

Available from: http://dx.doi.org/10.1007/11733836_33

Copyright © 2006 Springer-Verlag Berlin Heidelberg. The original publication is available at www.springer.com.

This is the author’s version of the work. It is posted here with the permission of the publisher for your personal use. No further distribution is permitted. If your library has a subscription to these conference proceedings, you may also be able to access the published version via the library catalogue.
Algorithm 1 FindQueryChild()
{Input an element node and trigger descendant operators}
    IF (isHashTerminalNode(element)) THEN output element;
    ELSE
        q <- HashBucketFirstnode(QueryNextnode(element));
        WHILE(q!=null)DO
            IF (q.fid==elemnet.fid) THEN q.val=element.val; FindQueryChild(q); q=q.next;
            ELSE FOR(p=element.hid;p!=null&&p.hid!=q.fid;p=p.next);
                IF (p.hid==q.fid) THEN q.val= element.val; FindQueryChild(q);q = q.next;
            END IF END FOR
        END IF
    END WHILE
END IF

Algorithm 1 and 2 change the corresponding values of the hash table to schedule triggering the descendant operator and inquiring the parent operator.

Algorithm 2 FindQueryParent()
{Input an element node and inquire parent operator}
    IF (HashQueryFirstnode(element)) THEN element.val=TRUE;
    ELSE
        q <- HashBucketFirstnode(QueryPrenode(element));
        WHILE(q!=null)DO

Fig. 10. XFPro Processing Example
FOR ( ; q!= null; q = q.next )
IF (q.fid==element.fid) THEN element.val = q.val;
ELSE
FOR (phid=q.hid; phid!= null; phid = phid.next);
IF(p.hid==element.fid) THEN element.val = q.value;
END IF END FOR
END IF END FOR
END WHILE
element.val=UNDECIDED;
END IF

5 Performance Evaluation

We have implemented the XFPro translator engine in Java, which rewrites XPath expressions into tid-tree based query plans for XML fragments. Our XFPro query engine on fragmented XML streams processes the optimized queries directly on the filler fragments before reconstructing the entire XML document.

All experiments are run on a PC with 2.6GHz CPU, 512M of memory and 80G hard disk. The operating system is WindowsXP. The experiments are run on data sets generated by the xmlgen program. We have written an XML fragmenter that fragments an XML document into filler fragments to produce an XML stream, based on the tag structure defining the fragmentation layout.

We have selected three representative queries (\(Q_1\), \(Q_2\) and \(Q_3\)) on the generated XML documents and compared the results with the XFrag Processor [4].

Query1: \(\text{doc}("book.xml")/\text{book/sections/section/subsection/title}\)
Query2: \(\text{doc}("book.xml")/\text{book/section[difficulty>="default"]}/\text{title}\)
Query3: \(\text{doc}("book.xml")/\text{book/title/section[difficulty>="default"]}\)

To illustrate the differences in the query execution methods on the filler fragments, consider the Query 1 that returns the subsection title of the books. Since “section”, “subsection” and “title” are in common filler fragments, according to the fragmentation information in tag structure, our query operates “subsection” and “title” over fragment only when the fragment tsid matches that of the operator. Furthermore, each fragment is only evaluated once and hashed to corresponding item if tsid matches. While in XFrag, each fragment needs to be passed on through the pipeline and evaluated step by step. In this way, our method performs better than XFrag. The results of the experiments are summarized in Figure 11.

From the experimental results, we observe that the XFPro method outperforms the XFrag method mainly on running time, while the memory cost of these two methods makes little difference. That is because both of the methods adopt the policy of keeping the output-related information of the fragments while hash buckets use less links than association table. For the query processing time, the XFPro method saves CPU time by avoiding subsumption operations. Furthermore, the XFrag method has to schedule the operations for each fragment, while the XFPro only changes the corresponding value of the hash table.
<table>
<thead>
<tr>
<th>Query</th>
<th>File size</th>
<th>Fragmented File Size</th>
<th>Method</th>
<th>Run time</th>
<th>memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>10Mb</td>
<td>11.04Mb</td>
<td>XFPro</td>
<td>518.27ms</td>
<td>0.36Mb</td>
</tr>
<tr>
<td></td>
<td>15Mb</td>
<td>17.56Mb</td>
<td>XFrag</td>
<td>1875.00ms</td>
<td>0.42Mb</td>
</tr>
<tr>
<td></td>
<td>20Mb</td>
<td>23.18Mb</td>
<td>XFrag</td>
<td>3926.50ms</td>
<td>1.35Mb</td>
</tr>
<tr>
<td></td>
<td>20Mb</td>
<td>23.18Mb</td>
<td>XFrag</td>
<td>3926.50ms</td>
<td>1.35Mb</td>
</tr>
<tr>
<td>Q2</td>
<td>10Mb</td>
<td>11.98Mb</td>
<td>XFPro</td>
<td>3015.92ms</td>
<td>1.87Mb</td>
</tr>
<tr>
<td></td>
<td>15Mb</td>
<td>19.20Mb</td>
<td>XFrag</td>
<td>7329.70ms</td>
<td>2.13Mb</td>
</tr>
<tr>
<td></td>
<td>20Mb</td>
<td>24.12Mb</td>
<td>XFrag</td>
<td>11444.55ms</td>
<td>6.95Mb</td>
</tr>
<tr>
<td>Q3</td>
<td>10Mb</td>
<td>11.78Mb</td>
<td>XFPro</td>
<td>3005.86ms</td>
<td>2.08Mb</td>
</tr>
<tr>
<td></td>
<td>15Mb</td>
<td>19.38Mb</td>
<td>XFrag</td>
<td>7219.07ms</td>
<td>2.03Mb</td>
</tr>
<tr>
<td></td>
<td>20Mb</td>
<td>24.33Mb</td>
<td>XFrag</td>
<td>11429.71ms</td>
<td>6.64Mb</td>
</tr>
</tbody>
</table>

Fig. 11. Experimental Results

6 Conclusions

This paper has presented a framework and a set of techniques for processing XPath queries over streamed XML fragments. We present techniques for enabling the transformation from XPath expression to optimized query plan. Our query model of tid tree helps to transform queries on element nodes to queries on XML fragments and analyze “redundant” operations in them. Furthermore, such transformations specify query operations such as “/” and “*” and reduce the query workload. Based on optimized tid tree, we present a scheme to map a tid tree directly into an XML fragment query processor, and thus efficient query execution plan is generated. Our experiments show that our framework performs well on saving processing power and memory space.

Acknowledgments This research was partially supported by the National Natural Science Foundation of China (Grant No. 60473074 and 60573089) and Specialized Research Fund for the Doctoral Program of Higher Education (SRFDP).

References