ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 1, January 2013

Survey over Adaptive Compression Techniques

Vijay S Gulhane, Dr. Mir Sadique Ali

Abstract— There are a number of XML compression techniques available today which were developed, and tested over the last few years. The problem with XML is that it is text-based, and verbose by its design (the XML standard explicitly states that terseness in XML markup is of minimal importance"). As a result, the amount of information that has to be transmitted, processed and stored is often substantially larger in comparison to other data formats. This can be a serious problem in many occasions, since the data has to be transmitted quickly and stored compactly. This research is for surveying the various existing compression and query processing of XML documents for Adaptive Compression Techniques and Efficient Query Evaluation. and Proposed new approach towards xml compression and Efficient Query Evaluation as - Feasible XML compression using data compression algorithm., Query Processor using Sax parsing and Interfaces .

***Index Terms*— XML compression, partial decompression, SAX parser, compressor Systems, Query processor, Ziv-Lempel algorithm.**

I. INTRODUCTION

The main objective of this research work is to initiate an enquiry XML compression in real-time database systems (RTDBS). Xml documents are identified as the major resources that need to be compressed and effectively partial decompression even at the compressed database level with support from underlying operating systems. The main criteria in assessing any XML compressor and decompressor, the success ratio in terms of the CR1 and CS (compression ratio and compressed size). The proposed research aims at the investigation of efficient Partial query processing techniques on compressed xml dataset in database systems. Focusing on query proceeding time and memory space.

II. SURVEY OF EXISTING COMPRESSOR

Various lossy and lossless compressor and query processor for XML document were investigated and algorithms minimum-redundancy coding [Huffman 1952], due independently to Shannon and Fano [Shannon and Weaver 1949; Fano 1949], Shannon-Fano Coding, Static Huffman Coding, Adaptive Huffman coding, adaptive Huffman algorithm of Vitter (algorithm V) , arithmetic coding was suggested by Elias [Abramson 1963] and algorithm FGK by Gallager [Gallager 1978] etc algorithm were used for various compressor such as XQueC,Xqzip,Xmill,Xgrind,Gzip,Xpress and Xcom. and Lempel-Ziv coding. A novel technique for Adaptive Compression Techniques and Efficient Query Evaluation is proposed. The idea The essential idea of these technologies is that, by utilizing the exposed structure information in the input XML document during the compression process, they pursue two important goals at the same time. First, they aim at achieving a good compression ratio and time compared to the generic text compressors mentioned above. Second, they aim at generating a compressed XML document that is able to support efficient evaluation of queries over the data.

A query processing based on the parsing (SAX), A program or module that checks a well-formed syntax and provides a capability to manipulate XML data element. Navigate through the XML document. Extract or query data elements Add/delete/modify data elements. SAX consists mostly of interfaces rather than classes, but the interfaces refer to two standard exception classes, and a third is provided for universal convenience. These classes are useful for both parser and application writers. A SAX parser, however, is much more space efficient in case of a big input document (because it creates no internal structure). Locator interface in the SAX, This simple interface allows users to find the current location in the XML source document What's more, it runs faster and is easier to learn than DOM parser because its API is really simple, and studied for different size xml documents. adaptive compression in Xml database

Data Compression shrinks down a file so that it takes up less space. This is desirable for data storage and data communication. Storage space on disks is expensive so a file which occupies less disk space is "cheaper" than an uncompressed file. Smaller files are also desirable for data communication, because the smaller a file the faster it can be transferred. A compressed file appears to increase the speed of data transfer over an uncompressed file

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 1, January 2013

several compression techniques have been adopted in these tools, and with varying results. that there are still many paths to be explored in the field of XML data compression. In this work, we present a syntactical compression scheme that is based on probabilistic modeling of XML structure. It does not need the DTD since it infers all the necessary information directly from the input XML data. Moreover, it works incrementally, making on-line compression and decompression possible. Transparent parsing of the compressed data using the SAX interface is possible. An XML compressor 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.

We have investigated different compressor for qurable Xml, like Xquec, Xgrind and Xmill.

XQueC:

Given a workload of typical queries, an attempt is made to determine the optimal container grouping, and assignment of the best available compression algorithm to apply to each group, in order to minimize the following costs

Decompression time

Storage costs for compressed data and source models

XGrind

XGrind is a compression tool for XML documents which supports querying the compressed document. At the same time, they claim that this tool retains the structure of the original XML document too. This facilitates reuse of standard XML techniques for processing the compressed document [P.M.Tolani et al 2002].

XMILL:

The main Aim of developers of Xmill was to design and implement a special purpose XML encoder that will compress an XML file better than a typical compressor just the data of that file .Xmill is a special purpose ,efficient software application for the compression of XML (Extensible Markup Language) Documents.

III. COMPRESSOIN ALGORITHM OVERVIEW

Query processor plays a very important role in xml database systems. However, efficient work has been done to study query processing in XML database systems. Query to xml database refers to a information or data that search where data is located in xml source file. The basic components of any query processor is an indexing scheme .query processing strategies attempt to investigate a more effective auxiliary structure, such as an indexing scheme, to aid querying compressed XML databases . The queriable compressors are themselves strengthened to support efficient querying over compressed XML data. This means the goal here is an analytical model for querying compressed databases, which optimize the query engine of a compressor.

The compression in xml file and the indexing for qury processor affects response time in two ways. First, before a compressor starts its execution, memory space has to be allocated to the proces. These memories are used to store the execution code, copies of files, and any temporary objects produced which is intermediate step for th e compressor and query processor. Second, some applications, such as DBLP complete dataset, have high demands on memory. Their executions will be significantly slowed down. There are many policies We have simulated the following policies.

A .Static Defined –Word Schemes

The classic defined-word scheme was developed over 30 years ago in Huffman's well-known paper on minimum-redundancy coding [Huffman 1952]. Huffman's algorithm provided the first solution to the problem of constructing minimum-redundancy codes. Many people believe that Huffman coding cannot be improved upon, that is, that it is guaranteed to achieve the best possible compression ratio. This is only true; however, under the constraints that each source message is mapped to a unique codeword and that the compressed text is the concatenation of the code words for the source messages. An earlier algorithm, due independently to Shannon and Fano [Shannon and Weaver 1949; Fano 1949], is not guaranteed to provide optimal codes, but approaches optimal behavior as the number of messages approaches infinity. The Huffman algorithm is also of importance because it has provided a foundation upon which other data compression techniques have built and a benchmark to which they may be compared. We classify the codes generated by the Huffman and Shannon-Fano algorithms as variable-variable and note that they include block-variable codes as a special case, depending upon how the source messages are defined.

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 1, January 2013

1.1: Shannon-Fano Coding

The Shannon-Fano technique has as an advantage its simplicity. The code is constructed as follows: the source messages $a(i)$ and their probabilities $p(\ a(i)\ )$ are listed in order of non increasing probability. This list is then divided in such a way as to form two groups of as nearly equal total probabilities as possible. Each message in the first group receives 0 as the first digit of its codeword; the messages in the second half have code words beginning with 1. Each of these groups is then divided according to the same criterion and additional code digits are appended. The process is continued until each subset contains only one message. Clearly the Shannon-Fano algorithm yields a minimal prefix code.

1.2 Static Huffman Coding

1) Huffman's algorithm, expressed graphically, takes as input a list of nonnegative weights $\{w(1), \ldots, w(n)\}$ and constructs a full binary tree [a binary tree is full if every node has either zero or two children] whose leaves are labeled with the weights. When the Huffman algorithm is used to construct a code, the weights represent the probabilities associated with the source letters. Initially there is a set of singleton trees, one for each weight in the list. At each step in the algorithm the trees corresponding to the two smallest weights, $w(i)$ and $w(j)$, are merged into a new tree whose weight is $w(i)+w(j)$ and whose root has two children which are the sub trees represented by $w(i)$ and $w(j)$. The weights $w(i)$ and $w(j)$ are removed from the list and $w(i)+w(j)$ is inserted into the list. This process continues until the weight list contains a single value. If, at any time, there is more than one way to choose a smallest pair of weights, any such pair may be chosen. In Huffman's paper, the process begins with a no increasing list of weights. This detail is not important to the correctness of the algorithm, but it does provide a more efficient implementation [Huffman 1952].

1.3 Arithmetic Coding

The method of arithmetic coding was suggested by Elias, and presented by Abramson in his text on Information Theory [Abramson 1963]. Implementations of Elias' technique were developed by Rissanen, Pasco, Rubin, and, most recently, Witten et al. [Rissanen 1976; Pasco 1976; Rubin 1979; Witten et al. 1987]. We present the concept of arithmetic coding first and follow with a discussion of implementation details and performance.

B. Adaptive Huffman coding

Adaptive Huffman coding was first conceived independently by Faller and Gallager [Faller 1973; Gallager 1978]. Knuth contributed improvements to the original algorithm [Knuth 1985] and the resulting algorithm is referred to as algorithm FGK. A more recent version of adaptive Huffman coding is described by Vitter [Vitter 1987]. All of these methods are defined-word schemes which determine the mapping from source messages to codewords based upon a running estimate of the source message probabilities. The code is adaptive, changing so as to remain optimal for the current estimates. In this way, the adaptive Huffman codes respond to locality. In essence, the encoder is "learning" the characteristics of the source. The decoder must learn along with the encoder by continually updating the Huffman tree so as to stay in synchronization with the encoder.

Another advantage of these systems is that they require only one pass over the data. Of course, one-pass methods are not very interesting if the number of bits they transmit is significantly greater than that of the two-pass scheme. Interestingly, the performance of these methods, in terms of number of bits transmitted, can be better than that of static Huffman coding. This does not contradict the optimality of the static method as the static method is optimal only over all methods which assume a time-invariant mapping. The performance of the adaptive methods can also be worse than that of the static method. Upper bounds on the redundancy of these methods are presented in this section. As discussed in the introduction, the adaptive method of Faller, Gallager and Knuth is the basis for the UNIX utility *compact*. The performance of *compact* is quite good, providing typical compression factors of 30-40%.

2.1 Algorithm FGK

The basis for algorithm FGK is the Sibling Property, defined by Gallager [Gallager 1978]: A binary code tree has the sibling property if each node (except the root) has a sibling and if the nodes can be listed in order of non increasing weight with each node adjacent to its sibling. Gallager proves that a binary prefix code is a Huffman code if and only if the code tree has the sibling property. In algorithm FGK, both sender and receiver maintain dynamically changing Huffman code trees. The leaves of the code tree represent the source messages and the weights of the leaves represent frequency counts for the messages. At any point in time, k of the n possible source messages has occurred in the message ensemble.

2.2Algorithm V

The adaptive Huffman algorithm of Vitter (algorithm V) incorporates two improvements over algorithm FGK. First, the number of interchanges in which a node is moved upward in the tree during a recomputation is limited to

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 1, January 2013

one. This number is bounded in algorithm FGK only by l/2 where *l* is the length of the codeword for $a(t+1)$ when the recomputation begins. Second, Vitter's method minimizes the values of SUM{ $l(i)$ } and MAX{$l(i)$} subject to the requirement of minimizing SUM{ $w(i)\ l(i)$ }. The intuitive explanation of algorithm V's advantage over algorithm FGK is as follows: as in algorithm FGK, the code tree constructed by algorithm V is the Huffman code tree for the prefix of the ensemble seen so far. The adaptive methods do not assume that the relative frequencies of a prefix represent accurately the symbol probabilities over the entire message. Therefore, the fact that algorithm V guarantees a tree of minimum height (height = MAX{ $l(i)$ } and minimum external path length (SUM{ $l(i)$ }) implies that it is better suited for coding the next message of the ensemble, given that any of the leaves of the tree may represent that next message.

C .Other Adaptive Methods

More adaptive data compression methods, algorithm BSTW and Lempel-Ziv coding, are discussed in this section. Like the adaptive Huffman coding techniques, these methods do not require a first pass to analyze the characteristics of the source. Thus, they provide coding and transmission in real time. However, these schemes diverge from the fundamental Huffman coding approach to a greater degree . Algorithm BSTW is a defined-word scheme which attempts to exploit locality. Lempel-Ziv coding is a *free-parse* method; that is, the words of the source alphabet are defined dynamically, as the encoding is performed. Lempel-Ziv coding is the basis for the UNIX utility *compress*. Algorithm BSTW is a variable scheme, while Lempel-Ziv coding is variable-block.

3.1Lempel-Ziv Codes

Lempel-Ziv coding represents a departure from the classic view of a code as a mapping from a fixed set of source messages (letters, symbols or words) to a fixed set of code words. We coin the term *free-parse* to characterize this type of code, in which the set of source messages and the code words to which they are mapped are defined as the algorithm executes. While all adaptive methods create a set of code words dynamically, defined-word schemes have a fixed set of source messages, defined by context (eg., in text file processing the source messages might be single letters; in Pascal source file processing the source messages might be tokens). Lempel-Ziv coding defines the set of source messages as it parses the ensemble.

The Lempel-Ziv algorithm consists of a rule for parsing strings of symbols from a finite alphabet into substrings, or words, whose lengths do not exceed a prescribed integer $L(1)$; and a coding scheme which maps these substrings sequentially into uniquely decipherable code words of fixed length $L(2)$ [Ziv and Lempel 1977]. The strings are selected so that they have very nearly equal probability of occurrence. As a result, frequently-occurring symbols are grouped into longer strings while infrequent symbols appear in short strings. This strategy is effective at exploiting redundancy due to symbol frequency, character repetition, and high-usage patterns. Effective compression is achieved when a long string is replaced by a single 12-bit code.

IV. CONCLUSION

Existing current research work on XML compression does not adequately analyze the related features. they are ineffective at large size of dataset or at different type of XML document. Since none of the compression policy is best, it can be observed from the studies that the disadvantage of one replacement policy can be overcome by another compression policy. Good compression and decompression times are achievable only by parsing and processing the XML document just once, such as by using an SAX parser. The memory consumption should also be taken into consideration of effective compression, since an XML document can be huge and it is infeasible to load the entire document into the memory to perform compression or decompression. Therefore, there should be a limit to the main memory window size for the compression and decompression processes. In addition, the window size should be independent of the size of the input document and preferably be a constant.

REFERENCES

[1] Ziv, J., and Lempel, A. 1977. A Universal Algorithm for Sequential Data Compression. IEEE Trans. Inform. Theory 23, 3 (May), 337-343.

[2] James Cheng and Wilfred Ng "XQzip: Querying Compressed XML Using Structural Indexing" E. Bertino et al. (Eds.): EDBT 2004, LNCS 2992, pp. 219–236, 2004. _c Springer-Verlag Berlin Heidelberg 2004

[3] Andrei Arion, Angela Bonifati, Ioana Manolescu, Andrea Pugliese "XQueC: A Query-Conscious Compressed XML Database" ACM Journal Name, Vol. , No. , 20, Pages 1–31.

[4] JunKi Min MyungJae Park ChinWan Chung "XPRESS: A Queriable Compression for XML Data" SIGMOD 2003, June 912, 2003, San Diego, CA. Copyright 2003 ACM 158113634X/ 03/06

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 1, January 2013

[5] Mustafa Atay, Yezhou Sun, Dapeng Liu, Shiyong Lu, Farshad Fotouhi "MAPPING XML DATA TO RELATIONAL DATA: A DOM-BASED APPROACH" Department of Computer Science Wayne State University, Detroit, MI 48202

[6] Sherif Sakr "XML compression techniques: A survey and comparison " National ICT Australia (NICTA), 223 Anzac Parade, NSW 2052, Sydney, Australia Journal of Computer and System Sciences 75 (2009) 303–322

[7] Pankaj M. Tolani Jayant R. Haritsa XGRIND: A Query-friendly XML Compressor" Proceedings of the 18th International Conference on Data Engineering (ICDE.02) 1063-6382/02 $17.00 © 2002 IEEE

[8] Wilfred Ng · Wai-Yeung Lam Peter T. Wood · Mark Levene "XCQ: A queriable XML compression system" Knowl Inf Syst (2006) DOI 10.1007/s10115-006-0012-z

[9] Wilfred Ng Lam Wai Yeung James Cheng "Comparative Analysis of XML Compression Technologies" Department of Computer Science The Hong Kong University of Science and Technology Hong Kong

[10] Weimin Li "XCOMP: AN XML COMPRESSION TOOL" A thesis presented to the University of Waterloo. Waterloo, Ontario, Canada, 2003

[11] Michael Ley "DBLP — Some Lessons Learned" VLDB '09, August 2428, 2009, Lyon, France Copyright 2009 VLDB Endowment, ACM 0000000000000/ 00/00.

[12] AlHamadani, Baydaa "Retrieving Information from Compressed XML Documents According to Vague Queries" July, 2011 University of Huddersfield Repository http://eprints.hud.ac.uk/

[13] Vojtech Toman "Compression of XML Data" Master Thesis Prague, March 20, 2003 Faculty of Mathematics and Physics Charles University

[14] Smitha S. Nair XML Compression Techniques: A Survey Department of Computer Science, University of Iowa ,Iowa City, Iowa, USA

[15] Wai Yeung, Lam Wilfred Ng, Peter T. Wood Mark Levene XCQ: XML Compression and Querying System Hong Kong University of Science and Technology, Birkbeck College, University of London

[16] Andrei Arion1, Angela Bonifati2, Gianni Costa2, Sandra D'Aguanno1, etel "Efficient Query Evaluation over Compressed XML Data" E. Bertino et al. (Eds.): EDBT 2004, LNCS 2992, pp. 200–218, 2004. _c Springer-Verlag Berlin Heidelberg 2004

[17] Gregory Leighton and Denilson Barbosa "Optimizing XML Compression (Extended Version)" arXiv:0905.4761v1 [cs.DB] 28 May 2009

[18] Michael Ley "DBLP XML Requests- Appendix to the paper "DBLP — Some Lessons Learned" (June 17, 2009) Michael Ley Universit ¨ at Trier, Informatik D–54286 Trier Germany

[19] Mustafa Atay, Yezhou Sun, Dapeng Liu, Shiyong Lu, Farshad Fotouhi "MAPPING XML DATA TO RELATIONAL DATA: A DOM-BASED APPROACH" arXiv : 1010 .1746 V1[CS.DB] 8 oct 2010 , http://arxiv.org/abs/1010.1746v1