Preprocessing of XML file via dictionary method for faster data compression and decompression.

Mrs. Leena.K.Gautam[1]

Lecturer

Prof.V.S.Gulhane[2]

Asst. Professor
Department of Computer Science

Sipna's college of Engg. And Technology, Amravati, Maharashtra, INDIA

Abstract: - **compression algorithms reduce the redundancy in data representation to decrease the storage required for that data. Data compression offers an attractive approach to reducing communication costs by using available bandwidth effectively. Dictionary-based code compression techniques are popular as they offer both good compression ratio and fast decompression scheme. XML is a popular self describing meta-language in widespread use across a variety of application domains. Being self describing grants XML its great flexibility and wide acceptance but on the other hand it is the cause of its main drawback that of being huge in size. The huge document size means that the amount of information that has to be transmitted, processed, stored, and queried is often larger than that of other data formats. The basic strategy proposed in this paper is to preprocess the XML document and transform it into some intermediate form by using an dictionary based approach which can then compressed by BWT (transforms a block of data into a format that is extremely well suited for compression.) With better efficiency and which exploits the natural redundancy of the language in making the transformation.**

Keywords: - Compression, decompression, dictionary based compression and decompression, MPM, BWT

I. INTRODUCTION

A. Compression

Data compression, in context is the science (the art) to represent information in a compact form. It is the process of converting an input data stream (the source stream or the original raw data) into another data stream (the output, or the compressed, stream) that has a smaller size [5]. Data compression is popular for two reasons: (1) People like to accumulate data and hate to throw anything away. No matter how big a storage device one has, sooner or later it is going to overflow. Data compression seems useful because it delays this inevitability. (2) People hate to wait a long time for data transfers. When sitting at the computer, waiting for a Web page to come in or for a file to download, we naturally feel that anything longer than a few seconds is a long time to wait [11]. There are two major families of compression techniques when considering the possibility of reconstructing exactly the original source. They are called lossless and lossy compression. Certain compression methods are lossy. They achieve better compression by losing some information. When the compressed stream is decompressed, the result is not identical to the original data stream. Such a method makes sense especially in compressing images, movies, or sounds. If the loss of data is small, we may not be able to tell the difference. In contrast, text files, especially files containing computer programs, may become worthless if even one bit gets modified. Such files should be compressed only by a lossless compression method. There are many known methods for data compression. They are based on different ideas, are suitable for different types of data, and produce different results, but they are all based on the same principle, namely they compress data by removing redundancy from the original data in the source file.

The e**X**tensible **M**arkup **L**anguage (**XML**) has been acknowledged to be 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. Due to the simplicity of its basic concepts and underlying theories, XML has been used in solving numerous problems such as providing neutral data representation between completely different architectures,

bridging the gap between software systems with minimal effort and storing large volumes of semi-structured data. XML is often referred to as *self-describing data* because it is designed in a way that the schema is repeated for each record in the document. On one hand, this self-describing feature provides XML with immense flexibility but on the other hand, it also introduces the main problem of *verbosity* of XML documents which results in huge document sizes. This huge size lead to the fact that the amount of information that has to be transmitted, processed, stored, and queried is often larger than that of other data formats. The usage of XML compression tools has many advantages such as: reducing the network bandwidth required for data exchange, reducing the disk space required for storage and minimizing the main memory requirements of processing and querying XML documents.[3]The XML document is first compressed by a dictionary based approach and then the entire document is decompressed when required.

B. Decompression

Any compression algorithm will not work unless a means of decompression is also provided due to the nature of data compression. When compression techniques are discussed in general, the word compression alone actually implies the context of both compression and decompression. In many practical cases, the efficiency of the decompression algorithm is of more concern than that of the compression algorithm. For example, movies, photos, and audio data are often compressed once by the artist and then the same version of the compressed files is decompressed many times by millions of viewers or listeners [5].

II. BACKGROUND

A. Textual Data Compression Methods

Various sophisticated algorithms have been proposed for lossless compression. A very promising development in the field of lossless data compression is the Burrows-Wheeler Compression Algorithm (BWCA), introduced in 1994 by Michael Burrows and David Wheeler. The algorithm received considerable attention since of its Lempel-Ziv like execution speed and its compression performance close to state-of-the-art PPM algorithms [1]. While these strategies differ in the details, at a higher level it is possible to distinguish between *statistical* and *dictionary-based* approaches. *Statistical* approaches work by generating a variable-length encoding scheme based on the frequencies with which symbols appear in the input data stream. For example, in a typical sequence of English text the letter `e' tends to appear more frequently than any other letter, so the variable length code would assign the shortest possible encoding to `e'. Conversely, the letter `z' would be assigned one of the longer encodings, since it occurs rarely. Examples of this approach include Huffman coding and arithmetic coding. *Dictionary-based* approaches work by building up a table (or \dictionary") of patterns as they are encountered within the input data stream. Compression results by replacing subsequent occurrences of a pattern with a pointer to the corresponding table entry. Dictionary-based approaches are usually very fast, since the process of building the table and generating the encoding can be completed during a single pass through the source stream [2]. Various preprocessing method is performed on the source text before applying an existing compression algorithm. The transformation is designed to make it easier to compress the source file. A number of strategies for achieving this have been proposed in recent years. XMill represents the first published work on XML-conscious compression applies three core principles in its compression routine: Separate the XML structure of a document from its data, Group related data items in a single container and then Apply semantic compressor(s) to each container. An alternate version of XMill called XBMILL uses bzip2 in place of gzip during container compression. Second XML compressor is XGRIND. Throughout its compression process, XGRIND retains the original structure of the XML document.

Compression in XGRIND proceeds as follows: As a first step it performs structural compression .In a second step it Perform enumerated-type attribute value compression and finally perform general element/attribute value compression.Several other XML-conscious compression strategies have been developed, including XPRESS, Millau, and XML-XPress XComp [3].

III. PROPOSED WORK AND OBJECTIVES

The goal of this project is to develop new transformations for lossless text compression to incorporate fast, secure and to achieve a good compression ratio in transmissions .The approach consists of exploiting the natural redundancy of the English language by encoding text into some intermediate form before applying the backend compression algorithm which increases the context for compression. The encoding scheme uses dictionaries to co-relate words in the file and the transformed words. At the receiver end the file is decompress by using the same backend algorithm and the intermediate form is converted to XML file by the use of dictionary.

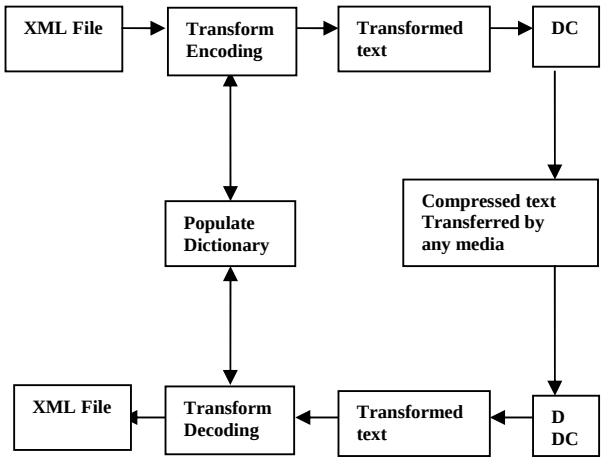

Fig: 1 Block diagram

To demonstrate the complete process a simple Xml file is used as an example1.

<? xml version=”1.0” encoding=”UTF-8”>

<class name=”cmps”>

<instructor>PRD</instructor>

<students>

<student id=”10”>Yatharth</student>

<student id=”11”>Parth</student>

</students>

</class>

A. Compression Strategy

The compression strategy is broken down into the following four steps.

Step1: Separate structure from content

The first phase of the compression process separates the XML structure of the input document (its tags and attribute names) from the content (attribute values and data values). A SAX parser is used to perform a single-pass parsing of the XML document. During the compression process, the following data structures are manipulated by the SAX parser.

Elements table and attributes table: the elements table is a hash table with entries corresponding to the XML elements encountered so far. Each entry is keyed by a byte value within the range [1, 124], and stores an associated data structure containing the qualified name of the element, along with a list of the data values belonging to encountered instances of this element. Key values are assigned in increasing order. The attributes table performs the same purpose for the XML attributes encountered within the input XML file; attribute key values fall within the range [132, 254].

Path stack : this is a stack used to keep track of which element or attribute is currently being processed; it also maintains proper ordering and nesting of elements. *DATA dictionary, comments dictionary, processing instruction dictionary, doc-type dictionary*: these are list structures used to record the contents of CDATA sections, comments, processing instructions, and the DOCTYPE declaration, respectively.

Structure string: a list of byte values which maintains the structuring of the source XML document. As each token is encountered by the SAX parser, an appropriate course of action is taken based on the token type.

Start tag for an XML element: A lookup is performed on the elements table for the element name. If the element name does not currently exist in the table, a new entry is added using the next available byte value. This key value is then appended to the structure string and pushed onto the path stack. Each attribute contained in the current element is processed in a similar way, by updating the contents of the attributes table.

End tag of an XML element: the byte value 130 is appended to the structure string and the top value is popped off the path stack. *Data value*: the byte value 128 is appended to the structure string, and a string containing the data value is added to the data value dictionary for the active element or attribute (the appropriate dictionary is determined by seeking to see the key of the top element on the path stack).*Comment*: a new entry is created in the comments list consisting of the text between the delimiting <!-- and --> sequences. The byte value 126 is appended to the structure string.*CDATA section*: a new entry is created in the CDATA dictionary, consisting of the full text between the delimiting <![CDATA[and]]> sequences. The byte value 127 is appended to the structure string. *Processing instruction*: a new entry is created in the processing instruction dictionary consisting of the full text between the delimiting <? and ?> sequences. The byte value 125 is appended to the structure string. *DOCTYPE declaration*: all text enclosed by the delimiting <!DOCTYPE and > sequences (including any internal DTD information) is stored in the doctype dictionary, and the byte value 129 is appended to the structure string[2].

Step:2. Apply MPM algorithm to the structure string

In this step, the MPM algorithm [MPMalgo] is applied to the structure string. This serves to condense repetitions within the XML structure. The MPM code is then written to the output file. Continuing with the same example, the following MPM code is produced:

$u0$ = (1 132 2 128 130 3 4 133 128 130 4 133 128 130 130 130)

$u1$ = (1 132 2 128 130 3 4 133, 128 130 4 133 128 130 130 130)

$u2$ = (1 132 2 128, 130 3 4 133, 128 130 4 133, 128 130 130 130)

$u3$ = (1 132, 2 128, 130 3, 4 133, 128 130, 4 133, 128 130, 130 130)

$S0(x)$ = (1 132 2 128 130 3 4 133, 128 130 4 133 128 130 130 130)

$S1(x)$ = (1 132 2 128, 130 3 4 133, 128 130 4 133, 128 130 130 130)

$S2(x)$ = (1 132, 2 128, 130 3, 4 133, 128 130, 4 133, 128 130, 130 130)

$S3(x)$ = (1, 132, 2, 128, 130, 3, 4, 133, 128, 130, 130, 130)

$T0$ = ($t0$; $t1$)

$T1$ = ($t0$; $t1$; $t2$; $t3$)

$T2$ = ($t0$; $t1$; $t2$; $t3$; $t4$; $t3$; $t4$; $t5$)

$T3$ = (1; 132; 2; 128; 130; 3; 4; 133; 128; 130; 130; 130)

Step:3 Compress data value dictionaries

The contents of the data value dictionaries associated with each element and attribute, along with those of the comment, CDATA, processing instruction, and DOCTYPE dictionaries, are compressed using backend algorithm (Burrows-Wheeler block-sorting algorithm) [5].The lossless Burrows-Wheeler compression algorithm has received considerable attention over recent years for both its simplicity and effectiveness. It is based on a permutation of the input Sequence the Burrows-Wheeler transformation which groups' symbols with a similar context close together. The BWT is performed on an entire block of data at once. Most of today's familiar lossless compression algorithms operate in streaming mode, reading a single byte or a few bytes at a time. But with this new transform, we want to operate on the largest chunks of data possible. Since the BWT operates on data in memory, you may encounter files too big to process in one fell swoop. In these cases, the file must be split up and processed a block at a time. A typical scheme of the Burrows-Wheeler Compression Algorithm (BWCA) is presented and consists of four stages.

I/p->BWT→MTF->RLE->Huffman->O/p

Each stage is a block transformation of the input buffer data and forwards the output buffer data to the next stage. The stages are processed sequentially from left to right for compression

B. Decompression Strategy

The first stage of the decompression process reverses the effects of the MPM and Burrows- Wheeler algorithms to obtain the original structure string and data dictionaries. Sub-

Sequently, the original XML file can be reconstructed by the decoder by conducting a single pass through the reconstituted structure string. Throughout the decompression process, the name of the last visited element is maintained using a stack, *S*.A specific set of actions is taken, depending on the type of the token currently being read from the structure string. Each time an *element* is encountered, a lookup is performed in the elements table for the corresponding element name. Then, the string \<elementname> " is written to the decompressed file, and the name of the element is pushed onto *S*. If the next symbol in the structure string is an *attribute*, then the attributes table is searched to determine the matching attribute name, which is written to the output file. Then, the first value contained in the data dictionary for that attribute is removed and emitted to the output file. The start tag of the element is terminated by emitting a `>' character. If a *data value* token is read from the structure string, then the first entry is removed from the data dictionary corresponding to the element or attribute that is currently atop *S*.A similar strategy is used whenever a *comment*, *CDATA*, *processing instruction*, or *DOCTYPE* token is read from the structure string, so that the first data value stored in the corresponding data dictionary is removed and then emitted to the output file. If an *end element* symbol is read from the structure string, then the topmost elementname is popped from *S*, and the string \</elementname>" is output.

IV. CONCLUSION

A Dictionary-based compression techniques use no statistical models. They focus on the memory on the strings already seen. Preprocessing is achieved by using a XML compressor which uses that uses a grammar-based strategy for structural compression. The compression algorithm and the decompression algorithm build an identical dictionary independently. The advantage of this is that the compression algorithm does not have to pass the dictionary to the decompressor.This scheme is an excellent improvement in data compression and added levels of security over the existing methods. This method will maintain the compression ratio with a good reconstruction at the receiver's end.

REFERENCES

[1] V.K. Govindan1 B.S. Shajee mohan2.,IDBE - An Intelligent Dictionary Based Encoding Algorithm for Text Data Compression for High Speed Data Transmission Over Internet.

[2] Gregory Leighton, James Diamond, Tomasz MÄuldner, A Grammar-based Approach for Compressing XML, Jodrey School of Computer Science, Acadia University, Wolfville, NS, Canada TR-2005-004 August, 2005

[3] Sherif Sakr 1 XML compression techniques: A survey and comparison, *National ICT Australia (NICTA), 223 Anzac Parade, NSW 2052, Sydney, Australia*

[4] Burrows, M, Wheeler, D. A Block-Sorting Lossless Data Compression Algorithm. Technical report, Digital Equipment Corporation, Palo Alto, California, 1994, URL (March 2006): http://citeseer.ist.psu.edu/76182.html..

[5] E-book on Fundamentals of data compression by ida mengui pu.

[6] Isal, R, Moffat, A, Ngai, A. Enhanced Word-Based Block-Sorting Text Compression. In Proceedings of the twenty-fifth Australasian conference on Computer science, Volume 4, January 2002, 129 138, 2002.

[7] [KrMu96] H. Kruse and A. Mukherjee. .Data Compression Using Text Encryption. *Proc. Data Compression Conference*, 1997, IEEE Computer Society Press,1997.

[8] [KrMu97] H. Kruse and A. Mukherjee. .Preprocessing Text to Improve Compression, IEEE Computer Society Press,1997, p. 556.

[9] [Welc84] T. Welch, .A Technique for High-Performance Data Compression.,*IEEE Computer*, Vol. 17, No. 6, 1984.

[10] [ZiLe77] J. Ziv and A. Lempel. .A Universal Algorithm for Sequential Data Compression., *IEEE Trans. Information Theory*, IT-23, pp.237-243.

[11] E-Book A complete Reference on data Compression by David Solomon.