# An Adaptive and Efficient XML Parser Tool for Domain Specific Languages

W. Jai Singh, S. Nithya Bala

Abstract— XML (eXtensible Markup Language) is a standard and universal language for representing information. XML has become integral to many critical enterprise technologies with its ability to enable data interoperability between applications on different platforms. Every application that processes information from XML documents needs an XML Parser which reads an XML document and provides interface for user to access its content and structure. However the processing of xml documents has a reputation of poor performance and a number of optimizations have been developed to address this performance problem from different perspectives, none of which have been entirely satisfactory. Hence, in this paper we developed a Fast Parser tool for domain specific languages. In this we can execute Parser user friendly without having any constraints.

**Index Terms**—XML, Parser Tool, Document Object Model, SAX, XML Document, Document Validation.

## 1 Introduction

The XML (eXtensible Markup Language) is now widely adopted within (networked) applications. Due to its flexibility and efficiency in transmission of data, XML has become the emerging standard of data transfer and data exchange across the application and Internet [1]. XML has potential as a back end solution as well as a marvelous standard for re-designing databases and other content. XML has become integral to many critical enterprise technologies with its ability to enable data interoperability between on different platforms. With various conversion tools now emerging in the marketplace, XML can be used to bridge between different applications. It's standard-based, set forth a design for structuring future content [1], [2].

XML delivers key advantages in interoperability due to its flexibility, expressiveness and platform-neutrality. As XML has become a performance-critical aspect of the next generation of business computing infrastructure. Tomorrow's computers will have more cores rather than exponentially faster clock speeds, and software will increasingly have to rely on parallelism to take advantage of this trend.

Every application that processes information from XML documents needs an XML Parser which reads an XML document and provides interface for user to access its content and structure. An XML parser facilitates in simplifying the process of manipulating XML documents. There are mainly two challenges for generic XML parsers. One is that the code size of the XML Parsers is restricted because of the limitation of memory. The other is the run time adaptability of XML Parsers is required due to the diversity of applications in terms of their dependency on XML syntax set.

• *W. Jai Singh, Assistant Professor, Department of MCA, Park College of Engineering and Technology, Coimbatore, Tamil Nadu, India – 641 659.*
• *S. Nithya Bala, PG Scholar, Department of MCA, Park College of Engineering and Technology, Coimbatore, Tamil Nadu, India – 641 659*

Several efforts have been made to address the parsing and validation performance through the use of grammar based parser generation by leveraging XML schema language such as DTD (Document Type Definition), XML schema at compile time. DOM (Document Object Model) and SAX (Simple API for XML) are the two most widely used XML parsing models, none of which has been entirely satisfactory [2], [5].

A parser can read the XML document components via Application Programming Interfaces (APIs) in two approaches. For stream-based approach such as SAX (also known as event-based parser) and tree-based approach such as DOM. DOM (Document Object Model) and SAX (Simple API for XML) are the two most widely used XML parsing models, none of which has been entirely satisfactory [2], [3], [5], [7]. A brief description of them is given as follows.

DOM is a tree-based interface that models an XML document as a tree of various nodes. The main advantage of this parse method is that it supports random access to the document. DOM parsers create a node object for each node that precisely models all the structure and content information [2], [3], [5]. DOM is an easy way to work with XML. However, DOM parsers take too much time and memory, making them unavailable for large XML documents. Moreover, no actual work can be done before completely parsing XML, which introduces significant delay and is unacceptable in enterprise applications.

SAX is an event-based parsing model that reads an XML document from beginning to end. Each time it encounters a syntax construction; it generates an event and notifies the application [2], [5], [7]. It does not preserve the structure and content information in memory, thus saving a large amount of memory space. Unfortunately, they lack the ability of random access and are forward only, which limits their use to a very small scope. SAX is memory efficient but writing a SAX parser is complex.

Several methods were presented to improve XML parsing from different viewpoints. There are a number of approaches trying to address the performance bottleneck of XML parsing. The typical software solutions include the pull-based parsing [9], lazy parsing [10] and schema-specific parsing [4].

Su Cheng Haw and G. S. V. Radha Krishna Rao have presented a model called “Comparative Study and Benchmarking on XML Parser”. In that, they compare the xerces and .NET parsers based on the performance, memory, usage and so on [2]. Giuseppe Psaila have been developed a system called “Loosely coupling Java algorithms and XML Parsers”. In that, he conducted a study about the problem of coupling java algorithms with XML parsers. Su-cheng Haw and Chien-Sing Lee have been presented a model called “Fast Native XML Storage and Qurey Retrieval”. In that, they proposed the INLAB2 architecture comprises of five main components namely XML parser, XML encoder, XML indexer, Data manager and Query processor [10]. Fadi El-Hassan and Dan Ionescu presented “An efficient Hardware based XML parsing techniques”. In that, they proposed hardware based solutions can be an obvious choice to parse XML in a very efficient manner.

The existing XML Parsers spend a large amount of time in tokenizing the input. To overcome all the drawbacks, here we have developed a new Fast Parser tool for domain specific languages. Though careful analysis of the operations required for parsing and validation, we are using hash table to store element information, this will enhance the speed of accessibility while searching for an element. More over we are using regular expressions to search for the tags and attributes, this will enhance the speed while reading XML contents.

## 2 Fast XML Parser

To parse an XML document in software, the processing sequence starts by loading the XML document, then reading its characters in sequence, extracting elements and attributes and then validating the XML document, writing parsed information and finally reading the resulting parsed data. Our initial approach separates the process of reading the XML document and stores the contents in to the hash table using regular expressions. Fig.1 shows the architecture of the fast XML parser tool.

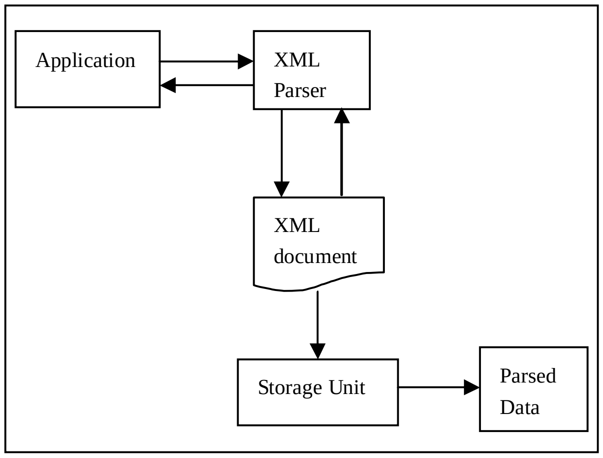

Fig.1: XML Parser Architecture

The Fast XML Parser Tool contains four modules. First, load an XML document in an application. Second, reading an XML document. Third, Writing an XML document into the application. Finally, knowledge based search of an XML document. The Fast XML Parser Tool is as follows:

Load an XML document: Before an XML document can be accessed and manipulated, it must be loaded into an XML Parser. The XML Parser reads XML document and converts it into a meaningful format. The job of the XML Parser is to make sure that the document meets the defined structure and constraints. The validation rule for any particular sequence of character data is determined by the type of definition for its enclosing element. Fig.2 shows the sample XML document.

Algorithm

```
Step 1: Read XML File
Step 2: Search for start tag using regular expression ‘<…>
Step 3: If a start tag found then
Step 4: Add attributes to hash table
Step 5: Search or end tag using regular expression ‘</…>
Step 6: Add element to hash table
Step 7: End if
Step 8: Repeat step 3 to 7 until End Of File
Step 9: Verify and validate each element in hash table
Step 10: End XML Parser
```

Reading an XML document: Reading an existing load XML file. It provides the createXMLReader function that returns an implementation of the XMLReader interface. It reads the root element first and reads the sub element and corresponding data. Finally creates the XML document and sends it to the user application as a XML document.

Writing an XML document: Read the XML document and separates the content and writes it to the corresponding hash table such as root element, sub element, attributes and data. It provides the CreateXMLWriter function to return an implementation of the XMLWriter function.

Knowledge based search: Search a particular element or content from an XML document by our fast XML Parser tool. Initially it checks the content in storage unit using hash key that is root element. If it is available then it goes to the sub element and corresponding data and displays it as output. If it is not available means it terminates the search.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Product data is the root element-->
<PRODUCTDATA>
<PRODUCT PROID="P001">
<PRODUCTNAME>Barbie
Doll</PRODUCTNAME>
<DESCRIPTION>This is a toy for children in
the age group of 5-10 years</DESCRIPTION>
<PRICE>$120</PRICE>
<QUANTITY>12</QUANTITY>
</PRODUCT>
</PRODUCTDATA>
```

Fig.2: Sample XML Document

## 3 EXPERIMENTAL RESULTS AND DISCUSSION

According to the experiment and considering the balance between memory and computing consumption, our parser tool is appropriate for parsing XML document with less memory. When XML documents size increases and gets larger, the time difference will increase accordingly.

We made some assumptions to facilitate our estimation. One of the most noticeable assumptions is that we simply ignore the intensive garbage collection operations in DOM. Garbage collection is actually very time-consuming, our parser tool can avoid it, because we does not recycle the unused memory space when parsing finishes. If we take this into account, the time difference would be even larger. So, we can draw the conclusion that our parser gains great advantage over DOM especially for large and complicated XML documents.

### 3.1. Testing Results

The testing result shows that great improvement has been achieved over present XML parsers, both in terms of memory computation and parsing performance. The tests are performed on our XML parser tool and MSXML. MSXML is a typical DOM parser which is widely used in enterprise applications. Similar tests are performed on other open-source XML parsers too, like Xerces-C++, Jaxen and Xalan. For clarity, we only present the testing results between our XML parser tool and MSXML. The testing codes are run in a HP G61 with 2.80GHz Intel(R) Core2 Due CPU and 4GB of RAM.

### 3.2. Memory Consumption

In this part, we test the ratio of memory space needed to parse an XML document to the original XML document size. The memory usage when many 1KB and 5KB XML documents are stored and accessed. The result shows in fig.3 that the memory consumption is acceptable in such an environment.

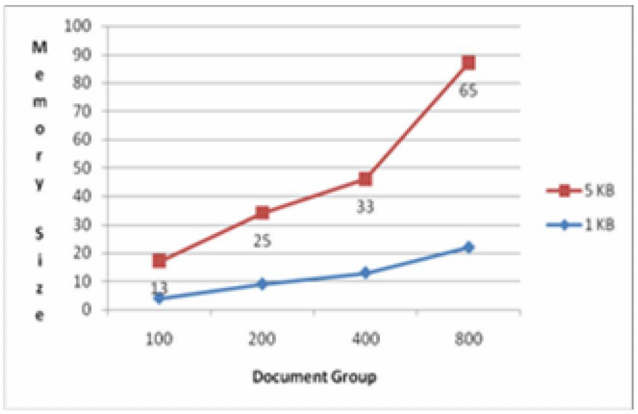

Fig.3: Memory Consumption Results

### 3.3 Parsing Performance

The results of parsing throughput per second are shown in figure. Note that our XML parser tool gets the lowest throughput, thus leading to low parsing performance. Figure 4 shows the cumulative parsing times for a group of similar XML documents. The X-axis is the file size of the XML documents that were parsed.

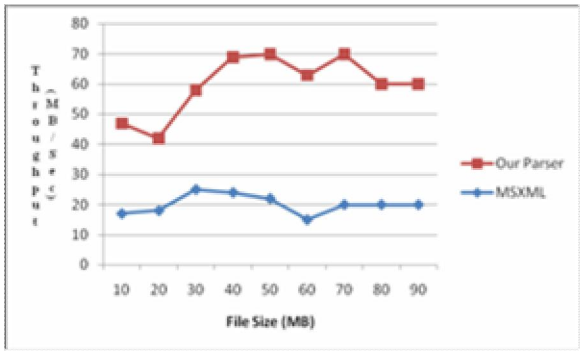

Fig.4: Parsing Throughput Results

### 3.4 Results Analysis

From above testing results we can see that our XML parser tool consistently outperforms MSXML in both memory consumption and parsing performance. Our XML parser tool consumes memory about approximately the same size of original XML document, while MSXML takes 3~4 times. Our XML parser tool deliveries up to 20~35MB/sec sustained throughput, while the average parsing throughput of MSXML is only 12MB/sec. Note that for MSXML, parsing throughput declines sharply when XML documents become large, which verifies the conclusion that too much creation and destroy of node objects in DOM can slow down parsing performance considerably.

## 4 CONCLUSION

From a proper analysis of positive points and constraints on the XML Parser tool, it can be safely concluded that the product is a highly efficient. This XML Parser tool is working properly and meeting to all user requirements. This tool can be easily plugged in domain specific languages. Our Parser tool can improved the XML parsing performance significantly and scales well. The computing efficiency will be improved in the future work.

## 5 REFERENCE

[1] “eXtensible Markup Language”, http:/www.w3.org/xml. 10th Feb 2011.

[2] Su Cheng Haw and G. S. V. Radha Krishna Rao, “A Comparative Study and Benchmarking on XML Parsers”, *International Conference on Computer science and Technology*, IEEE Computer Society,

pp.321-325, Feb-2007.

[3] Shiren Ye and Tat-Seng Chua, “Learning Object Models from Semistructured Web Documents”, *IEEE Transactions on Knowledge and Data Engineering*, pp. 334-339, 2006.

[4] Zhenghong Gao, Yinfei Pan, Ying Zhang and Kenneth Chiu, ”A High Performance Schema-Specific XML Parser”, *Third IEEE International Conference on e-Science and Grid Computing*, IEEE Computer Society, pp.245-252, 2007.

[5] A. Waheed, J. Ding, “Benchmarking XML Based Application Oriented Network infrastructure and Services”, *International Symposium on Applications and the Internet*, IEEE Computer Society, 2007.

[6] Wei Zhang and Robert A. van Engelen, “An Adaptive XML Parser for Developing High-Performance Web Services”, *Fourth IEEE International Conference on eScience*, IEEE Computer Society, 2008.

[7] Yunsong Zhang Lei Zhao* Jiwen Yang Liying Yu, “NEM-XML: A Fast Non-extractive XML Parsing Algorithm”, *Third International Conference on Multimedia and Ubiquitous Engineering*, IEEE Computer Society, 2009.

[8] Zhou Yanming and Qu Mingbin, “A Run-time Adaptive and Code-size Efficient XML Parser”, 30th Annual International Computer Software and Applications Conference, IEEE Computer Society, 2006.

[9] Giuseppe Psaila, “Loosely Coupling Java Algorithms and XML Parsers: a Performance-Oriented Study”, *22nd International Conference on Data Engineering Workshops*, IEEE Computer Society, 2006.

[10] Su-Cheng Haw and Chien-Sing Lee, “INLAB2: Fast Native XML Storage and Query Retrieval”, *3rd International Conference on Intelligent System and Knowledge Engineering*, IEEE Computer Society, pp.44-49, 2008.

[11] Fadi El-Hassan and Dan Ionescu, “SCBXP: An efficient hardware-based XML parsing technique”, IEEE Computer Society, pp.45-50, 2009.

[12] Wei Lu , Kenneth Chiu and Yinfei Pan, “A Parallel Approach to XML Parsing”, *Grid Computing Conference* , pp.223-230, 2006.

[13] Gang WANG, Cheng XU, Ying LI, Ying CHEN, “Analyzing XML Parser Memory Characteristics: Experiments towards Improving Web Services Performance”, *IEEE International Conference on Web Services*, IEEE Computer Society, 2006.