XQBench – A XQuery Benchmarking Service

Peter M. Fischer
ETH Zurich
<peter.fischer@inf.ethz.ch>

Abstract

Benchmarks have been a driving factor for acceptance and progress in the relational database area, as they gave researchers and engineers directions on the issues to tackle, and marketers the leverage to sell progress on these issues to customers. For more than two decades, standard benchmarks covering most application area of relational database have existed, with the most prominent example being the TPC suite.

XQuery has not (yet) reached this level of maturity. Benchmarks do exist for particular application scenarios (XMark, TPoX), microbenchmarking (MeMBer, The Michigan Benchmark), but most of the results shown by vendors and academics alike are specific to customers, implementations or usage scenarios, with little attempts to generalize them and compare over a wider range of implementations. In the first half of this decade, there was a short period in which academia showed interest in XQuery benchmarks, leading to series of proposals (XMark, XMach-1, XOO7, XBench). This interest has waned rather quickly, and industry has not really picked up the challenge (TPoX being the sole exception).

Instead of proposing another benchmark workload, we are working to provide a public service to run XQuery benchmarking workloads on a well-defined environment, including an installation of the most commonly used (open-source) XQuery implementation on a stable hardware and OS setting. By doing so, two main goals can be achieved: (1) Workloads can easily be compared on multiple implementations, leading to a broader coverage and applicability (2) A comprehensive collection of workloads can be established, highlighting performance aspects in particular areas and possibly leading to a general benchmark suite.

Currently, nearly all of the features of the benchmarking service are implemented. The service undergoes internal testing to ensure stability and correctness, and also some more work is needed on documentation. We expect the service to become publicly available around the time of the XML Prague workshop. Preliminary result have been established running XMark on Saxon B/HE, MonetDB, eXist, BerkeleyDB XML, Sedna, xQuilla and Zorba; we

also plan to run TPoX, MeMBer and some custom benchmarks until the general release.

The benchmarking service will be available on http://xqbench.org

Keywords: XQuery, Benchmark, Service

1. Benchmarking Service Requirements

The XQuery benchmarking service requires a system on which users can tests a set of queries and documents against a set of XQuery engines. Such as system needs to provide means for managing the queries and documents, specifying experiments, executing them reliably and reproducably and finally presenting and evaluating the results.

2. Data and Metadata Model

2.1. Overall Model

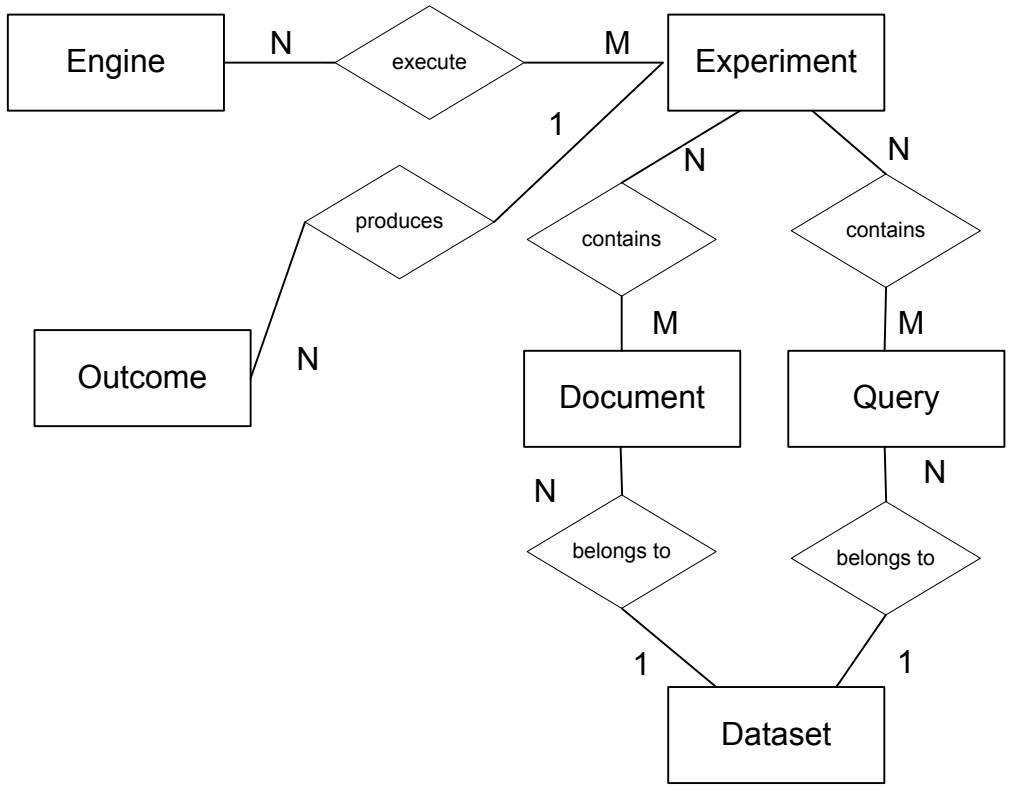

Figure 1. Overall Data Model of the Benchmarking Service

In order to maintain the data sets, specify experiments and evaluate the experiment results, extensive data and metadata is collected. Figure 1 shows the relationships in the data model behind the benchmarking service, based on the data model of XCheck[11]. An experiment contains documents and queries, which are executed on the documents. Documents and Queries belong to a Dataset specifation, (e.g. XMark), so that queries are only executed on matching documents. Experiments are executed on a set of XQuery engines, for every execution an outcome is produced.

In the following, the individual entities are explained on examples, the accompanying schemas are available on the benchmarking service.

2.2. Documents and Data Sets

Example 1. Sample document metadata

```
<documentEntries>
 <document>
   <docId>XMark_10.0</docId>
   <name>XMark 0.1 (1MB)</name>
   <dataset>XMark</dataset>
   <description>XMark , Scale Factor 0.1</description>
   <created>2008-06-23</created>
   <author>XMark</author>
   <generatorInfo>xmlgen.Linux -f 0.1 -o XMark_0.1.xml</generatorInfo>
   <file>XMark_0.1.xml</file>
   <size>10.0MB</size>
  </document>
 <document>
 ...
</documentEntries>
```

For a document, an identifier, a name and a description are saved. In addition, each document carries its dataset identifier, the date when the document was created and its author, the document generator information, the XML file name and the file size.

2.3. Queries

Example 2. Sample query metadata

```
<query>
    <name>XMark query 1</name>
    <id>XMark1</id>
    <descr>Return the name of the person with ID `person0'</descr>
    <dataset>XMark</dataset>
```

```
<created>2008-06-23</created>
<author>XMark</author>
<language>XQuery 1.0</language>
<categories>
    <cat>xpath</cat>
    <cat>flwor</cat>
    <cat>join</cat>
</categories>
<query file='query1.xq'>
    <![CDATA[
    let $auction := doc() return
    for $b in $auction/site/people/person[@id = "person0"]
     return $b/name/text()
    ]]>
    </query>
      <expectedresult document='XMark_0.01'>
        query1.xdm
      </expectedresult>
      </query>
```

For queries, again a name, an identifier and a description are stored, as well as the a creation date and the author. To further describe the properties of the query, the language requirements (XQuery 1.0) and the categorization in terms of the operations can be provided. Finally, the query text (both inline and as file reference) and a expected result (based on a given document) are to be provided.

2.4. Engines

Example 3. Sample engine metadata

```
<engines>
         <engine id="Zorba" type="xquery">
                <name>Zorba</name>
                <version>0.9.8</version>
                <language>XQuery 1.0</language>
                <language>XQuery 1.1</language>
                <language>XQuery Update 1.0</language>
                <language>XQuery Scripting 1.0</language>
                <homepage>http://www.zorba-xquery.com</homepage>
                <description>Zorba is a general purpose XQuery
                processor implementing in C++
                the W3C family of specifications.</description>
                <adapter>bin/CLAdapter.pl -e zorba-0.9.8.xml</adapter>
                <path>/local/zorba-0.9.8/build/bin</path>
                <cpu_time>n</cpu_time>
        </engine>
```

```
        ...
</engines>
```

For every XQuery implementation, there is an entry (with a unique identifier) that provides the name, the version, the supported languages, a home page and a description. To facilitate the actual execution of this "engine", information about the wrapper with the actual invocation information (called "adapter", see Example 7), and the install path in the file system are provided.

2.5. Experiments

Example 4. Sample Experiment Metadata

```
<experiment>
 <name>2010-01-28T12_45_59</name>
  <description>XCheck 2010-01-28T12_45_59</description>
 <engines>
 <engine>Zorba</engine><engine>XQuilla</engine></engines>
 <documents>
    <document id="XMark_10.0">
      <description>XMark , Scale Factor 0.1</description>
      <file>XMark0_1.xml</file>
    </document>
 </documents>
 <queries>
    <query id="q1">
      <description>Return the name of the
         person with ID `person0'</description>
      <filequery engine="all">
          XMark/query1.xq
      </filequery>
    </query>
 </queries>
 <groups>
 <group id="g0"><enginelist refs="all"/>
   <documentlist refs="XMark_10.0"/>
   <querylist refs="q1"/></group>
 </groups>
<customName>PaperExample</customName>
</experiment>
```

An experiment contains the internal name, a description, the list of engines (identified by the engine, see Section 2.4), the list of documents (identified by the docId, see Section 2.2) and the list of queries (identified by the query id, see Section 2.3). In addition to these identifers, explicit descriptions and file paths are given, in order

to make the evaluation more robust and transparent. Since slight variations in the syntax or semantics supported by various engines might require different versions of a query, alternative versions can be given, annotated with the engines on which they should be used. The total set of combinations of datasets, queries and engines can partitioned into groups, which are executed separately. This is useful for graphical evaluations (as to reduce the number of dimensions) and separating workloads containing different datasets.

2.6. Experiment Results

Example 5. Sample experiment result

```
<outcomes>
      <engine id="Zorba">
        <document id="XMark_10.0" size="11669705">
          <query id="q1">
            <result size="17171"></result>
            <doc_processing_time sd="0.000">0.000</doc_processing_time>
            <query_compile_time sd="0.001">0.015</query_compile_time>
            <query_exec_time sd="0.283">36.109</query_exec_time>
            <total_time sd="0.280">36.503</total_time>
          </query>
        </document>
      </engine>
      <engine id="XQilla">
        <document id="XMark_10.0" size="11669705">
          <query id="q1">
            <result size="17607"></result>
            <total_time sd="0.055">16.503</total_time>
          </query>
        </document>
      </engine>
      <info>
        <start>Thu Jan  28 12:46:44 2010</start>
        <finish>Thu Jan  28 13:20:32 2010</finish>
        <duration>2028</duration>
        <iterations>3</iterations>
        <cpu>AMD Opteron(tm) Processor 248</cpu>
        <ram>8240816 kB</ram>
        <system>Linux 2.6.18</system>
        <file_bench>/projects/xqbench/XCheck-0.2.0/experiments/
            2010-01-28T12_45_59/experiment.xml</file_bench>
        <save_results>false</save_results>
      </info>
    </outcomes>
```

Each execution of an experiments generates an outcome, either by successfully completing or by an unplanned termination. On success, the captured information is expressed as XML, HTML and a series of graphical plots. The XML file contains the complete information in a low-level, detailed form, whereas the other formats build on it and show a more human-readable and easy to understand presentation.

Given the execution strategy of XCheck, the results are grouped by engine, and then by document, each identified in the same way as in the experiment description. Document sizes, result sizes and total execution time will always be recorded, more fine-grained information like document loading time, query compilation time or query execution time only if the engine exposed it. In the example above, these times are available for Zorba, but not for XQuilla.

In addition to the measured outcomes, additional information on the execution circumstances are recorded, such as start and time, the number of iterations over which the results have been averaged, CPU, memory and operating system information and the file path of the experiment.

3. Benchmark Service Architecture

3.1. Overall Architecture

Given the long-running nature of many benchmarks, scalability is an import concern for the benchmarking service. As a result, the benchmarking service is split into a management frontend, repositories for both workloads and experiment results and a collection of backend nodes that perform the actual execution of the experiments. Figure 2 shows this architecture and the interactions of the components. The repositories and the frontend are fairly tightly coupled, whereas there is a more loose coupling among the backend nodes and among the backend and the rest of the system:

- Each node contains a (partial) replica of the XML documents and the queries needed for the experiments. This eliminites access delay and contention when accessing the main repository, thus making the experiments more predictable. In addition, changes in the repository (such new documents or modified queries) can be carried out from the frontend immediately. Before a new experiment is running on a backend node, synchronization will to reflect the updates on the repository.
- On submission, experiments are queued until a suitable execution backend becomes available. There are several reasons for that; (1) Since measuring the execution time should be as precise as possible, all competing accesses on a backend should be blocked until the currently running experiment ends, either by completion or timeout. (2) If failures like machine crashes occur, an experiment can easily be restarted, as all the relevant information is in the queue.

- On completion of an experiment (whether successful or not), the results are propagated to the repositoy of results. As a consequence, backend nodes can be removed without losing the experiments run on them. Search operations on the results will also not affect the backend

Before describing each of the components in more detail, the role of monitoring and notification should be clarified: To provide more control and transparency over the execution state of experiments, system performance parameters and service availability are monitored: collectd [9] records performance parameters such CPU or RAM usage, while Nagios [10] is used to check node and process activity.

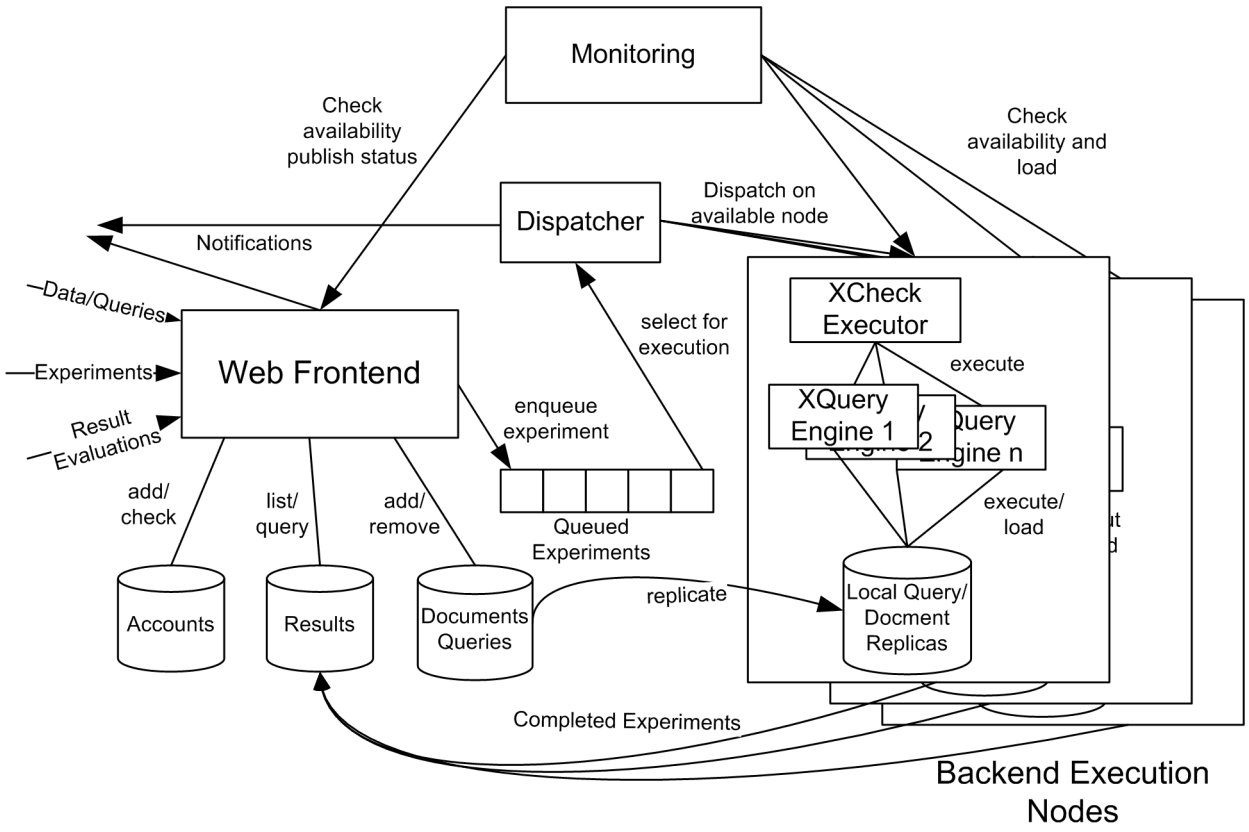

Figure 2. Benchmarking Service Architecture

3.2. Web-Based Frontend

The benchmarking service provides a web frontend to give users a convenient way of interaction. It provides features to

- manage user registrations and permissions
- upload and manage datasets and queries
- submit experiments
- show, search and evaluate the outcomes

A more detailed overview on the interaction and usage is presented in Section 4. Technically, the frontend is realized as a JSP/Servlet invoking an XQuery engine (MXQuery) for the data management and UI rendering. Authentication and user managment are handled by the JSP/Servlet container, storing the account data and user roles in a relational database.

3.3. Data and Results Repository

Since all data, metadata and result have an XML format, they can be stored as such. Queries and expected results may not be in XML format, but have at least a textual representation. To provide easy storage scalability, backup and replication, all data items are stored in the file system, using a folder hierarchy. All modifications, searches and transformations are performed using the file-based XQuery operations of MXQuery; replication and result consolidation using rsync and a set of shell scripts. In the long run, it might be suitable to load metadata as well as experiment and result data into a native XML database, as to benefit from indexing for faster search.

3.4. Workload Distribution and Monitoring

For a small set of backend execution nodes, a simple dispatching approach has been chosen: All pending experiments are written into a shared filesystem. Each backend executor checks these entries, and picks the first experiment that has not been picked up by another node by placing a locking. After some waiting period (as to prevent other node to also put a lock there), it will actually start the execution.

Monitoring utilizes the existing host monitoring infrastructure already in place at ETH. On the backend execution nodes and also the the web frontend node, data collection and status checking processes are run, which deliver their information into the monitoring and notification backends.

3.5. XCheck Experiment Executor

The benchmarking service builds on XCheck [11] as execution backend, which provides the functionality of running a set of queries on multiple implementations, collecting the results and providing graphical plots. XCheck is an open-source Perl script collection developed at the University of Amsterdam. It takes an experiment file as input, and executes the workloads as follows

Example 6. XCheck execution strategy

```
for $engine in ENGINES
   for $doc in DOCUMENT
       for $query in QUERIES
```

```
let $res := (for numrepetitions
  execute($engine,$doc,$query))
return average-values(fn:subsequence($res,2))
```

The averaged values and execution information are then put into an XML files, rendered into HTML with some simpler graphs and also into some more complex, standalone graphs.

By repeating the execution, and leaving out the first run, the effect of a "cold" system is minimized, and more reliable results are provided. In addition to averaging, XCheck provides a number of features to make experiment execution more stable, including timeouts to stop "runaway" experiments. For an XQuery engine to be executed from XCheck it requires a wrapper, called "adapter" that indicates how to execute the engine, how to interpret the results, etc.

Example 7. Example XCheck Adapter for Zorba

```
<adapter>
  <engine id="Zorba">
    <command>
      <executable><![CDATA[zorba -o #result -t #query >& #times]]>
      </executable>
      <file_query>y</file_query>
      <fullpath_doc>y</fullpath_doc>
    </command>
          <times>
            <factor_time>0.001</factor_time>
            <time id="t1">
              <line>Compilation time: (\d+) milliseconds</line>
            </time>
            <time id="t2">
              <line/>
            </time>
            <time id="t3">
              <line>Execution time: (\d+) milliseconds</line>
            </time>
            <doc_processing_time>0</doc_processing_time>
            <query_compile_time>#t1</query_compile_time>
            <query_exec_time>#t3</query_exec_time>
          </times>
      <error>Error on|Fatal error</error>
   </engine>
</adapter>
```

Adapters are again XML files, describing the command line with the parameters, possible setup and teardown operations, pattern matching and scaling factors for

different times, and patterns to detect errors in the output. XCheck will call the command line tool of the XQuery implementation according to the parameters given in the adapter, redirect the output, and parse relevant time and error information.

4. Workflow and Usage Guide

4.1. Public and Private Operations

Datasets and queries for common benchmarks are preinstalled, and benchmark "experiment" results on them are made publicly available. Users can upload their own datasets and queries, run experiments on both their own datasets and queries as well the publicly available ones. By default the data sets, queries and results contributed by users are private, but they can easily be made public, thus becoming usable and searchable by everybody. By making this distinction, it allows users to perform the experiments without disclosing them to other users or the general public; obviously they need to disclose their data and queries to the benchmarking service. From a benchmarking service point of view, knowing the users simplifies the operations, since every uploaded data and query can be traced back to an owner.

4.2. Experiment Creation and Submission

As a first step to test XQuery engines for a set of queries and documents, the user needs to create an experiment. The GUI provides a choice of XQuery engines, datasets with documents and queries. For all of these, the metadata is directly available; queries and document can also be completely downloaded. The user can choose documents, queries and engines, and provide a name for the experiment. In addition, he/she can specify at which stages of the experiment execution notifications should be sent:

- On Submission
- On Execution Start
- On Execution Completion

A registered user can upload his/her own data sets and queries, non-registered users are restricted to only choose the publicly available ones. After the user submits his/her workload, an experiment data set according to Section 2.4 is created, with a timestamp-based identifier in addition to the user-defined name. If the experiment contains multiple datasets (e.g. XMark and TPoX), execution groups for each dataset are created, so that XMark queries are only run on XMark documents, etc.

Since the execution of an experiment is the cross-product of the number of engines, number of documents, number of queries and number of repetitions, a single

experiment could easily run for weeks or months, if specified too broadly. In order not block the execution backend nodes for too long, the experiment is checked for its maximum runtime, based on the cross-product mentioned before and the timeout specified in XCheck for a single query execution. If this maximum runtime exceeds a threshold, the experiment is rejected. This threshold is set higher for registered users, as to provide more freedom for them, whereas "public" experiments should have lower priority. In addition to this limit on a single experiment, there is also a limit on the number of outstanding experiments for each registered users or for all non-registered - again to prevent overloading the service.

4.3. Result Presentation and Search

The execuction of an experiment in XCheck captures the run times of the individual engines (split into document loading, query compilation and query execution, where available) as well as result sizes. These results represented as HTML, XML and in a set of graphs. This result is transferred into the result repository, from which a user can access or view the experiment results. A notification is sent to the user at the end of the experiment, providing a direct link to these results. In addition, a user can view all his/her experiment results, and also can view the public experiments. The GUI lists all experiment and also provides a filter option on this list along with a general XQuery support.

- List of Experiments, completed and outstanding: The GUI lists all the experiments that have been submitted by the user in the past. Hence, an experiment is added to the list of all experiments, after its submission by the user. However, experiment results are only available once the experiment is completely executed and the results have been transferred to the repository.
- The list of all experiments can be filtered by expressing simple constraints using the GUI, which are treated as a conjunction. This selection is based on the user selection of XQuery engines, documents and queries, similar to the way how an experiment is built. If, for example, a user selects the Zorba and XQilla engines, the XMark 10MB document and the XMark 1 query, all those experiments would be listed that contain the Zorba and XQilla engines, the XMark 10MB document and the XMark 1 query. Additional engines, documents and queries are possible, but the listed ones need to be present.
- General query over the experiments and results: Since the simple filter above is not sufficient for more complex evaluations, the benchmarking service provides the possibility to search the experiment descriptions and results using XQuery. An example for such a complex evaluation would be searching for all queries in which Zorba beats XQuilla on one document, but loses on another. Since the experiment description and the results are both in XML format, the GUI provides a method to express full XQuery statements over the collections of experiments

and outcomes (limited to the user's and the public ones). A join between descriptions and outcomes is already provided, so that the query writer can focus on the actually relevant conditions. The results is provided as a direct XML download, which can be further evaluated locally.

4.4. Managing Documents and Queries

The Benchmarking service provides a number of preloaded public datasets and queries, among them XMark and TPoX. However, the more interesting use case is to add custom documents and queries, as to test them over a variety of implementation under controlled circumstances. Several different actions may need to be performed:

- Adding schemas for new XML datasets: For every document uploaded onto the benchmarking service, it is required to provide a data set, which is expressed as an identifier and a schema. The schema helps in checking the uploading XML document conformance with the schema, and also it prevents query belonging to one schema to run on documents having different structure.
- Adding a new document: To add a new document to the benchmarking service, the user needs to upload the document from the GUI, along with providing the details about the XML document, like schema (or dataset), name, description, etc. Only after validation it will be added to the list of documents that can be used for benchmarking. Due to the limitations of current browsers, files bigger than 2GB cannot be reliably uploaded. In such cases, manual administrator intervention is needed. For the long run, it is planned to include a document generation service, e.g. based on ToXGene[8], in which just the document generator input is uploaded.
- Adding new queries: When adding a new query to the benchmarking service, user need to provide a data set to which this query belongs, metadata information like the language features the query requires (update, schema), metadata on the operations used (path steps, joins), expected results, and finally, the actual query.

4.5. Providing Reproducability Information

An important factor in benchmarking is traceability and reproducibility. Several steps are taken to ensure them in XQBench: (1) the installation and configuration logs for all XQuery implementations on XQBench can be downloaded. (2) "Adapter" files to integrate the implementations into XCheck are linked from the implementation descriptions. (3) Data set and queries for particular experiments can be downloaded directly from the results; in the same way error messages. (4) Users are notified on the execution and completion of their benchmark experiment, and can follow the benchmark system load behavior, when required.

5. Preliminary Results and Status

Currently, nearly all of the features of the benchmarking service are implemented. The service undergoes internal testing to ensure stability and correctness, and also some more work is needed on documentation. We expect the service to become publicly available around the time of the XML Prague workshop.

Preliminary result have been established running XMark on Saxon B, MonetDB, eXist, BerkeleyDB XML, Sedna, xQuilla and Zorba. Several commercial XML database have also been tested, but since the licensing terms disallow publishing any results without permissions, we are currently not making these implementations available.

We also plan to run TPoX, MeMBer and some custom benchmarks until the general release.

6. Conclusion and Future Work

XQBench provides a general XQuery benchmarking service, allowing to execute standard and custom workloads over a well-defined set of XQuery implementations on controlled hardware. The results can be searched and downloaded as needed. The goal is to make XQuery benchmarking a much easier undetaking, thereby enabling vendors and users alike to understanding performance tradeoffs and advance the field.

Future work will focus on integrating more XQuery engines, broadening the base of benchmarks, and enciting users to contribute their own workloads. An important conceptual issue is to address the limits of XCheck, as it does not really deal will with the different interaction models of different XQuery implementations: the repeated invocation of a command line tool does not really cater for virtual machine startup cost or just-in-time compilation benefits, neglecting the benefits of repetition to such cases. Similarly, preloading or not preloading XML data into database-style implementations make give a huge performance difference, and needs to be expressed properly when running measurements.

Bibliography

[1] Afanasiev L., Manolescu I., and Michiels P. MemBeR: A Micro-benchmark Repository for XQuery Proceedings of the Third International XML Database Symposium, XSym 2005, Trondheim, Norway, August 28-29, 2005

[2] Böhme T. and Rahm E. XMach-1: a benchmark for XML data management In Proceedings of the German Database Conference BTW2001. Springer, Berlin, 2001, pp. 264–273

[3] Bressan S., Lee M, Li Y., Lacroix Z., and Nambiar U. The XOO7 Benchmark Proceedings of the VLDB 2002 Workshop EEXTT and CAiSE 2002 Workshop

DTWeb on Efficiency and Effectiveness of XML Tools and Techniques and Data Integration over the Web-Revised Papers

[4] Nicola M., Kogan I., and Schiefer B. An XML transaction processing benchmark In Proceedings of the ACM SIGMOD International Conference on Management of Data. ACM, New York, NY, USA, 2007, pp. 937–948

[5] Runapongsa K., Patel J.M., Jagadish H.V., Chen Y., and Al-Khalifa S. The Michigan benchmark: towards XML query performance diagnostics Inf. Syst., 31(2):73–97 (2006)

[6] Schmidt A., Waas F., Kersten M.L., Carey M.J., Manolescu I.,Busse R.: XMark: a benchmark for XML data management In Proceedings of the Very Large Database Conference: 974–985 (2002)

[7] Yao B.B., Özsu M.T., and Khandelwal N.: XBench Benchmark and Performance Testing of XML DBMSs ICDE 2004:621–633

[8] Denilson Barbosa, Alberto Mendelzon, John Keenleyside and Kelly Lyons. ToXgene: A template-based data generator for XML SIGMOD 2002 http://www.cs.toronto.edu/tox/toxgene/

[9] collectd – The system statistics collection daemon http://collectd.org/

[10] Nagios http://www.nagios.org/

[11] XCheck - A Platform for Benchmarking XML Query Processors http://ilps.science.uva.nl/Resources/XCheck