Available at: www.researchpublications.org

An Empirical Study: XML Parsing using Various Data Structures

Ms. V.M.Deshmukh[#], Dr. G.R.Bamnote[*]

[#]*Computer Science & Engineering Department, Prof. Ram Meghe Institute of Technology & Research, Badnera-Amravati.*

[1]msvmdeshmukh@rediffmail.com

[*]*Computer Science & Engineering Department, Prof. Ram Meghe Institute of Technology & Research, Badnera-Amravati*

[2]grbamnote@rediffmail.com

***Abstract*— XML has become a defacto standard for data representation and exchange, XML data processing becomes more and more important for server workloads like web servers and database servers. One of the most time consuming part is XML document parsing. Parsing is a core operation performed before an XML document can be navigated, queried, or manipulated. Recently, high performance XML parsing has become a topic of considerable interest In this paper, we are presenting a performance study of XML data parsing by evaluating these parsers using time as a parameter. The proposed design uses four different data structures linked list, stack, array queue. All these data structures are linear in nature. We evaluate the data parsing behaviour and study architectural characteristics. The proposed design analyses the performance of XML parsing techniques using various data structures. Based on observed analysis and graphical results it shows that the data structure based parser is efficient than SAX & DOM parsers..**

***Keywords*— XML, SAX, DOM parsers**

I. INTRODUCTION

XML stands for eXtensible Markup Language (XML). It is a meta-language derived from Standard Generalized Markup Language (SGML) and is used to store and exchange structured information. It was designed to provide flexible information identification in web documents [1]. However it has come to play an increasingly important role in representation and exchange of any kind of structured document because it is platform-independent, human readable and extensible. It offers developers great flexibility to define their own data formats. As commercial workloads and web services rely more and more on XML for data storage and communication, XML data processing becomes an important workload for web servers, database servers, etc.

Studies have shown that these servers spend a significant portion of their execution time in XML data processing [2], especially in XML data parsing. Data parsing is to convert the input XML document and break it into small elements. It is one of the most important portions in XML data processing because an XML document has to be parsed before any other operations can be performed. Studies have shown that data parsing consumes about 30% of web service applications [3], and has become a main performance bottleneck in real-world database servers [4].

The paper is organised as, in section 1 above introduces XML parsing and data structures. In section 2 Literature review for XML parsing is given. Section 3 and 4 proposed evaluation design and experimental results are discussed. Finally concluded in section 5.

II. LITERATURE SURVEY

Parsing is a core operation performed before an XML document can be navigated, queried, or manipulated. Recently, high performance XML parsing has become a topic of considerable interest [8].

A. XML Parsing technologies

There are two technologies for XML data parsing. One is DOM (Document Object Model) [2] and the other is SAX (Simple API for XML) [1] . DOM is a platform and language-neutral interface to represent XML document as an object-oriented model, which is usually a tree. Once the tree is built up, it allows applications to dynamically access and update its content as well as its structure. On the other hand, SAX is an event-driven, serial-access mechanism for accessing XML documents. A SAX parser reads

an XML document as a stream and invokes call back functions provided by the application. Compared to a SAX parser, a DOM parser is much more complex, and hence, much slower. In DOM the data retrieval time can be improved by using cache as a temporary file [5]. After retrieving data from database, the data is saved at cache. This model is very efficient compared to old model and gives better performance for data retrieval. A different approach described SEDOM [4], based on a new compression approach and a set of manipulation algorithms, which enable many DOM operations to be performed when the data are in the compressed format, and allows individual parts of a document to be compressed, decompressed and manipulated. It can be used to efficiently manipulate very large XML documents. The SAX like validating XML parser [13] uses a schema-specific approach. The schema compiler first transforms the schema into an intermediate representation called generalized automata which abstracts the computations required to parse XML documents. An implementation method for DTD-based XML parser is also presented [20] ,according to the requirements of the XML-based network management interface testing system and the syntax and semantic rules of XML. An efficient parsing framework was developed based on the idea of placing internal physical pointers within the XML document that allow the navigation process to skip large portions of the document during parsing[20]. The research shows how to generate such internal pointers in a way that optimizes parsing using constructs supported by the current W3C XML standard.

The tree representation is equally important in XML parsing. The technique presented by [21] allows to represent the tree structure of an XML document in an efficient way. The representation exploits the high regularity in XML documents by compressing their tree structure; the latter means to detect and remove repetitions of tree patterns.

B. Parallel XML Parsing

Parallel XML parsing (PXP) leverages the growing prevalence of multicore architectures in all sectors of the computer market and yields significant performance improvements. The parallel XML

parser consists of an initial preparsing phase to determine the structure of an XML document followed by a full parallel parse [8] .The results of preparsing are then used to help partition the XML document for data parallel processing. The XML document would be divided into some number of chunks, and each thread would work on the chunks independently. As the chunks are parsed, the results are merged. This parallel approach focuses on DOM-style parsing where a tree data structure is created in memory that represents the document.

A hybrid XML SAX parser has been implemented [21] efficiently. To handle inherent data dependencies in XML while still allowing reasonable scalability, a 4-stage software pipeline with a combination of strictly sequential stages and stages that can be further data-parallelized within the stage are utilised. Thus a hybrid between pipelined parallelism and data parallelism is proposed. The same research is being extended further to design a general model. The kernel of the model is a stealing-based dynamic load balancing mechanism by which multiple threads are able to process disjointed parts of the XML document in parallel with balanced load distribution [7]. The model also provides a novel mechanism to trace the stealing actions, and the equivalent sequential results are obtained by gluing the multiple parallel-running results together. The basic idea of stealing based scheme is that every thread works on its own local task queue and whenever it runs out of the task it steals the task from other thread's task queue. This model uses queue as a data structure.

C. XML Parsing in Databases

XML parsing is generally known to have poor performance characteristics relative to transactional database processing. Yet, its potentially fatal impact on overall database performance is being underestimated. XML parsing performance is a key obstacle to a successful XML deployment for real-world database applications. There is a considerable share of XML database applications which are prone to fail at an early and simple road block: XML parsing [9]. Processing model for storing and

building XML document in data transfer between XML and relational database is available [6]. In this model, a XML document is parsed and its elements are stored in a single table of database instead, it is not necessary to read the nodes according to their hierarchical structure, thus leveraging the workload of DOM building to memory by the algorithm called Tree-Branch inter growth. XML can be used to both store and enforce organizational data definitions, thus providing a synergetic framework for leveraging the potential of knowledge management (KM) tools [18]. XML provides a flexible markup standard for representing data models. KM provides IT processes for capturing, maintaining, and using information. [22] presents a parallel solution to XML query application through the combination of parallel XML parsing and parallel XML query. The XML parsing is based on arbitrary XML data partition and parallel sub-tree construction with the final merging procedure. After XML parsing, the region encodings of XML data are obtained for relation matrix construction in that the XPath evaluation in query procedure is based on relation matrix.

III. Proposed Work

A software program called an XML parser (or an XML processor) is required to process the XML document. The XML parser reads the XML document, checks its syntax, reports any errors and allows programmatic access to the documents contents. An XML document is considered well formed if it is syntactically correct. XML syntax requires a single root element, a start tag and end tag for each element, properly nested tags and attribute values in quotes. Furthermore, XML is case sensitive, so the proper capitalization must be used in element and attribute names. A document that properly conforms to this syntax is well formed document. Parsers can support the Document Object Model and Simple API for XML for accessing a document content programmatically using languages. A DOM based parser builds a tree structure containing the XML document's data in memory. A SAX based parser processes the document and generates events (i.e. notification to the application) when tags, text, comments etc are encountered. These events return data from the XML document [8].

For the design of XML parser, we focus on XML documents which include an elements, attributes, and text/values in XML documents. Although a text can appear anywhere within the start and end tag of an element, we shall first assume that it is strictly enclosed by start and end element tags, e.g., <author>Jack</author>.

An XML parser can be built by extracting tokens(e.g. start and end tags) from a document by reading it from the beginning. With the help of data structure, we can parse it. For example, A DOM tree can be built by extracting tokens (e.g. start and end tags) from a document by reading it from the beginning. A stack (S) is maintained and is initially empty. This stack essentially stores the information of all the ancestors (in the DOM tree) of the current element being processed in the document. When a start element tag say <e> is read, a DOM node (de) is created for element (e) and any (attribute, value) pair that is associated with the element is parsed and stored, by creating the necessary DOM nodes. If S is not empty, then this implies that (de)'s parent node has already been created. If (e) encloses text, then a DOM node for the text is also created and linked as a "text" child of de. When an end element tag say </e> is read, e is checked with the top of stack S. If the element names do not match, then the parsing is aborted as the document is not well-formed. Otherwise, the top of S is popped and the parsing continues. After the last character of the document is processed, if S is empty, then the entire DOM tree has been constructed. Otherwise, the document is not well-formed.

We serially parse the document with start tag and end tag. When start tag will be encountered, we add node in data structure and when end tag encounter then check the current node with end tag, if it matches then parsing continues. If the element names do not match, then the parsing is aborted as the document is not well-formed. Here we parse XML document with the help of three parsers. First is DOM parser, implemented using java API for XML and generates DOM tree by calculating time taken to parse the document.

Second is SAX parser, implemented using API for XML and generates events by calculating time taken to parse the document. And third, proposed parser is implemented using java string library with the help of data structures like Queue, Linked list, Array and Stack and generates events by calculating time taken to parse the document.

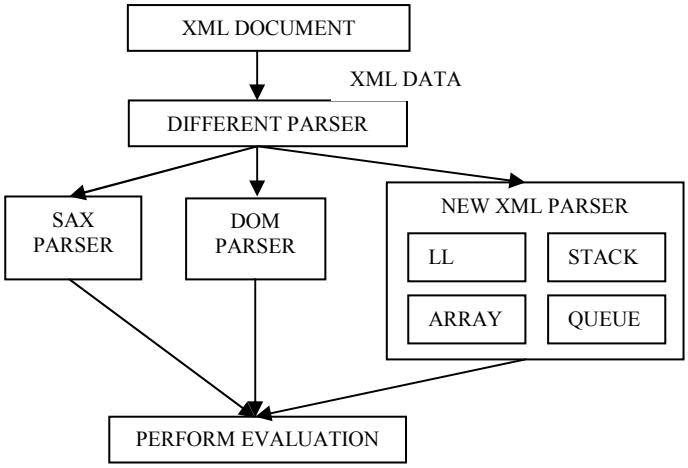

Fig. 1 Architecture View.

A. Algorithm

1) *Get XML file.*
2) *Initialize variable to calculate time and space.*
3) *Append "? " as a delimiter at the end of file.*
4) *Get start tag using String functions.*
5) *Add start tag to the Data Structure.*
6) *Get end tag and check with the last tag in the Data Structure.*
7) *If start tag and end tag matches, remove it from the Data Structure.*
8) *Repeat steps 4 to 7 until Data Structure become null.*
9) *Calculate space and time taken to parse XML file.*

IV. Experimental Results

In this work, XML documents are the input to the system. We have used SAX parser & DOM Parser. In implementation, we have implemented data structure based parser using java string library. SAX and DOM are implemented using java API for XML processing and all the elements, attributes, values are parsed according to the sequence. The parsed document and the parsing time and space are shown in text area. SAX and DOM parsers uses Stack as a data structure to parse the XML documents.

The proposed design uses different data structures Linked List, Queue, Stack and Array to parse XML document. All the elements, attributes, values are parsed according to the sequence. The parsed document is shown in text area and also the parsing time and space is displayed.

After parsing the document using above parsers, we compare SAX, DOM and data structure based parser as per the time and space taken to parse XML document and shows performance evaluation using graphs.

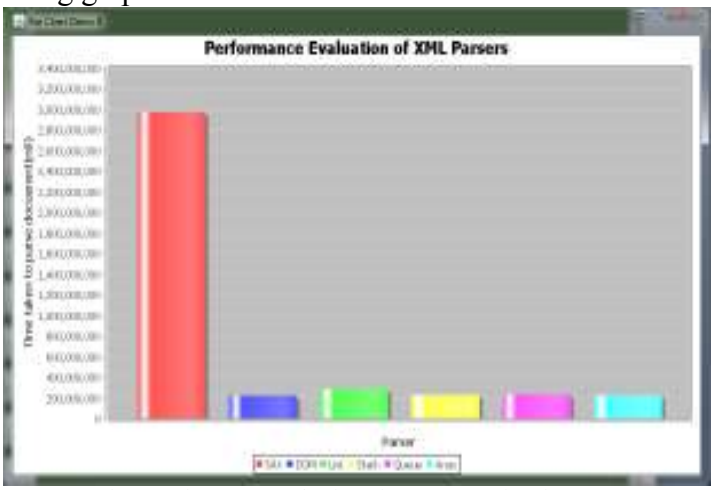

Fig. 2 Performance Evaluation of XML Parser for Time.

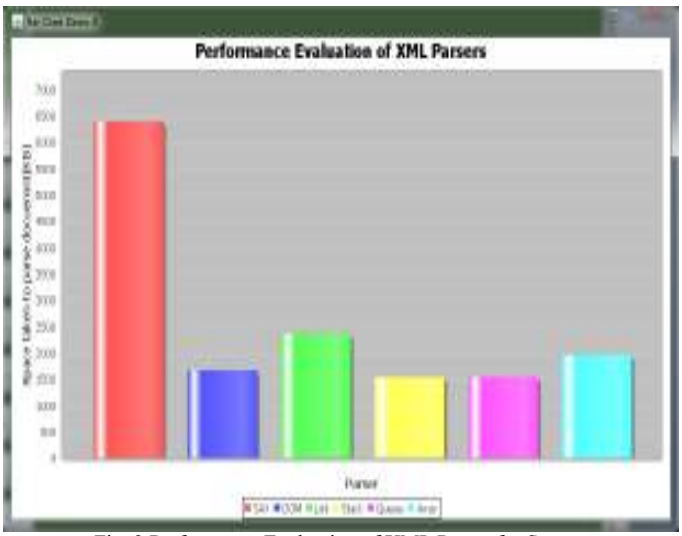

Fig. 2 Performance Evaluation of XML Parser for Space.

V. Conclusion

As a language for semi-structured documents, XML has emerged as the core of the web services architecture and is playing crucial roles in messaging systems, databases, and document processing. XML document must be parsed for validation. XML parsing is generally known to have poor performance characteristics relative to transactional database processing.

Traditionally SAX and DOM parsers are use to parse the XML document. DOM creates tree structure of whole XML document in main memory and parses the document. Unfortunately, DOM has some penalty over performance characteristics. This method involves reading the entire file and storing it in a tree structure, which may be inefficient, slow, and it can be a strain on resources. One alternative is SAX. SAX allows you to process a document as its being read, which avoids the need to wait for all of it to be stored before taking action. SAX generates events by fetching contents from secondary storage during parsing and unfortunately secondary storage is slower.

Data structure based parser works in main memory and uses various data structure for parsing. In the implementation, the proposed parser removes the elements from document and serially checks if the document is well formed or not using Linked list, Queue, Stack and Array simultaneously, which increases its performance over SAX and DOM parser.

In future, we can make more efficient XML parser using data structures and DTD file. DTD file is used for Validation purpose. We can also improve the performance by caching result of highly referenced files.

REFERENCES

[1] W3C, "Extensible markup language (XML)." [Online]. Available: http://www.w3.org/XML.

[2] W3C, "Document object model (DOM) level 2 core specification." [Online]. Available: http://www.w3.org/TR/DOM-Level-2-Core.

[3] Billy B.L. Lim, H. Joseph Wen, " The impact of next generation XML" in Journal of Information Management & Computer Security, 10/1, 2002, pp. 33-40, MCB University Press.

[4] Fangju Wang, Jing Li, Hooman Homayounfar, " A space efficient XML DOM parser" in Journal of Data & Knowledge Engineering,Volume 60, Issue 1, January 2007, Pages 185-207.

[5] Yusof Mohd Kamir, Mat Amin Mat Atar, "High Performance of DOM Technique in XML for Data Retrieval,"in International Conference on Information and Multimedia Technology IEEE, 2009.

[6] Li Gong,Liu Gao-Feng, Liu Zhong, Ru-Kui. "XML Processing by Tree-Branch symbiosis algorithm", in 2nd International Conference on Future Computer and Communication. IEEE 2010.

[7] Lu W., Dennis Gannon "Parallel XML Processing by Work Stealing ",in High performance Distributed Computing archive Proceedings of the 2007 workshop on Service-Oriented computing performance. 2008. Monterey Clifornia USA page31-38.

[8] Lu W., Y. Pan, and K. Chiu," A Parallel Approach to XML Parsing "in The 7th International Conference on Grid Computing, IEEE/ACM 2006.

[9] Nicola M. and J. John," XML Parsing: a Threat to Database Performance", in Proc. of the12th International Conference on Information and Knowledge Management, pages 175–178, 2003.

[10] Power James F., Brian A Malloy." Program annotation in XML: a parse-tree based approach", in Proceedings of the Ninth Working Conference on Reverse Engineering (WCRE'02) 2002.

[11] Seung Min Kim, Suk I. Yoo, "DOM tree browsing of a very large XML document: Design and implementation", in Journal of Systems and Software, Volume 82, Issue 11, November 2009, Pages 1843-1858.

[12] Hazem M., El-Bakry and Nikos Mastorakis, "Performance Evaluation of XML Web Services for Real-Time Applications", in International Journal of Communications, Issue 2, Volume 3, 2009.

[13] Gao Z., Y. Pan, Y. Zhang, and K. Chiu. 'A High Performance Schema-Specific XML Parser", IEEE Intl. Conf. on e-Science and Grid Computing, pages 245–252, Dec. 2007.

[14] Lu W., K. Chiu, A. Slominski, , and D. Gannon" A Streaming Validation Model for SOAP Digital Signature", in 14th IEEE International Symposium on High Performance Distributed Computing (HPDC-14) July 2005.

[15] Engelen Robert A.van, "Constructing Finite State Automata for High Performance Web Services", in Proceedings of the International Symposium on Web Services(ISWS), 2004.

[16] M. Huhns and M. P. Singh. Service-Oriented Computing: Key Concepts and Principles, IEEE Internet Computing, 9(1):75–81, Jan. 2005.

[17] R. A. V. Engelen. A framework for service-oriented computing with C and C++Web service components. ACM transactions on Internet Technology, 8(3):1–25, 2008.

[18] James R. Otto, James H. Cook and Q.B. Chung, " Extensible markup language and knowledge management", in Journal of Knowledge Management, Volume 5, No. 3, 2001, pp. 278-284, MCB University Press.

[19] Fernando Farfán, Vagelis Hristidis, Raju Rangaswami, "2LP double-lazy XML parser" in Journal of Information Systems, Volume 34, Issue 1, March 2009, Pages 145-163.

[20] Giorgio Busatto, Markus Lohrey, Sebastian Maneth, "Efficient memory representation of XML document trees", in Journal of Information Systems, Volume 33, Issue 4-5, June-July 2008, Pages 456-474.

[21] Pan Yinfei, Ying Zhang, Kenneth Chiu, "Hybrid Parallelism for XML SAX Parsing", in International Conference on Web Services IEEE, 2008.

[22] Chen Rongxin, Weibin Chen, "A Parallel Solution to XML Query Application" in the Proceedings of the International Conference on Computer Science and Information Technology IICCSIT), Vol. 6, pages 542-546,IEEE, 2010

[23] Kai Ning, Luoming Meng," Design and Implementation of the DTD-based XML Parser", in Proceedings of ICCT 2003.

[24] Kotsakis Evangelos, Klemens Böhm, "XML Schema Directory: A Data Structure for XML Data Processing", in First International Conference on Web Information Systems Engineering (WISE'00),Proceedings, pp 62-69, June 19-21, 2000, Hong Kong, China, IEEE CS Press.

TABLE I

RESULTS OF EXPERIMENTATION

Parser	File size:10KB		File size:50KB		File size:70KB		File size:100KB	
	Time (nS)	Space (KB)	Time (nS)	Space (KB)	Time (nS)	Space (KB)	Time (nS)	Space (KB)
SAX	299083693	4094	6656768001	7295	4765551424	4621	53047291539	7220
DOM	52152063	1454	64048291	1805	586442757	1759	65346592	2154
Linked List	47022073	3894	495643763	2165	496652337	4394	2453871218	4502
Stack	44546224	1088	399319486	3203	408594395	2070	1530300372	7251
Queue	41595062	1173	386120905	3203	403111464	2070	1529028005	3586
Array	41723920	1173	397318952	3763	404016710	2096	1485246010	7806