IBP: An Index-Based XML Parser Model

Haihui Zhang[1], Xingshe Zhou [1], Yang Gang[1], and Xiaojun Wu [2]

[1] College of Computer Science, Northwestern Polytechnical University,
Xi'an, Shaanxi, China, 710072
zhh409@tom.com, zhouxs@nwpu.edu.cn, yang.gang@gmai.com
[2] College of Information Engineering, Chang'an University,
Xi'an, Shaanxi, China, 710064
depender@yahoo.com

Abstract. With XML widely used in distributed system, the existing parser models, DOM and SAX, are inefficient and resource intensive for applications with large XML documents. This paper presents an index-based parser model (IBP), which contains validation and non-validation modes, supports nearly all the XML characteristics. IBP has the characters of speediness, robustness and low resource requirement, which is more suitable for mass information parsing. We presents the application of IBP in a real-time distributed monitoring prototype system, the results have shown IBP effective.

1 Introduction

XML (Extensible Mark-up Language) is a meta-language to describe other markup languages. Since it appeared, XML has got greatly development and become standard of information exchange. Apart from traditional documents, it also comprises textual files describing graphical objects, transactions, protocol data units and all other kind of imaginable structured data. Despite XML's well known advantages, it has one key disadvantage: document size. Indeed, the XML standard explicitly states that markup terseness was *not* a design goal. Consequently, XML documents can be many times larger than equivalent non-standardized text or binary formats [2].

XML-conscious compression techniques have been well researched, XMLZIP, by XML Solutions [3], and Liefke and Suciu's XMILL[1], of which we are aware, also recently the Wireless Access Protocol standard group's Binary XML Content Format (WBXML) [4] and MHM[5] which based on Prediction by Partial Match (PPM). They are designed to reduce the size of XML documents with no loss of functionality or semantic information.

In a large-scale distributed system, it is always needed to exchange information with some applications that adopt original XML expression, so compression is not for all occasions. As Matthias[6] pointed out that in real-world experiences of using XML with databases, XML parsing was usually the main performance bottleneck. XML parser takes a most important role in XML applications. There are two models of XML parsers, DOM and SAX, which are proved to be ineffective for large XML database. Another feasible measure is to optimize the XML parser.

In our research, we develop Index-based Parser Model (IBP), which support almost whole XML specifications. With it, we can get element from the large XML file by

H. Jin, D. Reed, and W. Jiang (Eds.): NPC 2005, LNCS 3779, pp. 65 – 71, 2005.

its' multilevel index. It has the characteristic that parsing cost does not increase with the document size. Especially when parsing large XML file, it can get significant performance improvement compared to SAX and DOM.

2 Existing XML Parser Models

A XML parser is a basic but also very important tool. Publicly available parsers in use today, such as IBM's XML4J, Microsoft' MSXML, Oracle's XML Parser, Sun's Java™ Project X and some open source code parsers such as Expat, OpenXML, Xerces, SXP. Most XML parsing libraries use one of two interfaces, Simple API for XML (SAX) [7] and Document Object Model (DOM) [8]. XML parsing allows for optional validation of an XML document against a DTD or XML schema, so classified with validation and non-validation.

2.1 Document Object Model (DOM)

DOM is a standard tree-based API specification and under constant development by the Document Object Model working group at the W3C (World Wide Web Consortium). A tree-based parser parses and compiles an XML document into an internal, in-memory tree structure that represents the XML document's logical structure, which then is made available to the application. The current version is DOM Level 2 proposed recommendation [9], which is a platform-and-language-neutral interface that allows applications to dynamically access and update the content and structure of documents. DOM does allow applications to perform tree operations such as node additions, modifications, conversions and removals.

Since a DOM parser constructs an internal tree representation of the XML document content in main memory, it consumes memory proportional to the size of the document (2 to 5 times, hence unsuitable for large documents) [6]. Also, when a DOM parser constructs a tree, it will take account of all objects such as elements, text and attributes. But if applications only pay attention constantly to small proportion of total objects, the resource occupancy by which rarely or never used is striking. Lazy DOM parsers materialize only those parts of the document tree that are actually accessed, if most the document is accessed, lazy DOM is slower than regular DOM.

2.2 Simple API for XML (SAX)

SAX is a simple, low-level event-based API specification [7] and developed collaboratively by the members of the XML-DEV mailing list, hosted by OASIS. SAX 2.0 was released on the 5th of May 2000, and is free for both commercial and non-commercial use. SAX reports parsing events (such as the start and end of elements) directly to the application through callbacks. The application uses SAX parser to implement handlers to deal with different events. The memory consumption does not grow with the size of the document and it is possible to get the desired data without parsing the whole document. In general, applications requiring random access to the document nodes use a DOM parser while for serial access a SAX parser is better.

On the other hand, SAX has some disadvantages that restrict its application. SAX events are stateless, so, it may result in repeatedly parsing of the document to get multi-elements that scattered in the file. More serious is that events are only used to find the

elements, applications must maintain lots of specially callback handles which are full of IF/ELSE structures. SAX is read only and difficult to carry out complex query.

3 Index-Based Parser Model (IBP)

Through our experiences of using XML with databases, we noted that XML documents contain one or more key tags just as index in relational database. So we introduce the index mechanism to the parsing process. There is an initial operation before the operation on XML document, during which key tag index tables and sub-tree index tables will be built. After that, IBP allow applications to perform operations based on these tables.

The following example shows the basic process of our IBP method. This is a simple XML document (named BookSet.xml) with DTD statements.

```
<?xml version="1.0" encoding="GB2312" ?>
<!ELEMENT BookSet (Book*)>
<!ELEMENT Book(ISBN, Name, Author+, Price*)>
<!ELEMENT ISBN(#PCDATA)>
<!ELEMENT Name(#PCDATA)>
<!ELEMENT Author(#PCDATA)>
<!ELEMENT Price(#PCDATA)>
<!ATTLIST Price currency (dollar | RMB | pound) 'Dollar'>
```

The IBP parser parses XML data and breaks the structural tree into many sub-trees with specified elements. In this document the best choice of element is 'Book', just logging all the positions of start tags (<Book>) and end tags (</Book>) to form sub-tree index table. If we know which sub-tree contains what we want, just search this one. How to know the sub-tree is just using the key tag index table. We can create index with any tag we needed, but the elements with exclusive text value are

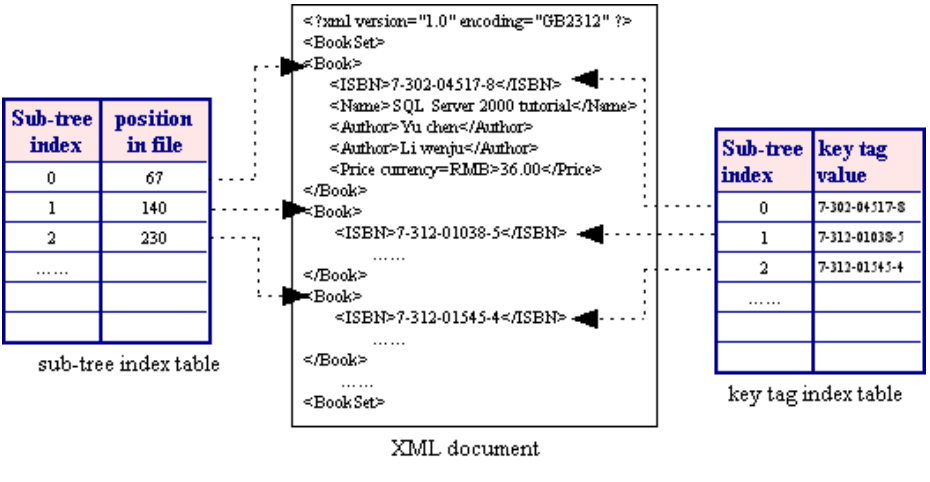


Fig. 1. Index tables after initial

recommendation. Therefore we take ISBN as the key tag and log the text between '<ISBN>' and '</ISBN>' as the value. After initial process, IBP builds the sub-tree index table and key tag index table (see Fig 1).

Now, if we want to find the author of a book with ISBN as '7-302-04517', we only load and search the fragment of document from position 67 to 140, just a string with length of 73.

3.1 Non-validation IBP

Non-validating parsers are only able to check if an XML document is well formed, whereas validating parsers can additionally check if the XML document conforms to its DTD (Document Type Definition).

IBP implements both modes. Non-validation mode of IBP (named IBP-nv), which works as above-mentioned, with which the XML document is read only. IBP-nv is suitable to lookup the XML database such as search tools and filters. IBP-nv is competent for the situation where SAX is used, but more efficient than SAX for its directly acquiring the special sub-tree by index tag, whereas SAX must parse the document at the beginning of the file.

After initial process, the optional operation can close the file handler, and reopen the file and read specified zone according to the sub-tree index table when needed. IBP-nv occupies very little memory, and it does not increase with the increase of file size.

3.2 Validation IBP

Validation mode of IBP (named IBP-v) does allow applications to perform updating operations such as node addition, modification, conversion and removal. In this mode, XML document is kept in memory, and the sub-tree index table is constructed in another way (see Fig 2).

Sub-tree index	Memory pointer	Size	Reallocated
0	0x0012fd40	73	False
1	0x0012fdb3	90	False
2	0x0012fd43	83	False
……			

Fig. 2. Sub-tree index table in IBP-v

After initial process, the XML document locates in a consecutive memory area. If an element needs to be modified, contents of the sub-tree must be copied to another reallocated memory first, and then change the sub-tree's memory pointer to the new address, and evaluate the 'Reallocated' with 'True' after update the element. The key of this mode is that a sub-tree is taken as an allocated memory unit, and memory

pointer permanently points to the latest memory area. Before closing the file, all sub-tree will write back to the disk one by one according to the index.

Which one (IBP-nv or IBP-v) will be used in application depends on whether the XML need to be updated. If only parsing to search document, IBP-nv is recommended. The cost of validation, reparsing or revalidating a full document as part of a small update is sometimes unacceptable. After much experiment work, we find that parsing even small XML documents with validation can increase the CPU cost by 2 to 3 times or more.

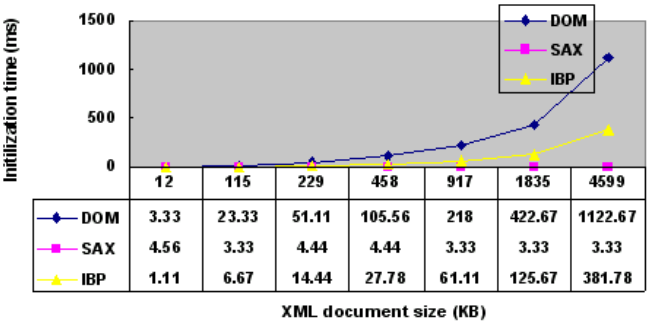
4 XML Parser Performance

Parser performance always depends on document characteristics such as tag-to-data ratio, heavy vs. light use of attributes [6], amount of sub-trees vs. average size of sub-tree, etc, but we do not strive to quantify these dependencies here. In our project, we have developed a prototype system, which has the above-mentioned features. Our goal is to relate the cost of Microsoft's MSXML4.0, which is used in windows system widely, and support SAX and DOM parsers.

We still take BookSet.xml as our example, its DTD statements has been given above. First, we create 7 XML documents of 12KB, 115KB, 229KB, 458KB, 917KB, 1835KB, 4599KB, which contain different amount of elements. Then, we make quantitative analysis of the cost of parser initialization and parsing time, using MSXML4.0 and IBP respectively.

4.1 Analysis of Initialization Time

XML documents are initialized 5 times with DOM, SAX, and IBP parsers. The average times are listed in Fig 3.



	12	115	229	458	917	1835	4599
DOM	3.33	23.33	51.11	105.56	218	422.67	1122.67
SAX	4.56	3.33	4.44	4.44	3.33	3.33	3.33
IBP	1.11	6.67	14.44	27.78	61.11	125.67	381.78

Fig. 3. Initialization times of DOM, SAX and IBP

In initial process, DOM needs to parse and compile a XML document into an internal, in-memory tree, the initial time is proportional to the size of the XML document. SAX only creates a handle to open the file, but does not read any data, so the cost is the least. IBP needs to parse and build index tables, so it takes more time.

4.2 Parsing Cost

The 7 XML documents are parsed 1,000 times, without grammar caching. Average cost of 1000 times' random parsing can reflect moderately the parser's performance.

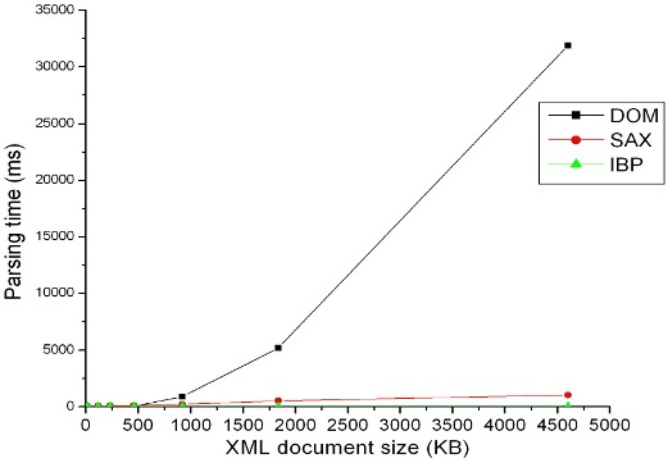


Fig. 4. Average parsing performance (1000 times)

DOM parser performs very well on small XML document, but cost for large document exceeds the limit of our tolerance. SAX parser's performance is better than DOM, but it is still not very efficient. After further research, we find DOM parser and SAX parser have the approximately same costs when the XML document size is 720KB. During the IBP parser initialization process, it creates sub-tree index table and key tag index table, so it's parsing process actually contain two operations: lookup the tables and then matches element in special sub-tree. In IBP, index tables are optimized with hash function, so the lookup time is almost invariable. And the searching time in a sub-tree is only related with the tree' size. Therefore, IBP parser has high performance even for very large XML documents.

In the worst case, IBP need to be re-initialized to build index table on a new tag. We compare the initialization and parsing time of IBP with the parsing time of DOM and SAX. IBP still has a better performance than DOM and SAX.

Document size (KB)	DOM (ms)	SAX(ms)	IBP(ms)
12	10	15.56	11.11
115	10	43.33	16.67
229	10	64.44	24.44
458	50	114.44	37.78
917	901	213.33	71.11
1835	5188	523.33	135.67
4599	31875	1034.33	391.78

Fig. 5. Average performance (1000 times): initialization and parsing time of IBP vs. parsing time of DOM and SAX

5 Future Work

Future research directions concern both theoretical and practical aspects of the research carried out in this paper. We plan to keep on developing IBP API for full XML characteristic and research more flexible parsing model. Another interesting direction is tighter integration of database system with IBP.

6 Summary

We discuss common XML parser models, DOM and SAX, point out they are not suitable to large XML documents. Aiming at large-scale distributed systems, we present a new parser model called IBP, which has low resource requirement and good performance, regardless of the XML document size. For large XML files, IBP parses much more faster than DOM and SAX. Via API with C++, IBP can be widely used in various kinds of applications with low parsing time cost, and present a new idea for XML parsing.

References

1. H. Liefke and D. Suciu. Xmill. an efficient compressor for XML data. In Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, pages 153-164,2000.
2. James Cheney, Compressing XML with Multiplexed Hierarchical PPM Models, ICDE2001
3. XML Solutions. XMLZIP. http: //www.xmls.com/.
4. WAP Binary XML Content Format, W3C NOTE 24 June 1999, http://www.w3.org/TR/wbxml/
5. James Cheney, Compressing XML with Multiplexed Hierarchical Models, in Proceedings of the 2001 IEEE Data Compression Conference, pp. 163–172.
6. Matthias Nicola and Jasmi John. XML Parsing: A Threat to Database Performance, CIKM'03, November 3–8, 2003
7. Megginson, David, SAX 2.0: The Simple API for XML, http://www.megginson.com/SAX/
8. Extensible Markup Language (XML). 1.0 W3C Recommendation 10-Feb-98. http://www.w3.org/TR/1998/REC-xml-19980210.pdf.
9. World Wide Web Consortium. Extensible Markup Language (XML) 1.0 (Second Edition).http://www.w3.org/TR/2000/REC-xml-20001006.
10. Quanzhong Li and Bongki Moon, Indexing and Querying XML Data for Regular Path Expressions. Proceedings of the 27th International Conference on Very Large Databases (VLDB'2001), pages 361-370, Rome, Italy, September 2001.