Indian Journal of Science and Technology

Providing a New Optimization Structure for Quickening the Performance of the Unstructured Solicitations

Shahla kiani[1*], Hossein Shirazi[2] and Mahdi Javanmard[3]

[1] Graduate Student of PNU University, Tehran, Iran; Sh.kiani.h@gmail.com
[2] Department of Computer Engineering, Malek-Ashtar University of Technology, Tehran, Iran; shirazi@mut.ac.ir
[3] Department of Computer Engineering, Payam Noor University, Tehran, Iran; javanmard@pnu.ac.ir

Abstract

Through this paper, we will demonstrate that how we can decrease the performance duration of the solicitations on XML large files; and why such files can't be read as a single file through the memory; or why doesn't the typical detection and derivation mechanism of XML (e.g. DOM – SAX) perform quickly for implementing solicitations, or why are they ineffective during processing. We defined a new concept as Skeletal Documents that is preserved and maintained in memory as a file, which by using of parser is read PULL. This affair is used to show the document structure and to select its components.

Keywords: XML-Documents, Skeletal Documents, Query, Unstructured, Semantic Web

1. Introduction

The problem of transmitting XML large files has been identified from years ago, and the best way to examine it is using STX method, which is a XML - based language for transmitting XML documents to other XML documents without producing an entire tree in memory. STX defines new transmitting language and questions, which add some limitations to standard XPath and XSLT. The object of such conversions is supporting that alternative patterns can be applied during conducting conditions in events which continuously and consecutively is perceived from a XML analyzer.STX depends on a data sample, which saves a node of the tree and a pile of its parents to the root during a certain period of time. Our procedure is a similar providing as well as STX structure, but it's much more moderate in the object completion. [1]

2. SAX

SAX that is an abbreviation for Simple API for XML is not a parser, but a Standard Interface which is set up by different parsers. Such an API contains several interfaces. SAX uses Stream Model to pars; i.e. XML document is converted into a stream of element by such a parser. Also XML document is once reviewed by parser and a tree schema is not created by this parser. So SAX is really useful, but it's not appropriate for advance tasks on document, for it requires designing and creating a schema at performance duration by developer that takes a long time and is error – prone (Figure 1).

3. DOM

Document Object Model is an interface independent to the language and the platform. Unlike to SAX, For accessing and updating XML document, DOM access to XML

** Corresponding author:*
Shahla kiani (Sh.kiani.h@gmail.com)

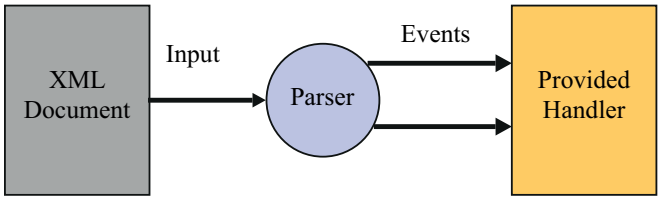


Figure 1. SAX Procedure.

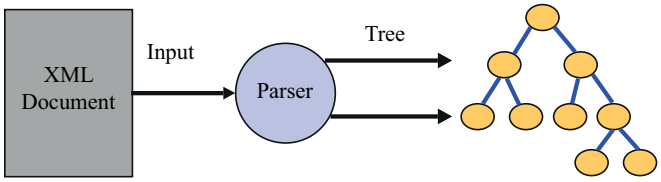


Figure 2. DOM Procedure.

documents via a structural tree, which is made from element nodes and text nodes (Figure 2).

This tree is produced in memory, thus it requires much memory, when using DOM. The advantage of DOM is that programming through it is much easier and simpler than SAX [2]. DOM tree is created just by using some lines of programming code, and it could be traversed toward any side, as it is produced.

4. XSLT

Extensible Stylesheet Language Transformation is applied to convert XML documents to any desirable format such as HTML, WML, XHML and so on. An engine or transformer

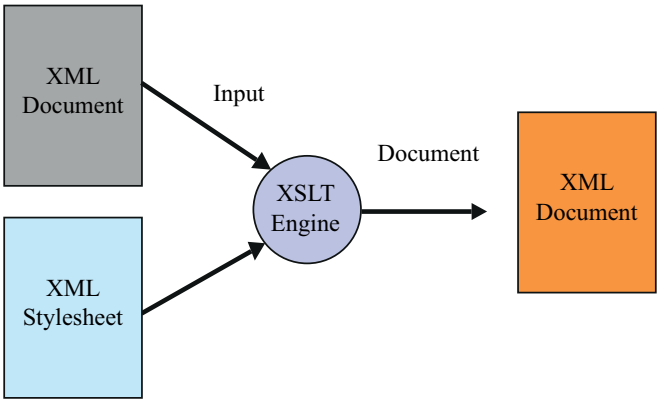


Figure 3. XSLT Procedure.

processor takes a XML document as an input and as you can see at Figure 3, turns it into a new document by XML.[3, 4]

5. Providing an Optimization Structure

XML is on the basis of a steady, simple and powerful model of data organization of generalized tree diagram. Left part of the Figure 4 shows a small XML file, which describe initial and basic marking. An optional name is given to a tree node between two marks, < and > that is called Initial Tag. Each thing which is corresponding to get a final tag forms content that could be a tree by itself. Components can contain characteristic data and composition characteristic

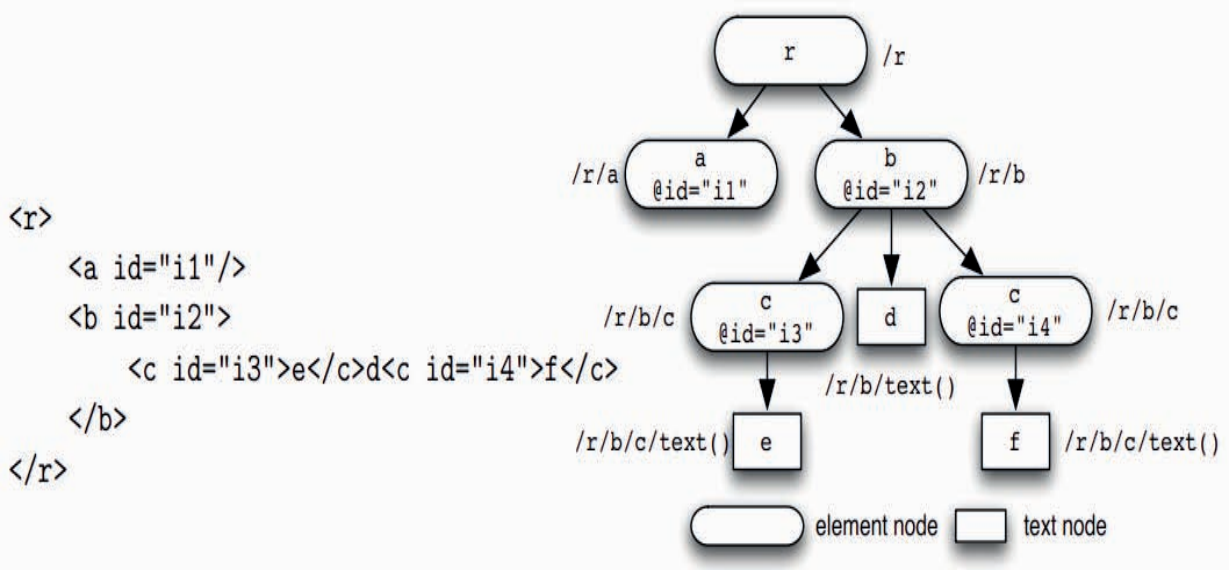


Figure 4. XML file and tree diagram corresponding to it.

components. XML elements having no contents with an immediate final tag can be shown that follow initial tag and can be shortened as a tag with no components.

One of the basic problems is the size of XML files, which can be developed very rapidly, and this makes accessing to their contents difficult and complex. Detection and derivation devices of XML files depend on XPath processors that evaluate the textual expression on the basis of tree structure toward data kind and location in the tree. To assess such expressions, XPath processor reads and makes firstly the file in the memory and inside tree structure of the file toward the evaluated phrase. Such a problem is not for small files, but it's problematic for larger files. It means tree structure file in memory is larger than the basic file, so it decreases the speed of the performance and response to requests on a XML file. To derive some small parts of the large textual files, we often recourse frequent and continual reading of each line, and decide for each one of the lines in any condition whether or not they are saved in memory [2, 5]. Utility and usefulness of this method is possible according to Grep operation so that it reads a file line by line and copies only the ones in a file, which are in conformity with regular expressions. [7]

Grep procedure is not useful for XML files with tree structure, because regular expression technique can't consider tags replacement structure, and the worse matter is that most of XML files are produced just from one line, for lines end do not mean any special and particular concept except for ordering characters present in XML file.

That is the tree structure, which is important and should be considered for derivation process. Through this paper, we have provided an equivalent line in XML file with tree structure, for being able to process the file continually with evaluating the limited shape and form of XML file, but useful of XPath expression to select some parts of XML documents without applying loading total structure of the tree in memory after evaluating regular expression and beneficial using of the memory.

The XPath expressions about evaluating nodes can be applied for perception of the structure of a XML document, and this output with limited and certain line can be composed with other Unix devices. As an instance, the direction of the first column content of Figure 5 is demonstrated content in central column of Figure 5 that shows the number of regular expressions on the basis of different level in XML document tree and repetition of regular expression and shows them in terms of a frequency reduction order and alphabetic order in equivalent repetition number.

Sort/ uniq – c/ sort – n – r – s – t –

We can introduce a simpler form and format to show the quantity, we wanted them to be systematic and regular XPath expression and not to change the method fundamentally. When we know the form of the created structure by -nodes in XML file, one of X Path expressions could be used for deriving the considered node efficiently that this is possible by evaluating its increasing in skeletal documents that is made by applying a same process [6].

Figure 5 shows the evaluation results of r/b/c in content of Figure 4. This structure is one of the formed standard XML documents, which is the content of the file node in accordance with XPath expression. The root elements names can be customized with order line reasoning and the style sheet of preceding XSLT can be used for each derived group. This allows deriving parts of XML file by using XPath expression within an appropriate and useful space, which is evaluated just in skeletal documents. XPath expressions are not limited only to the regular expressions that are

```
/r                  | 2 /r/b/c          | /r
/r/a                | 2 /r/b/c/text()   | /r/a[@id="i1"]
/r/b                | 1 /r              | /r/b[@id="i2"]
/r/b/c              | 1 /r/a            | /r/b[@id="i2"]/c[@id="i3"]
/r/b/c/text()       | 1 /r/b            | /r/b[@id="i2"]/c[@id="i3"]/text()[.="e"]
/r/b/text()         | 1 /r/b/text()     | /r/b[@id="i2"]/text()[.="d"]
/r/b/c              |                   | /r/b[@id="i2"]/c[@id="i4"]
/r/b/c/text()       |                   | /r/b[@id="i2"]/c[@id="i4"]/text()[.="f"]
```

Figure 5. XPath Expression and its Content Description.

produced by detection mechanism, but they should begin from the root, not from grandchild and grandchildren for every branch and sub-branch. If not the survey would be more complex and takes longer time, and addressing in each level would not be directly. In addition, to research the document other than this mode the entire branch should be loaded in memory that is not an appropriate method for optimizing memory utilization and decreasing the survey duration.

<root>
<c id=lli3[11]>e</c>
<c id=li4”>f</c>
</root>

5.1 The Detection of XML File

We show how we create the settings of the skeletal documents using single transit XML analyzer. Pull analyzer, which is available from Java 1.6 as an API Stream for XML (STAX) allows the programmer to control when reading a XML file via pointer concept that reviews documents length from the outset to the end. The pointer can only move toward ahead and points to a separate point in XML file. Just a small part of the entire XML document requires loading in memory in every time, so this procedure is really effective in memory utilization.

Since the programmer control that when there is a mark in a pointer, what is it called, there is a possibility of textual information storage in reviewing the mark. A mark can be an initial tag, a final tag, character content and so on.

To show the existent information setting in skeletal documents, we read each mark and produce a DOM structure.

When we move forward in XML file, skeletal documents are made as additive XML file and are produced as mark by mark as shown in algorithm 1.

Algorithm.1. *The Detection of XML File Using skeletal Documents*

While there are tokens left in the XML file do

token←next token

if tokenis a start-tag then

convert token to an element node;

add it as child of the bottom node of the skeleton document;

output the XPath expression describing the leaf of the document

else if tokenis a character-data then

convert token to a text-node;

add it as child of the bottom node of the skeleton document;

output the XPath expression describing the leaf of this document;

remove this character-element from the skeleton document

else if tokenis an end-tag then

remove the bottom node of the skeleton document

else if tokenis a whitespace only element then

skip it

5.2 Nodes Derivation from XML File

When only several nodes require to be extracted from XML file, internal structure for the entire file completely takes time for preserving some nodes after evaluation. Ideally, it is desirable to evaluate XPath expression in a structural dynamic structure as the file is read.

To do so, we suggest that the evaluation of XPath expression in skeletal documents is limited so that it could be made increasingly in a single transit XML file as described through the previous chapter. To assess XPath expression along skeletal document for similar results as well as the evaluation along the entire document, virtual XPath expression language was somehow limited with following restrictions.

XPath expression should return a list of nodes, not a numeral quantity, boolean and string or so, but it can return a textual node.

- Expressions should be started from the root and show each phase of the tree, not grandchild and grandchildren.
- Predicates expression should not point to adjacent nodes in a branch of the tree, but should point to situations that adjacent nodes have in a branch (brother – sister)

Table 1. Examples of division and separation process

XPath expression	Parts
/r/b/c	/r
	/r/b
	/r/b/c
/r/b [substring(©id, 2) >111”J / c	/r
	/r/b[substring(©id,2»11”J/c

or the number of adjacent nodes in a branch (brother – sister).
- Predicative expression can't use any variable XSLT.

To develop the evaluation usefulness of XPath expression, we applied the tree structure of the documents and XPath expression in accordance with increasing each phase of the tree. XPath expression is divided into parts, which will be evaluated as skeletal documents. While a phase includes a predicate, the divided components and parts are in equivalence with the phases of XPath expressions. Since a predicate evaluation can't predict measures and values for low parts of the tree, XPath expression is queried to the final phase.

Algorithm.2: *shows the process of the production of Xpath expression as in XML file moves forward.*

While there are tokens left in the XML file do

token←next token;

i = 0

if tokenis a start-tag or character data then

if i < n−1 then

convert token to a node (element or text node);

add this node at the bottom of the skeleton document;

if part i of XPath matches the skeleton document then

i = i+ 1;

continue the while loop;

Else {no match}

if token is a start-tag then

skip the rest of this element (including embedded elements)

else {i = n−1}

if token is a start-tag then

read the rest of the current element;

create an element node with its content

else{token is character-data}

convert the character-data to a text-node

add the created node at the bottom of the skeleton document;

if the last part of the XPath matches the skeleton document then

output the node as result

remove the bottom element of the skeleton document

else if tokenis a end-tag then

i = i−1;

remove the bottom element of the skeleton document

6. Performance and Establishment

To find how applicable is this procedure on XL real data, we did some establishments, and previous parts show about three rather large files which are made to relate with dictionary files.

As represented in the following table, the results achieved from these experiments and assessments demonstrate that skeletal documents structure is effective enough to extract information from XML large files.

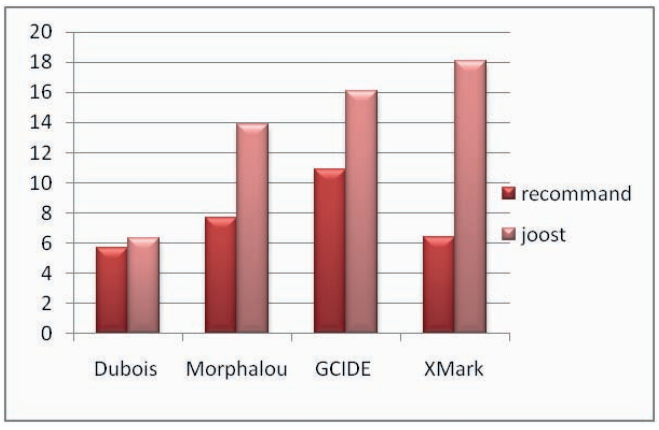
XPath expression	Recommended Method	Joost
Dubois	5.7	6.3
Morphalou	7.7	13.9
GCIDE	10.9	16.1
XMark	6.4	18.1

Times are per millisecond and are performed with 2.6 gigahertz Core 2 Duo in Windows-7.

7. Conclusion

Through this paper, we described a simple procedure to extract XML file contents and improve its contents using devices based on line. We also examined Grep devices to select nodes in XML file, and created a new XML file called skeletal documents so that using the method we do not need to save the entire file in memory.

This method could be fulfilled on hundreds lines of codes and doesn't require complex database, or a new formalism



or an interpreter. The results also show speed increasing and searching time decreasing in XML large files.

8. References

1. Cimprich P, Becker O et al. (2011). Streaming transformations for XML (STX). Technical report, Available from: http://stx.sourceforge.net/documents/spec-stx-20070427.html.
2. Damiani E, di Vimercati S D et al. (2010). Securing XML documents, Proceedings of the 7th International Conference on Extending Database Technology (EDBT2000), 121–135.
3. Chamberlin D, Florescu D et al. (2008). XQuery Update Facility 1.0, World Wide Web Consortium (W3C), Available from: http://www.w3.org/TR/2008/CR-xquery-update-10-20080314.
4. Java TM Platform, Standard Edition 6 API Specification, Available from : http://java.sun.com/
5. Özsu M T, Valduriez P (2009). Principles of distributed database systems, Prentice-Hall International, Upper Saddle River, N.J.
6. Sandhu R, Coyne E et al. (2007). Role-Based Access Control Models, IEEE Computer, vol 29(2), 38–47.
7. Sandhu R, Bhamidipati V et al. (2007). The ARBAC97 model for role-based administration of roles, ACM Transaction on Information and Systems Security, vol 2(1),105–135.