XCheck, a benchmark checker for XML query processors

Enrico Zimuel

zimuel@science.uva.nl, ezimuel@sci.unich.it

Informatics Institute, University of Amsterdam

Dipartimento di Scienze, Universita' "G.D'Annunzio" Chieti - Pescara (Italy)

Contents

- Benchmarking of XML engines
- What's XCheck?
- How to obtain reliable program runtimes
- Query elaboration times
- The architecture of XCheck
- The XML engines already supported
- Running phase
- Data analysis phase
- Examples of benchmarks
- References

Benchmarking

- In computing, a benchmark is a method to assess the relative performance of an object, by running a number of standard tests and trials against it.
- The practice of benchmarking can help vendors, developers, and users to evaluate the performance of an object.
- Moreover, it can help researchers to spot the weaknesses of the current technology and hence to propose improved solutions.
- Benchmarking is particularly important when a technology is young. This is the case of XML.

Benchmarking (2)

- Running a benchmark on different applications takes a lot of time and usually generates huge amounts of raw data.
- Interpreting the outcomes of the evaluation is a subtle and crucially important task.
- Moreover a single execution of an experiment is not sufficient to obtain reliable results. It's often necessary to run the experiment *n* times in order to reduce the standard deviation of the results.

XCheck

- XCheck is an *open source* software for automatic execution of a benchmark on XML query processors (XPath and XQuery processors).
- XCheck works in two phases: running and data analysis.
- In the running phase XCheck executes the benchmark on available XML engines and stores the engine execution times that it measures itself, as well as the execution times that the engines output.
- Optionally, it also stores the benchmark query results.

XCheck (2)

- In the data analysis phase XCheck elaborates some statistics on the execution times gathered in the running phase.
- As a result it generates performance reports (in XML and HTML formats) containing lots of plots (in PostScript and PNG formats).
- 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 comparing the sizes of the query results obtained from different engines.

Technical specification

- XCheck works with a Gnu/Linux operating system. In theory is possibile to use XCheck with another operating systems but we have tested it only on different versions of Gnu/Linux.
- XCheck is written in Perl and you need a Perl interpreter, ver. 5.8.5 or higher, installed on your computer in order to execute it. There are also two CPAN Perl modules you need to install.
- If you want to generate the plots of the benchmark you also need the Gnuplot software ver. 4.0 or higher.
- XCheck is released under the GNU General Public License

Reliable program runtimes

- Measuring reliable program runtimes is not as straightforward as it may appear. Computer's hardware and system software influence the runtime of a program.
- The CPU speed, the amount of main memory, the amount of cache, the operating system and the compiler all play significant roles.
- For Java applications, the Java virtual machine imposes another level of software between the program and the operating system that may alterate the runtimes of the applications under evaluation.

Reliable program runtimes (2)

- 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 bigger is the value of n, the less is the impact on the evaluation time of the warm-up time and of other costs that are external to the query evaluation.
- The value of n can be specified with a command line options of XCheck.

CPU time vs. *wall clock* time

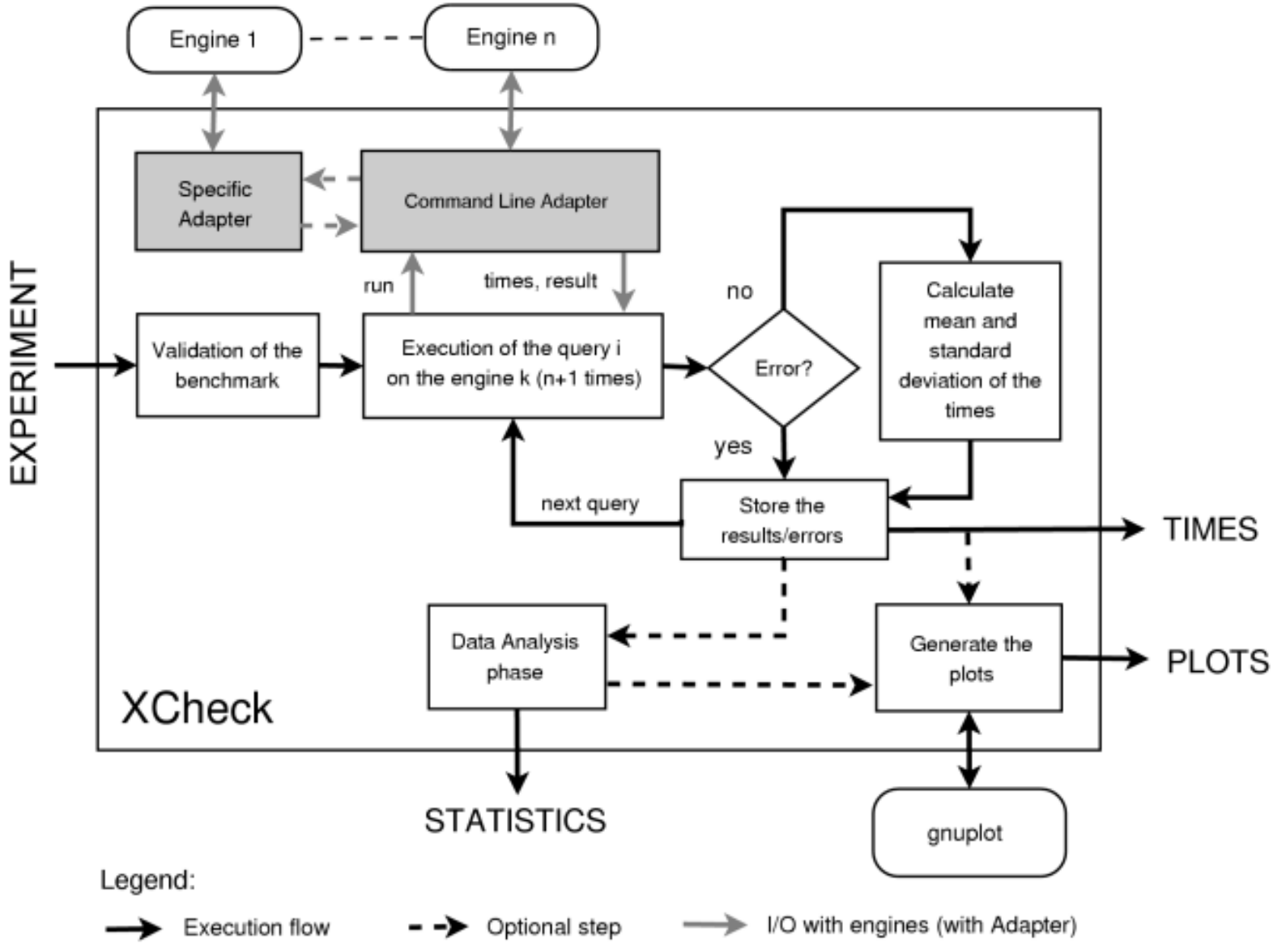
- Besides collecting the times that the engines report, XCheck itself measures the query *total execution time*
- This time is reported in *CPU* time. The *CPU* time (or *processor* time) is the amount of time the processor actually spends running a program.
- The other times are engine dependent and can be CPU time as well as *wall clock* time. The *wall clock* time is the elapsed time between when a process starts to run and when it is finished.
- It is important to remember this fact when comparing the performance of different engines, since, in principle, *CPU* time and *wall clock* time should not be compared.

Query elaboration times

XCheck uses the following types of query elaboration times:

- *document processing time* is the elaboration time of the XML document.
- *query compile time* is the elaboration time for the translation of the query in the internal language or data structure of the engine.
- *query execution time* is the engine elaboration time of the query.
- *serialization/output time* is the engine elaboration time for generate the file containing the output of the query result.
- *total time* is the response time of the engine. This time is always *CPU* time.

The architecture of XCheck



The XML engines already supported

Engines	Version	Type	Times
SaxonB	8.7	XQuery	D,QC,QE,T
Galax	0.5.0	XQuery	T
MonetDB/XQuery	0.10.3	XQuery	D,QC,QE,S,T
Qizx/open	1.0	XQuery	QE,S,T
eXist	1.0	XQuery	T
Qexo	1.8.1 alpha	XQuery	T
Blixem	16 Jun 2005	LiXQuery	T
XmlTaskForce	30 Sep 2004	XPath 1.0	T
Arb	?	CoreXPath	D,QE,T

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

Running phase

- An experiment consists of running a set of XML engines on a set of queries and a set of input XML documents.
- The user can specify the engines, the queries and the documents that participate in the experiment in an xml file, named `experiment.xml`.
- You can execute an experiment in the *running phase* of XCheck with the following command:

  ```
  $ ./XCheck.pl -run exp1
  ```

  where `exp1` is the name of the directory, under the directory `experiments`, where XCheck looks for the experiment specification file `experiment.xml`.

Running phase (2)

- During the running phase XCheck prints on the standard output some informations about the execution of the experiment. For each query XCheck uses the string `(ok)` to indicate an execution witout error and the string `(!?)` to indicate error.
- At the end of execution XCheck produces two files, named `outcome.xml` and `outcome.html`.
- These files are stored in the directory `/experiments/exp1/output/` of XCheck.

Running phase (3)

- As default option XCheck executes each query 4 times and takes the average and the standard deviation of the last 3 runs.
- If all four executions of a query produce errors XCheck stores the error and goes to the next query. If at least one execution gives results XCheck stores them.
- You can change the number of executions with the use of the option `-n num`. For instance, you can execute the queries one time (+1) with the following command:

```
$ ./XCheck.pl -run exp1 -n 1
```

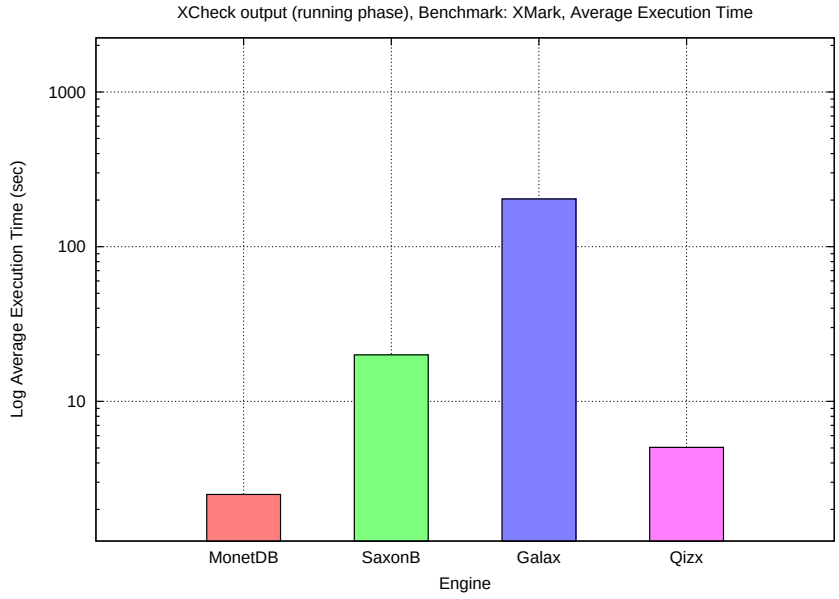
Running phase (4)

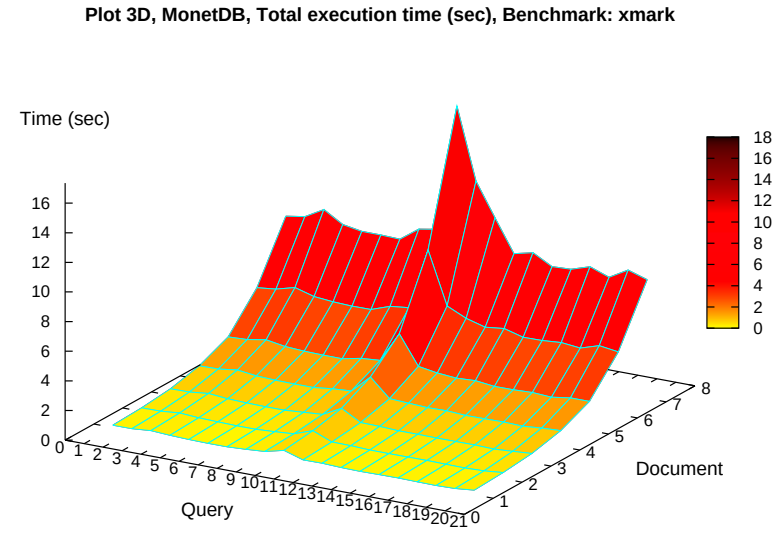
- The use of the `-p` option tells XCheck to generate diverse plots presenting the experiment execution times. For instance:

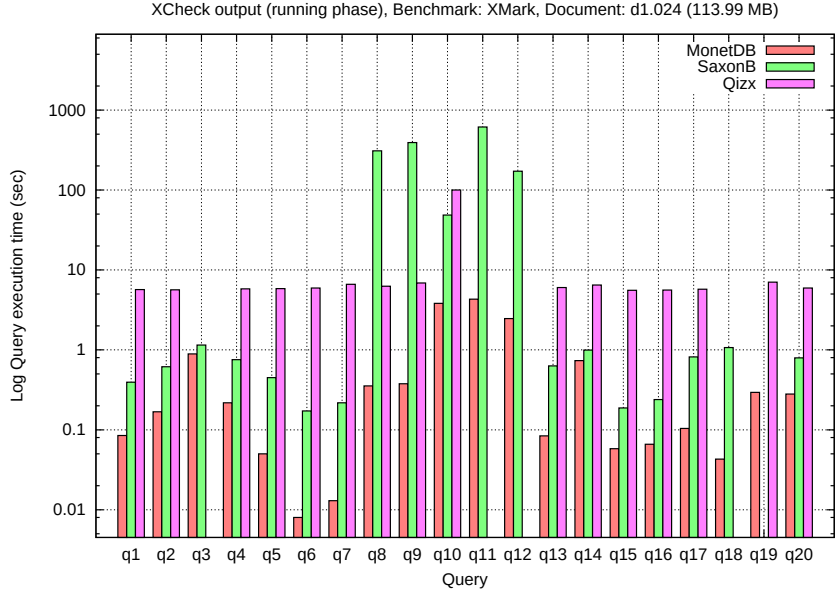
  ```
  $ ./XCheck.pl -run exp1 -p
  ```

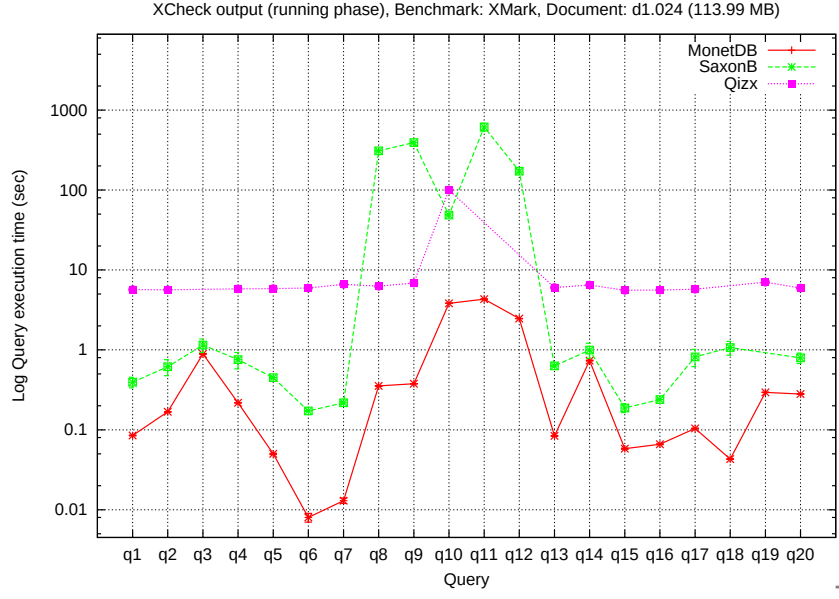
- After the execution of the experiment `exp1` XCheck generates several types of plots and places them in subdirectories of `/experiments/exp1/output/`.
- All the plots are also linked in the main report file `outcome.html`, so that after the execution of the running phase the user is able to see all the files generated by XCheck viewing only `outcome.html`.
- All the HTML files generated by XCheck are in standard W3C HTML 4.01

Example of plots









Running phase (5)

- By default, XCheck doesn't store the query results for the reason of saving disk space. The user can see only the size of the results in bytes in `outcome.html`
- If you want to store the results of the queries you need to use the option `-s` of XCheck, for instance:

```
$ ./XCheck.pl -run exp1 -s
```

- All the query results are stored in the directory `experiments/exp1/output/results`. The result files are also linked in `outcome.html`, in the last table of the report page.

The update option

- After the execution of an experiment if you want to insert new xml documents, queries or engines into the experiment without repeating the whole experiment from the beginning, you can use the `update` option.
- With this option XCheck keeps the old results of the experiment and executes only the new input. In this way the you don’t lose time with the re-execution of the old input.
- In order to execute the `update` option you must use the following command:

  ```
  $ ./XCheck.pl -run exp1 -update
  ```

  where `exp1` is the name of the experiment.

Data analysis phase

- After the execution of an experiment XCheck can help the user to analyze the results with the use of some *aggregation measures*, *statistics* and *plots*.
- The input of the data analysis phase is the output of the running phase, the file `outcome.xml` to be precise.
- XCheck uses the following *aggregation measures*: *sum*, *mean*, *median*, *minimum*, *maximum*, *standard deviation*, *medley relay data speed*, *medley relay query speed*, *data scalability factor*, *query scalability factor*, *similarity index*.

Data analysis phase (2)

- We define the *medley relay data speed* of E_k on the document set A and the query set B, denoted by $\mathrm{mrds}^k_{A,B}$, as follows:

$$\mathrm{mrds}^k_{A,B} = \frac{|B| \cdot \sum_{i \in A} \mathrm{size}_i}{\sum_{i \in A, j \in B} t^k_{i,j}}$$

- We define the *medley relay query speed* of E_k on the document set A and the query set B, denoted by $\mathrm{mrqs}^k_{A,B}$, as follows:

$$\mathrm{mrqs}^k_{A,B} = \frac{|A| \cdot \sum_{i \in B} \mathrm{length}_i}{\sum_{i \in A, j \in B} t^k_{i,j}}$$

Data scalability factor

- Whenever data scalability is a benchmark target, the following definition of *data scalability factor* becomes relevant. Let $I = (i_1, i_2, \ldots, i_k)$, for $k \geq 2$, be a sequence of indexes of documents of increasing sizes. We define the *data scalability factor* of E_k on the document sequence I and the query set B, denoted by $\mathrm{ds}^k_{I,B}$, as follows. If $k = 2$, then

$$\mathrm{ds}^k_{(i_1,i_2),B} = \frac{\mathrm{mrds}^k_{i_1,B}}{\mathrm{mrds}^k_{i_2,B}}$$

If $k > 2$, then

$$\mathrm{ds}^k_{I,B} = \frac{\sum_{j=1}^{k-1} \mathrm{ds}^k_{(i_j,i_{j+1}),B}}{k-1}$$

Query scalability factor

- Similarly, when *query scalability* is a benchmark target, the following definition of query scalability factor becomes relevant. Let $J = (j_1, j_2, \ldots, j_k)$, for $k \geq 2$, be a sequence of indexes of queries of increasing lengths. We define the *query scalability factor* of E_k on the document set A and the query sequence J, denoted by $\mathrm{qs}^k_{A,J}$, as follows. If $k = 2$, then

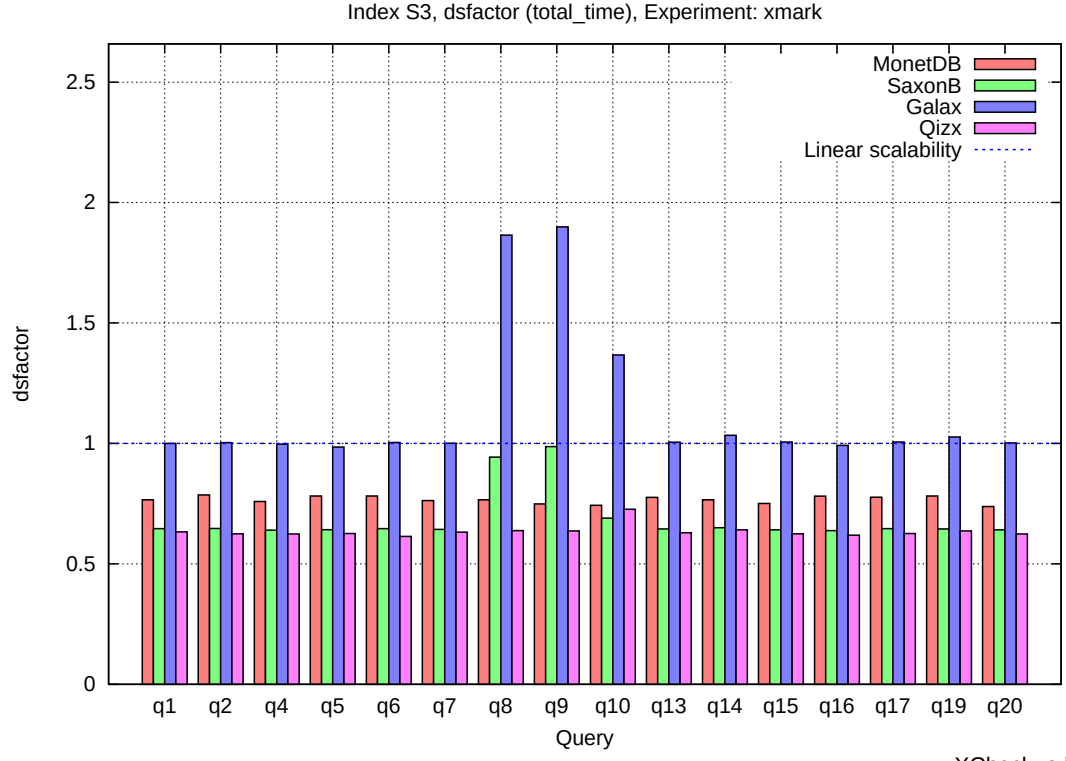
$$\mathrm{qs}^k_{A,(j_1,j_2)} = \frac{\mathrm{mrqs}^k_{A,j_1}}{\mathrm{mrqs}^k_{A,j_2}}$$

If $k > 2$, then

$$\mathrm{qs}^k_{A,J} = \frac{\sum_{i=1}^{k-1} \mathrm{ds}^k_{A,(j_i,j_{i+1})}}{k-1}$$

Scalability factor

- Notice that, by virtue of the definition of medley relay speed, a scalability factor (either for data or query) less than 1 (respectively, equal to 1, bigger than 1) corresponds to a *sub-linear* (respectively, *linear*, *super-linear*) increase of the total evaluation time when moving from one instance of the problem to a bigger one.



Similarity index

- XCheck uses a *similarity index* to compare the similarity of the behaviour of two engines.
- Let $X = (x_1, x_2, \ldots x_n)$ be a sequence containing the results on n experiments. For each $1 \leq i \leq n-1$, let $W_X^i = a_1 a_2 \ldots a_{n-i}$ where, for each $1 \leq j \leq n-i$, we have that $a_j = U$ if $x_i < x_{i+j}$, $a_j = D$ if $x_i > x_{i+j}$, and $a_j = E$ if $x_i = x_{i+j}$.
- Let $W_X = W_X^1 \circ W_X^2 \ldots \circ W_X^{n-1}$ be the *similarity word* associated to X, where $\circ$ is the concatenation operation.

Similarity index (2)

- Given two sequences $X = (x_1, x_2, \ldots x_n)$ and $Y = (y_1, y_2, \ldots y_n)$, the *distance* between X and Y is defined as the word distance between the words W_X and W_Y, where the *word distance* between two words of the same length is the number positions of the two words that contain a different character.
- For instance, the word distance between UUED and EUEU is 2 (the characters in both the first and last position differ in the two words).
- The *similarity index* is defined as the distance between X and Y over n, the length of the sequences. Low values of the index means best similarity (0= best, 1= worse).

Statistics indices

- XCheck generates six types of statistics indices:

S1 The elaboration of the *aggregation measure indices* computed on each engine, query and document.

S2 The elaboration of the *aggregation measure indices* for each document.

S3 The elaboration of the *aggregation measure indices* for each query.

S4 The plots in 3d for each engine with different elaboration times.

S5 The elaboration of the *similarity index* for each document and pair of engines.

S6 The elaboration of the *similarity index* for each query and pair of engines.

Statistics indices (2)

- For each statistics indices XCheck generates two files in XML and HTML formats.
- In the data analysis phase the user can also generate different plots of the elaboration times with different aggregations of the data.
- The user can choose to generate these plots: plots of times for each document (P1), plots of times for each query (P2), plots of times for each engine and document (P3), plots of times for each engine and query (P4).

Data analysis phase (3)

- In order to execute the *data analysis phase* the user must specify the *engines*, the *documents*, the *queries*, the type of *elaboration times* and the *aggregation measures* to be used in the elaboration.
- Moreover the user can specify the type of statistics (S1,...,S6) and plots (P1,...,P4) that should be generated.
- All these informations are reported in the `analysis.xml` file stored in the directory of the experiment.
- In order to execute *data analysis phase* you can use the following syntax:

```
$ ./XCheck.pl -data exp1
```

Examples of benchmarks

- We executed the following benchmarks with the use of XCheck:
- XMark, `http://monetdb.cwi.nl/xml/`
- Michigan, `http://www.eecs.umich.edu/db/mbench/description.html`
- XMach-1,

  `http://dbs.uni-leipzig.de/en/projekte/XML/XmlBenchmarking.html`
- X007, `http://www.comp.nus.edu.sg/~ebh/XOO7.html`
- XBench,

  `http://se.uwaterloo.ca/~ddbms/projects/xbench/index.html`
- All the results of these benchmarks are available on the XCheck web site.

References

- XCheck official web site, `http://www.xcheck.org`
- M. H. Kay, *Saxon. An XSLT and XQuery processor*, `http://saxon.sourceforge.net`
- M. Fernández et al., *Galax. The XQuery implementation for discriminating hackers*, `http://www.galaxquery.org`
- CWI Amsterdam, *MonetDB/XQuery. An XQuery Implementation*, `http://monetdb.cwi.nl/XQuery`
- Axyana software, *Qizx/open. An open-source Java implementation of XQuery*, `http://www.axyana.com/qizxopen`
- Wolfgang Meier, *eXist. Open Source Native XML Database*, `http://exist.sourceforge.net`
- *Qexo - The GNU Kawa implementation of XQuery*, `http://www.gnu.org/software/qexo/`
- University of Antwerp, *Blixem. LiXQuery engine.*, `http://adrem.ua.ac.be/~blixem/`
- *XML TaskForce XPath*, `http://www.xmltaskforce.com`
- Christoph Koch, *Arb*, `http://www.infosys.uni-sb.de/~koch/projects/arb/`