XMLGet -- Testing, Profiling, Benchmarking, and Proposed Future Work

by Neil Ferguson (neilferguson@mail.com)

Contents

1. Testing

1.1 Caveats

?? Brackets cannot be used within non-path set operations. This is due to an error made when designing the parser specification.

?? If a document node is returned from a query ('.' is actually a legal query), an exception will be thrown.

?? Display filters cannot be used in queries that are part of set operations (e.g. `'//name{content}/fore $union //sur'`. It is not clear exactly how this problem should be resolved (if it should), and currently the message, the query 'Error: Cannot use display filters with path-based set operations.' is displayed.

1.2 Profiling

In order to profile the various parts of the program and test its overall performance, a number of procedures were added to the program to time the following things:

?? Time taken to parse a document

?? Time taken to parse a query

?? Time taken to execute a query

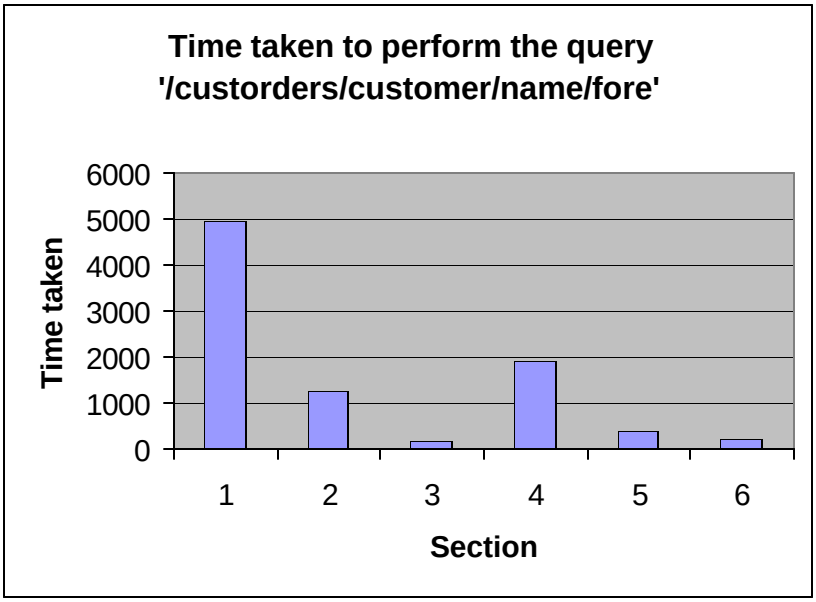
?? Time taken to solve a relative path structure

?? Time taken to output the results of a query

?? Overall time taken

The following query was executed on the performance testing document in Appendix B, containing 1000 records and being 436Kb in size:

```
/custorders/customer/name/fore
```



Number	Description	Time Taken (secs)
1	Overall time.	4.940
2	Output of elements	1.260
3	Solving 'custorders' part of the path structure	0.170
4	Solving 'customer' part of the path structure	1.920
5	Solving 'name' part of the path structure	0.390
6	Solving '' part of the path structure	0.220

From the above data we can see that finding the 'customer' elements takes far longer than the other elements, despite the fact that in the XML document there are identical numbers of 'customer', 'name' and 'fore' elements in the document. The 'customer' elements are all within one 'custorders' element, however, whereas there is one 'name' element within each 'customer' element, and one 'fore' element within each 'name' element. From this we can conclude that when an element has a large number of children, it will take longer for those children to be returned from that single element than it would take for them to be returned if they were spread over a number of elements. This could be due to the fact that a Vector object[1] is used to store matching elements when they are retrieved from the

[1] In Java a Vector is an expandable array. Useful when the amount of objects is unknown, but quite slow for large numbers of objects.

document, and future investigations could look at ways of improving performance in this area (see section 8.2).

The above data also shows that the actual outputting of the elements, as opposed to their retrieval from the document, takes quite a long time. Also, the time that this takes varies greatly depending on where they are being outputted to. As an illustration, the above data comes from when the elements are being output to a file, but when they are being output to an MS-DOS window it takes 3.460 seconds -- almost twice the time. Future extensions to the project should definitely look at ways of improving element output times, possibly by using some sort of buffer system.

Although not on the graph, parsing the query took 0.880 seconds; this seems reasonably efficient and, aside from using another parser generator and lexer, could probably not be improved upon.

Parsing the document took 2.8 seconds, and this also could probably not be improved upon much, although the results of an investigation comparing different parsers (including SAX ones) would be interesting.

The following query was also executed:

```
//fore
```

This query took a total of 4.445 seconds, showing that using recursive descent instead of a full path expression is more efficient.

More detailed profiling mechanisms are something that could easily be added to XMLGet in the future, enabling a more comprehensive investigation into the areas that could be improved upon.

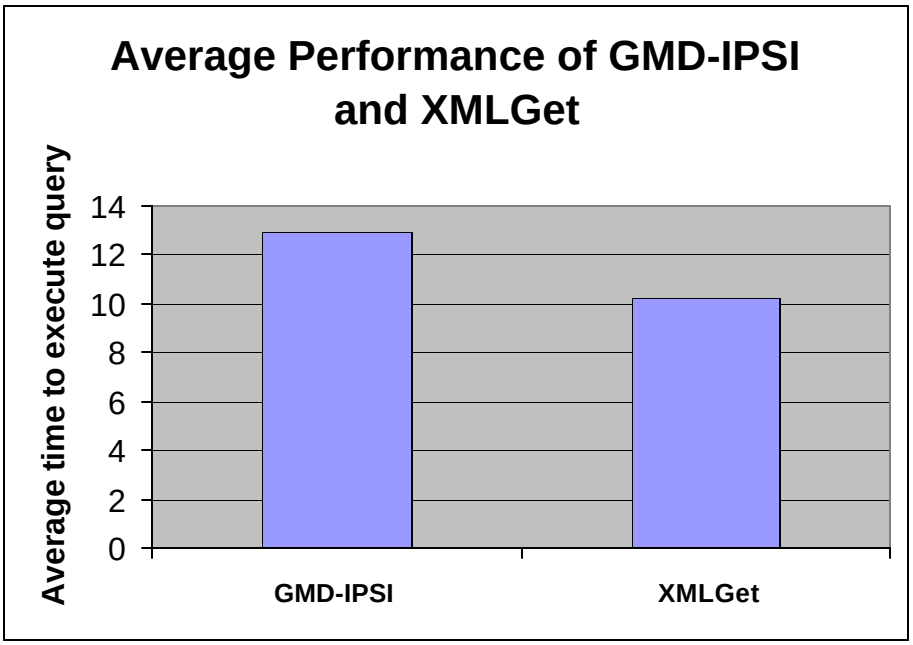
1.3 Benchmarking

The tool that was chosen for comparative testing was the GMD-IPSI XQL Engine, mainly because it is the only freely available XQL engine written in Java (it is, in fact, a demo. version of a commercial XML query tool).

The following table shows the comparative performance of the GMD-IPSI engine and XMLGet various queries (executed on the performance test document in appendix B with 1000 records, and including document parsing times; results correct 04 May 2000):

Query	GMD-IPSI	XMLGet
/custorders/customer/name/fore	10.92 secs.	8.52 secs.
//fore	10.98 secs.	7.30 secs.
/custorders/customer[@id$contains$'3']/email	11.47 secs.	8.02 secs.
/*/*/*	15.41 secs.	18.1 secs.
/*/*	15.71 secs.	9.17 secs.

These results show that XMLGet outperforms GMD-IPSI in most instances. The reason XMLGet was outperformed for the query '/*/*/*' is because it took around 12 seconds to print out all the elements (as mentioned above, this is definitely something that needs to be investigated), whereas the GMD-IPSI engine obviously has a better element display system. The graph below shows the average performance of XMLGet compared to GMD-IPSI. '1'.



1.4 Cache System Testing

To test the cache system, two similar queries were executed and the amount of time it took to retrieve the elements was measured each time. The results, for a 5000 record performance testing document (see appendix B), were as follows:

1. The query `'custorders/customer/name/fore'` was executed, and the time taken to retrieve the elements was **43.99** seconds.
2. The query `'custorders/customer/name/sur'` was executed, and the time taken to retrieve the elements was only **1.05** seconds!

Similar queries, like `'custorders/customer/email'`, were also executed and they all retrieved the elements in around 1 second.

From the above data we can conclude that XMLGet's query caching system offers very significant performance improvements when executing queries on similar sets of data.

2. Proposed Future Work

2.1 Future Work

There are many extensions and enhancements that could potentially be made to the solution, and some suggestions are as follows:

?? The issues in section 1.1 should be addressed.

?? A system for managing large document repositories could be implemented, probably consisting of:

  ?? Mechanisms for indexing large numbers of documents and for switching between file-system and in-memory storage, possibly using some sort of document cache system (see section 5.3.). Good reference points for this would be McHugh Widom, Abiteboul, Luo, and Rajaraman -- "Indexing Semistructured Data" and Dao, Sacks-Davis, and Thom -- "An indexing scheme for structured documents and its implementation".

  ?? Query optimisation techniques for large numbers of documents. A good reference point would be McHugh and Widom -- Query Optimization for XML.

?? The "display filter" system could be extended. Display filters could be implemented to count the number of nodes in a set, calculate the average value of a set of numerical nodes, return distinct nodes from a set, and some of the other operations that SQL can perform.

?? A version of the query engine could be produced that uses Java Servlet technology. This would queries to be made via. HTTP, and would enable large numbers of users to make queries over the internet without placing a high load on the network.

?? A more sophisticated profiling system could be implemented, hopefully leading to more efficient methods for performing certain operations. As identified from the existing profiling system some work could be done on the element outputting system and the procedure for retrieving large numbers of child elements.

?? When the W3C XML query working group publishes its recommendations for a query language, these could be incorporated into XMLGet.

?? XMLGet could be enhanced to support the full Unicode character set that XML supports. When specifying element names only the "western" character set is currently supported (plus the symbols . _ : -), but modifications could be made to the lexical analyser specification to support the full set.

?? Extensions could be made to enable a document's DTD to be considered when extracting elements from it. This could potentially improve overall speed, but would require very efficient DTD parsing tools to be used.

Appendix A. Test XML document.

```
<?xml version="1.0"?>

<custorders>

<customer>
        <name id="101">
                <fore>Bob</fore>
                <sur>Jones</sur>
        </name>
        <address>53 Dupont Av., Nice, France
                <postcode>re315ty</postcode>
        </address>
        <email>bob@yahoo.com</email>
        <fax>
                <countrycode>028</countrycode>
                <number>562234</number>
        </fax>
        <telephone>
                <countrycode>028</countrycode>
                <number>349876</number>
        </telephone>
        <order code="2343">lawnmower
                <itemid>bt95re</itemid>
                <date>02/04/99</date>
        </order>
        <order code="2762">lawnmower
                <itemid>cr36tr</itemid>
                <date>13/07/98</date>
        </order>
</customer>

<customer>
        <name id="775">
                <fore>Robson</fore>
                <sur>Smith</sur>
        </name>
        <address>123 Glexdale Rd., Birmingham, England
                <postcode>cv314tr</postcode>
        </address>
        <email>r.smith@mail.com</email>
        <fax>
                <countrycode>254</countrycode>
                <number>342465</number>
        </fax>
        <telephone>
                <countrycode>254</countrycode>
                <number>234565</number>
        </telephone>
        <order code="9537">lawnmower
                <itemid>tr45fg</itemid>
                <date>20/12/97</date>
        </order>
        <order code="7342">scissors
```

```
                <itemid>iu57vg</itemid>
                <date>30/05/99</date>
        </order>
</customer>

<customer>
        <name id="232">
                <fore>Alex</fore>
                <sur>Robson</sur>
        </name>
        <address>73 Faust Way, Munich, Genrmany
                <postcode>bt95re</postcode>
        </address>
        <email>alex@hotmail.com</email>
        <fax>
                <countrycode>667</countrycode>
                <number>665234</number>
        </fax>
        <telephone>
                <countrycode>667</countrycode>
                <number>342465</number>
        </telephone>
        <order code="3459">Scissors
                <itemid>vc456tt</itemid>
                <date>23/08/99</date>
        </order>
        <order code="7643">Reamer
                <itemid>vc876yt</itemid>
                <date>12/12/96</date>
        </order>
</customer>

<customer>
        <name id="667">
                <fore>Bob</fore>
                <sur>Bob</sur>
        </name>
        <address>456 De Gaulle Rd., Paris, France
                <postcode>ty56re</postcode>
        </address>
        <email>bob2@yahoo.com</email>
        <fax>
                <countrycode>028</countrycode>
                <number>553665</number>
        </fax>
        <telephone>
                <countrycode>028</countrycode>
                <number>454356</number>
        </telephone>
        <order code="2344">lawnmower
                <itemid>bt95re</itemid>
                <date>13/03/98/date>
        </order>
</customer>

<country>
        <countryname>France</countryname>
```

```
        <countrycode>028<countrycode>
</country>
<country>
        <countryname>England</countryname>
        <countrycode>254<countrycode>
</country>
<country>
        <countryname>Germany</countryname>
        <countrycode>667<countrycode>
</country>
```

Appendix B. Performance Testing Document

The document using for performance testing was generated by the following code:

```
package xmlget.server;

import java.util.Random;
import java.util.Vector;

public class GenerateXML {

        public static void main(String argv[]) {

                Random ranGen = new Random();

                int[] ranArray = new int[20];

                for (int count = 0; count < 20; count++) {

                        ranArray[count] = ranGen.nextInt(2000);

                }

                System.out.println ("<custorders>");

                for (int count = 0; count < new
Integer(argv[0]).intValue(); count++) {

                        System.out.println ("<customer id=\"" + count +
"\">");

                                System.out.println ("<name>");

                                        System.out.print ("<fore>");
                                                System.out.print
("fname" + count);
                                        System.out.println ("</fore>");

                                        System.out.print ("<sur>");
                                                System.out.print
("sname" + count);
                                        System.out.println ("</sur>");

                                System.out.println ("</name>");

                                System.out.print ("<address>");
                                System.out.print (ranGen.nextInt(99) + "
street, city, county");
                                        System.out.print ("<postcode>");
                                                System.out.print ("CV" +
ranGen.nextInt(999) + "AL");
                                        System.out.print
("</postcode>");
```

```
                                System.out.println ("</address>");

                                System.out.print ("<email>");
                                        System.out.print ("fname" +
count + ".sname" + count + "@whatever.com");
                                System.out.println ("</email>");

                                for (int count2 =0; count2 < 3;
count2++) {

                                        System.out.println ("<order
code=\"" + ranGen.nextInt(10000) + "\">");

                                                System.out.println
("<ordereditem id=\"" + ranArray[ranGen.nextInt(19)] + "\"/>");

                                                System.out.print
("<date>");
                                                        System.out.print
(ranGen.nextInt(30) + "/" + ranGen.nextInt(12) + "/" +
ranGen.nextInt(99));
                                                System.out.println
("</date>");

                                        System.out.println ("</order>");

                                }

                        System.out.println ("</customer>");

                }

        System.out.println ("</custorders>");

        }

}
```

This created an XML document with multiple “records” structured like this:

```
<custorders>
<customer id="0">
<name>
<fore>fname0</fore>
<sur>sname0</sur>
</name>
<address>19 street, city, county<postcode>CV545AL</postcode></address>
<email>fname0.sname0@whatever.com</email>
<order code="4809">
<ordereditem id="584"/>
<date>1/10/20</date>
</order>
<order code="8275">
```

```
<ordereditem id="1274"/>
<date>9/0/52</date>
</order>
<order code="9349">
<ordereditem id="584"/>
<date>22/0/59</date>
</order>
```

etc.

The customer ‘id’ attribute and the numerical part of fname and sname are incremented for each “record”, and the other numbers are randomly generated.