Parallel Processing Model for XML Parsing

Adriana Georgieva and Bozhidar Georgiev  
Technical University of Sofia, Sofia 1000, Bulgaria

Received: May 14, 2012 / Accepted: June 15, 2012 / Published: November 30, 2012.

Abstract: This paper presented some development problems and solutions concerning the parallel implementation of an algebraic method for XML data processing. The proposed parallel algorithm first partitions the XML document into chunks and then apply the parallel model to process each chunk of XML tree. The authors suggest a different point of view about XML parsers with the creation of advanced algebraic processor (including all necessary software tools, search techniques and programming modules). The possibilities of this linear algebraic model, combined with principles of parallel programming allow efficient solutions for parsing, search and manipulation over semi-structured data with hierarchical structures. Thus presented paper combines the building of an algebraic formalism for navigation over XML hierarchy with concepts of modern XML parser and their mutual work in parallel. The presented tests show higher rapidity and low consumption of resources in comparison with some existing commercial XML parsers.

Key words: XML transformations, algebraic modeling of XML structures, RP (reversal parser).

1. Introduction

With the advent of the information age and the ubiquitous use of the Internet, there is an unprecedented demand for effective and efficient techniques for data processing. XML is going to become the standard document format and with the use of XML query languages, users of XML retrieval systems are able to exploit the structural nature of the data and restrict their search to specific structural elements within an XML documents.

The current XML implementation landscape does not adequately meet the performance requirements of large scale applications. The applications using Web services have rather focused on XML protocol standardization and tool building efforts, and not on addressing the performance bottlenecks when dealing with large volumes of XML data. XML documents have some structural properties that make it more dependent on parallelized parsing than general context-free languages. XML parsers spend a large percentage of time tokenizing the input in an inherently serial process. In this article, the authors have been made efforts to parallelize XML parsing process. Recently, many XML researchers explore new techniques for parallelizing parsers for very large XML documents [1-2]. When thinking about a multithreaded solution it is necessary to consider at least the following strategies or some mixtures of them:

1. Creating multiple parsers and running them in parallel on the XML sources;
2. Rewriting parsing algorithms thread with the main goal to use safety only one instance of the parser;
3. Split the XML source into chunks and assign the chunks to multiple processing threads.

This paper proposes an approach to parallelize XML parsing process, where the XML document is split into fragments (two or more) and the parser works on different fragments in parallel. This model is well suited as for multi-core processors [2] as well as for multi-threading programming.

Bozhidar Georgiev, Ph.D., associate professor, research fields: software engineering, web technologies.

Corresponding author: Adriana Georgieva, Ph.D., associate professor, research fields: universal algebra, data modeling, XMLDB, DBMS. E-mail: adig@tu-sofia.bg.
The prevalence of large XML documents is another motivation for researchers in their efforts to optimize and parallelize XML parsers [3]. At the same time, multi-core processing is increasingly becoming available on desktop- and laptop-class computing machines. Parallelizing input documents into multiple threads is the key to performance improvement of XML parsing process.

The rest of the paper is organized as follows: Section 2 introduces the dividing process of the whole XML document into chunks and the following parallel handling of XML document. Here is explored reversal approach for speed up the XML parser mechanisms. Sections 3 describes in detail the previously suggested (from the same authors) functional XML parser [1, 2, 4, 5, 6]. Section 4 presents some XML parser architectures and program realizations along with an algebraic search and hierarchy access [4]. Finally, in Section 5, the general issues, conclusions, the further researches and some open problems are discussed.

The last section gives the performance evaluation results and makes a brief comparison with some similar approaches.

2. XML Documents Processing

The authors’ first step is by means of chunk partition to divide the whole or part of input XML document into several of approximately equal-sized chunks. The chunk size can be settled at run time and as a rule each chunk should be big enough to minimize the number of chunks and reduce the post processing manipulations. Actually, every XML document can be represented by a tree. Here is presented example of XML fragment:

```xml
<bank>New Bank
  <branches>Branch 1
    <clients>
      <assets>Account A</assets>
      <assets>Account B</assets>
    </clients>
  </branches>
  <branches>Branch 2
    <clients>
      <assets>Account C</assets>
      <assets>Account D</assets>
    </clients>
  </branches>
  <branches>Branch 3
    <clients>
      <assets>Account E</assets>
      <assets>Account F</assets>
    </clients>
  </branches>
</bank>
```

The hierarchical tree, corresponding to this XML code is shown below on Fig. 1:

![Hierarchical tree presentation of XML input file.](image)

After finishing parsing process, the information for the chunk is not complete. Then the parallel parser must carry out post processing for the parsed chunk, when all preceding chunks are parsed.

After that, the parsed chunks hold the complete infoset information for the corresponding input chunk and can be put into parsed chunk pool to be processed by next stage.

The nodes of XML tree are presented in corresponding parser table with 34 bytes for each node. The structure is following: first 16 bytes are allocated for the name of node; 1 byte is detached for
the type of node (element, attribute data etc.)—Fig. 2. This schema is followed by 1 byte for the level in hierarchy, next bytes 19-22 are allocated for the shifting in the row and last 12 bytes are separated for child nodes of this parent node [7]. Here can be stored three children of each corresponding parent node. As a practical realization of yet presented theoretical research, authors suggest simple parallel RP (reversal parser), based on standard SAX. The reversal parser (in short RP) is a two-streaming, two-way XML parser that begins handling the input XML string from its both sides. Reversal parser starts parsing process following directions from left to right and simultaneously from right to left. The results and analyses are shown in Section 4 to prove the theoretical grounds in so proposed solutions. There is built test schema with different by size and by complexity XML input documents.

3. An Accelerating Navigation over XML Documents

This section proposes an approach, inspired by explored theoretical formalisms [6], which directly addresses XML hierarchical components. This approach, offered by the same authors [3, 4, 5, 7], for extension of data processing possibilities in XML hierarchy [5], is applied here for accelerating navigation over XML documents in the form of hierarchical trees. A main goal of this analysis is to provide modern linear algebra tools for work on the XML document through a direct access to the nodes of appropriate XML tree.

The conceptual model of some hierarchy is presented as an algebraic structure $A = (A_1, A_2, ..., A_n)$—a family of modules $A_i$ over the rings $\alpha_i$, where $\alpha_i$ is the dimension of each database domain $D_i$ ($\alpha_1 < \alpha_2 < ... < \alpha_n$) and $n$ is the number of hierarchical levels. This conceptual model permits to work with the natural numbers only, i.e. the code values of XML database elements. That means simpler physical organization, because a physical address could be calculated for every hierarchical object with a finite sequence of the code values of its attributes. At that the computation is performed by ordinary algebraic operations with integers. This way it is provided an efficient direct access to every element of XML hierarchical data structure. This is very important for data structures organization and support, especially when the authors have parallel processing of XML documents.

The set $A = (A_1, A_2, ..., A_n)$ is considered as an algebraic model of hierarchical data structure with $n$ levels, which is partially ordered by inclusion: $A_1 \subset A_2 \subset ... \subset A_i \subset ... \subset A_n$.

So any object $r$ from level $n$ in this hierarchical structure with its attributes can be represented by the finite sequence $(a_1, a_2, ..., a_n)$ of the set $A$. In this algebraic model of XML data presentation, the transition “conceptual–internal” is defined as the mapping $\Phi: A \rightarrow P$, which correlates to every finite sequence: $(a_1, a_2, ..., a_n) \in A$ in one-to-one manner one fixed integer $p_r^c \in P$. Here the integer $p_r^c$ uniquely defines the place of the object $O_i^c$ from level $n$ in the real structure $M$, i.e. its address in the physical database design $P$.

![Fig. 2 Parallel XML parsing algorithm with FP (functional parser) as post mechanism](image)
For further formal and correct description of data transition “conceptual–internal” is defined the mapping $\Phi : A \rightarrow P$, which confront to any finite sequence $(a_1, a_2, ..., a_n) \in A$ one integer $p_n^r$.

This integer defines in simple way the position of the object $O_{n}^r$ into XML tree. For the physical data representation it is proved, that this mapping $\Phi : A \rightarrow P$, is bijective and linear mapping (linear function) [4-6].

As result of this theoretical model here is given the unique determination of the physical address of the object $O_{n}^r$ (object $r$ from level $k$) in common structure, i.e. the number $p_{k}^r$ by the following way:

$$p_{k}^r = \sum_{i=1}^{k-1} \alpha_i + a_k^l = a_1 + a_2 + ... + a_{k-1} + a_k^l = a_1.\Phi(h_1) + a_1.\Phi(h_2) + ... + a_1.\Phi(h_{k-1}) + a_k^l =$$

$$=a_1.\sum_{i=1}^{k-1} \Phi(h_i) + a_k^l \quad (1)$$

where:

$$\Phi(h_1) = c_0 = 1 \quad ; \quad \Phi(h_2) = c_0.c_1 = c_1 \quad ;$$

$$\Phi(h_3) = c_0.c_1.c_2 = c_1.c_2 \quad ; \quad .... \quad ;$$

$$\Phi(h_k) = c_0.c_1.c_2. ... .c_{k-1} = c_1.c_2. ... .c_{k-1}$$

are the transformed characteristic elements from the tree;

$c_0, c_1, c_2, ... , c_{k-1}$ are the number of children (subordinated elements) of any element from level $i$ to level $i+1$; $c_i \in \mathbb{R}$; ordinary $c_0 = 1$ and

$$a_k^l = \{...(a_i-1).c_i + a_{i+1}-1\}.c_{i+1} + ... +(a_{k-1}-1).c_{k-1} + a_k - 1 \quad (2)$$

Here $a_k^l$ is the code value $a_k$ in the hierarchical level $k$. The calculations in Eq. (2) are based on the formal description of the sets of code values of XML nodes components.

According to suggested formal algebraic description, each of the objects in a real XML hierarchical data structure can be accepted as an element of the corresponding hierarchical structure [4]. For XML physical data design is chosen one-dimensional address array with codes of all XML database elements from the type: $E (i_1 + k_1, i_2 + k_2, ... , i_n + k_n)$, which presents the XML data structure in increasing consistency in the order of the corresponding hierarchical levels. Here, an expression $k_m - i_m$ is a dimension of level $m$ in hierarchy for each $m = 1, 2, ..., n$.

So every object from the real XML hierarchy can be obtain on conceptual level as an element from the algebraic structure (n-sequence), respectively, on physical level as an address from corresponding physical structure (Eq. (3)). It is in dependence of the purposes and the character of the user application with the following representation:

$$f_1 \Phi\quad O_r \quad (a_1, a_2, ..., a_n) \quad \Phi \quad p_n^r \quad (3)$$

Finally, the proposed approach is different in comparison with many other well known query and transformational languages (as XSL, XSLT, XPath) in respect of their definition, expressiveness, and search techniques.

4. Practical Researches and Tests

As has been yet mentioned, practical results include the creation of high performance parallel reversal XML parser, and apply it as main component in parallel parsing strategies as well. The practical realization includes Windows application, which works in program development environment Eclipse 3.2. Actually, Java language is very suitable for the main goals of this research. This object-oriented language possesses set of possibilities for fast thread programming in real time. In this way, Java provides great effectiveness in the process of creating
contemporary multilevel parsers on new hardware systems with chip multiprocessors. In proposed in the article research, is chosen SAX Java Apache Xerces as a comparative parser.

The proposed reversal approach inserts two parsing processes working in parallel. It speeds up the management of huge XML document. Fig. 3 shows results of test examples. The diagram describes the differences between rapidity, while handling with these two types of parsers—classical SAX parser and reversal parser. Fig. 3 also resumes several different cases of file capacity and the corresponding time of parsing. When parallel parser is used, it accelerates the whole process with approximately 45% against traditional SAX parser.

This percentage depends on input file capacity. From Fig. 3, the authors can see that the bigger the XML document size is, the higher speedup of parallel parser can achieve. Because the huge XML document can be split into more subtasks to be parsed in parallel and can maximize the utilization of multi-processors. In so proposed reversal XML parser, simple typical and predictable error situations are resolved by means of Java exceptions software module.

5. Conclusions and Future Works

In this article, are described some possibilities for building a new parallel XML parsing algorithm with great performance. The concept about parallel parser is a possible solution for the acceleration of XML document processing. The embedding of this parallel mechanism in the process of XML parsing will satisfy the need for more effective and faster parser processing. This approach raises the rate of parsing and is especially useful in the parsing procedures for huge input XML documents.

The future work includes initial chunk partition of XML document into more than two chunks, using multiple functional parallel parsers, applying new contemporary algebraic methods, schema validations etc.. The further development of this research foresees to extend these theoretical ideas with practical examples as soon as it is possible. The authors hope that this direction of research is very important to advance query languages development and these new propositions in traditional theory of parsing, translation and compiling will take effect on XML DB and Web practice.

References