A Comparative Study and Benchmarking on XML Parsers

Su Cheng Haw
Faculty of Information Technology,
Multimedia University
63100 Cyberjaya
schaw@mmu.edu.my

G. S. V. Radha Krishna Rao
Faculty of Information Technology,
Multimedia University
63100 Cyberjaya
gsvradha@mmu.edu.my

Abstract — **Due to its flexibility and efficiency in transmission of data, XML has become the emerging standard of data transfer and data exchange across the Internet. XML document must always be checked for well formedness before data transfer and exchange can take place. To choose the right parser for an organization respective system is crucial and critical; since improper parser will lead to degradation in performance and decrease in productivity. In this paper, we will do an extensive comparative study and benchmarking on the popular XML parsers found in the market today. In addition, we also propose a non-validating SAX based XML parser, xParse. We implemented our technique and present the performance results, which prove the viability of our approach.**

Keywords — **XML, XML Parser, benchmark test, comparative study.**

1. Introduction

eXtensible Markup Language (XML) has become the de facto for data exchange and data transfer via the web medium [1]. However, XML would not be able to perform as desired before it has been parsed. Hence, the importance of XML parser, one of the core XML technologies, has become significant in this matter.

Currently there are a lot of XML parsers, and most of them evolve, improve and become sophisticated. Though all the parsers serve the same purpose, they vary in terms of specification, performance, reliability and also conformance to standards. If a wrong choice has been made, it is highly possible to leads to the problem of excessive hardware requirement, which will resulted in productivity degradation [2].

In this paper, comparative studies and extensive research on the features and information about the selected parsers are made. Furthermore, we also propose a non-validating SAX based XML parser, xParse, which is built on top of Java platform. Experiments to benchmark the performance of xParse with the two leading parsers in the market, Xerces (a Java based parser) and .NET parser (a Microsoft based parser) are to be conducted.

Our paper is organized as follows. Section 2 gives the background studies and literature reviews from the previous work done in this field. Section 3 presents the overview of xParse. Section 4 discusses on the experimental setup, findings and benchmark results. Section 5 concludes with conclusions and suggests future works.

2. Literature Reviews

2.1. Introduction to XML parser

XML parser plays the roles in reading, detecting its well formedness, and validating the XML documents against its schema. It can be classified along two independent dimensions: (1) validating versus non-validating [3], and (2) stream-based versus tree-based [4].

A validating parser uses a Document Type Definition (DTD) or a schema to verify that a document is properly constructed while a non-validating parser only require that the document must be well formed [5, 6]. Thus, a non-validating parser is relatively simpler compare to a validating parser.

A parser can read the XML document components via Application Programming Interfaces (APIs) in two approaches. For stream-based approach (also known as event-based parser), it reads through the document and signal the application every time a new component appears. As for tree-based approach, it reads the entire document into a memory resident collection of object as a representation of original document in tree structure [7, 8]. As a result, tree-based approach is not suitable for large-scale XML data because it can easily run out of memory.

Simple API for XML (SAX), StAX and XMLPull are stream-based approach API while Document Object Model (DOM), JDOM, ElectricXML, DOM4j are categorized as tree-based API. Most of the major XML parsers support both SAX and DOM. However, there are a few parsers that only support SAX, and at least a couple that only support their own proprietary API like ElectricXML and XMLPull parser.

A brief comparison of XML parser's APIs, with respect to their characteristics are depicted in Table 1.

2.2 Related Work

A study towards different XML parsers is beneficial when comes to determine the strength and weaknesses of the products. Various studies have been conducted which compare on conformance to standards, speed, memory usage and so on.

Table 1. Comparison on XML Parser's APIs

APIs	Advantages	Disadvantages
DOM	- Easy navigation - Entire tree loaded into memory - Random access to XML document - Rich set of APIs	- XML document must be parsed at one time - It is expensive to load entire tree into memory
SAX	- Entire document not loaded into memory which resulting in low memory consumption - Allows registration of multiple ContentHandlers	- No built-in document navigation support - No random access to XML document - No support for modifying XML in place - No support for namespace scoping
StAX/ Pull	- Contains two parsing models, for ease or performance - Application controls parsing, easily supporting multiple inputs - Powerful filtering capabilities provide efficient data retrieval	- No built-in document navigation support - No random access to XML document - No support for modifying XML in place and is still in an immature state
Electric XML	- Light weighted - Fast in performance	- No support for validating - Still in immature state

Michael and Elliotte have respectively conducted a SAX conformance test using W3C XML conformance Test Suite on a number of parsers. They compare based on the features supported such as well-formedness, validation, namespaces, XML schema, open source, exception handling, fatal error and so on. Based on their studies, Xerces emerges as the most conformant parser to SAX standard [6, 9].

Anex conducted a study to evaluate seven XML parsers on the conformance test, parsing speed and memory usage using OASIS Test Suite [10]. The study reveals that IBM java is an 'outstanding' parser where else Microsoft XML (MSXML) falls under 'good' ranking category.

Karre et al. conducted an empirical assessment on five java based parsers to measure each parser strengths and weakness based on three factors: (1) features (well-formedness, validity and namespaces), (2) percentage of acceptance and rejection rate for correct and incorrect XML documents, and (3) parsing speed. He concludes that Xerces is the best parser fulfill factors (1) and (2) while Aelfred which built on top of Java API for XML Processing (JAXP) has the fastest parsing speed.

Some other recent works include performance tests conducted by Mohseni and Sonoski respectively. Performance test conducted by Mohseni indicates that MSXML rivals other parser having the shortest loading time [11]. Sosnoski carried out a test on DOM based parsers using XMLBench. He tested on the execution speed and memory usage for a set of XML documents ranging from small-scale to large-scale file sizes. The test result shows that Xerces outperforms among the others [12]. Besides, Xerces parser is also voted as the best XML parser of the year by XML-Journal/Web Services Journal Readers' Choice Awards [13].

Since Xerces and MSXML outperform the rest of the parsers in most cases, we have decided to concentrate benchmarking our proposed parser, xParser against these two parsers.

3. Overview of xParse

xParse, a non-validating SAX parser is implemented under Java platform based on the following setting:-

- Get a new instance of SAXParserFactory

  ```
  SAXParserFactory factory =
  SAXParserFactory.newInstance();
  ```

- Create the parser

  ```
  SAXParser xParse =
  factory.newSAXParser();
  ```

- Parse the XML file

  ```
  xParse.parse(filename, this);
  ```

xParse fulfill the XML 1.0 specification requirements [14] except for parsing an internal DTD. With the rapid evolvement of XML schemas to replace DTDs, we think that supporting DTD parsing is no longer necessary [2]. Nevertheless, the limitation of xParse is that the XML input document must be in UNICODE.

xParse adopts the event-driven style of parsing. Figure 1 shows the entire XML document is transform into a series of SAX events. As the parser sequentially encounters each component such as element, text, attribute, comment, processing instruction, entity reference, declaration and so on, it reports to its delegate. Next, its delegate will process according to the implemented associated method. Figure 2 shows a sample of XML document.

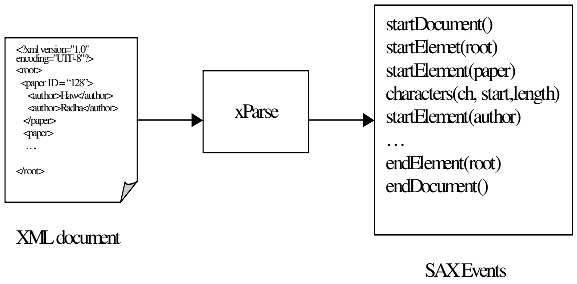


Figure 1. xParse Processing Model

```
<?xml version="1.0" encoding="UTF-8"?>
  <paper ID = "128">
     <author>Haw</author>
     <author>Radha</author>
  </paper>
```

Figure 2. A Sample of XML Document

Based on the sample document in Figure 2, xParse will report the events of :-

(1) Start parsing document
(2) Found start tag of element paper
(3) Found attribute ID of element paper, with value 128
(4) Found start tag of element author
(5) Found text with value Haw
(6) Found end tag of author
(7) Found start tag of element author
(8) Found text with value Radha
(9) Found end tag of author
(10) Found end tag of paper
(11) End parsing document

The overall flow of xParse is illustrated in Figure 3.

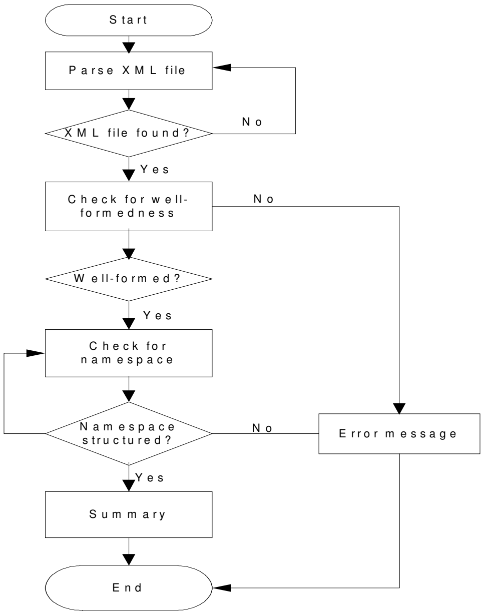


Figure 3. Overview of xParse System Flow

4. Benchmarking and Testing

4.1. Experimental Setup

XML Test 1.0 [15] is used as the benchmarking tools to evaluate the following model of XML parsers:

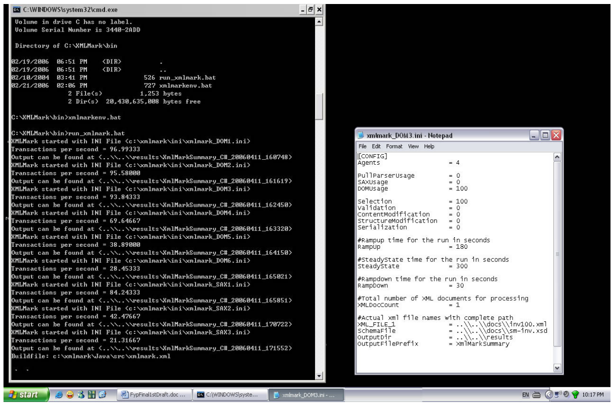
Table 2. XML Parsers Used for Benchmarking

	Xerces 2.6.2	**.NET 1.1**	**xParse**
Tree-based	DOM	DOM	-
Stream-based	SAX	PULL	SAX

Before running the test, the following setting and software are pre-requisite and version we used are hereby stated:

- jdk1.5.0 (Bundled with JAXP which contains Xerces 2.6.2 XML parser)
- Microsoft .NET Framework 1.1
- Jakarta ANT

All experiments are conducted on 1.7 GHz Pentium IV processor with 1.024 GB SDRAM running on windows XP system. Figure 4 shows the testing environment for XML Test 1.0.



```
 Volume in drive C has no label.
 Volume Serial Number is 3440-2ADD

 Directory of C:\XMLMark\bin

02/19/2006  06:51 PM    <DIR>          .
02/19/2006  06:51 PM    <DIR>          ..
02/10/2004  03:41 PM               526 run_xmlmark.bat
02/21/2006  02:06 PM               727 xmlmarkenv.bat
               2 File(s)          1,253 bytes
               2 Dir(s)  20,430,635,008 bytes free

C:\XMLMark\bin>xmlmarkenv.bat

C:\XMLMark\bin>run_xmlmark.bat
XMLMark started with INI File <c:\xmlmark\ini\xmlmark_DOM1.ini>
Transactions per second = 96.99333
Output can be found at <..\\..\\results\XmlMarkSummary_C#_20060411_160748>
XMLMark started with INI File <c:\xmlmark\ini\xmlmark_DOM2.ini>
Transactions per second = 95.58000
Output can be found at <..\\..\\results\XmlMarkSummary_C#_20060411_161619>
XMLMark started with INI File <c:\xmlmark\ini\xmlmark_DOM3.ini>
Transactions per second = 93.84333
Output can be found at <..\\..\\results\XmlMarkSummary_C#_20060411_162450>
XMLMark started with INI File <c:\xmlmark\ini\xmlmark_DOM4.ini>
Transactions per second = 69.64667
Output can be found at <..\\..\\results\XmlMarkSummary_C#_20060411_163320>
XMLMark started with INI File <c:\xmlmark\ini\xmlmark_DOM5.ini>
Transactions per second = 38.89000
Output can be found at <..\\..\\results\XmlMarkSummary_C#_20060411_164150>
XMLMark started with INI File <c:\xmlmark\ini\xmlmark_DOM6.ini>
Transactions per second = 28.45333
Output can be found at <..\\..\\results\XmlMarkSummary_C#_20060411_165021>
XMLMark started with INI File <c:\xmlmark\ini\xmlmark_SAX1.ini>
Transactions per second = 84.24333
Output can be found at <..\\..\\results\XmlMarkSummary_C#_20060411_165851>
XMLMark started with INI File <c:\xmlmark\ini\xmlmark_SAX2.ini>
Transactions per second = 42.47667
Output can be found at <..\\..\\results\XmlMarkSummary_C#_20060411_170722>
XMLMark started with INI File <c:\xmlmark\ini\xmlmark_SAX3.ini>
Transactions per second = 21.31667
Output can be found at <..\\..\\results\XmlMarkSummary_C#_20060411_171552>
Buildfile: c:\xmlmark\Java\src\xmlmark.xml
```

```
[CONFIG]
Agents                  = 4

PullParserUsage         = 0
SAXUsage                = 0
DOMUsage                = 100

Selection               = 100
Validation              = 0
ContentModification     = 0
StructureModification   = 0
Serialization           = 0

#Rampup time for the run in seconds
RampUp                  = 180

#SteadyState time for the run in seconds
SteadyState             = 300

#Rampdown time for the run in seconds
RampDown                = 30

#Total number of XML documents for processing
XMLDocCount             = 1

#Actual xml file names with complete path
XML_FILE_1              = ..\\..\\docs\\inv100.xml
SchemaFile              = ..\\..\\docs\\sm-inv.xsd
OutputDir               = ..\\..\\results
OutputFilePrefix        = XmlMarkSummary
```

Figure 4. XML Test 1.0 Testing Environment

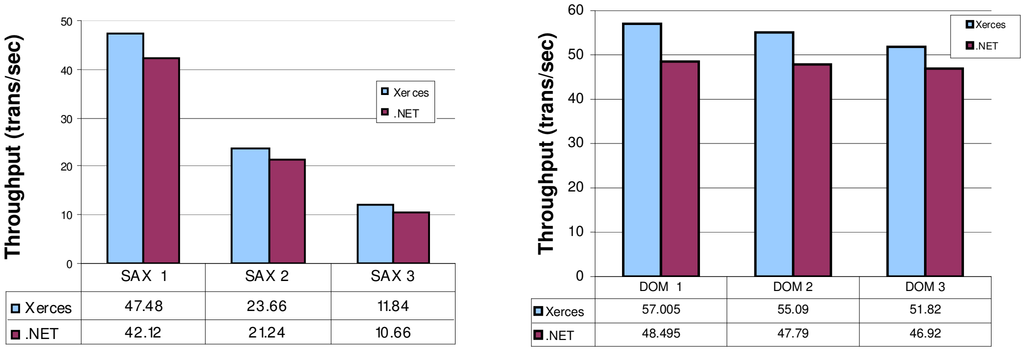
4.2. Results and Discussion

To analyze the benchmark results, we group them into four sets of test cases as below:

- SAX/Pull test – Compare the performances of the streaming based parsers, using 1000 transactions XML invoice document (about 900KB).
- DOM test (without serialization) – Compare the performances of DOM based parsers, using 100 transactions XML invoice document (about 90KB).
- XML parsers comparison – Compare the parsing time against various file sizes for DOM and SAX based parsers of Xerces Java, .NET and xParse using the modified invoice dataset.
- xParse performance analysis – Measuring the parsing time of xParse against various file sizes using Orders dataset obtained from University Washington repository [16].

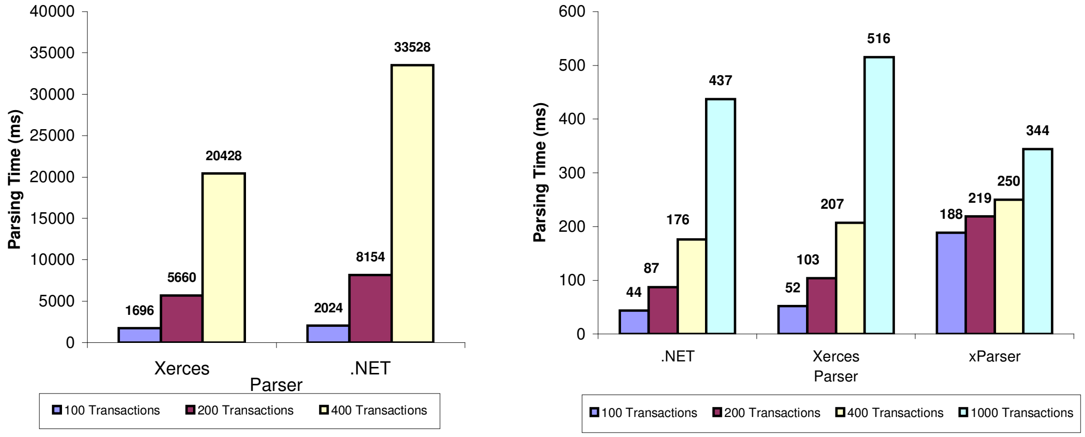
Figures 5 to 7 depict the test results. In Figures 5(a) and 5(b), the testing results are obtained by using the following setting. For SAX 1 and DOM 1 configuration respectively, the percentage of selection is 25%, followed by SAX 2 and DOM 2, where percentage of selection is 50% and SAX 3 and DOM 3, which is set to 100%. Selection is percentage of lineitems retrieved in the access phase (a phase which data is extracted from the elements and attributes of parts of the document into the application program). From the results, Xerces SAX parser outperforms .NET Pull parser.

In addition, the parsing time for DOM and SAX based parsers are shown in Figure 6(a) and 6(b) respectively. No matter parsing invoice document of 100 transactions (90KB), 200 transactions (181KB) or 400 transactions (356KB), Xerces Java emerges to be the fastest DOM based parser. Interestingly, .NET pull parser performs better for the SAX



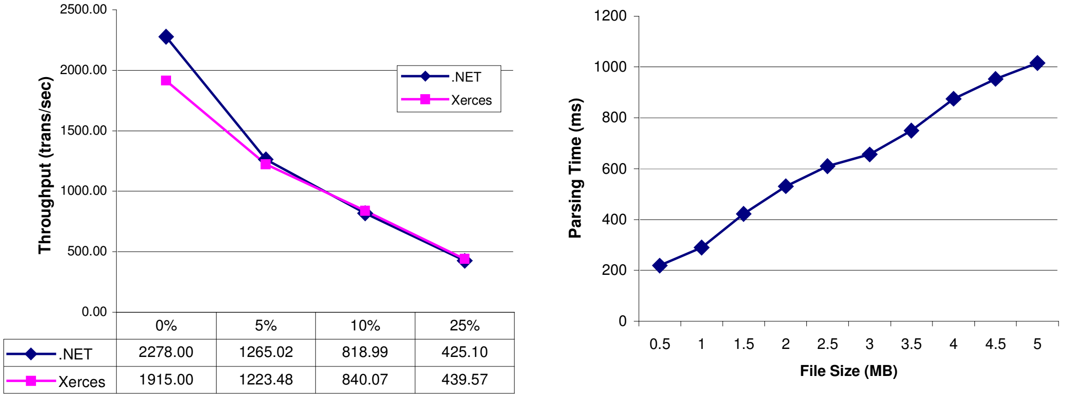
(a) **(b)**

Figure 5. Test Results on (a) SAX/Pull Test (b) DOM Test



(a) **(b)**

Figure 6. Test Results on (a) DOM Parsers' Performance (b) SAX Parsers' Performance



(a) **(b)**

Figure 7. Test Results on (a) SAX Parsers' Performance by Altering the Selection Criteria (b) xParse Parsing Time

parser category as compared to Xerces and xParse. This is because .NET pull parser has the ability to skip over unwanted content if involving parsing phase only [17]. However, xParse performs impressively when the file size is large. Thus, it can support large-scale dataset efficiently. On the other hand, in Figure 7(a), we observed that Xerces Java outperformed .NET when the selection is set to 10% onwards. Figure 7(b) shows the parsing time for xParse increase linearly against increasing file size of XML document as compared to most other parsers which increase drastically against the increasing file size (as shown in Figure 6(b)). Hence, xParse parser is suitable for supporting the large-scale dataset.

From the result summarized, it is clear that Xerces outperforms .NET parser from most of the test cases carried out. However, if parsing large-scale dataset is required, xParse may be the best choice.

5. Conclusions and Future Work

There have been a handful of studies and researches towards XML parsers. Nevertheless, most of them are not up to date. As XML parser is a technology, which is changing rapidly for the moment, there is no single study or research that would valid forever.

The result of the study indicates that Xerces Java has been the best parser in terms of performance. However, xParse outperformed in terms of supporting large-scale of dataset efficiently. Nevertheless, performance is not the only criteria; there are lots of factors to be considered when choosing XML parser, such as organization's need, API support, platforms and license fees.

Since the testing and benchmarking only involve in evaluating two most popular parsers, the .NET and Xerces parser, the study can be further extended in future. Some of the future approaches could includes 1): Compare the performance of new and established APIs DOM, SAX, StAX, Pull or electric XML together in a set of benchmarking tool, and 2): Compare and study the conformance of parsers to some of the new features such as support to new APIs and eXtensible Stylesheet Language Transformation (XSLT) ability.

Acknowledgments

We would like to thank Tzy Yee Yeoh and Chen Wai Lor for partially supporting this study.

References

[1] S.C. Haw and G.S.V.R.K.Rao, "Query Optimization Techniques for XML Databases", International Journal of Information Technology, Vol. 2, No. 1, 2005, pp. 97-104.

[2] Nicola, M. and John, J., "XML Parsing: a Threat to Database Performance" International Conference on Information and Knowledge Management, 2003, pp. 175-178.

[2] Slominski, A., "Design of a Pull and Push Parser System for Streaming XML", Indiana University, Technical Report TR550, 2001.

[4] Zisman, A., "An Overview of XML", Computing & Control Engineering Journal, 2000.

[5] Karre, S. and Elbaum, S., "An Empirical Assessment of XML Parsers", *6th* Workshop on Web Engineering, 2002, pp. 39-46.

[6] Michael, C., "XML Parser Comparison", 2000, http://www.webreference.com/xml/column22/2.html

[7] Tong, T. et al, "Rules about XML in XML", Expert Systems with Applications, Vol. 30, No.2, 2006, pp. 397-411.

[8] Kiselyov, O., "A better XML parser through functional programming", LNCS 2257, 2002, pp. 209-224.

[9] Elliotte, R.H., "SAX Comformance Testing", XML Europe, *2004*

[10] J. Anez, "Java XML Parsers- A Comparative Evaluation of 7 Free Tools", Java Report Online, 1999.

[11] Mohseni, P., "Choose Your Java XML Parser", 2001, http://www.devx.com/xml/Article/16921

[12] Sosnoski, D.M., "XMLBench", 2005 http://www.sosnoski.com/opensrc/xmlbench

[13] XMLJ News Desk, "Journal Readers choice Award", 2004 http://xml.sys-con.com/read/44008.htm

[14] XML 1.0 Specification, W3C, http://www.w3.org/TR/REC-xml/

[15] XML Performance Team, "XML Test 1.0", 2005 http://java.sun.com/developer/codesamples/webservices.html#Performance

[16] University of Washington Repository, http://www.cs.washington.edu/research/xmldatasets/

[17] Sun's white paper, http://java.sun.com/performance/reference/whitepapers/XML_Test-1_0.pdf