MemBeR: A Micro-benchmark Repository for XQuery

Loredana Afanasiev[1], Ioana Manolescu[2], and Philippe Michiels[3]

[1] University of Amsterdam, The Netherlands, `lafanasi@science.uva.nl`
[2] INRIA Futurs & LRI, France, `ioana.manolescu@inria.fr`
[3] University of Antwerp, Belgium, `philippe.michiels@uia.ua.ac.be`

Abstract. XQuery is a feature-rich language with complex semantics. This makes it hard to come up with a benchmark suite which covers all performance-critical features of the language, and at the same time allows one to individually validate XQuery evaluation techniques. This paper presents MemBeR, a *micro-benchmark repository*, allowing the evaluation of an XQuery implementation with respect to precise evaluation techniques. We take the view that a fixed set of queries is probably insufficient to allow testing for various performance aspects, thus, the users of the repository must be able to add new data sets and/or queries for specific performance assessment tasks. We present our methodology for constructing the micro-benchmark repository, and illustrate with some sample micro-benchmarks.

1   Introduction

The development of XML query engines is currently being held back by a lack of systematic tools and methodology for evaluating algorithms and optimization techniques. The essential role of benchmark tools in the development of XML query engines, or any type of data management systems for that matter, is well established. Benchmarks allow one to assess a system's capabilities and help determine its strengths or potential bottlenecks.

Since the introduction of XML query languages like XPath 1.0 [6], XPath 2.0 [9] and XQuery [3], many benchmark suites have been developed. Most of them, including XMark [24], XMach-1 [4], X007 [5] and XBench [25], fall into the category of *application benchmarks*. Application benchmarks are used to evaluate the overall performance of a database system by testing as many query language features as possible, using only a limited set of queries. The XML data sets and the queries are typically chosen to reflect a particular user scenario. The influence of different system components on their overall performance is, however, difficult to analyze from performance figures collected for complex queries. The evaluation of such queries routinely combines a few dozen execution and optimization techniques, which in turn depend on numerous parameters. Nevertheless, due to a lack of better tools, application benchmarks have been used for tasks they are not suited for, such as the assessment of a particular XML join or optimization technique.

Micro-benchmarks, as opposed to application benchmarks, test individual performance-critical features of the language, allowing database researchers to evaluate their query evaluation technique (e.g. query optimization, storage and indexing schemes etc.) in isolation. Micro-benchmarks provide better insight in how XQuery implementations address specific performance problems. They allow developers to compare performance with and without the technique being tested, while reducing to a minimum the interaction with other techniques or implementation issues.

To the authors' knowledge, the Michigan benchmark [23] is the only existing micro-benchmark suite for XML. It proposes a large set of queries, allowing one to assess an engine's performance for a variety of elementary operations. These queries are used on a parametrically generated XML data set. However, this micro-benchmark suffers from some restrictions. For instance, the maximum document depth is fixed in advance to 16, and there are only two distinct element names. Furthermore, the query classes identified in [23] are closely connected to a particular evaluation strategy.[4] The queries of [23] are very comprehensive on some aspects, such as downward XPath navigation and ignore others, such as other XPath axes, or complex element creation.

The need for micro-benchmarks and associated methodology The first problem we are facing is the lack of micro-benchmarks allowing system designers and researchers to get *precise and comprehensive* evaluations of XML query processing systems and prototypes. An evaluation is *precise* if it allows one to study language features *in isolation*, to understand which parameters impact a system's behavior on that feature, without "noise" in the experimental results due to processing other features. The need for precision, which is a general aspect of scientific experiments, leads to the choice of micro-benchmarks, one for each feature to study. For an evaluation to be *comprehensive*, several conditions must be met. For every interesting feature, there must be a micro-benchmark allowing to test that feature. When studying a given feature, *all* parameters which may impact the system's behavior in the presence of that feature must be described, and *it must be possible to vary their values in a controlled way*. Finally, *all interesting aspects of the system's behavior during a given measure must be documented*. For instance, a performance micro-benchmark should be accompanied by information regarding the memory or disk consumption of that measure.

Second, a *micro-benchmark user methodology* is needed, explaining how to choose appropriate micro-benchmarks for a given evaluation, why the micro-benchmark parameters are likely to be important, and how to choose value combinations for these parameters. Such a methodology will bring many benefits. It will ease the task of assessing a new implementation technique. It will facilitate comprehension of system behavior, and reduce the time to write down and disseminate research results. And, it will ease the task of reviewers assessing "Experiments" sections of XML querying papers, and help unify the authors' standards with the reviewers' expectations.

Our approach: micro-benchmark repository and methodology We are currently building a repository of micro-benchmarks for XQuery and its fragments, which will provide for precise and comprehensive evaluation. We endow micro-benchmarks with precise guidelines, easing their usage and reducing the risks of mis-use.

---

[4] Evaluating path expressions by an n-way structural join, where n is the total number of path steps. The query classification criteria are no longer appropriate e.g., if a structural index is used.

Given the wide range of interesting XQuery features, and the presence of interesting ongoing developments (such as extensions for text search [7], and for XML updates [8]), a fixed set of micro-benchmarks devised today is unlikely to be sufficient forever. Thus, we intend to develop our repository as a *continuing, open-ended community effort*:

○ users can contribute to the repository by *creating*, or *enhancing* an existing micro-benchmark;
○ additions to the repository will be subject to *peer review* from the other contributors, checking if the feature targeted by the addition was not already covered and if the addition adheres to the micro-benchmark principles, and ensuring the quality of the benchmark's presentation.

Such a repository will allow consolidating the experience of many individual researchers having spent time and effort in "carving out" micro-queries from existing application benchmarks unfit for the task. It will be open to the addition of new performance challenges, coming from applications and architectures perhaps not yet available today. This way, the micro-benchmarks will be continuously improved, and, we hope, widely used in the XML data management community. The repository is hosted on the Web and freely accessible. This should further simplify the task of setting up experimental studies, given that individual micro-benchmarks will be available at specific URLs.

This paper describes our approach. Section 2 formalizes the concepts behind the micro-benchmarks repository, and Section 4 illustrates them with several examples. Section 3 outlines a preliminary micro-benchmark classification and delves into more details of performance-oriented micro-benchmarks. Section 5 describes the test documents in the repository. Section 6 briefly describes the repository's Web interface, and concludes with some perspectives.

2 Our approach: a micro-benchmark repository

Our goal is *to build a repository of micro-benchmarks for studying the performance, resource consumption, correctness and completeness of XQuery implementations techniques.*

Performance: how well does the system perform, e.g., in terms of completion time, or query throughput? The primary advantage of a data management system, when compared with an ad-hoc solution, should be its efficiency.

Resource consumption: performance should be naturally evaluated against the system's resource needs, such as the size of a disk-resident XML store, with or without associated indexes, or the maximum memory needs of a streaming system.

Completeness: are all relevant language features supported by the system? Some aspects of XQuery, such as its type system, or its functional character, have been perceived as complex. Correspondingly, many sub-dialects have been carved out [16, 20, 22]. Implementations aiming at completeness could use a yardstick to compare against.

Correctness: does the output of the system comply with the query language specifications? For a complex query language such as XQuery, and even its fragments, correctness is also a valid target of benchmarking.

In this paper we will mainly focus on *performance and resource consumption micro-benchmarks*. Nevertheless, we stress the importance of correctness for interpreting performance results. In devising correctness and completeness benchmarks, we expect to draw
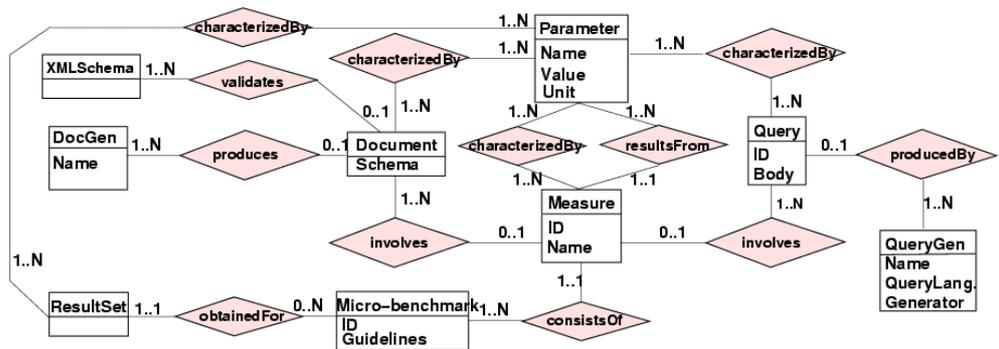
Fig. 1. Entity-Relationship diagram of the micro-benchmark repository contents.

inspiration from the use cases and examples used in the W3C XQuery specifications and from existing benchmarks, like [12].

We intend our benchmark repository mainly for system designers, to help them analyze and optimize their system.

Micro-benchmarking design principles We adopt the following design principles:

There should be a minimal number of micro-benchmarks for every language feature, and we will usually strive to keep this number to 1. However, if a new example provides a very different perspective, and the current ones cannot be extended to simulate it, then it will be included in the repository.

A micro-benchmark should explicitly list, and provide value ranges for all data, query and resource parameters which may impact the results. This is a crucial condition for benchmark results to be reproducible, interpretable, and trustworthy. In this way a micro-benchmark will contain well-documented and thorough measures.

The above implies that for any micro-benchmark measure, and any data parameter likely to impact the measure's result, *at least one data set can be constructed by controlling the value of that parameter in its interesting range.* This will have an impact on our choice of data sets (see Section 5).

A micro-benchmark should reduce to a minimum the influence of all but the tested language feature. Thus, if the purpose is to test path expressions navigating downward, the queries should not use sibling navigation, and vice versa. An important consequence is the following. The presence of an XML Schema for the input document enables a large number of optimizations, at the level of an XML store, XML index, XQuery rewriting and optimization, automata-based execution etc. *In any micro-benchmark measure where the focus is not on schema-driven optimizations, one should use documents without a schema.* Otherwise, schema-driven optimizations might effect the system's performance in a non-transparent manner and make results uninterpretable.

Measure (also) individual query processing steps. To get a precise evaluation, often is needed to measure individual processing steps, such as: query normalization, query rewriting, query optimization, data access, (structural) join processing, output construction etc. For instance, XPath micro-benchmarks may measure the time to *locate* the elements which must be returned (this often means finding their IDs). Measuring such processing steps requires

hooks into the execution engine. We consider it worth the trouble, as even if queries are chosen with care, query execution times may still reflect the impact of too many factors.

A micro-benchmark should be extensible. A micro-benchmark should aim to remain useful even when systems will achieve much higher performance. The parameters should therefore allow for a wide enough range. The micro-benchmarks should also be regularly updated to reflect new performance standards.

From these principles, we derive the following micro-benchmark repository structure (Fig. 1):

- *XML documents.* A document may have an XML Schema or not. It is characterized by a number of *parameters*, which we model as name-value pairs. Benchmarks typically benefit from using collections of documents similar in some aspects, but characterized by different parameters. For synthetic data sets, a *document generator* is also provided.
- *Queries.* Each query aims at testing exactly one feature. Queries can also be characterized by *parameters*, such as: number of steps in a path expression query, numbers of query nesting levels, selectivity of a value selection predicate etc. Similar queries make up a *query set*, for which a *query generator* is provided.
- *Measures*: a measure is one individual experiment, from which experimental results is gathered. We model an experimental result also as a parameter, having a name and a value. A measure may involve a document, or none; XML fragments are legal XQuery expressions, and thus an XML query may carry "its own data" (see micro-benchmark μB_4 in Section 4). A measure may involve zero, one, or more queries; the latter case is reserved to multi-query scenarios, such as, for instance, XML publish/subscribe. Intuitively, one measure yields "one point on a curve in a graph". We will provide examples shortly.
- *Micro-benchmarks.* A micro-benchmark is a collection of measures, studying a given (performance, consumption, correctness or completeness) aspect of a XML data management and querying. Intuitively, a micro-benchmark yields a set of points, which can be presented as "one or several curves or graphs", depending on how many parameters vary, in the documents and queries considered. A micro-benchmark includes *guidelines*, explaining which data and/or query parameters may impact the results and why, and suggesting ranges for these parameters. Measure methodologies may also specify the scenario(s) for which the measure is proposed, including (but not limited to): persistent database scenario, streaming query evaluation, publish/subscribe etc.
- *Micro-benchmark result sets*, contributed by users. A result set consists of a set of points corresponding to each of the micro-benchmark's prescribed measures, and of a set of parameters characterizing the measure enactment. Commonly used parameters describe hardware and software configurations. Other important parameters are the technique(s) and optimization(s) employed. For instance, a result set may specify the particular XML index used, or the fact that the query automaton was lazily constructed etc.

Micro-benchmark usage methodology Even a carefully designed (micro-)benchmark can be misused. As an attempt to limit this, we require micro-benchmark results to adhere to the following usage principles:

Always declare the language and/or dialect supported by the system, even for features not used by the micro-benchmark. Many efficient evaluation techniques are conditioned by some underlying language simplifications, such as: unordered semantics, simplified atomic types set, unsupported navigation axes, unsupported typing mechanism etc. Such simplifications,

if any, should be clearly stated next to performance figures. In relational query processing research, the precise SQL or Datalog dialect considered is always clearly stated. XQuery not being simpler than SQL, at least the same level of precision is needed.

Micro-benchmark results which vary less parameters than specified by the micro-benchmark are only meaningful if they are accompanied by a short explanation as to why the simplifications are justified. Omitting this explanation ruins the effort spent in identifying useful parameters, and compromises the comprehensive aspect of the evaluation.

Extra values for a parameter may always be used in results. Range changes or restrictions should be justified. In some special situations, new parameter values may be needed. In such cases, a revision of the micro-benchmark should be considered.

When presenting micro-benchmark results, parameters should vary one at a time, while keeping the other parameters constant. This will typically yield a family of curves where the varying parameter values are on the x axis, and the measure result on the y axis. For space reasons, some curves may be omitted from the presentation. In this case, however, *the measure points for all end-of-range parameter values should be provided.* Trying the measure with these values may give the system designer early feedback, by exposing possible system shortcomings. And, when performance is robust on such values, a convincing case has been made for the system's performance.

3   Preliminary taxonomy of measures and micro-benchmarks

In this section, we outline a general classification of measures (and thus, of micro-benchmarks). This classification guides a user looking for a specific micro-benchmark, and serves as a road map for our ongoing micro-benchmark design work.

A first micro-benchmark classification criterion, introduced in Section 2, distinguishes between *performance*, *consumption*, *correctness* and *completeness* benchmarks.

We furthermore classify micro-benchmarks according to the following other criteria:

○ The result metric: it may be *execution time*, *query normalization or optimization time*, *query throughput*, *memory occupancy*, *disk occupancy* etc. It may also be a simple boolean value, in the case of correctness measures.

○ Benchmarks may test *data scalability* (fixed query on increasingly larger documents) or *query scalability* (increasing-size queries on fixed documents).

○ Whether or not a micro-benchmark uses an *XMLSchema*, and the particular schema used.

○ The *query processing scenarios* to which a micro-benchmark applies, such as: persistent database (store the document once, query it many times), streaming (process the query in a single pass over the document), or programming language-based (the document is manipulated as an object in some programming language).

○ The *query language* and perhaps dialect which must be supported in order to run the micro-benchmark.

○ The *language feature being tested* in a micro-benchmark is a precise classification criteria. We strive to provide exactly one micro-benchmark for each interesting feature.

The next section contains several micro-benchmark examples together with their classification and methodology.

4 Examples of micro-benchmarks for XPath and XQuery

We start by a very simple micro-benchmark, involving a basic XPath operation.

Example 1 (Micro-benchmark μB_1: simple node location). In a persistent XML database scenario, we are interested in the time needed to locate elements of a given tag in a stored document. We study this time on increasingly large and complex documents (data scalability measure). We are not concerned with schema optimizations, thus, we will use schema-less documents. We measure the time to *locate* the elements, not to *return* their subtrees.

Let Q_1 be the query //a_1. Let n, t, d and f be some positive integers, and p be a real number between 0 and 1. Let $D_1(n, t, d, f, p)$ be a document whose n elements are labeled a1, a2, ..., at, having the depth d, the fan-out (maximum number of children of an element) f, and such that exactly $p * n$ elements are labeled a1. Elements may nest freely, that is, the parent of an element labeled ai can have any aj label, $1 \leq i, j \leq t$.

The measure $M_1(h, b)$ involves Q_1 and a document $D_1(n, t, d, f, p)$, for some $n, t, d, f \in \mathbb{N}$ and $p \in [0, 1]$. The parameter h may take values in $\{true, false\}$ and specifies whether the measure is taken with a hot cache ($h = true$) or with a cold cache ($h = false$). The parameter b is the size of the memory buffer in KB. For any pair of (h, b) values, $M_1(h, b)$ results in an execution time for locating the persistent identifiers of the elements in Q_1's result, in document $D_1(n, t, d, f, p)$. A M_1 measure is characterized by a (h, b, n, t, d, f, p) tuple.

The micro-benchmark μB_1 consists of applying M_1 on a subset of $\{true, false\} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times [0, 1]$, such as, for instance:

$h \in \{true, false\}$	$b \in \{50, 200, 1000\}$	$n \in \{10^6, 10^9, 10^{12}\}$	$d \in \{10, 20, 50\}$
$f \in \{2, 10, 20\}$	$t \in \{1, 5, 20\}$	$p \in \{0.001, 0.01, 0.1, 0.5, 0.8, 1.0\}$	

Parameters n and p impact the number of retrieved elements. Parameter t determines the document's structural complexity, which may impact performance if the evaluation relies on a path or structural index. Together, d and f determine whether the document is deep or shallow; this may impact performance for strategies based on navigation or structural pattern matching. Such strategies are also affected by n, even when $n * p$ is constant. Varying h and b allows to capture the impact of the system's memory cache and memory buffer size on the evaluation times.

μB_1 results in several thousand points, all of which are not needed in all circumstances. If the system implements M_1 by a lookup in a tag index as in [17], d, f and t may make little difference, thus μB_1 may be reduced to only 60 individual measures. As another example, if M_1 is implemented by CAML-based pattern matching as in [2], then all evaluation takes place in memory and b is irrelevant.

Example 2 (Micro-benchmark μB_2: simple node location in the presence of a schema). Important performance gains can be realized on a query like Q_1 if schema information is available. μB_2 aims at quantifying such gains. Let $D_{2,l}(n, t, f, d, p, \Sigma_l)$ be a document satisfying the same conditions as $D_1(n, t, f, d, p)$ from μB_1, but furthermore valid with respect to an XML Schema Σ_l, specifying that a1 elements can only occur as children of elements labeled a1, a2, ... al. Σ_l prescribes a maximum number of f children for any element, and does not constrain the structure of $D_{2,l}$ in any other way.

Measure $M_2(n, t, f, d, p, l)$ records the running time of query Q_1 on $D_{2,l}(n, t, f, d, p, \Sigma_l)$. We choose not to include h and b here, since μB_2 is only concerned with the impact of the schema, orthogonal to cache and buffer concerns.

The micro-benchmark μB_2 consists of running M_2 for some fixed n, t, f, d and p, and for all $l = 1, 2, \ldots, t$. A set of suggested values is: $n = 10^6$, $t = 15$, $f = 10$, $d = 10$, $p = 1/t$.

An efficient system would use Σ_l to narrow the search scope for a1 elements. This can take many forms. A fragmented storage, or a structural index, may make a distinction between elements a1, ..., al and the others, making it easier to locate a1s. A streaming processor may skip over subtrees rooted in a$(l + 1)$, ..., at elements etc.

Example 3 (Micro-benchmark μB_3: returning subtrees, no schema). This micro-benchmark is meant for the persistent database scenario. It captures the performance of sending to the output sub-trees from the original document. This operation, also known as serialization, or reconstruction, is challenging in many respects: systems whose storage is fragmented (e.g. over several relational tables) will have to make an effort to reconstruct the input; systems using a persistent tree will have to follow disk-based pointers, thus they depend on the quality of node clustering etc.

We aim at measuring the cost of reconstruction alone, not the cumulated cost of locating some nodes and then reconstructing them. Thus, we choose the query Q_2: /*, and will apply it on the root of some input document.

Document depth, fan-out, and size, all impact reconstruction performance. Furthermore, document leaves (in a schema-less context, interpreted as strings) also may have an impact. Some storage models separate all strings from their parents, others inline them, others inline only short strings and separate longer ones etc.

We vary string size according to a normal distribution $ssd(sa, sv)$, of average sa and variance sv. We vary the number of text children of a node according to another (normal) distribution $tcd(ca, cv)$ of average ca and variance cv. Let $D_3(n, t, d, f, tcd, ssd)$ be an XML document having n elements labeled a1, a2, ... at, of depth d and fanout f, such that the number of text children of given element is obtained from tcd, and the number of characters in each individual text child is given by ssd. As in D_1, elements nest arbitrarily. The actual string content does not matter, and is made of randomly chosen characters.

Measure $M_3(n, t, d, f, sa, sv, ca, cv)$ consists of measuring the execution time of Q_2 on $D_3(n, t, d, f, tcd(ca, cv), ssd(sa, sv))$. Micro-benchmark μB_3 consists of M_3 measures for:

$$n \in \{10^3, 10^6, 10^9\} \quad t \in \{2, 20\} \quad d \in \{5, 20, 50\} \quad f \in \{5, 10\}$$
$$(ca, cv) \in \{(2, 1), (5, 2), (10, 5)\} \quad (sa, sv) \in \{(5, 2), (100, 50), (1000, 500)\}$$

Example 4 (Micro-benchmark μB_4: XPath duplicate elimination). This micro-benchmark is taken from [15]. The purpose is to assess the processor's performance in the presence of potential duplicates. Let Q_3^i be the query:

`<a><b/><b/></a>/b/parent::a/b/parent::a ... /b/parent::a`

where the sequence of steps `/b/parent::a` is repeated i times. Any Q_3^i returns exactly one a element. An evaluation following directly the XPath [9] specification requires eliminating duplicates after every path step, which may hurt performance. Alternatively, duplicates may be eliminated only once, after evaluating all path expression steps. However, in the case of Q^i, this entails building intermediary results of increasing size: 2 after evaluating /b/parent::a,
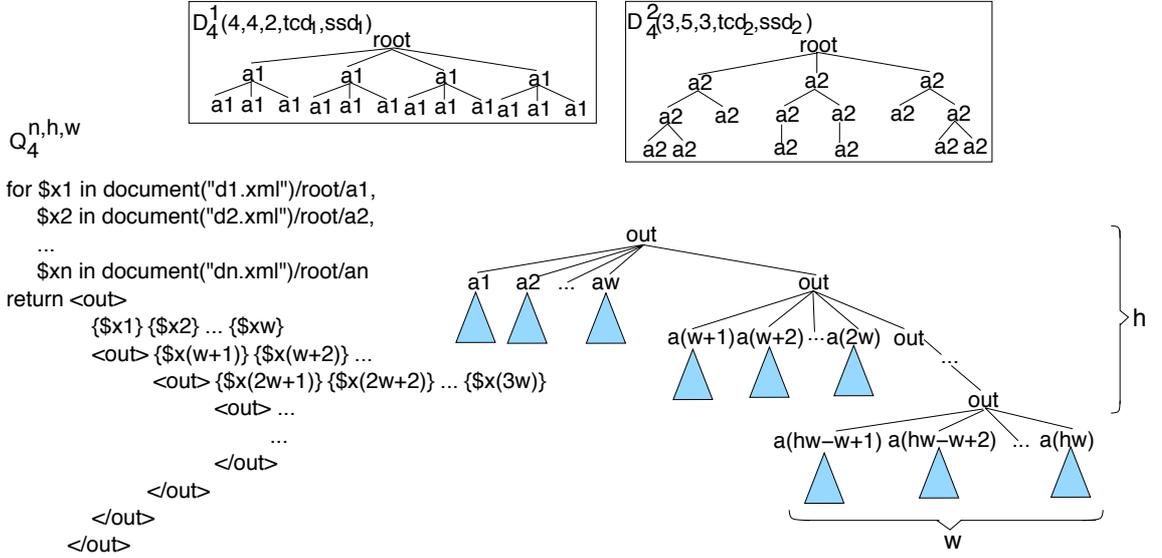
Fig. 2. Documents and queries involved in the micro-benchmark μB_5.

2^2 after evaluating `/b/parent::a/b/parent::a` etc., up to 2^i before the final duplicate elimination. Large intermediary results may eat up available memory and hurt performance.

$M_4(i)$ measures the running time of Q_3^i; it does not need a document. The micro-benchmark μB_4 consists of the measures $M_4(i)$, for $i \in \{1, 5, 10, 20\}$.

Example 5 (Micro-benchmark μB_5: element creation). This micro-benchmark targets the performance of new element construction, an important feature in XQuery.

The size, depth, and fanout of the copied input subtrees, as well as the text nodes therein, clearly impact performance. Another important factor is the complexity of the constructed XML output. Thus, we consider a set of documents $D_4^i(fr_i, n_i, d_i, tcd_i, ssd_i)$, as follows:
○ D_4^i's root element is labeled `root`. All other elements in D_4^i are labeled `a`i.
○ The root of any D_4^i document has exactly fr_i children elements.
○ Any sub-tree rooted in an `a`i child of the root is a tree of depth d_i consisting of n_i elements.
○ Root elements do not have text children. The number, and size, of text children of `a`i elements is dictated by the text child distribution tcd_i and the text length distribution ssd_i. These are normal distributions, as in the micro-benchmark μB_3 previously described.

For instance, $D_4^1(4, 4, 2, tcd_1, ssd_1)$ and $D_4^2(3, 5, 3, tcd_2, ssd_2)$ are outlined in the upper part of Fig. 2, for some unspecified distributions. Text nodes are omitted for readability.

To vary the complexity of result construction, we consider a family of queries $Q_4^{n,h,w}$, where: n is the number of input documents from which subtrees are extracted, h is the "stacking height" of such sub-trees in the output, and w is the "stitching width" of the subtrees in the output, where $h * w$ must be at most n. This is better understood by looking at $Q_4^{n,h,w}$, pictured in the lower part of Fig. 2, together with the outline of an element it produces. The shaded triangular shapes represent (deep copies of) the subtrees rooted in the `a`i input elements. This query uses very simple navigation paths in the `for` clause, in order to

$$Q_4^{3,1,3} \qquad\qquad\qquad Q_4^{3,3,1}$$

```
                                    for $x1 in document(''d1.xml'')/root/a1
  for $x1 in document(''d1.xml'')/root/a1      $x2 in document(''d2.xml'')/root/a2
     $x2 in document(''d2.xml'')/root/a2        $x3 in document(''d3.xml'')/root/a3
     $x3 in document(''d3.xml'')/root/a3  return <out>
  return <out>                                  {$x1} <out>
        {$x1} {$x2} {$x3}                             {$x2} <out> {$x3} </out>
        </out>                                        </out>
                                            </out>
```

Fig. 3. Sample queries from the micro-benchmark μB_5.

minimize the noise introduced in the measure by the effort spent in locating these subtrees. Two sample instances of $Q_4^{n,h,w}$ are shown in Fig. 3.

Measure M_5 records the execution time of query $Q_4^{n,h,w}$ for some n, h and w values, on n input documents D_4^1, D_4^2, ..., D_4^n. M_5 is characterized by numerous input parameters, including those describing each D_4^i.

Choosing the recommended parameters ranges for μB_5 without blowing up the number of measures is quite delicate. Without loss of generality, we restrict ourselves to the distributions:

- tcd_{low} with average 2 and variance 1, ssd_{low} with average 10 and variance 5; these correspond to a mostly data-centric document.
- tcd_{high} with average 10 and variance 2, ssd_{high} with average 1000 and variance 200; these correspond to a text-rich document.

To measure scale up with w, the following set of parameter ranges are recommended:

$n \in \{1, 2, 5, 10, 20\} \quad w = n \quad h = 1$
$fr_1 = 10^5$, choose a combination of fr_2, ..., fr_n values such that $\Pi_{i=1}^n fr_i = 10^8$
For any n and $i \in \{1, \ldots, n\}$, $(tcd_i, ssd_i) = (tcd_{high}, ssd_{high})$, $d_i = 5$, and $n_i = 100$

In the above, the values of related parameters, such as fr_i, are part of the micro-benchmark specification. To measure scale up with h, set $w = 1$, $h = n$, and proceed as above.

To measure scale up with the output size, set $n = 12$, $w = 4$, $h = 3$, set all distributions, first, to (tcd_{high}, ssd_{high}) and second, to (tcd_{low}, ssd_{low}), choose some fr_i such that $\Pi_{i=1}^n fr_i = 10^6$, and let each n_i take values in $\{1, 10, 25\}$. An output tree will thus contain between $12 + 3 = 15$ and $12 * 25 + 3 = 303$ elements.

All these five micro-benchmarks are performance-oriented. μB_1 and μB_2 measure node location time, while μB_3, μB_4 and μB_5 measure query execution time. μB_1, μB_2, μB_3 and μB_4 require downward XPath, while μB_4 requires XQuery node creation. Only μB_3 uses a schema. μB_1, μB_2 and μB_3 test data scalability; μB_4 tests query scalability.

In Table 1 and Table 2 we outline some other interesting XPath and XQuery performance-oriented micro-benchmarks; the list is clearly not exhaustive. We mark by *(d)* and *(q)* micro-benchmarks where data scalability (respectively query scalability) should be tested.

Other XQuery micro-benchmarks should target feature such as: explicit sorting; recursive functions; atomic value type conversions implied by comparison predicates; explicit type conversion to and from complex types; repeated sub-expressions etc. Two ongoing XQuery extension directions will require specific micro-benchmarks: *full-text search* [7], and *updates* [8].

μB_5 *(dq)* Simple linear path expressions of the form `/a1/a2/.../ak`.

μB_6 *(dq)* Path expressions of the form `//a1//a2//.../ak`.

μB_7 *(dq)* Path expressions of the form `/a1/*/*/.../a2`, where the number of `*` varies.

μB_8 *(d)* For each XPath axis [9], one path expression including a step on that axis . Interestingly, some node labeling schemes such as ORDPATH [21] allow "navigating" along several axes, such as parent, but also child, preceding-sibling etc. directly on the element label.

μB_9 *(dq)* Path expressions with a selection predicate, of the form `/a1/a2/.../ak[text()=`c`]`, where c is some constant.

μB_{10} *(dq)* Path expressions with inequality comparisons.

μB_{11} *(d)* Path expressions with positional predicates of the form `/a1[`n`]`, where $n \in \mathbb{N}$.

μB_{12} *(d)* Path expressions with positional predicates, such as `/a1[position()=last()]`.

μB_{13} *(dq)* Path expressions with increasingly many branches, of the form
`//a1[`$p1$`]//.../ak[`pk`]`, where each pi is a simple path expression.

μB_{14} *(dq)* Path expressions involving several positional predicates, of the form
`//a1[`$n1$`]//.../ak[`nk`]`, where each $ni \in \mathbb{N}$.

μB_{15} *(d)* Aggregates such as `count`, `sum` etc. over the result of path expressions.

Table 1. XPath performance micro-benchmarks (not an exhaustive list)

μB_{16} *(dq)*   The time needed to locate the elements to which are bound the `for` variables of a query. These variables can be seen as organized in a tree pattern of varying width and depth. For instance, width k and depth 2 yield the simple query Q_5^k (Fig. 4). This task is different from similar XPath queries, such as `//a1[//a2]...[//ak]`, or `//a1[//a2]...[//a(`$i-$`1)][//a(`$i+1$`)]...[//ak]//a`i, since unlike these XPath queries, all tuples of bindings for Q_5^k variables must be retained in the result.

μB_{17} *(dq)*   measures the time to locate the roots of the sub-trees to be copied in the output of query Q_6^k, shown in Fig. 4. Q_6^k will return some `a1` elements lacking some `a`i, while Q_5^k discards them. Thus, an evaluation technique using frequency estimations of `a`i elements to reduce intermediary results will improve performance for Q_5^k, and it would not affect Q_6^k.

μB_{18} *(d)*   measures the time needed to: locate the elements corresponding to `$x1` and `$x2` in the query Q_7^k, shown in Fig. 4, as well as its total evaluation time. In the query, θ stands for a comparison operator such as $=, <, \leq$ etc. When θ is $=$, separate micro-benchmarks should address: (*i*) the schema-less case, with the exact query above; then, replacing the condition with `$x/@y1=$x2/@y2`, (*ii*) the schema-less case, and (*iii*) the case when an XML schema specifies that the `y1` and `y2` attributes are in a key-foreign key relationship.

The time to locate the elements will show whether an efficient technique such as an index-based join is used to retrieve e.g. only those `$x2` elements with matching `$x1` elements. Measuring the total query evaluation time, especially for non-equality joins, exposes the join's performance, and how it interacts with serialization: If a `$x1` sub-tree must appear n times in the result, is it fully copied n times, or are some operations factorized ? The results of the micro-benchmarks based on Q_7^k should also be interpreted in conjunction with the results of μB_5 described in the previous section.

Table 2. XQuery performance micro-benchmarks (not an exhaustive list)

```
          for $x1 in document(''doc.xml'')//a1
              $x2 in $x1//a2                         for $x in document(''doc.xml'')//a1
  Q₅ᵏ         ...                            Q₆ᵏ    return <res>
              $xk in $x1//ak                               {$x//a1} {$x//a2} ... {$x//ak}
          return {$x1, $x2, ... $xk}                </res>
```

```
          for $x1 in document(''doc.xml'')//a1
              $x2 in document(''doc.xml'')//a2
  Q₇ᵏ     where $x1/text() θ $x2/text()
          return {$x1, $x2}
```

Fig. 4. Queries involved in the micro-benchmark μB_{16}, μB_{17} and μB_{18}.

5 Data sets for the micro-benchmark repository

Precise performance evaluation requires carefully characterizing the test documents. Some document characteristics were already considered in [23, 24]:

○ **Document size** is the most obvious parameter for data scalability measures. It is also one of the most mis-used. For instance, some studies use queries addressing the `category` hierarchy of a "500 Mb XMark document". If the fact that the category hierarchy makes up at most 3% of XMark documents is omitted, document size is probably misleading.
○ **Document tree depth** has an impact on document size, and determines the maximum length of downward path queries with non-empty results.
○ **Fan-out** is the maximum number of children of a node. It has an impact on document size; it may also impact some storage strategies, and thus query performance.

Size, depth and fan-out, however, are insufficient to account for all interesting characteristics of a document. We identify the following important data set characteristics:

Presence of schemas and constraints A DTD or XML Schema may be exploited for optimization purposes. More generally, one could envision XML query processors taking advantage of other classes of constraints, such as different type systems [2], or a-posteriori schemas extracted from schema-less data sets [13, 14]. A performance-oriented benchmark should state which constraints are used, if any. Proper type handling is one aspect of correctness.

Text-centric vs. data-centric Different XML data sets exhibit different ratios between the complexity of the XML structure tree, and the weight of the leaves (text). Both extremes are useful in different applications, and real-life data sets are in-between; the tree-to-text ratio may impact query performance. **Mixed contents** elements, frequent in text-centric documents, may raise correctness issues, as some systems do not support them.

Atomic value types The XQuery data model provides a rich set of atomic value types. However, most existing value-based XML indexing techniques proposed so far ignore these types and XQuery's many value coercions [10, 11]; meshing a value index with coercion-based semantics is quite complex [18].

Frequency of recursion Data recursion is an interesting feature, encountered in real-life XML documents [19], and affecting many evaluation techniques. Thus, precise evaluations must specify whether recursive elements were absent, rare, or frequent in the input.

Tag distribution In some documents, each tag may appear on only one path; in some others, numerous paths may lead to elements having the same name, maybe due to recursion, maybe not. For instance, in some (but not all) systems, the XMark-inspired query `//item//keyword`

would be evaluated by a structural join of all `item` and all `keyword` element IDs, even though many keywords do not appear under items.

Values The actual values found in the document's text nodes must also be described for measures using queries with conditions on values. Value distribution controlled in synthetic data sets [23]. Value domain, size distribution, and contents also impact query evaluation.

Partitioning of data in documents The XML and XQuery specifications give an important place to the notion of document, which can be seen as a physical segment of an XML data set. XQueries may combine information from several documents. Thus, it is important to understand how processors cope with input being fragmented over several documents.

Coming up with one unified data set, even a parametric one, on which all the above aspects can be varied at will, is hardly feasible. Notice that some parameters are inter-related and thus cannot be independently controlled, such as size, depth, and fan-out. We thus include in the benchmark repository two broad classes of synthetic documents. Documents in the first class are on purpose schema-less, and allow full control over the above mentioned parameters. Documents in the second class are schema-driven; we rely on ToXGene [1] to generate those documents. We briefly describe each class of documents next.

Schema-less parametric data set. This data set is produced by a data generator we implemented. It allows controlling: the maximum node fanout, maximum depth, total tree size (number of elements), document size (disk occupancy), the number of distinct element names in the document, and the distribution of tags inside the document. Required parameters are: either tree size or document size; and, either depth or fan-out. The number of distinct element names is 1 by default; elements are named `a1`, `a2` etc.

The distribution of tags within elements can be controlled in two ways. *Global* control allows tuning the overall frequency of element named `a1`, `a2`, ..., `a`n. Labels may nest arbitrarily. Uniform and normal distributions are available. *Per-tag* control allows specifying, for every element name `ai`, the minimum and maximum level at which `ai` can appear may be set; furthermore, the relative frequency of `ai` elements at that level can be specified as a number between 0.0 and 1.0[5]. Global distributions allow generating trees where any `ai` may appear at any level. Close to this situation, for instance, is the Treebank data set[6], corresponding to annotated natural language; tags represent parts of speech and can nest quite freely. Per-tag distributions produce more strictly structured documents, whereas e.g., some names only appear at level 3, such as `article` and `inproceedings` in the DBLP data set[7], other elements appear only below level 7, such as `keyword`s in XMark etc.

Fan-out, depth and tag distribution impact: the disk occupancy of many XML storage and structural indexing schemes; the complexity and precision of XML statistical synopses; the size of in-memory structures needed by an XML stream processor; and, the performance of path expression evaluation for many evaluation strategies. Thus, we will rely on this data set, and devise measures varying *all* these parameters, for assessing such aspects.

The number and size of text values follow uniform or normal distributions, as illustrated in μB_3 in Section 4. Values can be either filled with random characters, or taken from the Wikipedia text corpus (72 Mb of natural language text, in several languages). The latter is essential in order to run full-text queries; neither XMark nor MBench consider this issue.

---

[5] The generator checks the frequencies of several `ai`s at a given level for consistency.

[6] Available at `http://www.cs.washington.edu/research/xml/datasets`.
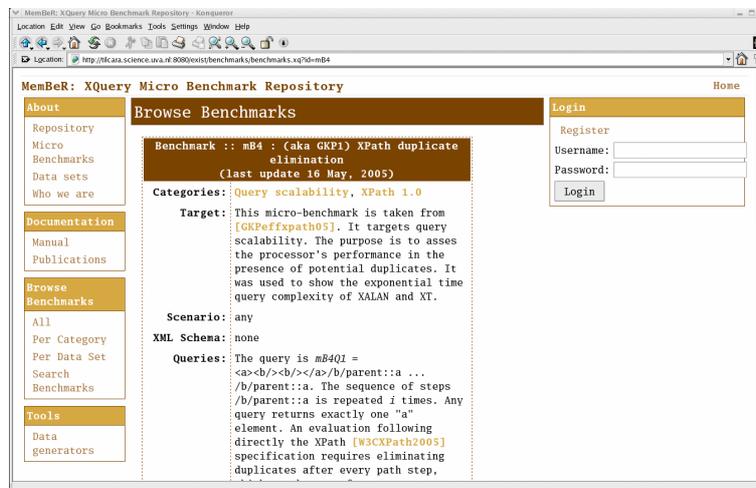
[7] Available at `http://dblp.uni-trier.de/xml`.

Fig. 5. Sample snapshot of Member's Web interface.

Schema-derived data sets The ToXGene [1] XML generator produces XML documents conforming to a type description expressed in a subset of XML Schema. Furthermore, ToX-Gene provides hooks for controlling: the frequency of a given element type, the simple values found in leaf nodes, the sharing of values in several nodes (thus, the size of various join results), the specific values to fill in specific places in the document etc. Thus, we adopt ToXGene as a useful tool for controlled generation of schema-endowed documents.

6 Conclusion and perspectives

We have described a micro-benchmark repository approach for evaluating XQuery processing techniques. Micro-benchmarks provide the proper tools for systematically evaluating approaches to XML query evaluation. We have described the design principles underlying our repository and we have given several examples of micro-benchmarks and their applications.

We have started implementing our repository over a Web interface, available at `http://ilps.science.uva.nl/Resources/MemBeR/`. For now, it contains some micro-benchmarks, with their categorization criteria, targets, scenarios etc. A sample snapshot corresponding to the page of micro-benchmark μB_4 is depicted in Figure 5.

We are currently adding to the repository several other micro-benchmarks, including those mentioned in Section 3. We plan to extract candidate features from the XQuery specification itself, and also from XML query processing articles recently published in major database conferences. These may also yield inspiration for micro-benchmark queries, although our methodology is more rigorous. In parallel, we are inviting researchers and system designers to send us suggestions of features, measures, or micro-benchmarks they deem interesting, or even better, contribute some benchmarks and become a micro-benchmark reviewers.

The need for precise and meaningful assessment of XML query processors is becoming stringent for research to advance, and for communicating results. Micro-benchmarks as a step forward; we started MemBeR to develop comprehensive, well-documented ones.

References

1. D. Barbosa, A. Mendelzon, J. Keenleyside, and K. Lyons. ToXgene: a template-based data generator for XML. In *WebDB*, 2002.
2. V. Benzaken, G. Castagna, and C. Miachon. A full pattern-based paradigm for XML query processing. In *PADL*, pages 235–252, 2005.
3. S. Boag, D. Chamberlin, M. Fernández, D. Florescu, J. Robie, and J. Siméon. XQuery 1.0 An XML Query Language, W3C Working Draft, April 2005. `http://www.w3.org/TR/xquery`.
4. Timo Böhme and Erhard Rahm. Xmach-1: A benchmark for XML data management. In *Proceedings of BTW2001, Oldenburg, 7.-9. Mrz, Springer, Berlin*, March 2001.
5. S. Bressan, G. Dobbie, Z. Lacroix, M. Lee, Y. Li, U. Nambiar, and B. Wadhwa. X007: Applying 007 benchmark to XML query processing tool. In *CIKM*, pages 167–174. ACM, 2001.
6. World Wide Web Consortium. XML path language (XPath) version 1.0 – W3C Recommendation, 2000. `http://www.w3.org/TR/xpath.html`.
7. World Wide Web Consortium. XQuery 1.0 and XPath 2.0 Full-Text – W3C Working Draft, July 2004. `http://www.w3.org/TR/xquery-full-text/`.
8. World Wide Web Consortium. W3C XQuery Update Requirements – W3C Working Draft, 2005. `http://www.w3.org/TR/xquery-update-requirements/`.
9. World Wide Web Consortium. XML path language (XPath) version 2.0 – W3C Working Draft, 2005. `http://www.w3.org/TR/xpath20/`.
10. World Wide Web Consortium. XQuery 1.0 and XPath 2.0 Formal Semantics – W3C Working Drafts, 2005. `http://www.w3.org/TR/xquery-semantics/`.
11. World Wide Web Consortium. XQuery 1.0 and XPath 2.0 Functions and Operators, 2005. `http://www.w3.org/TR/xpath-functions/`.
12. M. Francescet. XPathMark: an XPath benchmark for the XMark Generated Data. In *XSym*, 2005.
13. M. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim. XTRACT: Learning document type descriptors from XML document collections. *Data Min. Knowl. Discov.*, 1(7):23–56, 2003.
14. R. Goldman and J. Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In *VLDB*, pages 436–445, 1997.
15. G. Gottlob, C. Koch, and R. Pichler. The complexity of XPath query evaluation. In *PODS*, pages 179–190, 2003.
16. J. Hidders, J. Paredaens, R. Vercammen, and S. Demeyer. A light but formal introduction to XQuery. In *XSym*, pages 5–20, 2004.
17. H .V. Jagadish, S. Al-Khalifa, A. Chapman, L. V.S. Lakshmanan, A. Nierman, S. Paparizos, J. Patel, D. Srivastava, N. Wiwatwattana, Y. Wu, , and C. Yu. Timber: a native XML database. *VLDB Journal*, 11(4), 2002.
18. J. McHugh, J. Widom, S. Abiteboul, Q. Luo, and A. Rajaraman. Query optimization for semistructured data, 1998. Tech. report.
19. L. Mignet, D. Barbosa, and P. Veltri. The XML web: A first study. In *WWW Conference*, 2003.
20. G. Miklau and D. Suciu. Containment and equivalence for a fragment of XPath. In *PODS*, 2002.
21. P. O'Neil, E. O'Neil, S. Pal, I. Cseri, G. Schaller, and N. Westbury. ORDPATHs: Insert-Friendly XML Node Labels. In *SIGMOD*, pages 903–908, 2004.
22. S. Paparizos, Y. Wu, L. Lakshmanan, and H. Jagadish. Tree logical classes for efficient evaluation of XQuery. In *SIGMOD*, 2004.
23. K. Runapongsa, J. Patel, H.V. Jagadish, Y. Chen, and S. Al-Khalifa. The Michigan benchmark: Towards XML query performance, 2001. `http://www.eecs.umich.edu/db/mbench`.
24. A. R. Schmidt, F. Waas, M. L. Kersten, D. Florescu, M. J. Carey, I. Manolescu, and R. Busse. Why and How to Benchmark XML Databases. *SIGMOD Record*, 3(30):27–32, 2001.
25. B. Yao, T. Özsu, and N. Khandelwal. XBench benchmark and performance testing of XML DBMSs. In *ICDE*, pages 621–633. IEEE Computer Society, 2004.