a relation, to single out a number of dimensions and measures and provide a multidimensional representation that will be structured as a datacube, with aggregate data on suitable dimension levels. As a result, the document is reorganized according to some aggregation functions, resulting in a synthetic version of the original document. The authors also use lossless compression techniques for the documents markup structure and both lossy and lossless techniques for the data. Refer to the author's study for details regarding both the structural and content compression. The authors found (based on experimental documents) that structural compression ranged from about 79% to over 99% (based on number of structural elements), while content compression ranged from about 90% to over 96%, and outperformed both XMILL and Gzip. The authors further related that the scheme, based on document restructuring, makes sense in particular when *database-like* rather than *narrative* documents are considered.

Mertz's (2003) alternative is based on document transformation via blocking (grouping close together) relatively homogenous sets, like tags, attributes, and element bodies of different types. In essence, the transformed document contains the same information as the original document, but is structured in a more compression friendly style. Each block in the transformed document is then compressed by conventional means (Gzip, Bzip2, etc.), transmitted, decompressed, then reconstructed in a serial fashion. Based on experimentation, the author concluded that for small block sizes (less than 10k) compression is worse than using conventional file level compression; for block sizes around 10k, block-level compression appears adequately good; and for block sizes of 100k and greater, block-level compression is close to and sometimes better than file-level compression techniques.

Levene and Wood (2002) provide an algorithm to compress XML based on the knowledge encapsulated in the DTD. The method encodes information that is present in the XML document but not in it's DTD. The authors relate that the compression of the document contains three elements: (1) the DTD, (2) the

encoding of the document's structure, given the DTD, and (3) the textual data contained in the document, given the DTD. The outputs of these three elements can be compressed further by piping them through standard text compression tools. No experiments were related in the paper, hence no measures or comparisons for the scheme were provided.

Niedermeier et al. (2002), while working on the MPEG-7 standard, developed a binary format coding algorithm with special features for encoding XML data. The authors describe a schema-aware approach that exploits the knowledge of the standardized MPEG-7 syntax definition of the encoded XML document on the encoder and decoder side. Refer to the study (Neidermeier et al., 2002) for details regarding the MPEG-7 tool for compressing and streaming of XML data. The authors concluded, via experimentation, that their approach provides a good compression ratio, up to 98% for document structure.

Conclusions & Suggestions for Future Work

This paper provided a qualitative overview of existing and proposed schemes for efficient XML compression and made recommendations relating to efficient XML compression schemes based on three proposed categories of XML documents. These categories were defined as *Client Priority*, *Equal Priority*, and *Server Priority*. Existing studies of traditional compression and XML-conscious compression schemes were analyzed to determine which of the three categories each scheme was most suited for. The goal of categorization was to provide a summary of characteristics of XML compression schemes to aid the practitioner in selecting the appropriate scheme for application. It is concluded that there is not one "best" scheme for XML compression, but instead, each scheme should be considered in regards to the practitioner's desire for compression ratio, compression and decompression speed, and suitability to online/off-line processing.

Little attention has been given to measuring, comparing, and contrasting all traditional compression and current XML-conscious compression schemes across the board in regards to various metrics and XML document structures/content. Most studies have very limited measures and comparisons, wherein most measures provided ignore one or more major metrics, and most comparisons are limited to only a few schemes. Other ideas for future studies include analyzing all schemes using a variety of metrics, over a broad array of XML document structures, content types, file types, and file sizes.

References

ARMSTRONG, E. (2000) "Working with XML: Serial Access with SAX," from java.sun.com, July 13, 2000, http://java.sun.com/xml/jaxp-docs-1.0.1/docs/tutorial/sax/index.html

BOSAK, J. AND BRAY, T. (1999) "XML and the Second-Generation Web," *Scientific American*, May 1999.

BOSWORTH, A. (1998) "Microsoft's Vision for XML," in *SGML/XML Europe*, http://www.oasis-open.org/cover/bosworthXML98.html.

BROMBERG, P. (2001) "XML Data Compression / Decompression Over the Wire," from www.eggheadcafe.com, http://www.eggheadcafe.com/articles/20010604.asp

CAGLE, K. (2000) "A Tale of Two Parsers," from www.webtechniques.com, http://www.webtechniques.com/archives/2000/07/progrevu

CANNATARO, M., G. CARELLI, A. PUGLIESE, AND D. SACCA (2001) "Semantic Lossy Compression of XML Data," in *Proceedings of the 8th International Workshop on Knowledge Representation meets Databases*.

CHENEY, J. (2001) "Compressing XML with Multiplexed Hierarchical PPM Models," in *Proceedings of the IEEE Data Compression Conference*, pp. 163-172.

DODDS, L. (2000) "Good Things Come in Small Packages," from XML.com, March 23, 2000, http://www.xml.com/pub/2000/03/22/deviant/index.html

GAILY, J. AND ADLER, A. (2003) "GZIP," from Gzip.org, http://www.Gzip.org

GIRARDOT, M. AND SUNDARESAN, N. (2001) "Millau: an encoding format for efficient representation and exchange of XML over the Web," in *Proceedings of the 9th International World Wide Web Conference*.

GOEBEL, G. (2001) "Introduction / Lossless Data Compression," from Vectorsite.net, http://www.vectorsite.net/ttdcmp1.html.

.

GZIP (2001) “Gzip Manual” from Linuxprinting.org, http://www.linuxprinting.org/man/Gzip.1.html.
HALLOWAY, S. (2000) “Java, SAX & DOM: Technical Tips,” *Sun Microsystems*. June 27, http://developer.java.sun.com/developer/TechTips/2000/tt0627.html

ICT (2003) “Product Overview – Multi-filter Compression System,” white paper, from XpressFiles, http://www.ictcompress.com/XpressFilesWhitePaper.pdf

KING, A. (2003) “*Speed Up Your Site – Web Site Optimization*”, New Riders Publishing, Indianapolis, Indiana.

LEMPEL, A. AND ZIV, J. (1997) "A universal algorithm for sequential data compression," *IEEE Transaction on Information Theory*, 23(3): 337-343, May.

LEVENE, M. AND WOOD, P. (2002) “XML Structure Compression,” http://web.dcs.bbk.ac.uk/~mark/download/compress.pdf

LIEFKE, H. AND SUCIU D. (2000) “XMILL: An efficient compressor for XML data,” in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 153–164, Dallas, Texas.

MEGGINSON D. (2000) “SAX: History and Contributors,” from *Megginson Technologies*, http://www.megginson.com/SAX/index.html

MERTZ, D. (2001) “Exploring the Entropy of Documents,” *Xml Matters: Xml and Compression*, from *IBM*, http://www-106.ibm.com/developerworks/xml/library/x-matters13.html

MERTZ, D. (2003) “Compression and Streaming of XML Documents,” from *Intel Corporation*, http://cedar.intel.com/cgi-bin/ids.dll/content/content.jsp?cntKey=Generic+Editorial::xml_comp&cntType=IDS_EDITORIAL

NEIDERMEIER, U., J. HEUER, A. HUTTER, W. STECHELE AND A. KAUP (2002) “An MPEG-7 Tool for Compression and Streaming of XML Data,” in *Proceedings of IEEE International Conference on Multimedia and Expo*, Lausanne, Switzerland, August 26-29, 2002, pp. 521-524.

NEWCOMER, E. (2002) "*Understanding Web Services – XML, WSDL, SOAP, and UDDI*," Addison Wesley, Indianapolis, Indiana.

SCHMELZER, R. (2002) "Breaking XML to Optimize Performance," *ZapFlash e-newsletter*, October7, 2002, http://www.zapthink.com/flashes/10072002Flash.html
SEWARD, J. (2002) "*Bzip2*," from Redhat.com, http://sources.redhat.com/bzip2

SUNDARESAN, N. AND MOUSSA, R. (2001) "Algorithms and Programming Models for Efficient Representation of XML for Internet Applications," in *Proceedings of the World Wide Web Conference*, May 2-5, 2001, Hong Kong.

VEILLARD, D. (2000) "XML: Still validating while using SAX Interface?" from XMLSoft.Org, in *The XML C Library for Gnome*, October 15, 2000, http://xmlsoft.org

WALSH, J. (1998) "Web Designers Eye XML Data Compression," *Info World* [*Electric*], Vol. 20, Issue 18, May 4, 1998, http://www.infoworld.com/cgi-bin/displayStory.pl?980430.wcxml.htm

WHATIS.COM (2001) "Lossless and Lossy Compression," from *Whatis.com*, http://whatis.techtarget.com/definition/0,,sid9_gci214453,00.html. 2001.

WORLD WIDE WEB CONSORTIUM (W3C) (1998). "*Extensible Markup Language (XML) 1.0*," Second Edition, http://www.w3.org/TR/2000/ REC-xml-20001006.

XML SOLUTIONS (2003) "*XMLZip*," http://www.xmls.com/resources/xmlzip.xml