ISSN:2321-1091

A SURVEY OF XML PARSING ON MULTI-CORE PROCESSORS

Navreet Kaur, Er. Harwinder Singh Sohal
M-Tech Student , L.L.R.I.E.T. MOGA
navreet.online@yahoo.com
Asst. Prof. (I.T. Dept.), L.L.R.I.E.T. MOGA

ABSTRACT

XML Parallel parsing based upon multicore is a very efficient technique. It will also increase the performance of the system while xml parsing .The idea is to increase the number of threads based upon the CPU cores. By doing this the throughput & the performance of the CPU is increased as making the use of multiple threads on every CPU core. Hence it improves the overall CPU Utilaization & Speed-Up the Data read/write operations.

Indexing terms/Keywords

XML, Parsing, Core,CPU,Processors.

Council for Innovative Research
Peer Review Research Publishing System
Journal of Social Science Research
Vol.4, No.2
editor@jssronline.com
www.cirworld.com, member.www.jssronline.com

INTRODUCTION

Introduction to XML

XML stands for Extensible Markup Language. It is markup languages much like HTML and designed to carry data, not to display data.XML tags are not predefined. You must define your own tags. It is designed to be self-descriptive. By definition, an XML document is a string of characters. Almost every legal Unicode character may appear in an XML document. XML, a formal recommendation from the World Wide Web Consortium (W3C) is similar to the language of today's Web pages, the Hypertext Markup Language (HTML). Both XML and HTML contain markup symbols to describe the contents of a page or file. HTML, however, describes the content of a Web page (mainly text and graphic images) only in terms of how it is to be displayed and interacted with. For example, the letter "p" placed within markup tags starts a new paragraph. XML describes the content in terms of what data is being described. This means that an XML file can be processed purely as data by a program or it can be stored with similar data on another computer or like an HTML file that it can be displayed. For example, depending on how the application in the receiving computer wanted to handle the phone number, it could be stored, displayed or dialed. XML is "extensible" because unlike HTML the markup symbols are unlimited and self-defining. Early application is Chart Ware which uses XML as a way to describe medical charts so that they can be shared by doctors. Applications related to banking, e-commerce ordering, personal preference profiles, purchase orders, litigation documents, part lists and many others are anticipated.

Processor and Application

The processor analyzes the markup and passes structured information to an application. The specification places requirements on what an XML processor must do and not do, but the application is outside its scope. The processor (as the specification calls it) is often referred to colloquially as an XML parser.

Markup and content

The characters making up an XML document are divided into *markup* and *content*, which may be distinguished by the application of simple syntactic rules. Generally, strings that constitute markup either begin with the character < and end with a >, or they begin with the character & and end with a;.

Strings of characters that are not markup are content. However, in a CDATA section, the delimiters <![CDATA[and]]> are classified as markup, while the text between them is classified as content. In addition, the whitespace before and after the outermost element is classified as markup.

Tag

A markup construct that begins with < and ends with >. Tags come in three flavors:

- *start-tags*; for example: <section>
- *end-tags*; for example: </section>
- *empty-element tags*; for example: <line-break />

Element

A logical document component which either begins with a start-tag and ends with a matching end-tag or consists only of an empty-element tag. The characters between the start- and end-tags, if any, are the element's *content* and may contain markup, including other elements, which are called *child elements*. An example of an element is <Greeting> Hello World.</Greeting>. In this example “Hello World” is the content of “Greeting” tag.

Attribute

A markup construct consisting of a name/value pair that exists within a start-tag or empty-element tag. In the example (below) the element *img* has two attributes, *src* and *alt*:

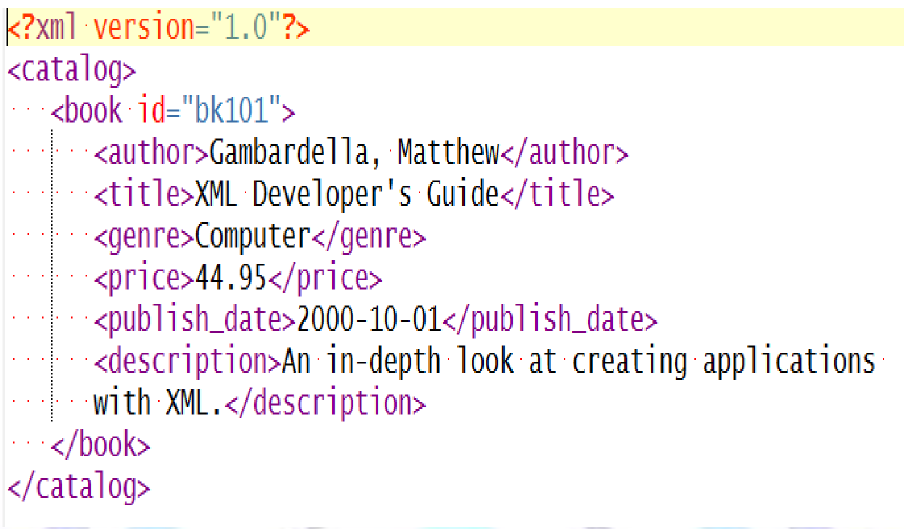
<img src="madonna.jpg" alt='Foligno Madonna, by Raphael' />

Another example would be

<step number="3">Connect A to B.</step>

where the name of the attribute is "number" and the value is "3".

An XML attribute can only have a single value and each attribute can appear at most once on each element. In the common situation where a list of multiple values is desired, this must be done by encoding the list into a well-formed XML attribute with some format beyond what XML defines itself. Usually this is either a comma or semi-colon delimited list or, if the individual values are known not to contain spaces, a space-delimited list can be used.



```
<?xml version="1.0"?>
<catalog>
   <book id="bk101">
      <author>Gambardella, Matthew</author>
      <title>XML Developer's Guide</title>
      <genre>Computer</genre>
      <price>44.95</price>
      <publish_date>2000-10-01</publish_date>
      <description>An in-depth look at creating applications 
      with XML.</description>
   </book>
</catalog>
```

Fig 1: Syntax of XML Code

XML Parsing

Parsing means "reading" the XML file/string and getting its content according to the structure usually to use them in a program. An XML parser is the piece of software that reads XML files and makes the information from those files available to applications and programming languages, usually through a known interface like the DOM.

For example if you have this XML fragment:

```
<root>
   <node1>value1</node1>
   <node2>value2</node2>
</root>
```

You may want to use these values in a data structure:

```
Class Root
   node1: string
   node2: string
```

so that, in the end:

```
Object obj = ClassRoot.new
parse(xml, obj)
puts(obj)
```

Yields would be something like:

```
obj[node1='value1'; node2='value2']
```

There are various ways of XML parsing. The most generic ways is to traverse the root tag and child nodes/tags underneath and store the data in some data structure with proper hierarchy of the input XML documents. There are number of ways to improve XML parsing performance. One approach would be to use job pool for XML segments. In this approach, XML can be split 1nto a number of segments at the second level of XML hierarchy. To execute each segment, the threads would be created based upon the system's processor to use the CPU efficiently, parse XML effectively without any data lose and without decreasing any system performance.

Types of XML parser

1. **Whole document parsers**: These read in a whole XML document in one go and provide the software which uses it a way to traverse the XML "tree" to examine the content. The DOM language-independent model is often used by "whole document" parsers, but other models do exist and there are several APIs for accessing such a model, even only considering Java. It's worth noting that just because a parser reads a whole XML document in one go, doesn't mean that it necessarily provides all possible tree-traversal options. For example, I often use an XML micro-parser (less than 10K of Java class files) to read application configurations and present the same interface as if they had come from a standard Java "properties" file.

2. **Sequential parsers**: These are typified by the SAX API, but there are many others available. To use a parser of this type, you initialize it and then add "callbacks" for some or all of the possible XML tags and content data."Sequential" parsers are especially useful if your application only needs to extract specific, information from a potentially large XML document. Unlike "whole document" parsers, a "sequential" parser never has to hold the whole document in memory, just the bits it's interested in.

3. **Query Based**: This is the modern and fasted technique of parsing. This feature is provided by .Net frame work is known as LINQ. LINQ to XML was developed with Language-Integrated Query over XML in mind from the beginning. It takes advantage of standard query operators and adds query extensions specific to XML. From an XML perspective, LINQ to XML provides the query and transformation power of XQuery and XPath integrated into .NET Framework languages that implement the LINQ pattern (for example, C#, Visual Basic and so on.). This provides a consistent query experience across LINQ enabled APIs and allows you to combine XML queries and transforms with queries from other data sources.

RELATED WORK

Yinfei Pan et. al [1] proposed that a number of techniques to improve the parsing performance of XML has been developed. Generally however these techniques have limited impact on the construction of a DOM tree which can be a significant bottleneck. Meanwhile the trend in hardware technology is toward an increasing number of cores per CPU. As shown in the previous work these cores can be used to parse XML in parallel resulting in significant speedups. They introduced a new static partitioning and load-balancing mechanism. By using a static, global approach they reduced synchronization and load-balancing overhead. Thus improving performance over dynamic schemes for a large class of XML documents. Their proposed work approached leverages libxml2 without modification which reduces development effort and shows that their approach is applicable to real-world, production parsers.

Zacharia Fadika et. Al [2] described an about the use of XML as the data format for many distributed scientific applications, with the size of these documents ranging from tens of megabytes to hundreds of megabytes. Their earlier benchmarking results revealed that most of the widely available XML processing toolkits do not scale well for large sized XML data. A significant transformation is necessary in the design of XML processing for scientific applications so that the overall application turn-around time is not negatively affected. They presented both a parallel and distributed approach to analyze how the scalability and performance requirements of large-scale XML-based data processing can be achieved. They adapted the Hadoop implementation to determine the threshold data sizes and computation work required per node, for a distributed solution to be effective. They also presented an analysis of parallelism using our PIXIMAL toolkit for processing large-scale XML datasets that utilizes the capabilities for parallelism that are available in the emerging multi-core architectures. Multi-core processors are expected to be widely available in research clusters and scientific desktops and it is critical to harness the opportunities for parallelism in the middleware instead of passing on the task to application programmers. Their parallelization approach for a multi-core node is to employ a DFA-based parser that recognizes a useful subset of the XML specification and convert the DFA into an NFA that can be applied to an arbitrary subset of the input. Speculative NFAs are scheduled on available cores in a node to effectively utilize the processing capabilities and achieve overall performance gains. They evaluate the efficacy of this approach in terms of potential speedup that can be achieved for representative XML data sets.

Wei Lu et. Al [3] described 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. However the processing of XML documents has a reputation for 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. In this paper they presented a seemingly quixotic but novel approach: parallel XML parsing. Parallel XML parsing leverages the growing prevalence of multicore architectures in all sectors of the computer market and yields significant performance improvements. This paper presented the design and implementation of parallel XML parsing. Their design consists of an initial preparsing phase to determine the structure of the XML document followed by a full parallel parse. The results of the preparsing phase are used to help partition the XML document for data parallel processing. The parallel parsing phase is a modification of the libxml2. XML parser shows that their approach applies to real-world, production quality parsers. Their empirical study shows the parallel XML parsing algorithm can improved the XML parsing performance significantly and scales well.

V.M. Deshmukh and G.R. Bamnote [4] described that extensible markup language XML has become the de facto standard for information representation and interchange on the Internet. As XML becomes widespread it is critical for application developers to understand the operational and performance characteristics of XML processing. The processing of XML documents has been regarded as the performance bottleneck in most systems and applications. XML parsing is a core operation performed on an XML document for it to be accessed and manipulated. XML processing occurs in four stages: parsing, access, modification and serialization. Parsing is an expensive operation that can degrade XML processing performance.

Adriana Georgieva and Bozhidar Georgiev [5] 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 suggested 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.

Ms. V.M.Deshmukh, Dr. G.R.Bamnote [6] described 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, they presented 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 and queue. All these data structures are linear in nature. They evaluate the data parsing behavior and study architectural characteristics. They 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.

CONCLUSIONS

In Generic methodologies of XML parsing the whole XML is read at once. Then the top root tag of the XML is looked, parse the attribute and element values for tags available underneath of root tag. This technique is quite expensive when there is large and huge number of XML files to parse and process. There are researches going on to speed-up the XML parsing by using the fixed number of threads to parse the different stages of the XML. In that approach XML parsing could be divided into a number of stages. Each stage would be executed by a different thread. This approach may provide speedup, but software pipelining is often hard to implement well, due to synchronization, load-balance and memory access costs. More promising is a data-parallel approach. Here, 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 approach has also many limitations; as if the XML is too big then the chunk count will be very big. So there is a need to create so many threads for those. Overall it will slow down the system performance. Also if we will fix the thread count then CPU may not be used properly.
Example let’s say there are 4 threads run in parallel then:

1. In case of single core, it may reduce the performance.
2. In case of multi-core, CPU may not be utilized properly.

So from the above literature it has been observed to create the threads based upon the CPU’s cores. The numbers for threads creation per CPU core are set and multiply it with the number of cores to run the threads parallel. By doing this XML parsing directly depend upon CPU performance. It will be much faster than the existing techniques and will not reduce the throughput. XML Parallel parsing based upon multicore is the very efficient technique. It will also increase the performance of the system by improving the throughput & CPU utilization while xml parsing. It also Speed-Up Data read/write operations

REFRENCES

[1] Yinfei Pan, Wei Lu, Ying Zhang and Kenneth Chiu *“A Static Load-Balancing Scheme for Parallel XML Parsing on Multicore CPUs”* CCGRID '07 Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid Pages 351-362

[2] Zacharia Fadika, Michael R. Head and Madhusudhan Govindaraju” *Parallel and Distributed Approach for Processing Large-Scale XML Datasets”* in Computer Science Department, Binghamton University P.O. Box 6000, Binghamton, NY 13902-6000, USA.

[3] Wei Lu, Kenneth Chiu “*A Parallel Approach to XML Parsing”* Computer Science Department, Indiana University 150 S. Woodlawn Ave. Bloomington, IN 47405, US

[4] V.M. Deshmukh and G.R. Bamnote *“Design & development of an Efficient XML Parsing Algorithm”* International Journal of Applied Science and Advance Technology January-June 2012, Vol. 1, No. 1, pp. 5-8.

[5] Adriana Georgieva and Bozhidar Georgiev *“Parallel Processing Model for XML Parsing*” in Journal of Communication and Computer 9 (2012) 1258-1262.

[6] Ms. V.M.Deshmukh, Dr. G.R.Bamnote “*XML Parsing using Various Data Structures”* International Journal of Computer Science and Applications, Vol. 6, No.2, Apr 2013 ISSN: 0974-1011.PP 400-405.