XCheck: A Platform for Benchmarking XQuery Engines

http://ilps.science.uva.nl/Resources/XCheck/

Loredana Afanasiev*
ISLA, University of Amsterdam
The Netherlands
lafanasi@science.uva.nl

Massimo Franceschet
Università "Gabriele D'Annunzio"
Italy
francesc@science.uva.nl

Maarten Marx
ISLA, University of Amsterdam
The Netherlands
marx@science.uva.nl

ABSTRACT

XCheck is a tool for assessing the relative performance of different XQuery engines by means of benchmarks consisting of a set of XQuery queries and a set of XML documents. Given a benchmark and a set of engines, XCheck runs the benchmark on these engines and produces highly informative performance output. The current version of XCheck contains all available XQuery benchmarks which are run against four XQuery engines: Galax, Qizx/open, Saxon and MonetDB/XQuery. XCheck's design makes it easy to include new engines and new benchmarks.

1. INTRODUCTION

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. The two main reasons to do XQuery benchmarking are (1) *correctness check* (does the output of engine X conform to the W3C standard?) and (2) *relative performance testing* (how well does engine X_1 perform, in terms of processing speed and memory use, compared to engines $X_2 \ldots X_n$?).

Running a benchmark *by hand* is tedious and time consuming. Even more so, when benchmarking the relative performance of several engines. One has to keep track of many system and engine parameters and the output is usually a huge amount of raw data that is difficult to interpret. A software tool is needed

1) to help XML query engine *developers* to evaluate the performance of their processor, also in comparison with other implementations.
2) to enable XML *researchers* to experimentally test their ideas, like a new optimization technique or the impact of language features, more easily.
3) to help XML *users* to compare and choose a query engine that performs well on their data.

This motivated us to create XCheck. The main goal of XCheck is to automate all the tasks involved in benchmarking XML query engines, except for the creative aspects of designing benchmarks and interpreting the results. In fact, XCheck is meant to minimize the technical effort of the last task. We implemented a platform that allows one to execute performance benchmarks on several query engines and helps in analyzing their relative performance. Ease of adding new engines and new benchmarks was an important constraint on XCheck's design.

XCheck already hosts all XQuery related benchmarks, and several engines: Galax [8], MonetDB/XQuery [11], Qizx/open [4] and Saxon [10]. To get an impression of the time-investment needed to run a benchmark: 43 hours were needed to run XMark on four engines with document sizes up to 113Mb.

XCheck supported Benchmarks
XMach-1[1] [5]
XMark [13]
XPathMark [9]
X007 [7]
XBench [14]
Michigan [12]

Table 1.

This abstract is organized as follows. Section 2 describes the design, architecture and usage of XCheck. Section 3 relates XCheck to other XML benchmarking platforms. Section 4 describes our planned demonstration.

2. SYSTEM DESCRIPTION

In this section we describe the design, architecture and usage of XCheck, which is delivered under the GNU General Public License. It is freely available at http://ilps.science.uva.nl/Resources/XCheck/. XCheck focuses on *performance benchmarks*, as opposed to *correctness benchmarks*. Thus, the benchmarks do not need to specify correct answers. Nevertheless, XCheck helps to detect incorrect answers by performing a comparison of the size of the answers given by the different engines.

The main idea behind the design of XCheck is that a user can learn most from *comparing* the performance of different engines on many benchmarks. Thus adding engines and

[1]XMach-1 is designed for a multi-user scenario with a query throughput under time constrains measure. XCheck executes only its single-user case.

*This work was sponsored by the Stichting Nationale Computerfaciliteiten (National Computing Facilities Fondation, NCF) for the use of supercomputer facilities, with financial support from the Nederlandse Organisatie voor Wetenschappelijk Onderzoek (Netherlands Organization for Scientific Research, NWO). L. Afanasiev and M. Franceschet/E. Zimuel are also supported by NWO, under grant numbers 017.001.190 and 612.000.207, respectively.

VLDB '06, September 12-15, 2006, Seoul, Korea.

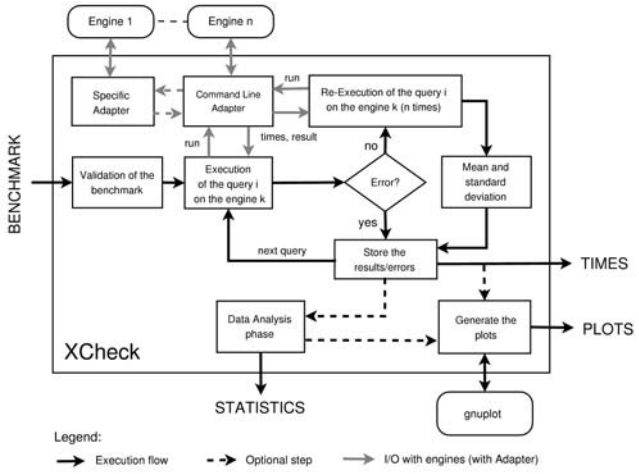

Figure 1: XCheck architecture

benchmarks had to be easy, and the output should be directed towards relative performance.

XCheck works in two phases. During the *running phase*, it executes a given performance test on a given set of query processors and produces detailed evaluation data, including diverse processing times and environment parameters. In the *data analysis phase*, XCheck applies statistical analysis and performance measures on the data obtained during the running phase. The output of the data analysis phase is a collection of high-level evaluation data presented in different formats (tables, plots, rankings).

The general process flow of XCheck is depicted in Figure 1. In this figure, BENCHMARK is an XML file specifying the experiment (which engines, which queries, which documents). The labels TIMES, STATISTICS and PLOTS denote the raw measurements, the analyzed data and the created data plots, respectively. Data analysis and creating plots is optional and can also be done in a latter stage, combining raw data from several experiments, if needed.

In the following, we describe the running and data analysis phases in more detail.

2.1 Running phase

Input The input of an experiment run on XCheck consists of: (i) a list of XQuery engines; (ii) a list of benchmark documents, or the commands to generate the documents whenever a document generator is available; (iii) a list of queries of the benchmark, or the commands to generate the queries whenever a query generator is provided. For each engine, for each document, XCheck runs all provided queries. This is implemented by leaving empty the argument of the `fn:doc()` function in the queries. XCheck will fill them with the appropriate file names. This design is sufficient to represent all XQuery benchmark published so far, including the multi-document scenario benchmarks.[2]

[2] XMach-1 and, partially, XBench are multi-document scenario benchmarks, i.e. a query is executed against a big collection of documents at once. For these benchmarks we used only one input document containing the list of the documents in the collection. Before each query execution, the input document is queried and a resulting sequence of document nodes is transmitted to the query as input.

Engine adapter design At the core of XCheck is a *pluggable engine adapter* design. There are many types of XML query engines, with different architectural design and running scenarios, implemented in different programming languages. XCheck calls each engine with a fixed input (query, set of documents) and receives a fixed output (query answer, several processing times, error messages). This is implemented via adapters which work as wrappers for the engines. In this way, XCheck can test any query engine as soon as an adapter for the engine is available.

There are two ways of implementing an adapter. If the engine is a command line executable, then an adapter is an XML document containing engine execution instructions and a description of the engine's output. XCheck processes this document and executes the instruction indicated there. Otherwise, a specific engine adapter in the form of a command line executable has to be implemented.

It is often desirable to measure the times taken by individual processing steps, such as, document processing time, query compilation time, query execution time, and result serialization time [3]. Measuring these times might be difficult or impossible, unless the engine provides this information. In such cases, XCheck captures these times and analyzes them.

Measurement accuracy Besides the times that the engines report, XCheck itself measures the total execution time, reported in CPU time. To reduce the possibility of unreliable results, XCheck executes each experiment a configurable number of times and outputs the average times, together with the standard deviation. We experienced that 3 runs are sufficient to obtain standard deviations which are less than 2% of the mean time. Moreover, XCheck records the hardware configuration of the machine on which it runs.

Error and crash handling Two types of errors might occur during execution. The first type is an error produced and handled by the engine, such as static and dynamic query processing errors. The second type, typically an engine crash, is handled by the platform. XCheck treats both types, and provides informative output to the user.

Finally, a pseudo-correctness test is implemented by comparing the size of the output of the different engines and producing a warning in case of large divergences.

Output The default output consists of an XML document containing the total query execution times, the processing times provided by the engines themselves, and the error messages whenever a query fails, grouped by engines, documents and queries. It also contains the total benchmark run time and the configuration of the testing environment. A more readable HTML format of this information is also provided. Optionally, XCheck saves the answers to the queries. Another optional output of the platform is a vast set of possible graphs displaying the data. The Gnuplot code for generating these graphs is also provided, so that the user can easily edit and modify the graphs.

Example As an example we run the XMark benchmark on the following XQuery engines: SaxonB 8.6.1, Galax 0.5.0, Qizx/open 1.0 and MonetDB/XQuery 0.10. The input query set consists of the 20 XMark queries and the document set consists of 7 documents corresponding to the scaling factors[3] 0.016 to 1.024 of size 1.80 MB to 113.99 MB, respec-

[3]XMark provides a document generator that produces documents whose sizes are proportional to a unique parameter

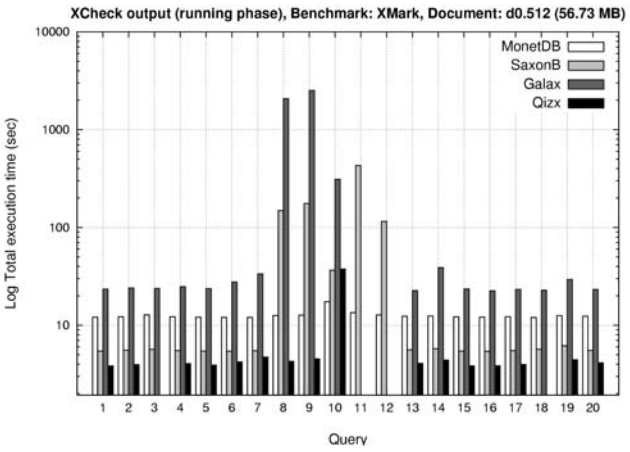

Figure 2: Total execution time for each query

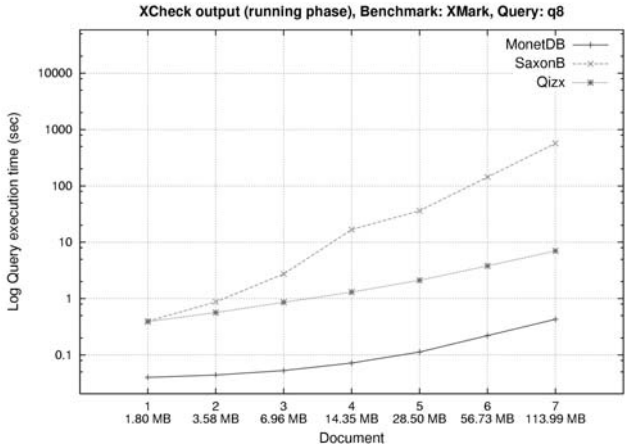

Figure 3: Query execution time for each document

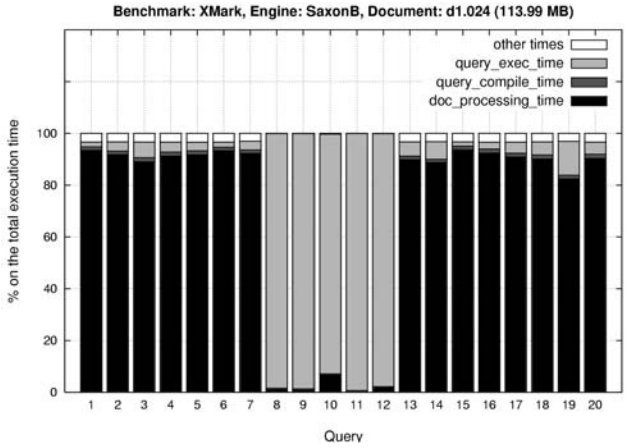

Figure 4: Query execution time for each document

tively. For each engine, for each document, every query was run four times. The times reported are the mean of the last three executions. The full output of XCheck on this example is accessible at http://ilps.science.uva.nl/Resources/XCheck/examples.html.

Figure 2 gives a first impression of the relative performance of the four engines, by showing the total execution times for each engine for each query on one specific document. Notice that Galax crashes on queries Q11 and Q12 (which are known to be hard) and Qizx/open does not parse queries Q3, Q11, Q12, and Q18. Moreover, Figure 3 shows how the different engines scale with respect to document size on query Q8. The times are those reported by the engines. Galax is not here, because, as confirmed by the authors, its reported query execution time is not reliable for the version we checked. In both cases, XCheck plotted the times in logarithmic scale.

Finally, Figure 4 provides a breakup, in terms of percentages, of the total execution time for Saxon on all XMark queries and on the biggest document in our sequence. Notice that on queries Q8–Q12 almost all the time is spent on query execution, while on all other queries the document processing time was the main factor.

2.2 Data analysis phase

After running the benchmark, XCheck can perform some data analysis tasks. Most measures are dependent on the specific benchmark and are not provided by XCheck. That given, XCheck performs some standard data analysis, as indicated in the following.

Quantitative analysis The goal is to analyze the *amount of time* spent by any engine for the processing of any evaluation phase on any fragment of the benchmark. As an example, we define the *medley relay speed* (measured in MB/sec) of an engine on the document set D and the query set Q as follows:

$$\mathrm{MRS}_{D,Q} = \frac{|Q| \cdot \sum_{d \in D} \mathsf{size}(d)}{\sum_{d \in D, q \in Q} \mathsf{time}(d,q)}$$

It summarizes the elaboration speed of the engine on the given documents and queries.

called the scaling factor. A scaling factor of 1 produces a document of about 100 MB.

Qualitative analysis The goal is to analyze the *qualitative behavior* of the performance of the engines. Relevant measures are the *qualitative distance* between two engines, which indicates the similarity between the evaluation strategies of the two engines, and the *stability measure* of an engine, which indicates the stability of the elaboration times of the engine.

Scalability analysis The goal is to analyze the performance of any engine either when the complexity of the input query grows (*query scalability analysis*) or when the size of the input document grows (*data scalability analysis*). The scalability analysis is significant only when the benchmark has been designed to target the engine scalability. For instance, let $D = (d_1, d_2, \ldots, d_k)$, for $k \geq 2$, be a sequence of documents of increasing sizes. We define the *data scalability factor* for an engine on the document sequence D and the query set Q as follows:

$$\text{if } k = 2, \text{ then } \quad \mathrm{DS}_{(d_1,d_2),Q} = \frac{\mathrm{MRS}_{d_1,Q}}{\mathrm{MRS}_{d_2,Q}},$$

$$\text{if } k > 2, \text{ then } \quad \mathrm{DS}_{D,Q} = \frac{\sum_{j=1}^{k-1} \mathrm{DS}_{(d_j,d_{j+1}),Q}}{k-1}.$$

Notice that, by virtue of the definition of medley relay speed (MRS), a data scalability factor less than 1 (respectively, equal to 1, bigger than 1) corresponds to a sub-linear (re-

spectively, linear, super-linear) increase of the elaboration time when the data size increases.

Below is an extract of the data analysis result on our example. We excluded from the computation queries Q3, Q11, Q12 and Q18 because they are not supported by all engines. Finally, as said above, we have no information on the query execution time for Galax.

	Total execution time		Query execution time	
Engine	**MRS**	**DS**	**MRS**	**DS**
MonetDB	4.356 MB/s	0.863	212.220 MB/s	0.768
Saxon	1.611 MB/s	1.145	1.927 MB/s	1.502
Qizx	7.629 MB/s	0.847	8.511 MB/s	0.946
Galax	0.131 MB/s	1.707	...	...

3. RELATED SYSTEMS

To the extent of the authors knowledge, there are two other automated testing platforms for evaluating XML query engines, BumbleBee [1] and XSLTMark [2]. BumbleBee is a test harness for evaluating XQuery engines and for validating queries expressed in the XQuery language. Although it measures the total execution times its main goal is to test engine's compliance with the language specification. The application can execute user defined tests containing reference answers for the correctness check. XSLTMark is a similar application for XSLT processor performance and compliance benchmarking. It comes with a collection of default test cases that are performance oriented. Both Bumblebee and XSLTMark have a fixed input/output pluggable adapter design.

In comparison, XCheck is optimized for executing user defined performance tests. It is based on a more flexible input/output adapter design that allows the users to customize the information the platform is managing. This allows one to obtain a detailed evaluation for the intermediate processing steps and other important engine parameters. XCheck performs statistical analysis of the data and outputs graphs, facilitating correct interpretation of the results. Although XCheck does not perform a proper correctness test, it implements a pseudo test by comparing the size of the query results of several engines relative to each other.

4. DEMONSTRATION SETUP

The demonstration of XCheck has two purposes. First, we show how XCheck works. As running a benchmark takes too much time, we focus on the preparation of the input and the interpretation of the output of XCheck. Second, we provide the *results* obtained with XCheck.
For the first purpose, we show in particular:

- how easy it is to add benchmarks and engines to XCheck;
- how the output of XCheck addresses the use-cases described in the Introduction;
- the under-the-hood design choices.

For the second purpose, visitors of the demo can see how their favorite engine performs on the benchmarks in Table 1. We are currently collecting these data and plan to run all these benchmarks on all XQuery processors described in [6], provided they are freely available. Notice that Figure 16 in [6] gives the results of running XMark on 17 XQuery engines, but the data are collected from the literature (and hence the corresponding experiments were run on different machines). Hence the results are not easy to compare. Using XCheck we can provide an independent and fair comparison since every test is performed under the same controlled circumstances.

5. ADDITIONAL AUTHORS

Enrico Zimuel, Università "Gabriele D'Annunzio", Italy. email: zimuel@science.uva.nl.

6. REFERENCES

[1] BumbleBee. http://www.xquery.com/bumblebee/.
[2] XSLTMark. http://www.datapower.com/xmldev/xsltmark.html.
[3] L. Afanasiev, I. Manolescu, and P. Michiels. MemBeR: a micro-benchmark repository for XQuery. In *Proceedings of XSym*, pages 144–161, 2005.
[4] Axyana software. Qizx/open. An open-source Java implementation of XQuery. http://www.axyana.com/qizxopen, 2006.
[5] T. Böhme and E. Rahm. XMach-1: A benchmark for XML data management. In *Proceedings of BTW2001*, Oldenburg, 2001.
[6] P. Boncz, T. Grust, M. van Keulen, S. Manegold, J. Rittinger, and J. Teuber. MonetDB/XQuery: a fast XQuery processor powered by a relational engine. In *Proceedings of SIGMOD*, 2006.
[7] 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 *Proceedings of CIKM*, pages 167–174, 2001.
[8] M. Fernández, J. Siméon, C. Chen, B. Choi, V. Gapeyev, A. Marian, P. Michiels, N. Onose, D. Petkanics, C. Ré, M. Stark, G. Sur, A. Vyas, and P. Wadler. Galax. The XQuery implementation. http://www.galaxquery.org, 2006.
[9] M. Franceschet. XPathMark: an XPath benchmark for the XMark generated data. In *Proceedings of XSym*, pages 129–143, 2005. http://www.science.uva.nl/~francesc/xpathmark.
[10] M. H. Kay. Saxon. An XSLT and XQuery processor. http://saxon.sourceforge.net, 2006.
[11] MonetDB/XQuery. An XQuery Implementation. http://monetdb.cwi.nl/XQuery, 2006.
[12] K. Runapongsa, J. Patel, H. Jagadish, Y. Chen, and S. Al-Khalifa. The Michigan Benchmark: A Microbenchmark for XML Query Processing Systems. In *Proceedings of EEXTT*, pages 160–161, 2002.
[13] A. R. Schmidt, F. Waas, M. L. Kersten, M. J. Carey, I. Manolescu, and R. Busse. XMark: A Benchmark for XML Data Management. In *Proceedings of VLDB*, pages 974–985, 2002. http://monetdb.cwi.nl/xml/.
[14] B. Yao, T. Özsu, and N. Khandelwal. XBench benchmark and performance testing of XML DBMSs. In *Proceedings of ICDE*, pages 621–633, 2004.