XML Parsing on Multicore Processors and Data Representation in .NET Tree Control

Navreet Kaur
M-Tech. (CSE)
LLRIET, Moga

Harwinder Singh Sohal
Assistant Professor, IT Deptt
LLRIET, Moga

ABSTRACT
The purpose of this research is to optimize the parsing process of the XML files. There are several ways to parse the XML files. But to comply with the advanced multicore CPUs and their fast performance the XML parsing logics need to be refined and optimized with parallel processing approach. The parallel XML parsing is a step towards this approach. It makes the reading of XML data faster because parser runs on more engines to extract the data. There are several advantages of parallel XML parsing like fast execution, high throughput, time saving, proper CPU utilization and load balancing. To perform the parsing processes simultaneously, the XML files need to be split in small uniform portions. Now it will execute the parsing logic on multiple threads on each CPU's core to parse the each portion of XML file without interfering with each others. In other words, an each segment will be an input to the parser running on different threads on different CPU cores. To enhance the system performance the multicore processors based devices have been introduced. Such system's processing is much faster than conventional sequential processing systems especially when it does repetitive calculations on vast amounts of data. This technique becomes more important when a candidate system or development application is model based application which operates on the XML files. This approach plays a significant role to enhance the application's capability to process large amount of data, improve application performances by providing quick results and eventually expeditious the application processing and dependent operations.

Keywords
XML, Parsing, CPU

1. INTRODUCTION
XML stands for Extensible Markup Language. It is a markup language which looks like HTML file. The difference between XML and HTML is, it is designed to carry the data and to transport the data to the client application whereas HTML is used to display the data only. In XML the tags are not predefined the user can define the tags according to the requirement. In case of HTML the tags are predefined. The XML is the technique which is used to create the common information. Both XML and HTML contains markup symbols to describe the contents of the web pages or files. It is very popular now days and adopted by many developers and companies to carry the computer product information and to share the information in consistent way. The design goal of

XML is simplicity and usability over the Internet. The data in XML is represented in the form of tree. XML is important and common tool used now days for data transmission. In XML the element tags are case sensitive and the tags cannot contain any space characters and any other characters. The root elements which are single contain the other elements. Parsing means reading the XML document/file according to the structure. An XML parser is the piece of software that reads the XML files extracts the valuable data and transport that information to the client application. It provides the methods for client application to work with XML document/file. It may have some validation rules to parse the document and formatting checks. One XML parser may not work for other XML files. It only operates on those types of files which comply the parser rules and reading logic. There are three kinds of parsers which are commonly used SAX, DOM and the pull.

2. RELATED WORK
Le Liu et al. (2008) [11] presented PSJ an efficient Parallel Structural Join algorithm for evaluating XPath. PSJ can skip many ancestor or descendant elements by evenly and efficiently partitioning the input element lists into some buckets. PSJ obtains high performance by evaluating XPath step in each bucket in parallel. It was very efficient to partition the input lists and was effective to evaluate XPath step in buckets and therefore PSJ achieves a high speed up ratio. They implemented proposed algorithm and the experimental results showed that PSJ algorithm achieved high performance and outperforms the existing state-of-the-art methods significantly.

Abdul Nizar M. and P. Sreenivasa Kumar(2009)[1] described the processing of backward XPath axes against XML streams was challenging for two reasons: (i) Data was not cached for future access.(ii) Query contained steps specifying navigation to the data that already passed by. While there were some attempts to process parent and ancestor axes, there were very few proposals to process ordered backward axes namely, preceding and preceding-sibling. For ordered backward axis processing the algorithm, in addition to overcoming the limitations on data availability had to take care of ordering constraints imposed by these axes. In this paper, authors showed how backward ordered axes can be effectively represented using forward constraints. They discussed an algorithm for XML stream processing of XPath expressions containing ordered backward axes. The algorithm used a layered cache structure to systematically accumulate query results. Their experiments showed that the new algorithm gains remarkable speed up over the existing algorithm without compromising on buffer space requirement.

Wei Lu and Dennis Gannon (2009) [12] introduced a general purpose parallel XML processing model ParaXML designed for multicore CPUs. The processing of the XML documents however had been recognized as the performance bottleneck in those systems as a result the demand for high-performance XML processing grows rapidly. On the hardware front the multicore processor is increasingly becoming available on desktop-computing machines with quad core shipping now and 16 core system within two or three years. Unfortunately almost all of the present XML processing

algorithms are still using serial processing model thus being unable to take advantage of the multicore resource. They believed that a parallel XML processing model should be a cost-effective solution for the XML performance issue in the multicore era.

Zacharia Fadika et al. (2009) [24] 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 PIXIMAL toolkit for processing large-scale XML datasets that utilizes the capabilities for parallelism that were 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.

Martin Krulis and Jakub Yaghob (2010) [14] stated Current XPath processors use direct approach to query evaluation which was quite inefficient in some cases and usually implemented serially. This may be a problem in case of processing complex queries on large documents. They proposed algorithms and XML indexing techniques which were more efficient and which can utilize standard parallel templates. Their implementation was highly scalable and outperforms common XML libraries.

Rongxin Chen and Weibin Chen (2010) [14] introduced that various XML query applications had come to the fore recently performance optimization becomes the research hotspot. With the popularity of multi-core computing condition parallelization appears as an important optimization measure. This paper presented a parallel solution to XML query application through the combination of parallel XML parsing and parallel XML query. The XML parsing is based on arbitrary XML data partition and parallel sub-tree construction with the final merging procedure. After XML parsing the region encodings of XML data were obtained for relation matrix construction in that the XPath evaluation in query procedure was based on relation matrix. The matrix construction procedure and query primitives are parallelized to boost performance. As a whole their solution makes use of multi-core environment through parallelization of key execution stages in query process.

Adriana Georgieva and Bozhidar Georgiev (2012) [6] introduced some development problems and solutions concerning the parallel implementation of an algebraic method for XML data processing. They proposed parallel algorithm which 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.

V.M. Deshmukh1 and G.R. Bamnote (2012) [22] proposed that extensible markup language XML had 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 had 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 was an expensive operation that can degrade XML processing performance.

Peter Ogden et al.(2013)[16] proposed in online social networking network monitoring and financial applications there was need to query high rate streams of XML data but methods for executing individual Xpath queries on streaming. XML data had not kept pace with multicore CPUs. For data-parallel processing, a single XML stream is typically split into well-formed fragments which were then processed independently. Such an approach however introduced a sequential bottleneck and suffers from low cache locality limiting its scalability across CPU cores. They described a data-parallel approach for the processing of streaming XPath queries based on pushdown transducers.

3. PROBLEM FORMULATION

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 the thread count will be fixed then CPU may not be used properly. So here to overcome the above problems, the proposed idea is 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.

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 here to overcome the above problems, the proposed idea is 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.

4. OBJECTIVES

XML Parallel parsing based upon multicore is the very efficient technique. It will also increase the performance of the system while XML parsing. The objectives of this technique are following:

1. Utilizing multicore rather than single core.

2. Multithreading on core capability to enhance computation process.
3. Load balancing using segmented parse set.
4. To speed up the data read operation.

5. METHODOLOGY

In order to speed up the XML parsing process and to utilize the CPU properly the number of threads per CPU core is to be set as two. By creating two threads it will not slowdown the CPU performance and reduces the XML parsing time. In this process there are take five phase to execute the whole process. These phases are Block identification, Build job Queue/Pool, Thread Manager, XML Parsing and Data Representation. Thread Manager will allocate each thread to each manager to speed up the process. Different phase have different working. This process improves the speed of the XML parsing and improves the throughput and also improves the XML search At last the XML data has been represented which is to be parsed into the tree representation form. The five different phase of the process are:

5.1 Parsing Process Description

1. Block Identification: In this module the XML file is first loaded and identify the segments at second level of data. As shown in the below example segments will be identified as N1, N2…Nn.

```
<Message>
<Note name =“N1”>
        <heading>Reminder</heading>
                <body>Don't forget me this
weekend! </body>
    </Note>
<Note name =“N2”>
        <heading>Reminder</heading>
                <body>Don't forget me this
weekend!</body>
    </Note>
<Note name =“Nn”>
        <heading>Reminder</heading>
                <body>Don't forget me this
weekend!</body>
</Note>
</Message>
```

2. Build Job Queue/pool: In this module all possible segments found at second level are read that will act as small XML. Here LINQ queries are used provided by .Net frameworks to separate the parallel XML element segments. Each segment will be queued in a list which will read by first to last index for parsing.

3. Thread Manager: In this section the CPU type will be identified and set the thread count. The count of CPU cores will be checked and multiply with 2 and set the max threads count for parsing. It will also traverse the job queue and execute thread for every element for parsing. For this “Parallel foreach” looping method is recommended to effectively use the system CPU cores.

4. XML Parser: It will take the input as XML segment and parse the data. In this section the query approaches will be followed which is the fastest way to read the XML data. XML parser reads the whole data.

```
XDocument xdoc = XDocument.Load("Seg1"));
var lv1s = from lv1 in xdoc.Descendants("Note")
        select lv1.Attribute("heading").Value;
```

It will fill the appropriate data structure and store the objects as values and the Names as keys in a MAP. It will help in easy data retrieval from the parsed data. In the query phase use the LINQ query is used for faster execution.

5. Data Representation: After completing the parsing processes, this module will read the map/data dictionary filled by XML parser and display the XML contents visually in .Net tree control to in proper hierarchy and readable form for better understanding. In the data representation phase the data would be represented in the tree form so that it should be clearly understandable by the user. In the data representation phase the whole segmented data is merged into one place.

5.2 Block Diagram

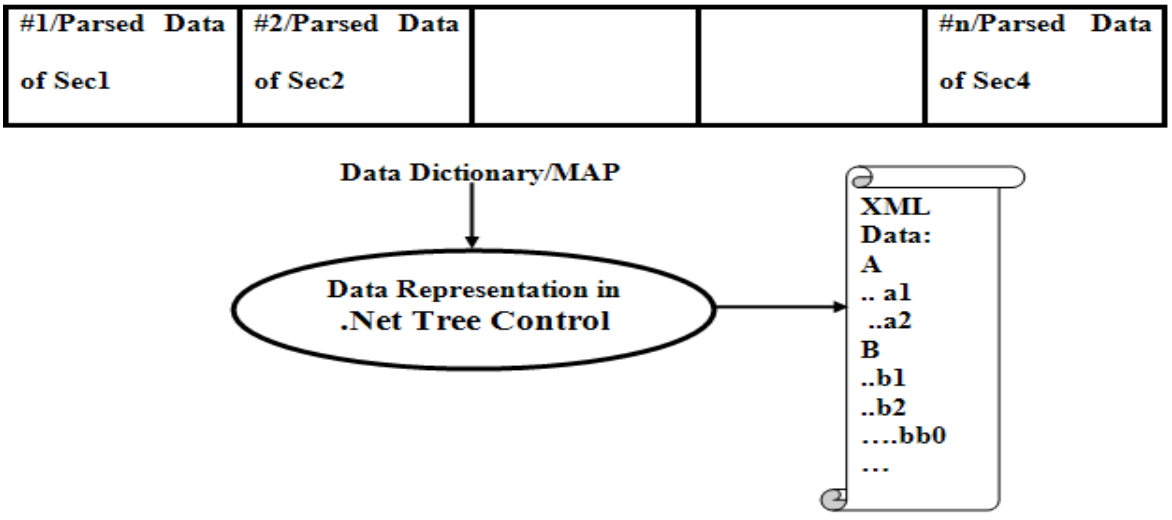

Figure 1: Block diagram showing parsed data of every section

5.3 Flow Chart of the Parsing Process

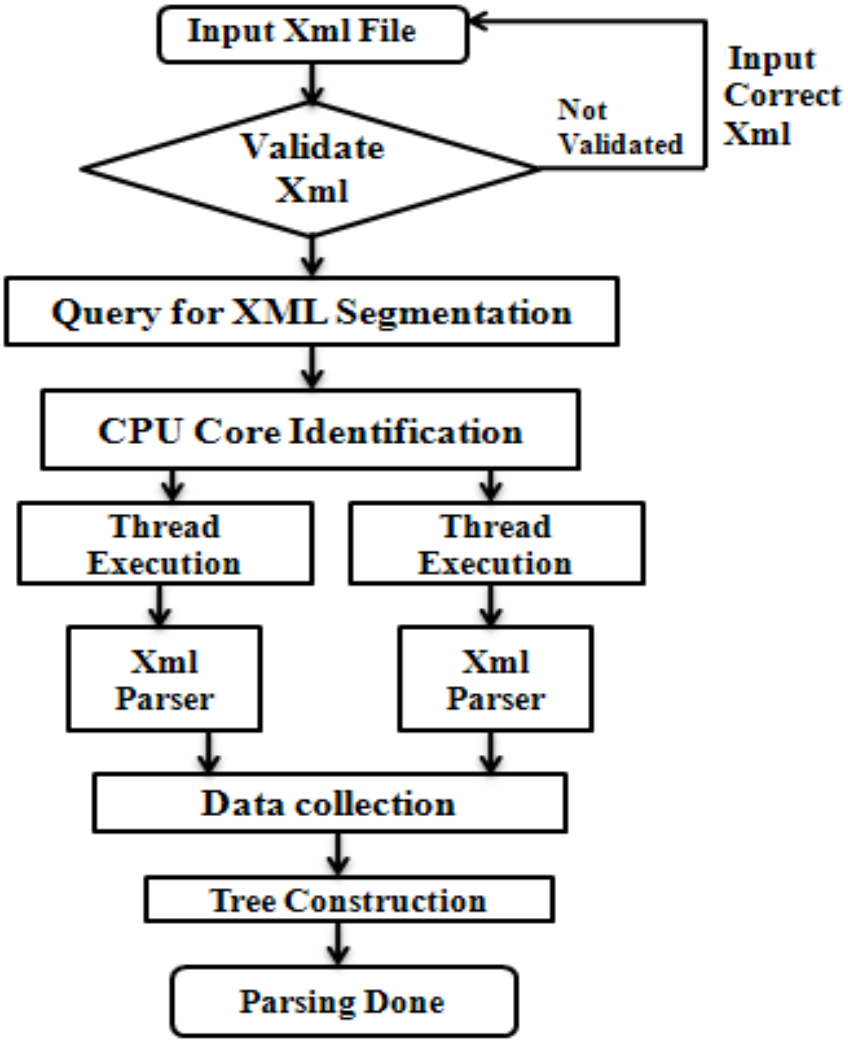

Figure 2: Flow Chart of the Parsing Process

6. RESULT

As evidence, the proposed idea has been in a tool. It has been tested on different system configurations. It is giving and results as expected and showing adequate saving too. It is properly balancing the load and parsing the data on simultaneously on multi cores. So that, on different runs it will have different time consumption. Underlying results from XML parsing on multicore using multiple threads practices illustrates the behavior of parallel parsing technique. It improves the execution time by running the certain number of multiple threads on each core. There are few examples given below as evidence.

6.1 XML parallel parsing with different threads on 2 cores CPU

I. **XML parallel parsing with 2 threads on 2 cores CPU:** XML parsing is done at this level by selecting 2 threads on 2 cores CPU. The total number of threads on 2 cores processor will be run 4 at this level. The total time it will take to parse the data is 109.2002 milliseconds.

II. **XML parallel parsing with 3 threads on 2 cores CPU:** XML parsing is done at this level by selecting three threads at 2 cores CPU. On each core three threads will be run. The total number of 6 threads will be run on the 2 cores processor to parse the XML document. Total time taken by parser to parse the data is 78.001 milliseconds.

III. **XML parallel parsing with 4 threads on 2 cores CPU:** XML parsing is done at this level on 2 cores CPU with 4 threads that run parallel on each cores CPU. The total number of 8 threads will be run on 2 cores processor to parse the selected XML document. Four threads will be run on each core to parse the XML document. Total time taken by parser to parse the data is 156.003 milliseconds.

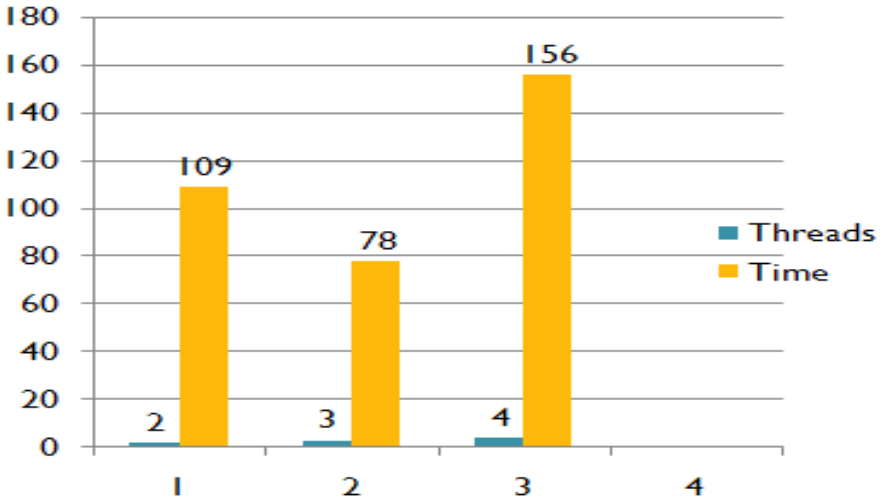

Figure 3: Execution time taken by different threads at 2 cores CPU

6.2 XML parallel parsing with different threads on 4 cores CPU

I. **XML parallel parsing with 2 threads on 4 cores CPU:** XML parsing is done at this level by selecting 2 threads on 4 cores CPU.The total number of eight threads will run on 4 cores. It will take less time to parse the whole document as compared to the single processor and multiple processors. Total time taken by parser to parse the data is 94.0054 milliseconds.

II. **XML parallel parsing with 3 threads on 4 cores CPU:** XML parsing is done at this level by selecting 3 threads on each 4 cores CPU. As the four threads will run on each core, the total number of threads will be run 12 to parse the whole document. It will take less time to parse the whole document as compared to parsing with parallel processing with 2 threads on each core. It makes the execution faster and utilizes the CPU processor more efficiently. Total time taken by parser to parse the data is 72.011 milliseconds.

III. **XML parallel parsing with 4 threads on 4 cores CPU:** XML parsing is done at this level by selecting 4 threads on each 4 cores CPU. The four threads will run on each core, the total number of threads will be run 16 to parse the whole document. It will take less time to parse the whole document as compared to parsing with parallel processing with 2 threads on each core. It makes the execution faster and utilizes the CPU processor more efficiently. Total time taken by parser to parse the data is 55.003 milliseconds.

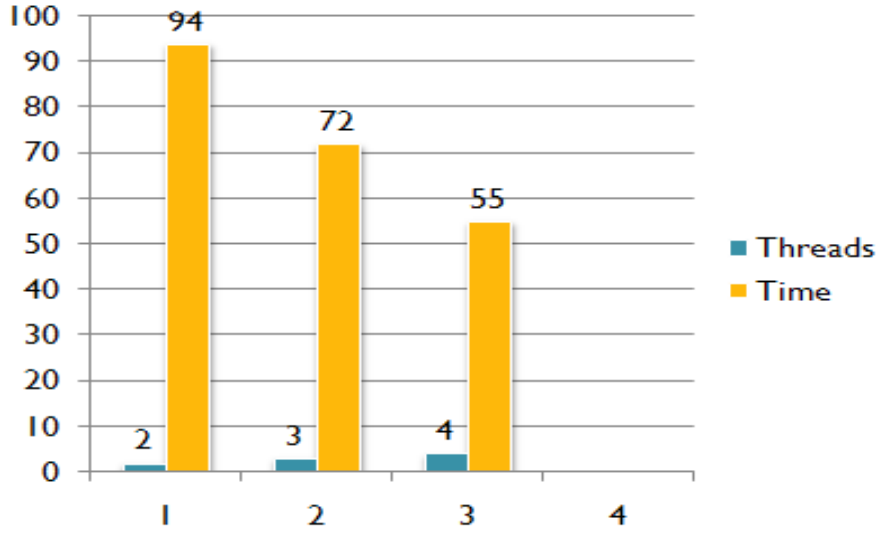

Figure 4: Execution time taken by different threads at 4 cores CPU

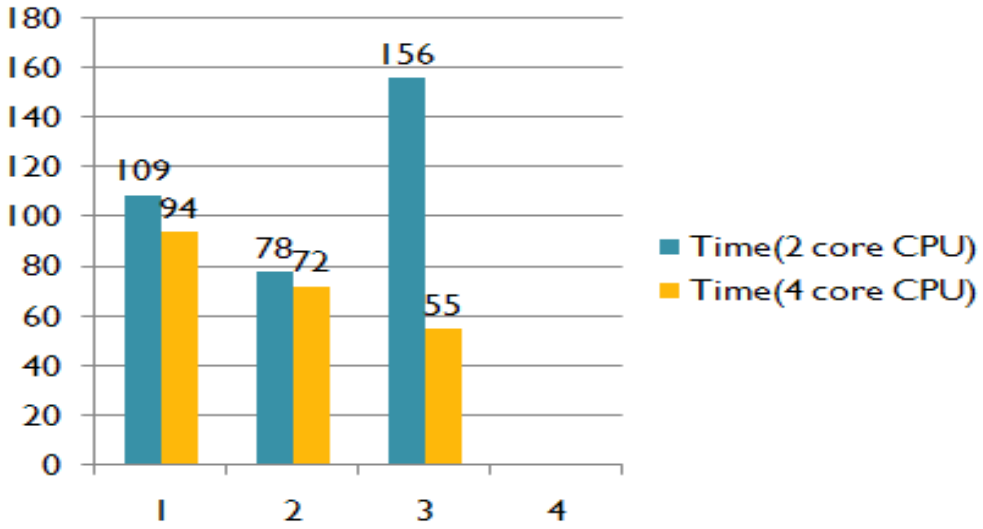

Figure 5: Comparison between 2 cores and 4 cores CPU

Table1: Execution Time of different cores

No. of cores	No. of threads	Execution Time (in milliseconds)
2	2	109.2002
	3	78.001
	4	156.003
4	2	94.0054
	3	72.011
	4	55.003

7. CONCLUSION & FUTURE WORK

7.1 Conclusion

By doing all these exercises and data analysis it can be said that it is very much important to tune up the XML parsing algorithms where the systems are dependent upon the XML based data. XML data is sequential in nature, so data reading is time consuming task if user has to extract the data from each segment one by one in single flow to improve the performance of the systems which use the XMLs a lot and expect the desired data at the earliest. In such systems, if user can save even a microsecond matters a lot. It becomes more important when system needs to perform XML based operations when frequent updates are there or the support files are referred in the XML file's contents. In such scenarios, if system uses the multiple threads on multi-cores for parallel parsing of XML components then it will utilize the system resources properly and desired results and further operations will be faster. In addition, the use of LINQ queries is beneficial to collect the parallel XML segments and for quick element data reading which is very much optimized on .Net Frameworks.

7.2 Future work

Present work can be improved further for extracting of XML files. The research in XML parsing has accumulated many advantages. In general there are three types of parsing techniques are sequential parsing, pipelining and the parallel parsing technique. As there is scope of improvement in different parsing techniques. The present work discussed about the data extraction of XML files and their segments on multi-core CPUs simultaneously. As the part future work, it can be enhanced for searching the contents in the XML files. It has been observed that it is very much time consuming if user search the contents in set of XML files by one by one. To precede this approach further, there is a need to run the individual/multiple threads for searching the contents logic on multi-core CPU. It will be very quicker than normal sequential approach.

8. REFERENCES

[1] Abdul Nizar M. and P. Sreenivasa Kumar (2009) "Ordered Backward XPath Axis Processing against XML Streams" XSym '09 Proceedings of the 6th

International XML Database Symposium on Database and XML Technologies Pages 1 - 16.

[2] Barbosa D(2002) ToXgene: a template-based data generator for XML, In : Proceedings of ACM Management of Data (SIGMOD), pp. 616.

[3] Fernando F et.al (2009) 2LP Double-lazy XML Parser in Journal of Information Systems, pp. 145-163.

[4] Gao Z. (2007) A High Performance Schema-Specific XML Parser, IEEE Intl. Conf. on e-Science and Grid Computing, pp. 245-252.

[5] Gong Li (2010) XML Processing by Tree-Branch symbiosis algorithm, 2nd International Conference on Future Computer and Communication, Volume1.

[6] Georgieva A and Georgiev B (2012) Parallel Processing Model for XML Parsing in Journal of Communication and Computer, 1258-1262.

[7] G.R.Bamnote(2013) An Empirical Study: XML Parsing using Various Data Structures, International Journal of Computer Science and Applications, Vol. 6, No.2.

[8] James R. Otto et.al (2001) Extensible Markup Language and Knowledge Management in Journal of Knowledge Management, 5(3), pp. 278-284, MCB University Press.

[9] Jie Tang et.al(2013) Acceleration of XML Parsing through Prefetching, IEEE TRANSACTIONS ON COMPUTERS, VOL. 62, NO. 8.

[10] Kwon, J et.al (2005) FiST: the scalable XML document _ltering by sequencing twig patterns, In: Proceedings of the 31st international conference on Very Large Databases (VLDB), pp. 217 – 228

[11] Le Liu et al. (2008) "Parallel Structural Join Algorithm on Shared-memory Multi-core Systems".

[12] Li Lu W. and Gannon, D. (2008) ParaXML: A Parallel XML Processing Model on Multicore CPU, Techincal Report.

[13] Li Xiaosong (2009) Key Elements Tracing Method for Parallel XML Parsing in Multi-coreSystem, in International Conference on Parallel and Distributed Computing, ApplicationsandTechnologies, IEEE.

[14] Martin Krulis and Jakub Yaghob(2010) "Efficient Implementation of XPath Processor on Multi-Core CPUs" J. Pokorn_y, V. Sn_a_sel, K. Richta (Eds.): Dateso 2010, pp. 60{71, ISBN 978-80-7378-116-3.

[15] Nicola M. and J. John(2003) XML Parsing: A Threat to Database Performance, Proc. 12th Int'l Conf. Information and Knowledge Management (CIKM 03), ACM Press, pp.175-178.

[16] Peter Ogden et al.(2013) "Scalable XML Query Processing using Parallel Pushdown Transducers" Proceedings of the VLDB Endowment, Vol. 6, No. 14.

[17] Rongxin c. et.al (2002) A Parallel Solution to XML Query Application in Computer Engineering College, Jimei University.

[18] S. Chen et al. (2006) Twig2Stack: Bottom-up processing of generalized-tree-pattern queries over XML documents In VLDB, pages 283–294.

[19] Su Cheng Haw and G. S. V. Radha Krishna Rao(2007) A Comparative Study and Benchmarking on XML Parsers, Advanced Communication Technology, The 9th International Conference (Volume:1) ISSN :1738-9445 , pp. 321 – 32.

[20] Seung Min Kim and Suk Yoo(2009) DOM Tree Browsing of a Very Large XML Document: Design and Implementation in Journal of Systems and Software, 82(11), pp. 1843-1858.

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

[22] V.M. Deshmukh and G.R. Bamnote(2012) Design And Development Of An Efficient XML Parsing Algorithm, International Journal of Applied Science and Advance Technology , Vol. 1, No. 1, pp. 5-8.

[23] Y. Pan et al.(2007) Parallel XML Parsing Using Meta-DFAs,Proc. 3rd IEEE Int'l Conf. e-Science and Grid Computing (e-Science 07), IEEE CS Press, pp. 237-244.

[24] Zacharia Fadika (2009) 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