A Novel Method for Representing XML DOM Data

Puspha rani Suri[a] , Neetu Sardana[b].

a: Computer Science & Application Department, Kurukshetra University, Kurukshetra
b: Apeejay School Of Management, Sector-8, Institutional area, Dwarka, New Delhi

Summary
XML is an acceptable standard for data representation and exchange on the World Wide Web. In the XML (Extensible Markup Language) based web applications like time series analysis, risk analysis, share marketing, forecasting system where data is homogeneous in nature, complex calculations are being done to analyze the data. We can't compromise with large processing time of mathematical processing system due to preparing data set for processing. That's why time efficient buffer creation is primary goal, so that total time taken in processing and generating pattern should be significantly less. This paper emphasis is to store XML data after DOM parsing in proposed way to get best response time if we are dealing with homogeneous data.
An investigation is being conducted in simulated environment for getting XML data from database and parses it to store in proposed way then after processing was done on data set. It is being found that while parsing XML data if tag and structure both are stored in a presented architecture of single buffer then it will give high performance in time for parsing and responding result of computation.
Keywords: *XML, DOM, DTD.*

1. Introduction

Extensible Markup Language (XML) is gaining acceptance in the global market as the standard format for storing structured and unstructured electronic documents on large database. Response time for retrieving or storing information from large database is always a challenge. Most of the work reported in literature is dealing with this issue [1-6].
But often we would like to do manipulation or analysis on XML data in applications like weather forecasting, financial modeling risk analysis etc and explore the data. For such type of activity we need an environment where we can retrieve data quickly from the large data set in XML format and further complex mathematical processing can be done.
In order to analyze XML data, XML has to be represented in some standard format via DOM (Document object model). DOM is a platform- and language-neutral interface that allows programs and scripts to dynamically access or update the content, structure and style of XML documents. Programming solutions using DOM are quite convoluted, even for such a simple task. DOM programmers have accepted that their code as tedious, hard to read and maintain, and thus prone to errors [7]. In DOM complete XML document exists in memory and it can be easily processed and manipulated but if documents are large then it imposes a large memory requirement. As the size of a DOM tree created from an XML document is as large as 10 times of the size of the original document [8]. This is the reason we have to port XML via DOM in some platform like C++, JAVA, OCTAVE or Matlab®.

This paper emphasis is on how to port the XML data onto some platform quickly so that further manipulation can be done on that data. Proposed methodology is using single buffer storage technique for preparing XML data for further processing in place of existing generic multiple data storage technique. It is being found that in most of XML based web applications, data used is homogeneous in comparison to normal data. So in place of using existing generic XML storage technique, a new proposed methodology that suits requirement for data analysis, is presented for reading these types of XML data and makes it usable for further processing.
The proposed method is simulated for gold price data using Matlab® platform.
Entire Paper is divided in following sections:
Section 2 describes the problem, section 3 introduces some preliminaries concerning XML, section 4 describes the storage structure for porting XML data in Multiple array system, section 5 introduces the storage structure for porting XML data using proposed Single array system, section 6 describes Time complexity using both methods, section 7 discusses a case study, section 8 concludes this paper.

2. Problem definition

We are developing a system that can take raw XML data so that the data can be used for analysis like forecasting future expected behavior / or future possible pattern.
For porting XML data is taken as an input and is parsed. Parsed XML data is stored in temporary buffer storage and finally complex processing is being done on the data, and output patterns are generated.
Currently we are using DOM parser for reading XML data as XML DOM defines a standard way for accessing

Manuscript received April 5, 2010
Manuscript revised April 20, 2010

and manipulating XML documents and it presents an XML document as a tree-structure [9].
In tree structure XML data consists of nodes and edges that represent elements attributes, text. Parsing of XML data is done using DOM parser, in which all nodes of data are parsed according to node type. This node type determines the characteristics and functionality of the node. The various node types in XML DOM are Document, DocumentType, DocumentFragment, Entityreference, Element, Attr, ProcessingInstruction, Comment, text, CDATAsection, Entity, Notation etc [10]. For parsing data, complete scanning of XML document is done. We have to process this tree structure for pattern generation of forecasting and it's not feasible to use this raw tree directly as all data stored there are in object form. In order to use it we need temporary buffer where we can store data of XML file result of we are parsing this tree and we are selecting node then recursively looking for its children.
In the following section we provide a brief introduction to the XML language, the related Document Type Definition language and Document Object Model using an example for better understanding. Further same example is used to elaborate existing and proposed storage system.

3. Working with XML documents

3.1 Extensible Markup Language:

XML is a data description language standardized by W3C (World Wide Web Consortium). XML is a sophisticated subset of SGML (Standard Generalized Markup Language: ISO 8879), and designed to describe document content using arbitrary tags. As its name implies, the extensibility is a key feature of XML; users or applications are free to declare and use their own tags and attributes. Therefore, XML ensures both the logical structure and content of semantics-rich information is retained. XML emphasizes description of information structure and content as distinct from its presentation. That is the reason it has become a standard for data representation and exchange on the Internet. It is expected that XML will become a universal format for data exchange on the Web [11][12].
An XML document has a logical structure. The logical structure allows the document to be divided into units called elements. These elements can contain other elements in turn thus allowing for a complex logical structure to be defined.
For example, to describe a gold price we need a date in terms of year and month element,
a format element and price element. Also, gold will have a exchange name that could be stored as an attribute.

Here is how this Gold.xml an XML document can be expressed as [13].

```
- <DailyGoldPrice>
    - <Gold exchange="Kitco">
            <Year>2008</Year>
            <Month>01</Month>
            <Format>USD</Format>
            <price>200</price>
  </Gold>
  - <Gold exchange="Kitco">
             <Year>2008</Year>
             <Month>02</Month>
             <Format>USD</Format>
             <price>230</price>
    </Gold>
- <Gold exchange="Kitco">
            <Year>2008</Year>
            <Month>03</Month>
            <Format>USD</Format>
            <price>240</price>
  </Gold>
- <Gold exchange="Kitco">
            <Year>2008</Year>
            <Month>04</Month>
            <Format>USD</Format>
            <price>250</price>
  </Gold>
    </DailyGoldPrice>
```

Figure1: Gold.xml

3.2 DTD (Document Type Definition):

DTD describes the grammar for XML. A DTD defines the document structure with a list of legal building blocks of XML. It defines elements and attributes of an XML document. DTD is basically used to validate an XML document. Document should conform to DTD specification. The following is a sample DTD that can be used to enforce the logical structure of the example presented previously in the XML section.

```
<?xml version="1.0" encoding="UTF-8"
standalone="no" ?>
<!DOCTYPE DailyGoldPrice [
   <!ELEMENT Gold (Exchange)>
   <!ELEMENT Gold (Year)>
   <!ELEMENT Gold (Month)>
   <!ELEMENT Gold (Format)>
<!ELEMENT Gold (Price)>
]>
```

Figure2: DTD for Gold.xml

3.3 DOM (Document Object Model):

The W3C has defined a standard interface for accessing XML files called Document Object Model (DOM) [8]. The DOM defines a language- and platform-neutral API that allows accessing, navigating, querying and manipulating XML documents. Methods for manipulating the tree or its components are provided by the specific parser implementation.
DOM consists of variety of nodes such that every node provided with its type that determines its functionality. The DOM tree for the XML document is shown in figure

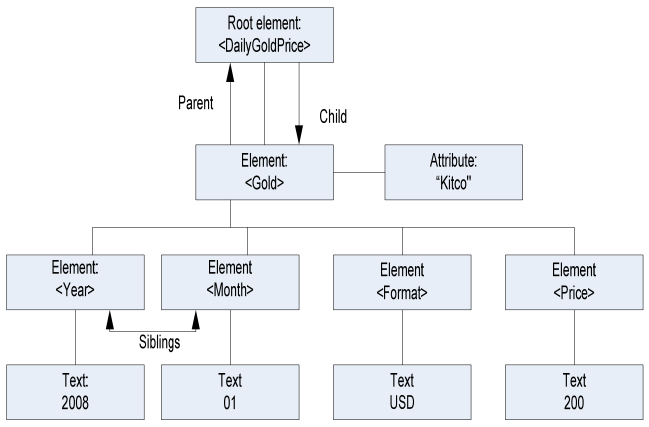


Figure 3: Sample DOM tree for Gold.xml

3.4 XML DOM API

The XML DOM (XML Document Object Model) defines a standard way for accessing and manipulating XML documents. It contains methods to traverse XML trees. It views XML documents as a tree-structure. All elements can be accessed through the DOM tree using its node.
The Node interface is the primary datatype for the entire Document Object Model. It represents a single node in the document tree. While all objects implementing the Node interface expose methods for dealing with children, not all objects implementing the Node interface may have children. For example, Text nodes may not have children and adding children to such nodes results in a DOMException being raised.
The attributes nodeName, nodeValue and attributes are included as a mechanism to get at node information without casting down to the specific derived interface. In cases where there is no obvious mapping of these attributes for a specific nodeType (e.g., nodeValue for an Element or attributes for a Comment), this returns null. Note that the specialized interfaces may contain additional and more convenient mechanisms to get and set the relevant information. [14]

FIELD SUMMARY

ATTRIBUTE_NODE
The node is an Attr.
CDATA_SECTION_NODE
The node is a CDATASection.
COMMENT_NODE
The node is a Comment.
DOCUMENT_FRAGMENT_NODE
The node is a DocumentFragment.
DOCUMENT_NODE
The node is a Document.
DOCUMENT_TYPE_NODE
The node is a DocumentType.
ELEMENT_NODE
The node is an Element.
ENTITY_NODE
The node is an Entity.
ENTITY_REFERENCE_NODE
The node is an EntityReference.
NOTATION_NODE
The node is a Notation.
PROCESSING_INSTRUCTION_NODE
The node is a ProcessingInstruction.
TEXT_NODE
The node is a Text node

METHOD SUMMERY

getAttributes()
A NamedNodeMap containing the attributes of this node (if it is an Element) or null otherwise.
NodeList getChildNodes()
A NodeList that contains all children of this node.
Node getFirstChild()
The first child of this node.
Node getLastChild()
The last child of this node.
String getLocalName()
Returns the local part of the qualified name of this node.
String getNamespaceURI()
The namespace URI of this node, or null if it is unspecified.
Node getNextSibling()
The node immediately following this node.
String getNodeName()
The name of this node, depending on its type; see the table above.
short getNodeType()
A code representing the type of the underlying object, as defined above.

String getNodeValue()

The value of this node, depending on its type; see the table above.

boolean hasAttributes()

Returns whether this node (if it is an element) has any attributes.

boolean hasChildNodes()

Returns whether this node has any children.

4. Storage structure for Multiple buffer system:

In Multiple buffer system, each type of node is stored in separate array depending upon its node types, which is linked to one another with some index value. In the output data structure field names are based on XML tags [15].

Here two buffers are used to store complete XML data. We are calling them as element buffer and data buffer. Element buffer is used to store element, attribute etc. of XML and data buffer is used to store actual data. Figure 4 shows structure of both types of buffer.

Element buffer consists Element Lable , AttribType, data count and number of child in node. In this system root node, data count will be null and child count will set equivalent to certain number depending upon the child it has. Node other then root node will have data count and child count value. Same logic is applied for other types of node. Data buffer will accommodate only data.

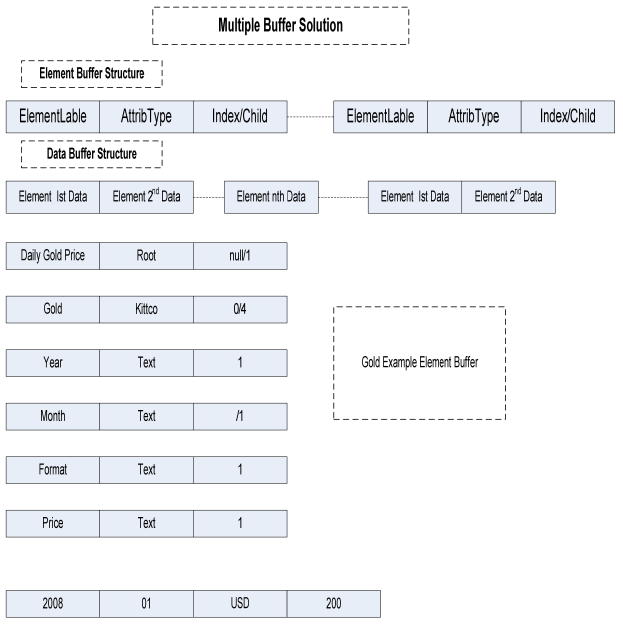


Figure 4: Multiple Buffer System:

To explain it again we are using same gold.xml example.

Element <Daily Gold Price> is root node and it have data count value as null and It has single child <gold> so child count value for it is one.

Field in data buffer is not required for root node.

Another element is gold and it has no data but have for child so data count is set to zero and child count is set to 4.

Element Year has data as year but no child so its data count is set as 1 and no value is set for child count.

Data for this node is filled in data buffer. In same way each node will be parse and both element and data buffer will be update as per there element type.

Step 1: Pass XML file as input

Step 2: Read XML file in memory using DOM

Step 3: Read entire tree represented by DOM by one by one node

Step 4: Check Node and its element type.

Step 5: Set Element label field of Element structure in Element buffer

Step 6: Set Element attrib field of element structure in Element buffer

Step 5: If node is root node set Element data count as null

Step 6: Set Element child count field as no of child.

Step 7: If node is not root node then set data count and child count.

Step 8: Set Data buffer index as 0 and next index for another root set as number of child count.

Step 9: If node are having leaf then fill data in Data buffer.

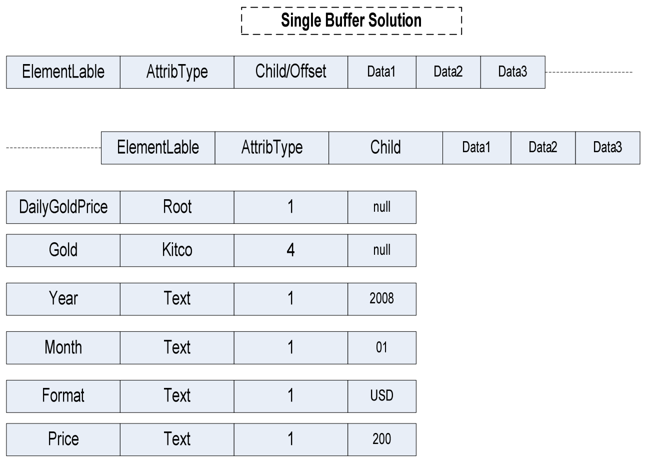
Step 10: Data for this leaf will be stored at index set but element buffer.

Step 11: Data for next leaf will store at next location in data buffer.

At the end of this process, entire XML document will be stored in element and data buffer. In this buffer complete data of single node will be stored in continuous memory and data for next node will start from index set by previous node that will be equal to number of child in previous child. This data buffer can be use for further processing depending query.

Now if we want to process this data based on some processing then first we need to search full element buffer and then we need to search complete data buffer. Searching to satisfy query in element and data buffer can we done by using any search algorithm, but minimum two full search loop required in this type of arrangement. Time complexity for search operation will come n^2 in this type of data storage arrangement [16][17].

5. Storage structure for Single Buffer System:



ElementLable	AttribType	Child/Offset	Data1	Data2	Data3
ElementLable	AttribType	Child	Data1	Data2	Data3

DailyGoldPrice	Root	1	null
Gold	Kitco	4	null
Year	Text	1	2008
Month	Text	1	01
Format	Text	1	USD
Price	Text	1	200

Figure 5: Single buffer system

In this work, primary goal is time efficient buffer creation so that this buffer can be further used for processing of XML data. Generic multiple buffer solution is efficient for other XML based application where we have data of heterogeneous type [18] but is not suitable for the application where we are using homogeneous data so this paper is presenting, a simple and straight forward method that takes less time in comparison with existing multiple buffer solution.

Here in place of creating separate buffer for element and data of XML tree we are creating single buffer that accommodate both data and element.
Storage structure of single buffer:
Our buffer consists of elementlabel, element attrib, child count and offset and array of continuous memory to store all data in a node of XML tree.
Now taking same gold example to show how our data will be represented in proposed single buffer solution. Again we are reading XML file using DOM. first it will give us root node and our root node will have only one child. Root node have no data. Element label will be set as daily Gold price. Attrib type will set as root. In our given gold example only one child is there so child count will be set as 1. Root node has no data so data count will be set as 1. Our root node have one child so next time we will read this child and fill our buffer accordingly. In presented example element label for first child is ‘gold’, its attrib type is ‘kitco’. First child itself have 4 child in tree so child count will be set as 4, same way full file will be parsed and structure in buffer will be update as per node / leaf of DOM tree. End of this we, will get single buffer that is having full xml document stored in it.
Step 1: Pass XML file as input
Step 2: Read XML file in memory using DOM
Step 3: Read entire tree represented by DOM by one by one node
Step 4: Check Node and its element type.
Step 5: Set Element label field of Element structure in Element buffer
Step 6: Set Element attrib field of element structure in Element buffer
Step 5: If node is root node set Element data count as null
Step 6: Set Element child count field as no of child.
Step 7: If node is not root node then set data count and child count.
Step 8: If node are having leaf then fill data in Data buffer.
Step 9: Data for this leaf will be stored at index same place with its element

This way full node will be stored in single buffer along with it data. In next section we will explain time complexity in single buffer in comparison to double buffer.

6. Time Complexity in XML DOM data parsing

6.1 Multiple buffer system:

Parsing of XML data is done using DOM parser, in which all nodes of data are parsed according to its node type. For parsing data complete scanning of document is being done and elements are stored as a node of tree and data is stored as a leaf of tree. All elements (node) of same DOM structure type are stored in a one array and leaf (data) is stored in another array. This is one of the common methods used for storing and parsing XML data in a buffer. Here we are using two buffers, one for storing element and another for data. By using simple mapping algorithm both element and data can be retrieved and saved.
For storing and retrieving data, we need two indexes one to get element type and another to get actual data. Here we require two loops to get actual data from node. In worst case time complexity will increase.
A discrete amount of time will be taken to execute each of the instructions involved with carrying out this algorithm.

Pseudo algorithm based on some query will come as following

Step-1 Read XML file and create Element and data buffer
Step -2 for (int i=0; i<N; i++)   N is total size of buffer after processing XML file
Step -3 if (A[i] == "Query Condition") Element Query
{
Step-4 for (int j=index; j<N; j++) Data Buffer(index got from first loop)

```
Step-5    if (B[j] == "condition")// Some query based condition in Data buffer
Retrive data    for math processing

}
Step-6  fun() //pass data  to do mathematical processing
```

The action carried out in step 1 are considered to consume time T, step 2 uses time T, and so forth.
In this pseudocode step, 1, 3, 5 and 6 will execute only once. Step 2 will take n+1 time to evaluate (note that an extra step is required to terminate the 'for loop', hence n + 1 and not n executions). Total time taken by inner loop will be control by other loop and it will take (n+1)*(n) time.
Total time for running this procedure in worst case will come as ~ n^2+n.
One can assume that the highest-order term in any given function dominates its rate of growth and thus defines its run-time order. Here it is n^2.Our function will come as $F(n) = n^2$.
Big O notation for this algorithm will come in order of n^2.

6.2 Reference Architecture using Single buffer:

Storing data in multiple buffer is good approach if data is heterogeneous in nature. In applications where we are dealing with time series data most of data will be homogeneous in nature there we will have data with similar type of element tag. We are proposing a new algorithm of single buffer that is faster than multiple buffer approach.
Here all DOM element and data are stored in a single buffer and using element search in same buffer corresponding data can be retrieved.

Pseudo algorithm for operation on single buffer based on some query will come as following

```
Step-1 Read XML file and create single data buffer
Step -2 for (int i=0; i<N; i++)   N is total size of buffer after processing XML file
Step -3 if (A[i] == "Query Condition")  Element section in a buffer
{
Step-5    if (A[i+offset] == "condition")// Some query based condition in Data section of same buffer
        Retrive data
}
Step-6  fun() //pass data  to do mathematical processing
```

In this case we are eliminating inner loop as storing data at single buffer. Our query based data retrieval is controlled by single loop. In worst case computation time for this also will come as n.

Big O notation for this algorithm will come in order of n.

7. Case Study

We implemented two different algorithms and tested them on several sets of test data. The processing times you obtained are in shown in given table, Test results are for generated for large XML pages with homogeneous data that in gold.xml [19]. For reading XML file we are using DOM structure. Entire simulation was done on Matlab® environment as Matlab® provides rich library to do lots of mathematical computation [20]. We are using same logic for mathematical processing difference is only in reading XML file in generalize method vs. reading XML file and create data set for further processing so we can also proof our logic on actual environment

No of time parsing done	Generalize Method(sec)	Proposed Method(sec)
1000	1.44e+002	7.13e+001
1500	2.12e+002	1.10e+002
2000	2.85e+002	1.15e+002
2500	3.61e+002	1.79e+002
3000	4.12e+002	2.14e+002
5000	6.89e+002	3.89e+002

Table 1: Comparative study between Generalize Method and Proposed Method

8. Conclusion

This paper has presented a methodology to port XML data onto Matlab® platform using single buffer solution that proved to be better then existing double buffer solution as it takes less time if we are dealing with data which is homogeneous in nature.

References:

[1] Ronald Bourret, "XML and databases", 2001.
[2] D. Florescu and D. Kossmann, Storing and querying XML data using an RDBMS, IEEE Data Engineering Bulletin, Vol. 22(3), pp. 27 – 34 (1999).
[3] J.Shanmugasundaram et al., Efficiently Publishing Relational Data as XML Document, VLDB, 2000.
[4] Irena Mlynkova and Jaroslav Pokornya , XML in the world of (object-) Relational database systems, pp 1 – 14.

[5] Carl-Christian Kanne, Guido Moerkotte, Efficient storage of XML data, Technical Report, university of Mannheim, 1999.
[6] R. Goldman, J. McHugh, and J. Widom. Lore : A Database Management System for XML. Dr. Dobbs Journal, 25(4):76-80. April 2000
[7] Lauren Wood, Programming the Web: The Web: The W3C DOM Specification at URL: http://computer.org/internet/, IEEE Internet Computing, January. February1999.
[8] Fangju Wang, Jing Li, Hooman Homayounfar, A space efficient XML DOM parser, Data & Knowledge Engineering 60 (2007) 185–207.
[9] xmlread at URL: http://www.mathworks.com/access /helpdesk/help/techdoc/ref/ xmlread.html.
[10] World Wide Web Consortium,"Document Object Model (DOM) Level 1 Specification Version 1 .0" at URL: http://www.w3.org/DOM, October1998.
[11] N. Bradley,"The XML Companion", Addison-Wesley, 1998.
[12] Bray, T et.al. (ed.). Extensible Markup Language (XML) 1.0. W3C Recommendation, February 1998.
[13] Extensible Markup Language (XML) Commutation on non-linear data using conventional and soft computing method.(Accepted for publication in Serial Publication Journal International Journal of Computing and Application).
[14] DOM API at URL: http://xerces.apache.org/xerces-j/apiDocs/org/w3c/dom.
[15] xml_io_tools at URL: http://www.mathworks.fr/matlabcentral/fileexchange/ loadFile.do?objectId=12907&objectType=file#.
[16] Lecture Notes on Computational Complexity, Luca Trevisan1,2004 at URL: http://www.cs.berkeley.edu/~luca/notes/complexitynotes02.pdf
[17] R.M. Karp. Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher, editors, Complexity of Computer Computations, pages 85{103.Plenum Press, 197.
[18] Ngamnij Arch-int and Peraphon Sophatsathit, "A semantic information gathering approach for heterogeneous information sources on WWW", *Journal of Information Science*, Vol. 29, No. 5, 2003, pp. 357-374.
[19] Live chart, historical chart & data for gold price at URL: www.kitco.com.
[20] Matlab 7.0 at URL: http://www.mathworks.com.

Dr. Pushpa.Rani Suri is a Reader in the department of computer science and application at Kurukshetra University, Haryana, India. She has supervised number of Ph.d students. She had published number of papers in national and International Journals and conference proceedings.

Ms. Neetu Sardana is Lecturer(Sr. grade) in Apeejay School of Management, Delhi, India. She has Master's degree in Computer Science. At present she is pursuing her Ph.d in Computer Science. Her area of research is XML data analysis.