Available online at www.sciencedirect.com
ScienceDirect
Information Systems 33 (2008) 182–202
www.elsevier.com/locate/infosys

Toward microbenchmarking XQuery

Philippe Michiels[a], Ioana Manolescu[b,*], Cédric Miachon[c]

[a]University of Antwerp, Belgium
[b]INRIA Futurs, France
[c]LRI—Université Paris-Sud 11, France

Abstract

A substantial part of the database research field focusses on optimizing XQuery evaluation. However, there is a lack of tools allowing to easily compare different implementations of isolated language features. This implies that there is no overview of which engines perform best at certain XQuery aspects, which in turn makes it hard to pick a reference platform for an objective comparison. This paper is the first to give an overview of a large subset of the open source XQuery implementations in terms of performance. Several specific XQuery features are tested for each engine on the same hardware to give an impression of the strengths and weaknesses of that implementation. This paper aims at guiding implementors in benchmarking and improving their products.

Keywords: XML; Query; XQuery; Benchmark; Microbenchmark; Performance

1. Introduction

In the recent past, a lot of energy has been spent on optimizing XML querying. This resulted in many implementations of the corresponding specifications, notably XQuery and XPath. Usually, little time and space is spent on thorough measurements across different implementations. This complicates the task of implementors to compare their implementations to the *state-of-the-art* technology, since no one really knows what system actually represents it.

As is pointed out in [1], there are two possible approaches for comparing systems using benchmarks. Application benchmarks like XMark [2], XMach-1 [3], X007 [4] and XBench [5] 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. As such, these kinds of benchmarks are not very useful for XPath/XQuery implementors, since they are mainly interested in isolated aspects of an implementation that need improvement.

Micro-benchmarks, on the other hand, are designed to verify the performance of isolated features of a system. We believe that microbenchmarks are crucial in order to get a good understanding of an implementation. Moreover, it rarely happens that one platform is the fastest on all aspects. Only microbenchmarks can reveal which implementation performs best for isolated features. Our focus is to benchmark a set of important XQuery constructs that form the foundation of the language and thus greatly impact the overall query engine performance. These features are:

- XPath navigation
- XPath predicates (including positional predicates)

*Corresponding author.
E-mail address: Ioana.Manolescu@inria.fr (I. Manolescu).

0306-4379/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.is.2007.05.003

- XQuery FLWORs
- XQuery node construction.

The selected XQuery processors are chosen to represent both in-memory and disk-based implementations of the language.

We hope to continue this effort using automated tools such as XCheck [6,7]. This continuation involves the population of a repository with a large amount of ready-made micro-benchmarks as well as the benchmarking of many more platforms. We hope that this work can guide XQuery implementors to improve their products based on objective, thorough and relevant measurements.

Limitations: We take the view that detailed performance measures should document as much as possible the times spent by an XQuery processing engine in each stage of query evaluation—for instance, separating query optimization from query execution and from the XML result serialization time. From our experience, in the case of large-result queries, the serialization time can easily dominate the other evaluation times (sometimes by orders of magnitude)! Unfortunately, some engines do not provide a means to isolate the serialization time from the other execution components, e.g. when execution is streamed. Therefore, we have decided to measure the time to run each query as such, and then the time simply *counts the query results*, *with the hope that the latter time is a reasonable approximation of the time to run the query* without serializing the result.

We are aware of two possible problems of this approach. First, a very simplistic implementation may serialize the results and then count them, thus including, against our will, the serialization time in the counting query running time. Second, a sophisticated implementation may answer counting queries from some data statistics, e.g. histograms or indexes, without actually accessing the data. In this case, the execution time is incomparable with the time to run the simple query, without the count. Despite these shortcomings, we found the counting queries useful in practice as a means to approximate the otherwise inaccessible XML serialization time.

2. Settings

In this section, we present the documents (Section 2.1) and queries (Sections 2.2 and 2.4) used for the performance measures in this paper, as well as the rationale for choosing them. Section 2.5 describes our hardware and software environment, and the system versions used.

All documents, queries, settings and (links to) the systems used in these measures can be found at [8].

2.1. Documents

In order to have full control over the parameters characterizing our documents, we used synthetic ones, generated by the MemBeR project's XML document generator [1,9]. MemBeR-generated documents consist of simple XML elements, whose element names and tree structure are controlled by the generator's user. Each element has a single attribute called **@id**, whose value is the element's positional order in the document. The elements have no text children.

Some of the systems we tested are based on a persistent store, while the others run completely in memory. While we are aware of the inherent limitations that an in-memory system encounters, we believe it is interesting to include both classes of systems in our comparison, since performant techniques have been developed independently on both sides, and the research community can learn interesting lessons from both. To enable uniform testing of all systems, we settled for moderate-sized documents of about 11 MB, which most systems can handle well. As such, the stress testing of the systems below has a focus on query scalability, rather than data scalability.

To these documents, we added a family of 10 more documents of varying size, going from 100,000 nodes to 1,000,000 nodes. The purpose of this last document family was to enable an analysis in the way query engines process path queries. The interesting feature of such queries is that a naive implementation requires sorting and duplicate elimination to be performed after each XPath step, whereas efficient engines are able to avoid it. Our 10 chosen documents allow tracing the data scalability of an engine on path queries, which in turns allows some inferences about the engine's inner workings.

The document structures are outlined in Fig. 1. In this figure, white nodes represent elements, whose names range from **t1** to **t19**; dark nodes represent @id attributes. The exponential2.xml, layered.xml and mixed.xml documents have a depth of 19, which we chose so that complex navigation can be studied, and in accordance with the average-to-high