Querying XML: benchmarks and recursion

Afanasiev, L.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: https://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
In Chapters 3 and 4, we identified the need for a software tool that helps executing benchmarks and reporting performance results. In this chapter, we address the question whether it is possible to automate the execution of performance benchmarks on many XML query engines and the comparison of their performance? As a solution, we present XCheck, a tool for running performance benchmarks that measure execution times on sets of XML documents and sets of queries, formulated in an XML query language, such as XPath and XQuery. Given a benchmark and a set of engines, XCheck runs the benchmark on these engines, collects performance times, query results, and the testing environment configuration. Version 0.2.0 of XCheck runs the XPath and XQuery benchmarks that were openly available in 2007 on 9 different XPath and XQuery engines. XCheck’s design makes it easy to include new engines and new benchmarks.

Part of this work was published in Afanasiev et al. 2006.

5.1 Introduction

The task of running a performance benchmark (T1) is notoriously tedious and time consuming [Jain 1991]. This task becomes even more problematic, when benchmarking engines in a relatively new research field, like XML querying, where the engines are still immature and require a lot of tuning and error-handling (see Section 3.6 of Chapter 3). The difficulties grow when evaluating the relative performance of several engines. In Section 3.6, we have seen that such evaluations are very useful, but tedious to implement—one has to keep track of the execution of many engines, handling their output and performance times.

As we argue in Chapter 4, one important aspect of performance evaluation is its repeatability. In order to make an evaluation repeatable, one needs to rigorously document all performance critical parameters of the engines and the testing
environment; evaluations that do not take in consideration this information can lead to misleading conclusions \cite{Jain1991}. We call this task \textit{documenting the benchmark experiment} (T2). This is yet another tedious task that many performance evaluators fail to complete (see Chapter 4).

After running the benchmark and gathering the measurements comes the most difficult task: \textit{analyzing the benchmark results} (T3). This requires mostly intellectual work, though analyzing large numbers of measurements is impossible without statistical tools and data visualization.

To a large degree, the three tasks described above—running a performance benchmark (T1), documenting the benchmark experiment (T2), and analyzing benchmark results (T3)—can be automated. A software tool is needed to help execute these tasks under a single roof, from running the experiments to aggregating the results and presenting them in an easily-readable format. Such a tool is valuable to

- \textit{developers and researchers}, for evaluating the performance of their engine or research prototype, also in comparison with other implementations; and
- \textit{users}, for comparing and choosing a query engine that performs well on their data.

We are aware of only two open source tools, BumbleBee \cite{BumbleBee2006} and XSLTMark \cite{XSLTMark2006}, that target the automatic execution of benchmarks in the area of XML querying. BumbleBee is a test harness aimed at testing the correctness of XQuery engines, rather than performance. XSLTMark is a performance benchmark and testing platform for XSLT engines only. The goals of both tools are too narrow for covering the tasks described above. Moreover, as of September 2008, both testing platforms are no longer available. We discuss these tools in more detail in Section 5.4. Further, there are no generic testing platforms that aim at automating tasks T1–T3 and at accommodating any XML query engine and any benchmark.

In this chapter we address the following question:

\textbf{5.1. \textit{Question.} Is it possible to build a generic tool for automating the tasks T1–T3 and what are the design choices that need to be made?}

We answer these questions by developing XCheck. XCheck is a testing platform for running performance benchmarks measuring \textit{performance times}. It runs benchmarks whose atomic measure is the performance time of processing \textit{one query} possibly against a \textit{document/collection} in a \textit{single-user scenario}. The platform can test any XML query engine, stand-alone or DBMS, that has a \textit{command line interface}. It allows one to test several query engines in one experiment and helps analyzing their relative performance; new benchmarks and engines can easily be added.
5.2 XCheck

XCheck is delivered under the GNU General Public License and it is freely available at http://ilps.science.uva.nl/Resources/XCheck/

This chapter is organized as follows. In Section 5.2 we describe XCheck’s functionalities and architecture. In Section 5.3 we give an example of XCheck’s usage and describe XCheck’s coverage. In Section 5.4 we describe the two related tools mentioned above, BumbleBee and XSLTMark. We conclude in Section 5.5.

5.2 XCheck

In this section, we describe XCheck’s goals, functionalities, and architecture. We give a general overview of XCheck; for a complete description of the platform and its options, see XCheck’s webpage: http://ilps.science.uva.nl/Resources/XCheck/

XCheck’s goal is to automate tasks T1–T3 described in the introduction while satisfying the following requirements: (i) it should have a good coverage of existing XML query engines and performance benchmarks; (ii) it should be easy to integrate new engines and benchmarks; (iii) it should have a flexible interaction with the engines, e.g., it can selectively collect the data that the engine outputs; (iv) it should allow for the comparison of performance of several engines; (v) it should have an output that is easily readable for humans, but also allows for further automatic processing; and (vi) it should be easy to run.

As a result, XCheck is a testing platform that takes as input a performance benchmark workload consisting of a set of XML documents and a set of queries, and runs it on a given set of XML query engines. The platform targets benchmarks whose atomic measure is the performance times of processing one query, possibly against a document/collection. It can test stand-alone or DBMS XML query engines that have a command line interface. It communicates with the engines via an engine adapter that allows the users to specify all the relevant information about the engine, including running instructions and output format. It has an easy-to-run command line interface, and a workflow oriented towards minimizing the total time the user spends on execution and analysis of the benchmark. It stores the experimental data in XML for further automatic processing. It also aggregates the results, builds plots and presents the information in HTML format for a quick overview and human interpretation.

XCheck has the following functionalities:

1. Running performance benchmarks, collecting performance times, errors, and optionally, query results. These functionalities address tasks T1 and T2 presented in the introduction;

2. Documenting engine configuration and the configuration of the testing environment. These functionalities address task T2;
3. Facilitating the analysis of performance measurements, by computing statistics and generating plots of the results. These functionalities address task T3.

In the following subsection, we present XCheck’s architecture and workflow in more detail.

### 5.2.1 Architecture and workflow

XCheck works in two steps. During the first step, XCheck runs a given experiment specification describing which XML query engines to run on which documents and on which queries and it gathers the engines’ outputs and processing times. During the second step, XCheck aggregates the data obtained during the running step, and documents the configuration of the engines and of the testing environment. The output of the running step is raw performance time measurements and results presented in a machine-readable format, while the output of the augmenting step is a collection of raw and aggregate data presented in an integrated human-readable format.

The general workflow of XCheck is shown in Figure 5.1. The **EXPERIMENT SPECIFICATION** label denotes XCheck’s input. The **TIMES**, **RESULTS**, and **ERRORS** labels denote raw performance time measurements, query processing results, and errors, respectively. The **PLOTS** label denotes different plots created by the system.
for visualization of the raw performance times. The INTEGRATED OUTPUT label denotes an easy-to-read output that integrates all the raw measurements and aggregate data collected by the platform. A detailed description of XCheck components and workflow follows below.

**Input** XCheck takes as input an *experiment specification*. An experiment specification consists of: (i) a non-empty list of XML query engines; (ii) a possibly empty list of documents/collections, or the commands to generate the documents whenever a document generator is provided; (iii) a non-empty list of queries, or the commands to generate the queries whenever a query generator is provided. The list of documents can be empty since, for example, XQuery queries do not necessarily need an input document, a query can construct a document. This design is sufficient to represent all XQuery benchmarks presented in Chapter 3. The input is represented in XML. Since XML is both human and machine readable, it is a natural choice for this purpose.

**Engine adapters** XCheck communicates with the XML query engines via *engine adapters*. The platform requires that the XML query engine has a command line interface for the execution of query processing operations: document processing, query compilation and processing. Then XCheck executes the engine via a system call on a particular input and stores the engine’s output. The engine adapter contains: (i) the engine’s command line execution instructions, (ii) a formal description of the engine’s output format, and (iii) natural language documentation of the engine.

The execution instructions are specified as a template with the running parameters, like specific query and/or document, to be filled in by XCheck at the running step. The output is described with the help of regular expressions that allow XCheck to extract pieces of relevant information from the engine’s output. The documentation of the engine contains the description of the performance critical engine parameters, like the software version, software compilation options, engine execution options, etc. The adapters are represented in XML, again chosen because it is both human and machine readable. The *engine handler* shown in Figure 5.1 is responsible for the validation and interpretation of the engine adapters.

**Kernel** The role of the kernel of XCheck consists of executing one atomic performance measurement: running one engine on one given query and possibly on a document/collection and measuring the query processing times. If the engine crashes or outputs an error at the first execution, the kernel stores this information. Examples of errors that an engine outputs are: document parsing errors, static and dynamic query processing errors [World Wide Web Consortium 2007b]. Otherwise, the kernel re-executes the same measurement another \( N \) times \((N + 1\)
Chapter 5. XCheck: a Tool for Benchmarking XQuery Engines

times in total) and stores the mean and the standard deviation of different processing times obtained during these last \( N \) executions. The times obtained during the first execution are ignored. This is done to increase the accuracy of the time measurement (see Section 5.2.2). Optionally, XCheck can store the query results output by the engine. In Figure 5.1 the kernel is indicated with a dotted line.

The general running strategy of XCheck is to iterate through each engine, each document/collection, and each query in the order given by the experiment specification, and to execute the kernel for each triplet (engine, document, query) or pair (engine, query), in case the document list is empty. This is the job of the experiment handler from Figure 5.1.

**Data aggregator** The data aggregator does the following: (i) it aggregates the raw performance times and presents them in a easily-readable format like tables and plots; and (ii) it documents the configuration of the engines and testing environment. It also performs a “pseudo-correctness” test of the query results. This test is meant to signal possibly wrong query results that might invalidate the performance studies.

When the experiment has finished executing, the data aggregator collects the raw performance times and computes simple descriptive statistics over them: the sum and the average processing times per engine, per engine and document, and per engine and query. Also, it keeps track of the total running time. The performance times and statistics are presented in tables. Optionally, the data aggregator calls the times plotter (Figure 5.1) to generate several types of plots. XCheck plots the times for: (i) each engine, (ii) each document, (iii) each query, (iv) each engine and document, and (v) each engine and query. The plots are generated with Gnuplot [Williams et al., 2008]. To make it easy for the user to edit the plots, XCheck stores the Gnuplot code used to generate them. Examples of plots produced by XCheck are given in Section 5.3.

Further, the data aggregator collects and documents the configuration of the engines and of the testing environment. The environment information such as computer details: CPU frequency, cache memory size, available hard disk size, and the operating system, are obtained automatically. The engine configuration—performance critical parameters such as version, compilation options, and execution parameters—are retrieved from the engine adapters.

XCheck’s main goal is to automate the execution of performance benchmarks as opposed to correctness tests. A considerable effort is done by the W3C XML Query Testing Task Force to develop and maintain the XML Query Test Suit (XQTS) [World Wide Web Consortium, 2006a] that provides query results and guidelines for assessing the correctness of an XQuery implementation. Executing such correctness tests is out of the scope of XCheck—when evaluating the performance of an engine, the correctness of its results is assumed. However, in practice, implementations are often incomplete or erroneous, therefore the data
aggregator also performs a test meant to signal the user when an engine outputs possibly wrong results. This test is done by comparing the size of the output of the different engines. A warning is produced if the size of a result output by an engine significantly differs from the average size of the results output by the other engines participating in the experiment. This test is not accurate and it can be applied only when there is more than one engine in the experiment, nevertheless, it proved to be a useful tool for the benchmark analysis presented in Chapter 3.

**Output**  The default output consists of an XML document containing the query processing times and the error messages of the failed queries, grouped by engines, documents and queries. It also contains the total experiment running time, engine and testing environment configuration. Optionally, XCheck saves the answers to the queries. Another optional output is a large set of plots displaying the performance times. The Gnuplot code for generating these plots is also provided, so that the user can easily edit and modify them. We chose the XML format to store this information in order to facilitate future automatic processing. The user can compute any aggregate measure or statistics that are not implemented by XCheck by using an XML query engine.

A more readable HTML format containing all the information collected or computed by XCheck is also provided and browsable from a single HTML webpage. The HTML webpage lists the following information: a natural language description (provided by the user) of the experiment; the total time it took to execute the experiment; a link to the XML file containing the input experiment specification; the list of engines, their description and configuration; the list of documents, their description, and links to the actual files; the list of queries, their description, and links to the files containing the queries; the configuration of the testing environment; a link to the XML file containing the output; a list of tables, one per engine, containing the performance times or errors per document (the rows) per query (the columns); a list of plots containing the average processing times for each engine, the average times for each document and engine, the average times for each query and engine; 5 links to HTML galleries of plots, one per each type of plot that XCheck outputs; and finally, a table containing the results of the pseudo-correctness tests—one row per document, one column per query, each cell contains the result sizes obtained with each engine. The HTML presentation greatly improves the readability of the performance results produced by the experiment. An example of XCheck’s output is given in Figures 5.2–5.10 in Section 5.3.

### 5.2.2 Collecting performance times

XCheck measures the total time it takes to run an engine on one query and zero or one document/collection, i.e., one run of the kernel. The platform measures the CPU time with the Unix command `time`. The unit of measure is seconds.
XCheck also keeps track of the wall-clock time elapsed from the beginning of the experiment until its end. This time is reported to the user as an indication of the total amount of time it takes to run the experiment. This time is not used for comparisons.

To avoid unreliable results, XCheck runs the same experiment $N + 1$ times and takes the average and the standard deviation of the last $N$ evaluation times. The first measurement is ignored because it usually contains the warm-up time of the operating system, i.e., the time spent on loading the application in the system’s cache. For Java applications, the Java virtual machine imposes another level of software between the application and the operating system that may alter the runtimes of the applications under evaluation. Taking the average over the last $N$ evaluation times improves the accuracy of the measurement. In our experiments of executing the standard XQuery benchmarks on 4 different engines (see Chapter 3), we took $N = 3$ and obtained standard deviations within 2% of the mean time. Based on this observation we set the default value of $N$ to 3. The user can change this with a command line option of XCheck.

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 [Manolescu et al., 2008b]. Measuring these times is difficult or impossible, unless the engine provides this information. In the latter case, XCheck captures these times. XCheck defines and collects, if available, the following types of query processing times:

- **document processing time** is the time that an engine takes to parse the input XML documents and create the internal document representation (in main memory or disk).

- **query compile time** is the time an engine takes to parse the query and translate it in an internal formalism of the engine. This time includes query normalization and rewriting, if implemented by the engine.

- **query execution time** is the time the engine takes to execute the query. This time includes only the time it takes to locate and/or construct the query results without outputting them. Usually, this is the most interesting elaboration time for comparing different query processing techniques and implementations.

- **serialization/output time** is the time it takes an engine to serialize and output the query result.

- **total time** is the total time an engine takes to process a query, starting with the engine invocation until the engine outputs results.

Out of all these times, the total time is measured by XCheck, while the rest can be provided by the engines. It is not always possible to separate the query
processing steps, and it is not always the case that an engine outputs detailed
times, but whenever it does, XCheck can record them. Note that XCheck is not
responsible for the accuracy of these times, nor can it determine what unit of
measure they use, CPU or wall clock time. It is important to remember this fact
when comparing the detailed performance times of different engines, since the
measurements might not always be comparable.

5.3 XCheck in action

In this section we give an example of XCheck’s usage by running the most pop-
ular XQuery benchmark, XMark [Schmidt et al., 2002], on 4 XQuery engines:
Saxon [Kay, 2009], Galax [Fernández et al., 2006], Qizx/Open [Axyana Software,
2006], and MonetDB/XQuery [Boncz et al., 2006]. Further, we show XCheck’s
coverage in terms of the engines and benchmarks it already accommodates.

5.3.1 Running XMark

As an example of XCheck usage and output, we run the XMark benchmark
on the following XQuery engines: SaxonB 8.7, Galax 0.5.0, Qizx/Open 1.0 and
MonetDB/XQuery 0.10.3. The input query set consists of the 20 XMark queries
and the document set consists of 7 documents corresponding to the scaling factors
0.016 to 1.024 of size 1.80 MB to 113.99 MB, respectively. The times reported
are the mean of the last three \( N = 3 \) executions. The experiment was run on
a machine with the following specifications: Intel(R) Xeon(TM) CPU 3.40GHz,
with 2 GB of RAM, running Debian Gnu/Linux version 2.6.16.

Figures 5.2–5.10 give a first impression of the HTML webpage output by
XCheck. The full output of this example is accessible at http://ilps.science.
uva.nl/Resources/XCheck/results/Lisa/xmark/output/outcome.html.

Figures 5.11, 5.12, and 5.13 give a closer look at 3 plots output by XCheck. Figure 5.11 shows the relative performance of the four engines, by showing the
total execution times for each query on one specific document of size 57MB (doc-
ument scaling parameter \( f=0.512 \)). This time is measured by XCheck, and it is
the CPU time of one kernel run measured in seconds. Note that there are a few
missing values in the plot: Galax crashes on queries Q11 and Q12, which contain
nested for-loops and data value joins; Qizx/open outputs static type checking er-
rors on Q3, Q11, Q12, and Q18, which contain the \texttt{op:numeric-multiply(A,B)}
operation on arguments with static type \texttt{xs:anySimpleType}. The plot gives a
quick overview of the relative performance when the engines are treated as off-
the-shelf (no special tuning) and on-the-fly (no document pre-processing) XQuery

\footnote{As detailed in Chapter 3, XMark provides a document generator that produces documents
whose sizes are proportional to a unique parameter called the scaling factor. A scaling factor
of 1 produces a document of about 100 MB.}
Figure 5.2: XCheck’s HTML output listing: the name of the experiment; a short description of the experiment; a list of the total time it took to run the experiment; a list of engines that were tested, their description and specifications; a list of the documents, their names, sizes, and description.
Figure 5.3: XCheck’s HTML output listing: how the time measurements were computed; the hardware and software specifications of the machine on which the experiment was run; detailed document processing times for MonetDB/XQuery.
Figure 5.4: XCheck's HTML output listing detailed query execution times for Qizx/open. The table cells with a red background indicate an error obtained during the execution of a query (given by the column name) on a document (given by the row name). The links given in these cells lead to the error messages output by XCheck or by the engine.
Figure 5.5: XCheck’s HTML output containing: three bar-plots showing the average query execution times per engine, per document and engine, per query and engine; links to plots, one for each types of plots.
Chapter 5. XCheck: a Tool for Benchmarking XQuery Engines

![Table showing query results sizes](image)

Figure 5.6: XCheck's HTML output containing the sizes (in KB) of the query results. Each cell lists the sizes of the results obtained by the four engines on the respective query and document. The table cells with red background indicate large deviations of the result size output by an engine from the average query result sizes output by the other engines. In this way, an incorrect query result is signaled.
Figure 5.7: XCheck’s HTML output containing a gallery of bar and line plots showing the performance times for each query in the experiment per document and engine. For each query and for each time measure used there are two plots presented, a bar and a line plot. Under each plot there are links to the raw data and to the GnuPlot source code that generated the plot. Note that the plots visible in this screenshot show the document processing times only for MonetDB/XQuery and SaxonB. The other two engines, Galax and Qizx/open, do not report these times.
Chapter 5. XCheck: a Tool for Benchmarking XQuery Engines

Figure 5.8: XCheck’s HTML output containing a gallery of bar and line plots showing the performance times for each document in the experiment per query and engine. For each document and for each time measure used there are two plots presented. Under each plot there are links to the raw data and to the Gnuplot source code that generated the plot. Note that the plots visible in this screenshot show the document processing times only for MonetDB/XQuery and SaxonB. The other two engines, Calx and Qizx/open, do not report on those times.
5.3. XCheck in action

Figure 5.9: XCheck’s HTML output containing a gallery of bar and line plots showing the performance times for each engine in the experiment per query and time measure used. For each engine and document in the experiment there are two plots presented. Under each plot there are links to the raw data and to the Gnuplot source code that generated the plot.
execution time is mostly influenced by the document processing time. Figure 5.10: XCheck's HTML output containing a gallery of bar and line plots showing the performance times for each engine in the experiment per document and time measure used. For each engine and query in the experiment, there are two plots presented. Under each plot there are links to the raw data and to the Gnuplot source code that generated the plot. Note how the query execution time for “q1” gradually grows as the document size increases, and how the total execution time is mostly influenced by the document processing time.
5.3. XCheck in action

engines. It also shows that Q8-Q12 are challenging queries.

Figure 5.12 shows how the query execution times scale with respect to document size on query Q8. The times are those reported by the engines: MonetDB/XQuery reports CPU time; Saxon and Qizx/open report wall-clock time. The times for Galax are not plotted, since version 0.5.0 of the engine reports incorrect query execution times and we configured the engine adapter for Galax not to store these times. Note that these times cannot be compared directly; in this plot only the slopes of the lines can be compared.

Q8, given below, contains a nested for-loop and a data value join.

```xml
let $auction := doc("name") return
  for $p in $auction/site/people/person
  let $a :=
    for $t in $auction/site/closed_auctions/closed_auction
      where $t/buyer/@person = $p/@id
    return $t
  return <item person="{$p/name/text()}">{count($a)}</item>
```

The number of person and closed_auction elements grows linearly with the size of the document. Thus, we can expect the performance of this query to be bound from above by a quadratic function in the size of the document. The lines in Figure 5.12 show a super-linear growth, with SaxonB having the steepest line.

Figure 5.13 shows the detailed performance times (document processing time, query compilation time, query execution time, and total execution time) output by Saxon on all XMark queries and on the document of size 114MB (f=1.024). Note that the document processing time and the query compilation time are constant over all queries; the document processing time dominates the query execution time for all queries but the queries Q8-Q12, thus the total execution time is mostly determined by the document processing time. For the difficult queries, most of the total time is spent on the query execution, which is 1–2 orders of magnitude larger than the document processing time. This plot also shows that the total time can be an inadequate measure for evaluating query execution techniques, when the total time is dominated by the performance of another engine component.

5.3.2 XCheck’s coverage

The current version of XCheck, version 0.2.0, includes adapters for 9 XML query engines listed in Table 5.1: the first column contains the engine’s name and reference, the second column contains the engine’s version, the third column contains the query language that the engine implements, and the last column contains the detailed execution times that the engine outputs.

XCheck powers the execution of all the performance studies presented in this thesis, including the execution and analysis of 5 XQuery benchmarks on
Figure 5.11: Total execution time for XMark queries on a document of size 57MB (f=0.512). The plot presents the CPU time measured in seconds. Missing values: Galax crashes on Q11 and Q12; Qizx/open outputs static type checking errors on Q3, Q11, Q12, and Q18.

Figure 5.12: Query execution time of XMark query Q8 on documents scaling from 1MB (f=0.016) to 114MB (f=1.024) (note that the document size doubles from one document to the next). The plot shows the times measured by the engines: MonetDB/XQuery reports CPU time; Saxon and Qizx/open report wall-clock time.
5.4. Related systems

At the time of development of XCheck, there were two other open source automated testing platforms for evaluating XML query engines, BumbleBee [BumbleBee, 2006] and XSLTMark [XSLTMark, 2006]. 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 an engine’s compliance with the XQuery 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

---

2 As of September 2009, a Google Scholar search shows 9 citations for [Afanasiev et al., 2006], the paper presenting XCheck.
3 As of September 2008, both testing platforms are no longer available.
Chapter 5. XCheck: a Tool for Benchmarking XQuery Engines

<table>
<thead>
<tr>
<th>Engines</th>
<th>Version</th>
<th>Query language</th>
<th>Times</th>
</tr>
</thead>
<tbody>
<tr>
<td>SaxonB [Kay 2009]</td>
<td>8.7</td>
<td>XQuery</td>
<td>D, QC, QE, T</td>
</tr>
<tr>
<td>Galax [Fernández et al. 2006]</td>
<td>0.5.0</td>
<td>XQuery</td>
<td>T</td>
</tr>
<tr>
<td>MonetDB/XQuery [Boncz et al. 2006b]</td>
<td>0.10.3</td>
<td>XQuery</td>
<td>D, QC, QE, S, T</td>
</tr>
<tr>
<td>Qizx/open [Axyana Software 2006]</td>
<td>1.0</td>
<td>XQuery</td>
<td>QE, S, T</td>
</tr>
<tr>
<td>eXist [Meier 2006]</td>
<td>1.0</td>
<td>XQuery</td>
<td>T</td>
</tr>
<tr>
<td>Qexo [Qexo 2006]</td>
<td>1.8.1 alpha</td>
<td>XQuery</td>
<td>T</td>
</tr>
<tr>
<td>Blixem [University of Antwerp 2006]</td>
<td>16 Jun 2005</td>
<td>LiXQuery</td>
<td>T</td>
</tr>
<tr>
<td>XmiTaskForce [Koch 2004]</td>
<td>30 Sep 2004</td>
<td>XPath 1.0</td>
<td>T</td>
</tr>
<tr>
<td>Arb [Koch 2006]</td>
<td>unknown</td>
<td>CoreXPath</td>
<td>D, QE, T</td>
</tr>
</tbody>
</table>

where D= document processing time, QC=query compile time, QE=query execution time, S=serialization/output time and T=total time.

Table 5.1: The list of engines for which XCheck provides adapters.

compliance benchmarking. It comes with a collection of default test cases that are performance oriented. XSLTMark outputs more information about the execution, such as (document) preprocessing time, total execution time, the size of engine input/output, the engine’s throughput in size per second, and the correctness check. Neither Bumblebee nor XSLTMark processes an engine’s output for extracting detailed times or error messages.

In comparison with BumbleBee, XCheck targets the execution of user defined performance tests rather than correctness tests. 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. In comparison with XSLTMark, XCheck is a generic testing platform that targets XML query engines, rather than XSLT engines. On top of this, XCheck is based on a more flexible input/output adapter design than either of the two platforms. This adapter design allows users to customize what information output by the engines the platform is documented. Moreover, XCheck performs a statistical analysis of the data and outputs graphs, facilitating interpretation of the results.

5.5 Summary and conclusion

The question we pursued in this chapter (Question 5.1) is whether a generic (engine and benchmark independent) tool for running performance benchmarks is feasible and what design choices need to be made in order to build it. The realization and success of XCheck is a clear answer to this question.

During the development of XCheck we had to address several issues. First, we had to decide how XCheck should communicate with the tested engines. The command line adapter design that XCheck implements is elegant and easily implementable—many of the XML query engines have a command line interface. Second, we had to decide what atomic measure XCheck should implement. Cur-
rent XQuery benchmarks measure performance times of a set of queries on a set of documents/collections, where the atomic measure is the performance time of processing one query on a document/collection. If the engines provide more detailed performance times, e.g., document processing, query compilation, etc., XCheck also collects these times. Third, we had to decide on how to obtain accurate performance measures. XCheck computes the atomic measure $N + 1$ times and takes the mean and the standard deviation of the last $N$ runs. Finally, we had to decide how to store and present the performance results. XCheck uses XML to store the raw measurement data, and it uses HTML and plots to present it to the user in an easily readable format.

We use XCheck to execute all experimental studies presented in this thesis. In Chapter 3, we use XCheck to execute existing XQuery benchmarks; in Chapters 6 and 7, we use XCheck to execute two micro-benchmarks; and in Chapter 8, we use XCheck to evaluate the performance of a proposed optimization technique for recursion in XQuery. All experiments are conducted on multiple XQuery engines.