XPathMark
An XPath benchmark for XMark

Massimo Franceschet

Informatics Institute
University of Amsterdam, The Netherlands
Department of Sciences
University of Chieti and Pescara, Italy

Abstract

We propose XPathMark, an XPath benchmark for XMark. It consists of a set of queries which covers the main aspects of the language XPath 1.0. These queries have been designed for XML documents generated under XMark, a popular benchmark for XML data management. We suggest a methodology to evaluate the XPath benchmark on a given XML engine and, by way of example, we evaluate two popular XML engines against the proposed benchmark.

1 Introduction

XMark [5] is a well-known benchmark for XML data management. It consists of a scalable document database modelling an Internet auction website and a concise and comprehensive set of XQuery queries which covers the major aspects of XML query processing.

XQuery [9] is much larger than XPath [6], and the list of queries provided in the XMark benchmark mostly focuses on XQuery features (joins, construction of complex results, grouping) and provides little insight about XPath characteristics. In particular, only `child` and `descendant` XPath axes are exploited. In this paper, we propose XPathMark [4], an XPath 1.0 benchmark for XMark. We have developed a set of XPath queries which covers the major aspects of the XPath language including different axes, node tests, Boolean operators, references, and functions. The queries are concise, easy to read and to understand. They have a natural interpretation with respect to the semantics of XMark generated XML documents. Moreover, we have thought most of the queries in such a way that the sizes of the intermediate and final results they compute, and hence the response times as well, increase as the size of the document grows. XMark comes with an XML generator that produces XML documents according to a numeric scaling factor proportional to the document size.

The targets of XPathMark are:

- *functional completeness*, that is, the ability to support the features offered by XPath;
- *correctness*, that is, the ability to correctly implement the features offered by XPath;
- *efficiency*, that is, the ability to *efficiently* process XPath queries;
- *data scalability*, that is, the ability to efficiently process XPath queries on documents of increasing sizes.

Since XPath is the core retrieval language for XSLT [7], XPointer [8] and XQuery [9], we think that the proposed benchmark can help vendors, developers, and users to evaluate these targets on XML engines implementing these technologies.

Our contribution is as follows. In Section 2 we take a close look to the XQuery benchmark proposed in XMark. In Section 3 we describe the proposed benchmark XPathMark, while in Section 4 we suggest how to evaluate XPathMark on a given XML engine. Moreover, by way of example, we evaluate, using XPathMark, two popular XML engines, namely Saxon [3] and Galax [1]. In the appendix of the paper we include the DTD for XMark documents.

2 XMark: a benchmark for XML data management

In this section we focus on the XQuery benchmark proposed in XMark. We do so in order to: (i) investigate which of the queries of the XQuery benchmark are rewritable into equivalent XPath queries and which are not. This may be useful to understand the differences between XQuery and XPath. For some applications, XPath could be enough as a query language, and using the more powerful XQuery instead may not be a wise choice; (ii) show that the XPath fragment used in the XQuery benchmark is not large enough to be considered a benchmark itself.

We proceed as follows: we first write the XQuery query in natural language, then, if it has a correspondent in XPath, we write first the XQuery version and then the corresponding XPath one, otherwise we explain why the query in not expressible in XPath. In the appendix we have included the XMark DTD.

Q1 *The name of the item with ID 'item20748' registered in North America*

```
for $b in doc("auction.xml")/site/regions/namerica/item[@id="item20748"]
return $b/name/text()
```

```
/site/regions/namerica/item[@id="item20748"]/name/text()
```

Q2 *The initial increases of all open auctions*

```
for $b in doc("auction.xml")/site/open_auctions/open_auction
return <increase>{$b/bidder[1]/increase/text()}</increase>
```

```
/site/open_auctions/open_auction/bidder[1]/increase/text()
```

Q3 *The first and current increases of all open auctions whose current increase is at least twice as high as the initial increase*

This query has no XPath correspondent since its result is a set of pairs of nodes, and not a set of nodes. For the same reason, queries Q8, Q9, Q11, Q12, Q13, and Q19 cannot be expressed in XPath.

Q4 *List the reserves of those open auctions where a certain person issued a bid before another person*

```
for $b in doc("auction.xml")/site/open_auctions/open_auction
where $b/bidder[personref/@person="person18829"]/following-sibling::
         bidder[personref/@person="person10487"]
return <history>{$b/reserve/text()}</history>
```

```
/site/open_auctions/open_auction[bidder[personref/@person="person18829"]
/following-sibling::bidder[personref/@person="person10487"]]/reserve/text()
```

Q5 *How many sold items cost more than 40?*

```
count(for $i in doc("auction.xml")/site/closed_auctions/closed_auction
      where  $i/price/text() >= 40
      return $i)

count(/site/closed_auctions/closed_auction[price >= 40])
```

Q6 *How many items are listed on all continents?*

```
for $b in doc("auction.xml")/site/regions
return count ($b//item)

count(/site/regions//item)
```

Q7 *How many pieces of prose are in our database?*

```
for $p in doc("auction.xml")/site
return count($p//description) + count($p//annotation) + count($p//email)

count(/site//description | /site//annotation | /site//email)
```

Q10 *List all persons according to their interest; use French markup in the result*

Query Q10 cannot be expressed in XPath since XPath is not able to group and reconstruct information. For the same reason, query Q20 is not expressible in XPath.

Q14 *The names of all items whose description contains the word 'gold'*

```
for $i in doc("auction.xml")/site//item
where contains($i/description,'gold')
return $i/name/text()

/site/regions/*/item[contains(description,'gold')]/name/text()
```

Q15 *The keywords in emphasis in annotations of closed auctions*

```
for $a in doc("auction.xml")/site/closed_auctions/closed_auction/annotation/
          description/parlist/listitem/parlist/listitem/text/keyword/emph/text()
return <text>{$a}</text>

/site/closed_auctions/closed_auction/annotation/description
/parlist/listitem/parlist/listitem/text/keyword/emph/text()
```

Q16 *The identifiers of the sellers of those auctions that have one or more keywords in emphasis*

```
for $a in doc("auction.xml")/site/closed_auctions/closed_auction
where not(empty($a/annotation/description/parlist/listitem/parlist/
                listitem/text/keyword/emph/text()))
return <person id = "{$a/seller/@person}" />

/site/closed_auctions/closed_auction[annotation/description/parlist
/listitem/parlist/listitem/text/keyword/emph/text()]/seller/@person
```

Q17 *Which persons don't have a homepage?*

```
for $p in doc("auction.xml")/site/people/person
where empty($p/homepage/text())
return <person name = "{$p/name/text()}" />

/site/people/person[not(homepage/text())]/name/text()
```

Q18 *Convert the currency of the reserve of all open auctions to another currency*

This query is not expressible in XPath since there are no user defined functions in XPath.

It turns out that 13 over 20 queries of the XQuery benchmark can be expressed in XPath.

3 XPathMark: an XPath benchmark for XMark

XPathMark has been designed in XML and is available at the XPathMark website [4]. In this section we describe the benchmark queries.

We first motivate our choice of developing the benchmark as XML data. This solution has all the advantages of XML [2]. In particular:

- the benchmark can be easily red, shipped, and modified;
- the benchmark can be queried with any XML query language (also with XPath itself);
- it is easier to write a *benchmark checker*, that is an application that automatically checks the benchmark against a given XML engine, that computes performance indexes, and that shapes the performance outcomes in different formats (plain text, XML, HTML, Gnuplot).

Figure 1 contains the Document Type Definition (DTD) for the XML document containing the benchmark. The root element is named `benchmark` and has the attributes `targets` (the targets of the benchmark, for instance, functional completeness), `language` (the language for which the benchmark has been written, for instance XPath 1.0), and `authors` (the authors of the benchmark). The benchmark element is composed of a sequence of `document` elements followed by a sequence of `query` elements. Each `document` element is identified by an attribute called `id` of type ID and contains, enclosed into a Character Data (CDATA) section, a possible target XML document for the benchmark queries. Each `query` element is identified by an attribute called `id` of type ID and has an attribute called `against` of type IDREF that refers to the document against which the query must be evaluated. Moreover, each `query` element contains the following child elements:

```
<!DOCTYPE benchmark [

<!ELEMENT benchmark     (document*,query*)>
<!ELEMENT document      (#PCDATA)>
<!ELEMENT query         (type,description,syntax,answer)>
<!ELEMENT type          (#PCDATA)>
<!ELEMENT description   (#PCDATA)>
<!ELEMENT syntax        (#PCDATA)>
<!ELEMENT answer        (#PCDATA)>

<!ATTLIST benchmark targets   CDATA #REQUIRED
                    language  CDATA #REQUIRED
                    authors   CDATA #REQUIRED>
<!ATTLIST document  id        ID #REQUIRED>
<!ATTLIST query     id        ID #REQUIRED
                    against   IDREF #REQUIRED>
]>
```

Figure 1: The benchmark DTD

- `type`, containing the category of the query;
- `description`, containing a description of the query in English;
- `syntax`, containing the query formula in the benchmark language syntax;
- `answer`, containing the result of the evaluation of the query against the pointed document, enclosed within a CDATA section. The result is always a sequence of XML elements with no separator between two consecutive elements (not even a whitespace).

We have included in the benchmark two target documents. The first document corresponds to the XMark document generated with a scaling factor of 0.0005. A document type definition is included in this document. The set of queries that have been evaluated on this document are divided into the following 5 categories: axes, node tests, Boolean operators, references, and functions. In the following, for each category, we give the corresponding benchmark queries . See the appendix of this paper or [5] for the XMark DTD.

- **Axes**

  These queries focus on the navigational features of XPath, that is on the different kinds of axes that may be exploited to browse the XML document tree. XPath contains 13 axes: `child`, `descendant`, `descendant-or-self`, `parent`, `ancestor`, `ancestor-or-self`, `following-sibling`, `preceding-sibling`, `following`, `preceding`, `attribute`, and `self`.

  - **Child**

    One short query (Q1) with a possibly large answer set, and a deeper one (Q2) with a smaller result. Only the **child** axis is exploited in both the queries.

    Q1 *All the items*

    ```
    /site/regions/*/item
    ```

Q2 *The keywords in annotations of closed auctions*

```
/site/closed_auctions/closed_auction/annotation
/description/parlist/listitem/text/keyword
```

– **Descendant axes**

The tag `keyword` may be arbitrarily nested in the document tree and hence the following queries can not be rewritten in terms of child axis. Notice that **listitem** elements may be nested in the document. During the processing of query Q4 an XPath processor should avoid to search the same subtree twice.

Q3 *All the keywords*

```
//keyword
```

Q4 *The keywords in a paragraph item*

```
/descendant-or-self::listitem/descendant-or-self::keyword
```

– **Parent**

Items are children of the world region they belong to. Since XPath does not allow disjunction at axis step level, one way to retrieve all the items belonging to either North or South America is to combine the **parent** axis with disjunction at filter level (another solution is Q22 that uses disjunction at query level).

Q5 *The (either North or South) American items*

```
/site/regions/*/item[parent::namerica or parent::samerica]
```

– **Ancestor axes**

The tag `keyword` may be arbitrarily deep in the document tree hence the ancestor operator in the following query may have to ascend the tree of an arbitrarily number of levels.

Q6 *The paragraph items containing a keyword*

```
//keyword/ancestor::listitem
```

Q7 *The mails containing a keyword*

```
//keyword/ancestor-or-self::mail
```

– **Sibling axes**

The `bidder` elements of a given open auction are siblings, and the XPath sibling axes may be exploited to explore them. As for query Q4 above, during the processing of query Q9, the XPath processor should take care to visit each bidder only once.

Q8 *The open auctions in which a certain person issued a bid before another person*

```
/site/open_auctions/open_auction[bidder[personref/@person='person0']
/following-sibling::bidder[personref/@person='person1']]
```

Q9 *The past bidders of a given open auction*

```
/site/open_auctions/open_auction[@id='open_auction0']
/bidder/preceding-sibling::bidder
```

– **Following and preceding**

`following` and `preceding` are powerful axes since they may potentially traverse all the document in document or reverse document order. In particular, `following` and `preceding` generally explore more than `following-sibling` and `preceding-sibling`. Compare query Q8 with query Q11: while in Q8 only sibling bidders are searched, in Q11 also bidders of different auctions are accessed.

Q10 *The items that follow, in document order, a given item*

```
/site/regions/*/item[@id='item0']/following::item
```

Q11 *The bids issued by a certain person that precedes, in document order, the last bid in document order of another person*

```
/site/open_auctions/open_auction/bidder[personref/@person='person1']
/preceding::bidder[personref/@person='person0']
```

– **Attribute**

We have already used the `attribute` axis in combination with other axis. The following two queries focus on this axis in isolation.

Q12 *All the featured items*

```
//item[@featured='yes']
```

Q13 *The elements having declared an attribute* id

```
//*[@id]
```

– **Namespace**

XMark generated documents do not declare namespaces except the implicit one with prefix xml, which is declared at the document root and hence is valid for the entire document.

Q14 *Person elements in scope of some a namespace with prefix* xml

```
//person[namespace::xml]
```

– **Self**

The `self` axis is useful to filter elements according to their string values, like in the next query.

Q15 *The increases grater than 20*

```
//increase[. > 20]
```

- **Node tests**

There are 8 node tests in XPath, namely: `name`, `*`, `prefix:*`, `node()`, `comment()`, `processing-instruction()`, `processing-instruction('target')`, and `text()`. The first two node tests have been already demonstrated above (see, e.g., query Q1). The

next six queries focus on the remaining node tests. Notice that XMark documents do not contain comments and processing instructions and do not define namespaces.

Q16 *The elements whose namespace URIs are the same as the namespace URI to which the prefix* xml *is mapped*

```
//xml:*
```

Q17 *Any children node of the root*

```
/node()
```

Q18 *The children nodes of the root that are comments*

```
/comment()
```

Q19 *The children nodes of the root that are processing instructions*

```
/processing-instruction()
```

Q20 *The children nodes of the root that are processing instructions with target robots*

```
/processing-instruction('robots')
```

Q21 *The text nodes that are contained in the keywords of the description element of a given item*

```
/site/regions/*/item[@id='item0']/description//keyword/text()
```

- **Boolean operators**

  Queries may be disjuncted with the | operator, while filters may be arbitrarily combined with conjunction, disjunction, and negation. This calls for the implementation of intersection, union, and set difference on context sets. These operations might be expensive if the XPath engine does not maintain (document) sorted the context sets.

  Q22 *The (either North or South) American items*

  ```
  /site/regions/namerica/item | /site/regions/samerica/item
  ```

  Q23 *People having an address and either a phone or a homepage*

  ```
  /site/people/person[address and (phone or homepage)]
  ```

  Q24 *People having no homepage*

  ```
  /site/people/person[not(homepage)]
  ```

- **References**

  References break down the typical tree symmetry of XML documents. A reference may potentially point to any node in the document having an attribute of type ID. Chasing a reference implies the ability of coping with arbitrary jumps in the document tree. However, references are crucial to avoid redundancy in the XML database and to implement joins in the query language. In summary, references provide data and query flexibility and they pose new challenges to the query processors.

Reference chasing is implemented in XPath with the function `id()` and may be static, like in query Q25, or dynamic, like in queries Q26-Q29. The `id()` function may be nested (like in query Q27) and its result may be filtered (like in query Q28). The `id()` function may also be used inside filters (like in query Q29).

Q25 *The name of a given person*

```
id('person0')/name
```

Q26 *The open auctions that a given person is watching*

```
id(/site/people/person[@id='person1']/watches/watch/@open_auction)
```

Q27 *The sellers of the open auctions that a given person is watching*

```
id(id(/site/people/person[@id='person1']/watches/watch/@open_auction)
/seller/@person)
```

Q28 *The American items bought by a given person*

```
id(/site/closed_auctions/closed_auction[buyer/@person='person4']
/itemref/@item)[parent::namerica or parent::samerica]
```

Q29 *The items sold by Alassane Hogan*

```
id(/site/closed_auctions/closed_auction
[id(seller/@person)/name='Alassane Hogan']/itemref/@item)
```

- **Functions**

  XPath defines 27 built-in functions for use in XPath expressions. We have categorized them as follows:

  - **Node-set functions**

    These are the following functions: `position()`, `last()`, `count()`, `id()`, `local-name()`, `name()`, `namespace-uri()`, and `lang()`. The function `id()` has been already demonstrated above.

    Q30 *The initial and last bidder of all open auctions*

    ```
    /site/open_auctions/open_auction
    /bidder[position()=1 and position()=last()]
    ```

    Q31 *The open auctions having more than 5 bidders*

    ```
    /site/open_auctions/open_auction[count(bidder)>5]
    ```

    Q32 *The elements with local name* item

    ```
    //*[local-name()='item']
    ```

    Q33 *The elements with qualified prefixed name* svg:item

    ```
    //*[name()='svg:item']
    ```

Q34 *The elements with a defined namespace URI*

```
//*[boolean(namespace-uri())]
```

Q35 *The elements written in Italian language*

```
//*[lang('it')]
```

– **String functions**
This section contains examples about the following functions: `concat()`, `contains()`, `normalize-space()`, `starts-with()`, `string()`, `string-length()`, `substring()`, `substring-after()`, `substring-before()`, and `translate()`.

Q36 *The items whose description contains the word 'gold'*

```
/site/regions/*/item[contains(description,'gold')]
```

Q37 *People having a name starting with 'Ed'*

```
/site/people/person[starts-with(name,'Ed')]
```

Q38 *Mails sent on the 10th*

```
/site/regions/*/item/mailbox/mail[substring-before(date,'/')='10']
```

Q39 *Mails sent in September*

```
/site/regions/*/item/mailbox/mail
[substring-before(substring-after(date,'/'),'/')='09']
```

Q40 *Mails sent in 1998*

```
/site/regions/*/item/mailbox/mail
[substring-after(substring-after(date,'/'),'/')='1998']
```

Q41 *Mails sent in the 21st century*

```
/site/regions/*/item/mailbox/mail[substring(date,7,2)='20']
```

Q42 *Items with a space-normalized description longer than 1000 characters*

```
/site/regions/*/item
[string-length(normalize-space(string(description))) > 1000]
```

Q43 *People with an address longer than 30 characters*

```
/site/people/person[string-length(translate(concat(address/street,
address/city,address/country,address/zipcode)," ",""")) > 30]
```

– **Number functions**
This section contains examples about the following functions: `ceiling()`, `floor()`, `number()`, `round()`, and `sum()`.

Q44 *Open auctions with a total increase greater or equal to 70*

```
/site/open_auctions/open_auction[floor(sum(bidder/increase)) >= 70]
```

Q45 *Open auctions with a total increase less or equal to 70*

```
/site/open_auctions/open_auction[ceiling(sum(bidder/increase)) <= 70]
```

Q46 *Open auctions with an average increase greater than 8*

```
/site/open_auctions/open_auction
[round((number(current) - number(initial)) div count(bidder)) > 8]
```

– **Boolean functions**
This section contains examples about the following functions: `boolean()`, `true()`, `false()`, and `not()`.

Q47 *People with both an email address and a homepage*

```
/site/people/person[boolean(emailaddress) = true() and
not(boolean(homepage)) = false()]
```

XMark documents do not contain any comment or processing instruction. Moreover, they do no declare namespaces and language attributes. Although we have used these features above, the corresponding queries do not give interesting insights when evaluated on XMark documents, since their answer sets are trivial. Therefore, we have included in the benchmark a second document and a different set of queries in order to test these features only. The document has been adapted from Example 4-4 in [2]. The benchmark queries are the following:

A1 *The children nodes of the root that are comments*

```
/comment()
```

A2 *The children nodes of the root that are processing instructions with target* robots

```
/processing-instruction('robots')
```

A3 *The elements in scope of a namespace with prefix* xlink

```
//*[namespace::xlink]
```

A4 *The elements in scope of an XLink namespace (that is, a namespace bound to the URI http://www.w3.org/1999/xlink)*

```
//*[namespace::* = 'http://www.w3.org/1999/xlink']
```

A5 *The elements whose namespace URIs are the same as the namespace URI to which the prefix* xlink *is mapped*

```
//xlink:*
```

A6 *The elements in scope of a namespace with prefix* svg

```
//*[namespace::svg]
```

A7 *The elements in scope of a Scalable Vector Graphics namespace (a namespace bound to the URI http://www.w3.org/2000/svg)*

```
//*[namespace::* = 'http://www.w3.org/2000/svg']
```

A8 *The elements whose namespace URIs are the same as the namespace URI to which the prefix* svg *is mapped*

```
//svg:*
```

A9 *The elements with a defined namespace URI*

```
//*[boolean(namespace-uri())]
```

A10 *The elements with local name* ellipse

```
//*[local-name()='ellipse']
```

A11 *The elements with qualified prefixed name* svg:ellipse

```
//*[name()='svg:ellipse']
```

A12 *The elements written in Italian language*

```
//*[lang('it')]
```

4 Evaluation of XML engines

In this section we suggest how to evaluate XPathMark on a given XML engine. Moreover, by way of example, we evaluate, using XPathMark, two popular XML engines, namely Saxon [3] and Galax [1].

4.1 Evaluation methodology

XPathMark can be checked on a set of XML processors and conclusions about the performances of the processors can be drawn. In this section, we suggest a method to do this evaluation.

We describe a set of *performance indexes* that might help the evaluation and the comparison of different XML processors that have been checked with XPathMark. We say that a query is *supported* by an engine if the engine processes the query without giving an error. A supported query is *correct* with respect to an engine if it returns the correct answer for the query. We define the *completeness index* as the number of supported queries divided by the number of benchmark queries, and the *correctness index* as the number of supported and correct queries divided by the number of supported queries. The completeness index gives an indication of how much of the benchmark language (XPath in our case) is supported by the engine, while the correctness index reveals the portion of the benchmark language that is correctly implemented by the engine.

XMark benchmark offers a document generator that generates XML documents of different sizes according to a numeric scaling factor. The document size grows linearly with respect to the scaling factor. For instance, factor 0.01 corresponds to a document of (about) 1,16 MB and factor 0.1 corresponds to a document of (about) 11,6 MB. Given an XMark document and a benchmark query, we can measure the time that the XML engine takes to evaluate the queries on the document. The *query response time* is the time taken by the engine to give the answer for the query on the document, including parsing of the document, parsing, optimization, and processing of the query, and serialization of the results. It might be interesting to evaluate the *query processing time* as well, which is the fraction of the query response time that the engine takes to process the query only, excluding the parsing of the document and the serialization of the results. We define the *query response speed* as the size of the document divided by the query response time. The measure unit is, for instance, MB/sec.

We may run the query against a documents series of documents of increasing sizes. In this case, we have a *speed sequence* for the query. The *average query response speed* is the average of the query response speeds over the document series. Moving from one document

(size) to another, the engine may show either a positive or a negative *acceleration* in its response speed, or the speed may remain constant.

The concept of speed acceleration is intimately connected to that of *data scalability*. Consider two documents d_1 of size s_1 and d_2 of size s_2 in the document series with $s_1 < s_2$, and a query q. Let t_1 and t_2 be the response times for query q on documents d_1 and d_2, respectively. Let $v_1 = s_1/t_1$ be the speed of q over d_1 and $v_2 = s_2/t_2$ be the speed of q over d_2. The *data scalability factor* for query q is defined as:

$$\frac{v_1}{v_2} = \frac{t_2 \cdot s_1}{t_1 \cdot s_2}$$

If the scalability factor is lower than 1, that is $v_1 < v_2$, then we have a *positive speed acceleration* when moving from document d_1 to document d_2. In this case, we say that the scalability is *sub-linear*. If the scalability factor is higher than 1, that is $v_1 > v_2$, then we have a *negative speed acceleration* when moving from document d_1 to document d_2. In this case, we say that the scalability is *super-linear*. Finally, if the scalability factor is equal to 1, that is $v_1 = v_2$, then the speed is constant when moving from document d_1 to document d_2. In this case, we say that the scalability is *linear*. A sub-linear scalability means that the response time grows less than linearly, while a super-linear scalability means that the response time grows more than linearly. A linear scalability indicates that the response time grows linearly. For instance, if $s_2 = 2 \cdot s_1$ and $t_2 = 4 \cdot t_1$, then the scalability factor is 2 and the time grows quadratically on the considered segment.

Once again, we may run query q against a document series of documents of increasing sizes and generate a *data scalability sequence* for query q. The *average data scalability factor* for query q is the average of the data scalability factors for query q over the document series.

All these indexes can be computed for a single query or for an arbitrary subset of the benchmark. Of particular interest is the case when the whole benchmark is considered. Given a document d, we define the *average benchmark response time* for d as the average of the response times of all the benchmark queries on document d. Moreover, the *benchmark response speed* for d is defined as the size of d divided by the average benchmark response time. Notice that the benchmark response speed is different from the average of the response speeds for all the benchmark queries. Finally, the *data scalability factor for the benchmark* is defined as above in terms of the benchmark response speed. If we take the average of the benchmark response speed (respectively, data scalability factor for the benchmark) over a document series we get the *average benchmark response speed* (respectively, *average data scalability factor for the benchmark*). The former indicates how fast the engine processes XPath, while the latter reveals how well the engine scales-up with respect to XPath when the document size increases.

The outcomes of the evaluation for a specific XML engine should be formatted in XML. In Figure 2 we suggest a DTD for this purpose. The document root is named `benchmark`. The engine under evaluation and its version are specified as attributes of the element `benchmark`. The benchmark element has an `index` child and zero or more `query` children.

The `index` element contains the performance indexes of the engine and has attributes describing the testing environment. The testing environment contains information about the processor (`cpu`), the main memory (`memory`), the operating system (`os`), the time unit (`time_unit`), and the time type, that is, either response or processing time (`time_type`). The performance indexes are: the completeness index (`completeness`), the correctness index (`correctness`), a sequence of average benchmark response times for a document series (`times`, a sequence of `time` elements), a sequence of benchmark response speeds for a document series (`speeds`, a sequence of `speed` elements), a sequence of data scalability factors

```
<!ELEMENT benchmark        (indexes,query*)>

<!ELEMENT indexes          (completeness,correctness,
                            times?,speeds?,scalas?,aqrs?,ads?)>
<!ELEMENT completeness     (#PCDATA)>
<!ELEMENT correctness      (#PCDATA)>
<!ELEMENT times            (qrt+)>
<!ELEMENT speeds           (qrs+)>
<!ELEMENT qrt              (#PCDATA)>
<!ELEMENT qrs              (#PCDATA)>
<!ELEMENT scalas           (scala+)>
<!ELEMENT scala            (PCDATA)>
<!ELEMENT aqrs             (#PCDATA)>
<!ELEMENT ads              (#PCDATA)>

<!ELEMENT query            (type,description,syntax,supported,error?,
                            correct,given_answer?,expected_answer?,
                            times?,speeds?,scalas?,aqrs?,ads?)>
<!ELEMENT type             (#PCDATA)>
<!ELEMENT description      (#PCDATA)>
<!ELEMENT syntax           (#PCDATA)>
<!ELEMENT supported        EMPTY>
<!ELEMENT error            (#PCDATA)>
<!ELEMENT correct          EMPTY>
<!ELEMENT given_answer     (#PCDATA)>
<!ELEMENT expected_answer  (#PCDATA)>

<!ATTLIST benchmark engine    CDATA #REQUIRED
                    version   CDATA #REQUIRED>
<!ATTLIST query     id        ID #REQUIRED>
<!ATTLIST indexes   cpu       CDATA #IMPLIED
                    memory    CDATA #IMPLIED
                    os        CDATA #IMPLIED
                    time_unit (msec | csec | dsec | sec)  #IMPLIED
                    time_type (elapsed | cpu) #IMPLIED>
<!ATTLIST supported value     (yes | no) #REQUIRED>
<!ATTLIST correct   value     (yes | no) #REQUIRED>
<!ATTLIST qrt       factor    CDATA #REQUIRED>
<!ATTLIST qrs       factor    CDATA #REQUIRED>
<!ATTLIST scala     factor1   CDATA #REQUIRED
                    factor2   CDATA #REQUIRED>
```

Figure 2: The DTD for a benchmark outcome

for the benchmark for a document series (`scalas`, a sequence of `scala` elements), the average benchmark response speed (`avgspeed`), and the average data scalability factor for the benchmark (`avgscala`). Each element of type `time` and `speed` has an attribute called `factor` indicating the factor of the XMark document on which it has been computed. Moreover, each element of type `scala` has two attributes called `factor1` and `factor2` indicating the factors of the two XMark documents on which it has been computed.

Each `query` element contains information about the single query and is identified by an attribute called `id` of type ID. In particular, it includes the category of the query (`type`), a description in English (`description`), the XPath syntax (`syntax`), whether or not the query is supported by the benchmarked engine (`supported`, it must be either `yes` or `no`), the possible error message (`error`, only if the query is not supported), whether or not the query is correctly implemented by the benchmarked engine (`correct`, it must be either `yes`, `no`, or `undef`. The latter is used whenever the query is not supported), the given and expected query answers (`given_answer` and `expected_answer`. They are used for comparison only if the query is not correct), a sequence of query response times for a document series (`times`, as above), a sequence of query response speeds for a document series (`speeds`, as above), a sequence of data scalability factors for the query for a document series (`scalas`, as above), the average query response speed (`avgspeed`), and the average data scalability factor for the query (`avgscala`).

The solution of composing the results in XML format has a number of advantages. First, the outcomes are easier to extend with different evaluation parameters. More importantly, the outcomes can be queried to extract relevant information and to compute performance indexes. For instance, the following XPath query retrieves the benchmark queries that are supported but not correctly implemented:

```
/benchmark/query[supported="yes" and correct="no"]/syntax
```

Moreover, the following XQuery computes the completeness and correctness indexes:

```
let $x := doc("outcome_engine.xml")/benchmark/query let $y :=
$x[supported="yes"] let $z := $x[correct="yes"] return <indexes>
         <completeness> {count($y) div count($x)} </completeness>
         <correctness> {count($z) div count($y)} </correctness>
       </indexes>
```

Finally, the following XQuery computes the average query response time of queries over the axes category when evaluated on the XMark document with scaling factor 0.1:

```
let $x := doc("outcome_engine.xml")/
          benchmark/query[type="axes" and correct="yes"]
let $y := sum($x/times/time[@factor="0.1"]) let $z := count($x) return
<average_time> {$y div $z} </average_time>
```

More generally, one can easily program a benchmark checker that automatically tests and evaluates different XML engines with respect to XPathMark.

4.2 Evaluating Saxon and Galax

We ran the XPathMark benchmark on two state-of-the-art XML engines, namely Saxon [3] and Galax [1]. Saxon technology is available in two versions: the basic edition Saxon-B, available as an open-source product, and the schema-aware edition Saxon-SA available on a commercial license. We tested Saxon-B 8.4, with Java 2 Platform, Standard Edition 5.0.

Galax is the most popular native XQuery engine available in open-source and it is considered a reference system in the database community for its completeness and adherence to the standards. We tested version 0.5. We ran all the tests on a 3.20 GHz Intel Pentium 4 with 2GB of main memory under Linux version 2.6.9-1.667 (Red Hat 3.4.2-6.fc3). All the times are response CPU times in seconds. For each engine, we ran all the supported queries on XMark documents of increasing sizes. The document series is the following (XMark factors):

(0.001, 0.002, 0.004, 0.008, 0.016, 0.032, 0.064, 0.128, 0.256, 0.512, 1)

corresponding to the following sizes (in MB):

(0.116, 0.212, 0.468, 0.909, 1.891, 3.751, 7.303, 15.044, 29.887, 59.489, 116.517)

It is worth noticing that in the computation of the completeness index we did not consider queries using the `namespace` axis, since this axis is no more supported in XQuery [9].

The whole evaluation outcomes can be accessed from the XPathMark website [4]. This includes the outcomes in XML for both the engines and some plots illustrating the behaviour of the performance indexes we have defined in this paper. In order to compare efficiency and scalability of the two engines, we also evaluated the subset of the benchmark corresponding to the intersection of the query sets supported by the two engines (which are different). This common base is the query set {Q1-Q9,Q12,Q13,Q15-Q24,Q30-Q47} of cardinality 39. In the following we report about our findings.

1. **Completeness and correctness.** The completeness and the correctness indexes for Saxon are both 1, meaning that Saxon supports all the queries in the benchmark (excluding queries using the `namespace` axis, which are not counted) and all supported queries give the correct answer. The completeness index for Galax is 0.85. In particular, the axes `following` and `preceding` (which are in fact optional in XQuery) and the `id()` function are not supported by Galax. However, all the supported queries give correct answers, hence the correctness index for Galax is 1.

2. **Efficiency.** On the common query set, the average benchmark response speed for Saxon is 2.80 MB/sec and that for Galax is 2.50 MB/sec. This indicates that Saxon is faster than Galax to process the (checked subset of the) benchmark. The average response time for a query in the benchmark, varying the document size, is depicted in Figure 3 (left side is from factor 0.001 to factor 0.032 and right side is from factor 0.032 to factor 1). Interestingly enough, Galax outperforms Saxon in the first track, corresponding to small documents (up to 3.7 MB), but Saxon catches up in the second track, corresponding to bigger documents. This trend is confirmed by the behaviour of the benchmark response speeds (see Figure 4 corresponding to the same segments of the document series).

3. **Scalability.** On the common query set, the average data scalability factor for the checked benchmark is 0.80 in the case of Saxon and it is 0.98 in the case of Galax. This indicates that Saxon scalas-up better than Galax when the size of the XML document increases. Figure 5 compares the data scalability factors for the two engines. Notice that Saxon's scalability is sub-linear up to XMark factor 0.256 (29.9 MB), and it is super-linear for bigger files. Galax's scalability is sub-linear up to XMark factor 0.032 (3.7 MB), and it is super-linear for bigger documents. This trend is confirmed by the behaviour of the benchmark response speeds (Figure 4). In particular, notice that Saxon's response speed increases (with a decreasing derivative) up to XMark factor 0.256, and then it decreases, while Galax has a positive acceleration up to XMark factor 0.032, and then the acceleration becomes negative. From this analysis, we

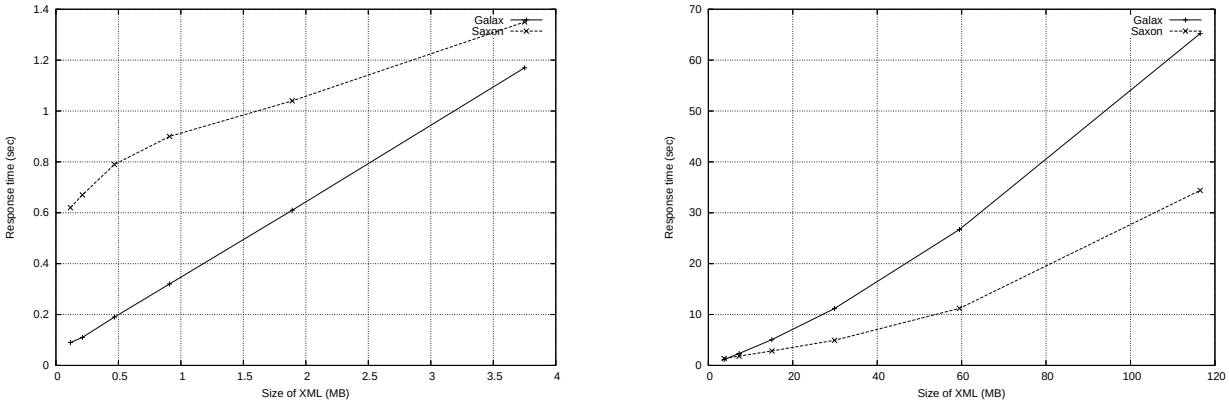

Figure 3: Average benchmark response times

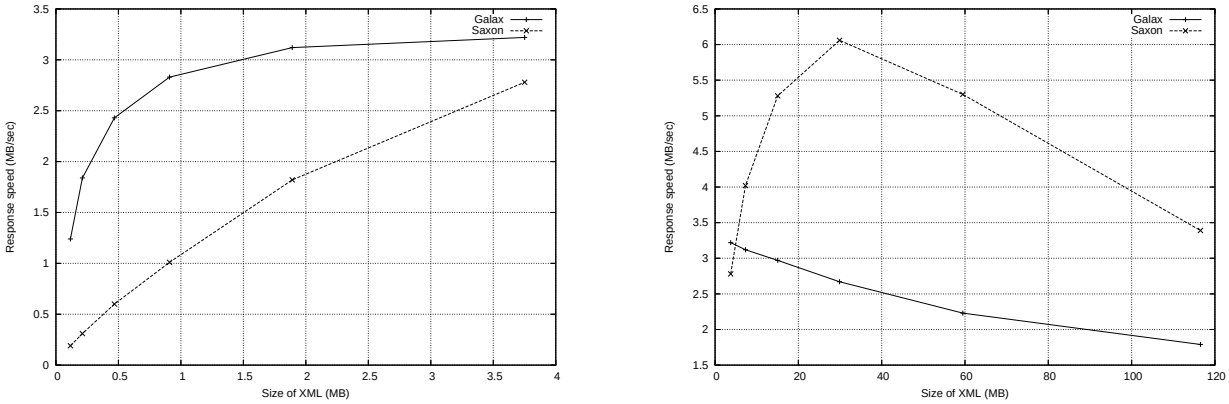

Figure 4: Benchmark response speeds

conclude that, under our testing environment, Saxon is well performing up to a *break point* corresponding to an XML documents of size 29.9 MB, while the break point for Galax corresponds to a smaller file of 3.7 MB.

Finally, Figures 6 and 7 depict, for each query, the average response speeds and the average data scalability factors over the document series. Interestingly, the *qualitative* behaviour of the response speeds is the same for both the engines, with Saxon outperforming Galax in all the queries but Q35 (The elements written in Italian language: `//*[lang('it')]`). This might indicate that the two engines implement a similar algorithm to evaluate XPath queries. The data scalability factor for Galax is almost constant for all the queries, and it is less but close to linear scalability. The data scalability factor for Saxon is less stable. It is far below linear scalability for all the queries but the problematic Q35. In particular, the scalability factor for Q35 is higher than 3 in the last segment of the document series, indicating that the response time for Q35 grows more then quadratically (probably Saxon doesn't understand Italian very well!). Notice that Q35 is not problematic in Galax.

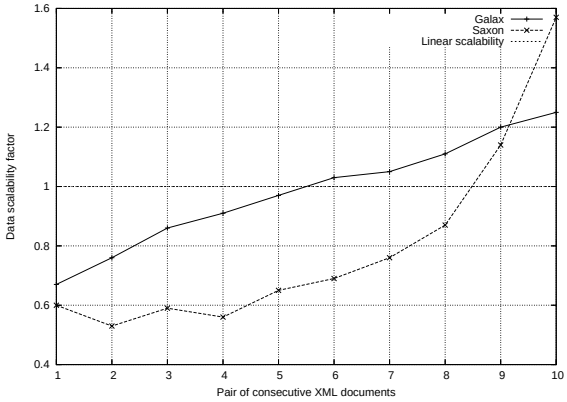

Figure 5: Data scalability factors for the benchmark

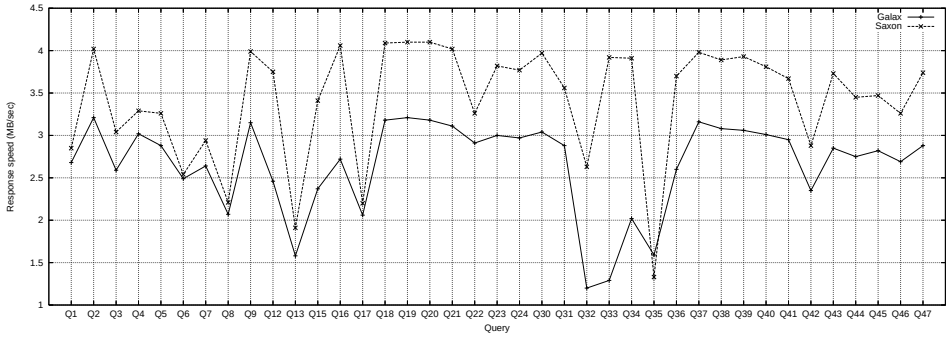

Figure 6: Average query response speeds

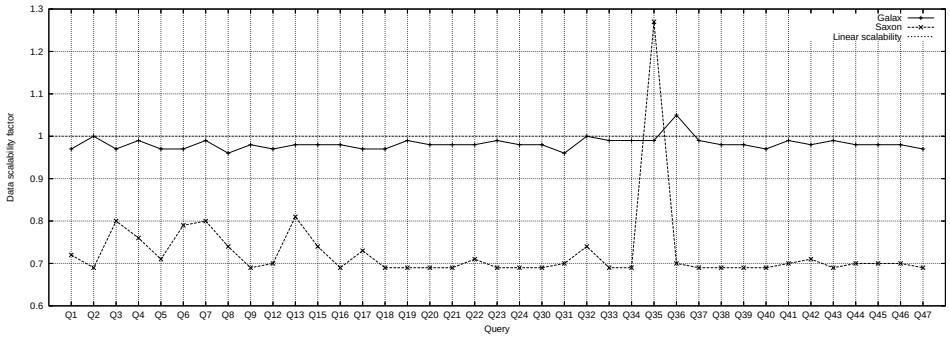

Figure 7: Average data scalability factors for queries

5 Future work

We have a lot of fresh ideas for the future:

1. To devise a larger set of *performance indexes* for the evaluation of XML engines.
2. To develop a benchmark to test *query scalability*. Query scalability is the ability of an XML engine to process queries of increasing lengths. Human-composed queries tend to be short, but computer-generated ones might be very long. This benchmark should include "crash-me queries", which are queries that seriously challenge current XML engines but that eventually might be solved efficiently.
3. To program a *benchmark checker* to automatically check the benchmark on a particular XML engine. The application should also compute the performance indexes and format the evaluation outcomes in different formats (plain text, XML, HTML, Gnuplot).
4. To develop a similar benchmark environment for XQuery.

References

[1] M. Fernández, J. Siméon, C. Chen, B. Choi, V. Gapeyev, A. Marian, P. Michiels, N. Onose, D. Petkanics, C. Ré, M. Stark, G. Sur, A. Vyas, and P. Wadler. Galax. The XQuery implementation for discriminating hackers. `http://www.galaxquery.org`, 2005.

[2] E. R. Harold and W. S. Means. *XML in a Nutshell*. O'Reilly, 3rd edition, 2004.

[3] M. H. Kay. Saxon. An XSLT and XQuery processor. `http://saxon.sourceforge.net`, 2005.

[4] M. Franceschet. XPathMark: An XPath benchmark for XMark. `http://www.science.uva.nl/~francesc/xpathmark`, 2005.

[5] A. R. Schmidt, F. Waas, M. L. Kersten, M. J. Carey, I. Manolescu, and R. Busse. XMark: A Benchmark for XML Data Management. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 974–985, 2002. `http://monetdb.cwi.nl/xml/`.

[6] World Wide Web Consortium. XML Path Language (XPath) Version 1.0. `http://www.w3.org/TR/xpath`, 1999.

[7] World Wide Web Consortium. XSL Transformations (XSLT). `http://www.w3.org/TR/xslt`, 1999.

[8] World Wide Web Consortium. XML Pointer Language (XPointer). `http://www.w3.org/TR/xptr`, 2002.

[9] World Wide Web Consortium. XQuery 1.0: An XML Query Language. `http://www.w3.org/TR/xquery`, 2005.

6 Appendix - The XMark DTD

```
<!ELEMENT site            (regions, categories, catgraph, people,
                           open_auctions, closed_auctions)>

<!ELEMENT categories      (category+)>
<!ELEMENT category        (name, description)>
<!ATTLIST category        id ID #REQUIRED>
<!ELEMENT name            (#PCDATA)>
<!ELEMENT description     (text | parlist)>
<!ELEMENT text            (#PCDATA | bold | keyword | emph)*>
<!ELEMENT bold            (#PCDATA | bold | keyword | emph)*>
<!ELEMENT keyword         (#PCDATA | bold | keyword | emph)*>
<!ELEMENT emph            (#PCDATA | bold | keyword | emph)*>
<!ELEMENT parlist         (listitem)*>
<!ELEMENT listitem        (text | parlist)*>

<!ELEMENT catgraph        (edge*)>
<!ELEMENT edge            EMPTY>
<!ATTLIST edge            from IDREF #REQUIRED to IDREF #REQUIRED>

<!ELEMENT regions         (africa, asia, australia, europe, namerica, samerica)>
<!ELEMENT africa          (item*)>
<!ELEMENT asia            (item*)>
<!ELEMENT australia       (item*)>
<!ELEMENT namerica        (item*)>
<!ELEMENT samerica        (item*)>
<!ELEMENT europe          (item*)>
```

```
<!ELEMENT item            (location, quantity, name, payment, description,
                           shipping, incategory+, mailbox)>
<!ATTLIST item            id ID #REQUIRED
                          featured CDATA #IMPLIED>
<!ELEMENT location        (#PCDATA)>
<!ELEMENT quantity        (#PCDATA)>
<!ELEMENT payment         (#PCDATA)>
<!ELEMENT shipping        (#PCDATA)>
<!ELEMENT reserve         (#PCDATA)>
<!ELEMENT incategory      EMPTY>
<!ATTLIST incategory      category IDREF #REQUIRED>
<!ELEMENT mailbox         (mail*)>
<!ELEMENT mail            (from, to, date, text)>
<!ELEMENT from            (#PCDATA)>
<!ELEMENT to              (#PCDATA)>
<!ELEMENT date            (#PCDATA)>
<!ELEMENT itemref         EMPTY>
<!ATTLIST itemref         item IDREF #REQUIRED>
<!ELEMENT personref       EMPTY>
<!ATTLIST personref       person IDREF #REQUIRED>

<!ELEMENT people          (person*)>
<!ELEMENT person          (name, emailaddress, phone?, address?, homepage?,
                           creditcard?, profile?, watches?)>
<!ATTLIST person          id ID #REQUIRED>
<!ELEMENT emailaddress    (#PCDATA)>
<!ELEMENT phone           (#PCDATA)>
<!ELEMENT address         (street, city, country, province?, zipcode)>
<!ELEMENT street          (#PCDATA)>
<!ELEMENT city            (#PCDATA)>
<!ELEMENT province        (#PCDATA)>
<!ELEMENT zipcode         (#PCDATA)>
<!ELEMENT country         (#PCDATA)>
<!ELEMENT homepage        (#PCDATA)>
<!ELEMENT creditcard      (#PCDATA)>
<!ELEMENT profile         (interest*, education?, gender?, business, age?)>
<!ATTLIST profile         income CDATA #IMPLIED>
<!ELEMENT interest        EMPTY>
<!ATTLIST interest        category IDREF #REQUIRED>
<!ELEMENT education       (#PCDATA)>
<!ELEMENT income          (#PCDATA)>
<!ELEMENT gender          (#PCDATA)>
<!ELEMENT business        (#PCDATA)>
<!ELEMENT age             (#PCDATA)>
<!ELEMENT watches         (watch*)>
<!ELEMENT watch           EMPTY>
<!ATTLIST watch           open_auction IDREF #REQUIRED>
```

```
<!ELEMENT open_auctions   (open_auction*)>
<!ELEMENT open_auction    (initial, reserve?, bidder*, current, privacy?, itemref,
                           seller, annotation, quantity, type, interval)>
<!ATTLIST open_auction    id ID #REQUIRED>
<!ELEMENT privacy         (#PCDATA)>
<!ELEMENT initial         (#PCDATA)>
<!ELEMENT bidder          (date, time, personref, increase)>
<!ELEMENT seller          EMPTY>
<!ATTLIST seller          person IDREF #REQUIRED>
<!ELEMENT current         (#PCDATA)>
<!ELEMENT increase        (#PCDATA)>
<!ELEMENT type            (#PCDATA)>
<!ELEMENT interval        (start, end)>
<!ELEMENT start           (#PCDATA)>
<!ELEMENT end             (#PCDATA)>
<!ELEMENT time            (#PCDATA)>
<!ELEMENT status          (#PCDATA)>
<!ELEMENT amount          (#PCDATA)>

<!ELEMENT closed_auctions (closed_auction*)>
<!ELEMENT closed_auction  (seller, buyer, itemref, price, date, quantity,
                           type, annotation?)>
<!ELEMENT buyer           EMPTY>
<!ATTLIST buyer           person IDREF #REQUIRED>
<!ELEMENT price           (#PCDATA)>
<!ELEMENT annotation      (author, description?, happiness)>

<!ELEMENT author          EMPTY>
<!ATTLIST author          person IDREF #REQUIRED>
<!ELEMENT happiness       (#PCDATA)>
```