Chapter 6

A Repository of Micro-Benchmarks for XQuery

In Chapter 3, we identified a need for precise and comprehensive tools for experimental evaluation, as well as a need for general methodology for experimental evaluation. In this chapter, we propose a micro-benchmarking methodology, as well as a repository of micro-benchmarks for XQuery, called MemBeR. First, we refresh our findings from Chapter 3 that motivate our work (Section 6.1). Then, we describe the micro-benchmarking methodology associated with MemBeR (Section 6.2). Further, we describe the MemBeR repository of micro-benchmarks (Section 6.3). To illustrate the MemBeR methodology, we also give an example micro-benchmark (Section 6.3.1). Finally, we discuss the benefits and weaknesses of our approach and conclude (Section 6.4).

This chapter is based on work previously published in Afanasiev et al., 2005a.

6.1 Introduction

The existence of suitable performance evaluation tools is imperative for the development of the XML processing engines. Performance benchmarks have proven to be a successful catalyst for the development of relational databases [Jain, 1991]. Since the introduction of XML and its query languages, many XML benchmarks have been proposed. In Chapter 3, we surveyed and analyzed five XQuery benchmarks publicly available in 2006: the Michigan benchmark (MBench) is a micro-benchmark suite, while XMach-1, XMark, X007, and XBench are application benchmarks. Among other questions, we investigated how the benchmarks are used in the database research community and whether they can be used for in-depth analysis of XML query processing techniques.

As a result of surveying scientific articles on XML query processing, we observed that the benchmarks are rarely used for performance evaluations of presented research (in less than 1/3 of the surveyed articles). Instead, ad-hoc exper-
Chapter 6. A Repository of Micro-Benchmarks for XQuery

Experimental evaluations are used. While the empirical evaluations used in scientific articles are focused on a particular language feature or query processing technique, the application benchmarks aim at the evaluation of a whole system in a particular application scenario. Our analysis showed that application benchmarks are not suitable for detailed and systematic evaluations of query processing techniques. The micro-benchmark suite MBench targets evaluation of XQuery language features in isolation and it is a good starting point for detailed analysis; however, it provides a query workload that is insufficient for a systematic and conclusive evaluation with respect to the tested language feature. Based on this, we concluded that the development of XML query engines is being held back by a lack of systematic tools and methodology for evaluating performance of query processing and optimization techniques.

The first problem we are facing is the lack of performance assessment tools allowing system developers and researchers to obtain precise and comprehensive evaluations of XML query processing techniques and systems. An evaluation is precise if it explains the performance of a processing technique or a system component on one language feature in isolation. In other words, it allows us to obtain an understanding of which parameters impact the performance of the target component on the target feature without “noise” in the experimental results due the performance of other components or other features. An evaluation is comprehensive if it considers all parameters that may impact the performance of the evaluation target and it explains the impact of every important parameter-value pair in a systematic way.

Second, a standard methodology is needed, explaining how to choose or develop appropriate performance evaluation tools for a given target, how to choose the parameters that are likely to be important, how to choose the value combinations for these parameters, and how to analyze the results. A standard methodology brings many benefits. It eases the task of performance evaluation. It also reduces the effort spent on experimental design, on dissemination of experimental results and comparison.

The research question that we address in this chapter is:

**6.1. Question. What is a suitable methodology for precise and comprehensive performance evaluation of XML query processing techniques and systems?**

As an answer to this question, we are proposing MemBeR, a structured repository of micro-benchmarks and related methodology. The micro-benchmarks target XML query processors on XML query language features in isolation (the main focus is on XQuery and its fragments). We find that micro-benchmarks are the most fitting tools for a precise and comprehensive performance evaluations. We also endow the repository with a micro-benchmarking methodology for facilitating the correct usage and creation of suitable micro-benchmarks. MemBeR is intended mainly for system developers and researchers, to help them analyze and optimize their techniques and systems.
Given the wide range of interesting XQuery features, ongoing development of XML processing engines, and ongoing developments of language extensions, such as full text search [World Wide Web Consortium, 2009a] and XML updates [World Wide Web Consortium, 2009b], a fixed set of micro-benchmarks devised today is unlikely to be sufficient and/or relevant in the future. Thus, we develop the repository as an open-ended community effort:

- repository users can contribute by creating new micro-benchmarks or by enhancing existing ones,
- quality control is guaranteed by a peer-review process, verifying that the proposed addition or change adheres to the micro-benchmarking methodology and it is not yet covered by the repository content.

MemBeR allows for continuous addition and improvement of micro-benchmarks, also targeting new performance challenges coming from applications and architectures perhaps not yet available today. In this manner, we hope that MemBeR will grow to provide a complete coverage of XQuery language features.

With MemBeR we aim to consolidate the experience of individual researchers that spend time and effort in designing micro-benchmarks for performance evaluation of their query optimization and processing techniques. We hope MemBeR will provide the necessary performance evaluation tools and methodology and will be widely used in the XML data management community.

MemBeR has a web-based interface and it is freely accessible at http://ilps.science.uva.nl/Resources/MemBeR/. Currently, MemBeR consists of 8 registered users, 5 contributors, and it contains 34 micro-benchmarks targeting different XQuery language features.

In the next section, we describe the MemBeR micro-benchmarking methodology.

### 6.2 The MemBeR micro-benchmarking methodology

Our goal is to build a repository of micro-benchmarks for studying the performance of XQuery processing techniques and engines. We identify four aspects of performance:

**Efficiency:** how well does a 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:** a system’s efficiency should be naturally evaluated against its resource needs, such as the size of a disk-resident XML store, with
or without associated indexes; the maximum memory size required by a streaming system, etc.

**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 for benchmarking.

**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 \[\text{Hidders et al., 2004, Miklau and Suciu, 2002, Paparizos et al., 2004}\]. Implementations aiming at completeness could use a yardstick to compare against.

In this chapter, our focus is mainly on benchmarks for testing efficiency and resource consumption. Nevertheless, we stress the importance of the other measures for a correct interpretation of performance. For devising correctness and completeness benchmarks, one can build on top of the XML Query Test Suite (XQTS) \[\text{World Wide Web Consortium, 2006b}\]. Although XQTS is not officially meant to test for an engine’s compliance to the XQuery standard, it is the best compliance test available today.

### 6.2.1 Micro-benchmark design principles

A well designed micro-benchmark is one for which the analysis of results is straightforward and the impact of each benchmark parameter is clear. In the following, we list the design principles for micro-benchmark creation that we adopt for MemBeR.

In some sense, these principles can be seen as refining our view on what micro-benchmarks are. Recall that \[\text{Runapongsa et al., 2002}\] were the first to propose micro-benchmarking in the context of XQuery. Two of the principles below, P1 and P3, were already implicit in \[\text{Runapongsa et al., 2002}\], and played a role in the design of MBench. The fourth principle, P4, was inspired by our analysis of MBench in Chapter 3 where we showed that the queries of MBench that are intended to test join processing are not sufficient for a detailed analysis since they vary several parameters simultaneously. All other principles are introduced here and we have not been able to trace them back to earlier literature on XML benchmarking. Note that P6 is not a design principle for individual micro-benchmarks, but rather concerns the structure of the entire micro-benchmark repository.

**P1**: A micro-benchmark should reduce to a minimum the influence of all but the tested system functionality and language feature. This can be achieved by designing a focused workload. For example, the presence of an XML Schema for the input document enables a large number of optimizations, at the level of an XML store and indices, at the level of XQuery rewriting and optimization, etc.
For any micro-benchmark whose target 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. If the purpose is to test path expressions navigating downward, the queries should not use sibling navigation, and vice versa.

**P2:** A micro-benchmark should (also) measure individual query processing steps. To get an accurate evaluation of a system component or functionality, it is often required to measure individual processing steps, such as query normalization, query rewriting, query optimization, data access, output construction, etc. For instance, XPath micro-benchmarks may measure the time to locate the elements that must be returned (this often means finding their IDs). Measuring such processing steps might require hooks into the tested engine. Even if the workload is carefully designed to trigger one system component and reduce the influence of the others, the total query execution times may still reflect the impact of too many factors.

**P3:** A micro-benchmark should strive to explicitly list and provide value ranges for all document, query, and other system or environment parameters that may impact the performance results. This is important in order for the benchmark results to be interpretable, reproducible, and comprehensive. In this way, a micro-benchmark provides well-documented measures.

At the time of creating a micro-benchmark, the creator most likely has a particular testing scenario in mind. Typically, the effort needed to invest in covering other testing scenarios as well is considerable, which makes P3 difficult to adhere to in practice. Nevertheless, we stress the importance of the principle.

**P4:** Whenever possible, micro-benchmark measures should vary one benchmark parameter at a time. This allows for analyzing the impact of each parameter on the performance measure. A micro-benchmark should aim at explaining the target’s performance in terms of the impact of the benchmark parameters. In this way, a micro-benchmark provides systematic 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 has an impact on the choice of data sets (see Section 6.2.5).

**P5:** A micro-benchmark should be extensible. A micro-benchmark should aim to remain useful even when systems undergo substantial development and achieve higher performance. The benchmark parameters should therefore allow for a wide enough range of values. The micro-benchmarks should also be regularly updated to reflect new performance standards.

**P6:** The number of micro-benchmarks for any given language feature in the repository should be kept to a minimum. This principle is meant to keep the repository focused. Instead of having two micro-benchmarks targeting two different aspects of the same language feature, the difference could be captured by a parameter in a single unified micro-benchmark. Still, there should be a balance
6.2.2 Micro-benchmark structure

Following the design principles stated above, we propose a more detailed specification of MemBeR micro-benchmarks.

A MemBeR micro-benchmark is an experimental tool for the performance evaluation (efficiency, consumption, correctness, or completeness) of a given component or functionality of an XML query processing system on a query language feature in isolation. The feature that the micro-benchmark studies (e.g., the system functionality, such as the query optimizer, on a language feature, such as structural joins) it is called the target of the micro-benchmark. The target also specifies the system type(s) and scenario(s) for which the micro-benchmark is proposed, for example a persistent database scenario or streaming scenario, etc.

The micro-benchmark includes a measure, which is a function of a parametrized workload and of other input parameters (e.g., parameters of the targeted system, or of the experimental set-up).

The workload consists of parametrized an XML data set and queries. The XML data set is parametrized by XML data characteristics that might impact performance results, such as document size, tree-depth, element fan-out, etc. (For a comprehensive list of XML data characteristics see Section 2.1.2) The value ranges for these parameters (and other relevant information, such as value distribution, value units, etc.) are provided. The data set can be empty, since XML fragments are legal XQuery expressions and thus an XML query may carry “its own data.” It might have an associated schema (e.g., DTD, XML Schema) or not.

The query set is characterized by its query language (e.g., XPath 1.0, XQuery 1.0). They can also be parametrized by query characteristics that might impact performance results, such as number of steps in a path expression query, numbers of query nesting levels, selectivity of a value selection predicate, etc. The
value ranges for these parameters (and other relevant information, such as value distribution, value units, etc.) are provided. The query set can also be empty, e.g., a benchmark that targets document pre-processing in the persistent database scenario does not contain queries.

Finally, a micro-benchmark is endowed with running scenarios that are guidelines on how to execute the benchmark and interpret the results. A running scenario specifies how to vary the values of one benchmark parameter in order to determine its impact on performance. Intuitively, it yields a family of curves where the varying parameter values are on the x axis and the measurements are on the y axis. A curve corresponds to a parameter-value configuration of the background parameters. The number of running scenarios in a micro-benchmark depends on the number of benchmark parameters.

Note that this micro-benchmark structure differs slightly from the structure presented in [Afanasiev et al., 2005a]. The differences concern only the presentation and are not conceptual. Specifically, we use a hierarchical model while in [Afanasiev et al., 2005a], an Entity Relationship model was used.

### 6.2.3 Micro-benchmarking methodology

In order to facilitate micro-benchmark execution and interpretation of results, we provide a list of general guidelines that comprises MemBeR benchmarking methodology.

When executing a micro-benchmark and analyzing results, the benchmark parameters should vary one at a time, while keeping the other parameters constant. This typically yields a family of curves where the varying parameter values are on the x axis, and the measured results on the y axis.

A micro-benchmark can have many parameters and the space of all parameter-value pairs, and thus measurements, can be huge (e.g., a measure involving 7 parameters with at least 3 values each that generates minimum $3^7 = 2187$ measurements). Performing and analyzing all the measurements might be not feasible. In such cases, at least the measurements for all end-of-range parameter values should be provided. Trying the measure with these values may give the system developer early feedback, by exposing possible system shortcomings. More measurements can be added to the analysis in a pay-as-you-go fashion further clarifying the initial results.

Micro-benchmark executions that vary less parameters than specified by the micro-benchmark require proper motivation. Neglecting parameters without further explanation compromises the correct interpretation of benchmark results and the comprehensiveness of the evaluation. Changes or restrictions of the parameter value-ranges should also be motivated, while testing extra values for a parameter is encouraged. In the case that the measurements obtained on the extended value-range reveal a significant difference in performance results in comparison with the measurements obtained on the original value-range, a revision of the
Chapter 6. A Repository of Micro-Benchmarks for XQuery

micro-benchmark should be considered.

*Determine and declare all hidden parameters that may impact the benchmark results.* Besides the parameters that the micro-benchmark explicitly varies, there might be hidden parameters that impact the benchmark results. For example, many efficient query processing 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 can be considered as hidden parameters and their values should be made clear when interpreting benchmark results. *When reporting results, declare the language and/or dialect supported by the system, even for features not used by the micro-benchmark.*

6.2.4 Preliminary classification of micro-benchmarks

The micro-benchmarks can be organized in the repository conform their target and measure. In this section, we outline a general classification 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 distinguishes between efficiency, consumption, correctness, or completeness. Further, we can classify micro-benchmarks according to the following criteria:

- The metric used by the benchmark measure, for example *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) and/or *query scalability* (increasing-size queries on fixed documents).

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

- The *type of the targeted engine*, such as: persistent database (store the document once, query it many times), streaming (process the query in a single pass over the document), and main-memory (the document is parsed and queried entirely in main-memory).

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

- The *targeted language feature* in a micro-benchmark is a precise classification criteria. We strive to provide exactly one micro-benchmark for each interesting feature.
There might be other classification criteria as the repository grows. Similarly to micro-benchmark principle P5, the repository should be easily extensible and its organization should also be regularly updated to provide the best classification of micro-benchmarks.

### 6.2.5 Data sets for micro-benchmarking

In order to satisfy micro-benchmark principles (P1, P3, P4, and P5), micro-benchmarks must provide parametrized data sets that allow for systematic varying of its parameters. This can be achieved only with synthetic data sets. Synthetic data generators are suitable for obtaining easily customizing test data. Nevertheless, coming up with one unified data set, even a synthetic one, on which all important characteristics can be varied at will, is hardly feasible. Notice that some parameters, such as size, depth, and fan-out are inter-related and thus cannot be independently controlled.

In this section, we consider two broad classes of synthetic documents for the MemBeR repository. Documents in the first class are schema-less and allow full control over the basic XML document characteristics. We propose a syntactic document generator for this class. Documents in the second class are schema-driven. We propose using existing declarative document generators for obtaining data sets in this class. These classes of documents and their generators are easy-to-use solutions for obtaining data sets for micro-benchmarking and we believe that they cover the basic needs for data sets for MemBeR. Nevertheless, other data sets can always be added to the repository.

Below, we briefly describe the two classes of documents.

**Schema-less parametric data generator**

For the class of schema-less documents, we propose a synthetic data generator that allows controlling: (i) the maximum node fanout, (ii) maximum depth, (iii) total tree size (number of elements), (iv) document size (disk occupancy), (v) the number of distinct element names in the document, and (vi) the distribution of tags inside the document. Out of these, the following parameters are required: (i) either tree size or document size; and (ii) either depth or fan-out.

The number of distinct element names is 1 by default; elements are named a1, a2 etc. The distribution of element tags within a document can be controlled in two ways. Global control allows tuning the overall frequency of element named a1, a2, ..., an. 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. Global distributions allow generating trees where any ai

---

1The generator checks the frequencies of several element tags at a given level for consistency.
may appear at any level. Close to this situation, for instance, is the Treebank
data set\textsuperscript{2} containing 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 \texttt{article}
and \texttt{inproceedings} in the DBLP data set\textsuperscript{3} other elements appear only below
level 7, such as \texttt{keywords} in XMark etc.

Fan-out, depth, and tag distribution have impact on different aspects of XML
query processing. For example, they have an impact on the disk occupancy of
many XML storage and structural indexing schemes, on the complexity and pre-
cision of XML statistical synopses, on the size of in-memory structures needed by
an XML stream processor, and on the performance of path expression evaluation
for many evaluation strategies. Thus, we will rely on this data set for varying
these parameters and for assessing such aspects.

The number and size of text values follow uniform or normal distributions.
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.

\textbf{Schema-derived data sets}

For the class of schema-derived data sets, we propose using ToXGene \cite{Barbosa02},
a declarative data generator that produces synthetic data according
to schema specifications of the desired test data. ToXGene relies on XML Schema
specifications. It allows for the specification of skewed probability distributions
for elements, attributes, and textual nodes. Being based on XML Schema, the
tool supports different data types, as well as id/idref constraints. It also offers
a simple declarative query language that allows one to model relatively complex
dependencies among elements and attributes involving arithmetic and string ma-
nipulation operations. For instance, it allows one to model that the total price
of an invoice should be the sum of the individual prices of the items in that in-
voice multiplied by the appropriate tax rates. Finally, ToXgene offers support for
generating recursive XML documents.

\section{The MemBeR repository}

In this section, we discuss the current version of the MemBeR repository and
give an example of a MemBeR micro-benchmark. We then discuss how well the
example meets the micro-benchmarking principles P1–P6.

A web-based interface to the MemBeR repository is located at \url{http://ilps.
science.uva.nl/Resources/MemBeR/}. The current version of MemBeR has the

\footnotesize{\begin{itemize}
  \item \textsuperscript{2}Available at \url{http://www.cs.washington.edu/research/xml/datasets}
  \item \textsuperscript{3}Available at \url{http://dblp.uni-trier.de/xml}
\end{itemize}}
6.3. The MemBeR repository

following components:

- Micro-benchmarks organized in categories based on the preliminary micro-benchmark classification presented in Section 6.2.4;
- Micro-benchmark results corresponding to the respective micro-benchmark and contributed by the repository users;
- The synthetic XML data generator presented in Section 6.2.5;
- Repository users that contribute micro-benchmarks and micro-benchmark results; and
- A list of XQuery benchmarking resources, such as links to the five XQuery benchmarks discussed in Chapter 3, the corrected and standardized queries of the five benchmarks; a list of open-source XQuery engines; link to XCheck, the tool for automatizing the process of running a benchmark and analyzing the results presented in Chapter 5, etc.

Currently, MemBeR contains 34 micro-benchmarks, contributed by 5 out of 8 registered users. It also contains micro-benchmark results corresponding to 14 micro-benchmarks.

6.3.1 An example of MemBeR micro-benchmark

In this section, we give an example of a MemBeR micro-benchmark created and published by Manolescu, Miachon, and Michiels. The benchmark can be found at: http://ilps.science.uva.nl/Resources/MemBeR/CHILD-ATTRIB.html. This micro-benchmark is part of a family of seven MemBeR micro-benchmarks targeting the performance XPath child navigation. We choose this example for its simplicity and in order to illustrate the micro-benchmarking design principles at work. In the next section, we discuss the extent to which the benchmark meets the MemBeR design principles.

In the next chapter, Chapter 7, we present a more comprehensive micro-benchmark (with respect to the number of tested parameters) for evaluating value-based joins processing techniques. For even more examples of MemBeR micro-benchmarks we refer to the MemBeR website and related publications [Afanasiev et al., 2005a, Manolescu et al., 2008b].

Target

The performance of child axis navigation of any type of XML query engine.

\(^4\)In compliance with the design principle P6, these micro-benchmarks should be combined in one if possible.
Measure

The benchmark measure is the query processing time as a function of the benchmark parameters. The benchmark varies two parameters: the number of child steps in a path expression and the child step selectivity. The number of child steps varies from 1 to 19 and it is controlled by a query set of 19 queries, one for each path depth. The child step selectivity varies from 1 to $2^{18}$ items in an exponential manner and it is controlled by setting the fan-out of one of the benchmarks XML documents to 2. Note that the child step selectivity is also responsible for the size of the context sequence of any intermediate child step.

The measure can be seen as query scalability with respect to the number of child steps in a path expression and with respect to query intermediate and end result size. Controlling both the path depth and the query selectivity at each depth is important, since these parameters have important independent impacts on the performance of child-navigation queries.

The unit of measurement is CPU seconds.

Data set

The data set consists of two documents, “layered.xml” and “exponential2.xml”, constructed with the MemBeR synthetic data generator. The first document is of size 12.33 MB. The root of the document is labeled “t1”, and it has 32,768 children labeled “t2”. At every level $i$ comprised between 3 and 19, there are 32,768 nodes labeled “ti”. Each element labeled “ti”, with $3 \leq i \leq 18$, has exactly one child labeled “t(i + 1)”. Elements labeled “t19” are leaves.

The second document is a binary tree of size 11.39 MB. At level $i$ (where the root is considered level 1), the document has $2^{(i-1)}$ elements labeled “ti”. Elements labeled “t19” are leaves. Every element of both documents have a unique attribute “@id”.

The two documents have the same tree depth, size, element-name distribution over the tree layers, etc. They differ only in the average tree fan-out—the first document has an average tree fan-out 1.06, while the second document has tree fan-out 2.

Note that the tree shapes of “layered.xml” and “exponential2.xml” are extremely regular; real-life documents are likely to be somewhere in between them in terms of average tree fan-out.

Query set

The query set consists of nineteen queries of the form
\[
doc(name)/t1/t2/.../tn/data(@id)
\]
where $n$ varies from 1 to 19 and $name$ is “layered.xml” or “exponential2.xml”. The queries retrieve the IDs of nodes at increasing depths of the document. Note
that all the queries have non-empty results. Query selectivity depends on the document against which the query is evaluated and on the number of child steps. When evaluated on “layered.xml” the query selectivity is constant for all the queries; when evaluated on “exponential2.xml” the query selectivity is exponential in the number of child steps.

The attribute step at the end of the child path is made in order to reduce the result serialization time. In this manner, the query processing time gets closer to the actual time it takes the query engine to locate the nodes.

This query is designed to test the ability of the query processor to deal with increasing lengths of child-path expressions and with increasing intermediate and end result size. For instance, materialization of intermediate path results has an increasing performance impact for longer queries.

**Running scenario**

The benchmark has one running scenario *varying the number of child steps* while fixing the child step selectivity by fixing the document on which the queries run. The results are represented by two curves, one for each document in the data set.

For assessing the query selectivity with respect to the number of child steps, the benchmark measures the query processing time of the query set on “layered.xml”. Since the number of returned nodes is constant ($2^{16}$) for all queries on this document, the performance time might be influenced only by the number of child steps. The number of nodes visited by a naive XPath evaluation strategy is in $O(n)$, thus the processing times of a query engine should behave, in the worst case, as a linear function of the number of child steps.

For assessing the query selectivity with respect to the intermediate and end result size, the benchmark measures the query processing time on “exponential2.xml”. The number of returned items is exponential in the number of child steps. The number of nodes visited by a naive XPath evaluation strategy is also in $O(2^n)$, thus the processing times of a query engine should be behave, in the worst case, as an exponential function of the number of child steps.

**Benchmark results**

To illustrate the benchmark in action, we execute it on four open-source XQuery engines: SaxonB v9.1 [Kay, 2009], Qizx/Open v3.0 [Axyana Software, 2009], Galax v0.5.0 [Fernández et al., 2006], and MonetDB/XQuery v0.30.0 [Boncz et al., 2006b].

The experiments are conducted on a Fedora 8 machine, with a 64 bit compilation, with 8 CPUs, Quad-Core AMD Opteron(tm) of 2.3GHz, and 20GB RAM. When running the Java implementations, SaxonB and Qizx/Open, we set the Java Virtual Machine maximum heap size value to 10GB. The experiments are run with XCheck, the testing platform presented in Chapter 5. The time mea-
measurements are computed by running each query 4 times and taking the average performance time(s) of the last 3 runs. SaxonB, Qizx/Open, and MonetDB report on detailed processing times, including the query processing time. Thus, conform the benchmark specification, we record the query processing time reported by the engines. Galax does not report on detailed times, thus we measure the total query execution time.

Figure 6.2 contains the micro-benchmark results for the four engines. On “layered.xml”, MonetDB/XQuery shows a linear increase in performance times with respect to the number of child steps. The other engines, though their corresponding curves are more irregular, do not show a steady linear increase of performance times. On “exponential2.xml”, SaxonB, Galax, and MonetDB/XQuery show a super-linear increase in the performance time with respect to the number of child steps of the query and a linear increase with respect to the child step selectivity. All three curves grow slowly or are almost constant on the first dozen queries and grow steeply on the rest. Qizx/Open shows a linear behavior on this document, with a drastic increase of the curve slope at the query of path depth 12.

Further analysis is needed to check whether the curves for testing the impact of child step selectivity are indeed linear or sub-linear. For example, we know that MonetDB/XQuery implements staircase join with pruning for evaluating XPath axes, thus it does one scan of both the context sequence and the document (with skipping) for every child step of the path expression. [Grust et al., 2003]
6.3.2 Meeting the design principles

In this section, we discuss how well the example micro-benchmark conforms to the design principles stated in Section 6.2.1.

Conform P1, the micro-benchmark isolates the impact on the performance of the tested language feature and of the two parameters (the number of child steps, and the child-step selectivity). The data set and query set are simple and they do not vary other features than the tested ones. By measuring the query processing time rather than the total query execution time, the benchmark conforms to design principle P2. As a result, the benchmark measures the performance of the dynamic evaluation phase of the XQuery processing model in isolation from the other phases.

The benchmark running scenario varies the number of child steps, while fixing the document on which the queries run. On “layered.xml”, only the number of child steps varies. On “exponential2.xml”, the child step selectivity varies together with the number of child steps. Conform P4 (and conform our micro-benchmarking methodology), a second running scenario is needed that varies only the child step selectivity while keeping the number of child steps constant. This can be achieved by executing the same query on different documents and comparing the results. Though the impact of the second parameter can be already seen on the results of the first running scenario, one needs to be aware that these results are dependent also on the number of child steps, thus the impact of the second parameter is not measured in isolation.

The benchmark tests for two parameters only. Arguably, there are many other parameters and parameter values that might impact the performance of child navigation. But whether they all need to be included in this micro-benchmark or separate benchmarks need to be created might be a subjective matter. The authors of this micro-benchmark developed six other micro-benchmarks testing the performance of child navigation (i) in the presence of position predicates, (ii) where the navigation is included in a predicate expression or not, and (iii) where the result nodes are retrieved or only located [Manolescu et al., 2008b]. Whether to assemble these micro-benchmarks together or not is subject to argumentation and might be a matter of opinion. Thus design principles P3 and P6 allow for different interpretations and debates.

Since these principles are difficult to fully adhere to when creating a micro-benchmark, we propose a pay-as-you-go strategy: allowing MemBeR users to submit and use micro-benchmarks that do not fully conform to P3 and P6 yet, and allow for updates to the benchmarks with new parameters or merge micro-benchmarks as the need arises.

Note that due to its simplicity, the above example benchmark can easily be extended with new parameters or, conform P5, with new values for the two parameters.
6.4 Conclusions

In this chapter, we tackled the problem of performance evaluation of XML query processors and the lack of suitable tools for *precise* and *comprehensive* performance evaluations. As a solution to this problem, we proposed MemBeR, an open-ended, community driven, repository of micro-benchmarks. We endowed the repository with micro-benchmarking design principles and methodology, with a fixed micro-benchmark structure, with suggestions for potentially interesting parameters, and tools for generating parametrized data sets.

MemBeR is freely accessible on the web and serves as a proof of concept of the MemBeR vision, its use, and potential.

In the next chapter, Chapter 7, we present a micro-benchmark for evaluating value-based joins processing techniques that follows the MemBeR methodology.