FlexBench: A Flexible XML Query Benchmark*

Maroš Vranec and Irena Mlýnková

Department of Software Engineering, Charles University in Prague, Czech Republic
`maros.vranec@gmail.com,mlynkova@ksi.mff.cuni.cz`

Abstract. In this paper we propose a new approach to XML benchmarking – a flexible XML query benchmark called *FlexBench*. The flexibility is given by two aspects. Firstly, FlexBench involves a large set of testing data characteristics so that a user can precisely describe the application. And, secondly, FlexBench is able to adapt the set of testing query templates to the particular set of synthesized testing data. Hence, contrary to the existing works, the testing is not limited by the fixed set of queries and basic data characteristics (usually only size) to a single (and often simple) application. We depict the advantages of the proposed system using a set of preliminary experiments.

1 Introduction

Since XML [11] has become a de-facto standard for data representation and manipulation, there exists a huge amount of so-called *XML Management Systems* (XMLMSs) that enable one to store and query XML data. Hence, being users, we need to know which of the existing XMLMSs is the most sufficient for our particular application. On the other hand, being vendors, we need to test correctness and performance of our system and to compare its main advantages with competing SW. And, being analysts, we are especially interested in comparison of various aspects of existing systems from different points of view. Consequently the area of benchmarking XMLMSs has opened as well.

In general, a *benchmark* or a *test suite* is a set of testing scenarios or test cases, i.e. data and related operations which enable one to compare versatility, efficiency or behavior of *system(s) under test* (SUT). In our case the set of data involves XML documents, possibly with their XML schema(s), whereas the set of operations can involve any kind of XML-related data operations. Nevertheless, the key operations of an XMLMS are usually XML queries. Currently, there exist several XML query benchmarks which provide a set of testing XML data collections and respective XML operations that are publicly available and well-described. However, although each of the existing benchmarks brings certain interesting ideas, there are still open issues to be solved.

In this paper we focus especially on the key persisting disadvantage of all the existing approaches – the fact that the sets of both data and queries are

* This work was supported in part by the Czech Science Foundation (GAČR), grant number 201/09/P364.

fixed or the only parameter that can be specified is the size or the number of XML documents. We propose a new approach to XML benchmarking – a flexible XML query benchmark called *FlexBench*. The flexibility is given by two aspects. Firstly, FlexBench involves a large set of testing data characteristics so that a user can precisely describe the tested application. But, to ensure user-friendliness and simplicity (i.e. one of the key requirements for a benchmark) we also provide a set of predefined settings which correspond to classical types of XML documents identified in an analysis of real-world XML data. And, secondly, FlexBench is able to adapt the set of testing queries to the particular set of synthesized testing data. Hence, contrary to the existing works, the benchmarking is not limited by the fixed set of queries and basic data characteristics to a single simple application. We depict the advantages of the proposed system using a set of preliminary experiments.

The paper is structured as follows: Section 2 overviews existing XML benchmarks, their classification and (dis)advantages. Section 3 describes FlexBench in detail. Section 4 involves the results of preliminary tests made using FlexBench. And, finally, Section 5 provides conclusions and outlines possible future work and open issues.

2 Related Work

In general, there exists a large set of XML query benchmarks. The seven best known representatives are XMark [13], XOO7 [17], XMach-1 [10], MBench [16], XBench [24], XPathMark [18] and TPoX [21].

From the point of view of purpose we can differentiate so-called *application-level* and *micro* benchmarks. While an application-level benchmark is created to compare and contrast various applications, a micro-benchmark should be used to evaluate performance of a single system in various situations. In the former case the queries are highly different trying to cover all the key situations, whereas in the latter case they can contain subsets of highly similar queries which differentiate, e.g., in selectivity. Most of the seven benchmarks are application-level; the only representative of micro-benchmarks is MBench.

Another set of benchmark characteristics involves the number of users it is intended for, the number of applications it simulates and the number of documents within its data set. Most of the benchmarks are single-user, single-application and involve only a single document. The only exception, XBench, involves (a fixed set of) four classes of XML applications with different requirements. On the other hand, XMach-1 and TPoX are multi-user benchmarks and enable one to test other XML management aspects, such as, e.g., indexing, schema validation, concurrency control, transaction processing, network characteristics, communication costs, etc.

Another important aspect of XML benchmarks are characteristics of the data sets. All the representatives involve a data generator, but in most cases the only parameter that can be specified is the size of the data. Most of the benchmarks involve own simple data generator, some of them (i.e. XBench and TPoX) exploit a more complex data generator, but pre-set most of its parameters. Expectably, all the benchmarks involve also one or more schemas of the data representing the simulated applications.

The last important set of characteristics describes the operation set of the benchmarks. All the benchmarks involve a set of queries, some of them (i.e. XMach-1, MBench and TPoX) also a set of update operations. The two multi-user benchmarks also support additional, less XML-like operations with the data. The most popular operations are XQuery [8] queries, whereas the benchmarks try to cover various aspects of the language, such as, e.g., ordering, casting, wildcard expressions, aggregations, references, constructors, joins, user-defined functions, etc. However, in all the cases the sets of queries are fixed.

3 FlexBench Benchmark

As we have mentioned in the Introduction, the aim of FlexBench is to deal with the problem of fixed parameters of the existing benchmarks. From one point of view it is an advantage since one of the general requirements for benchmarking is simplicity [9]. It is even proven by the analysis of existing XQuery benchmarks [5] that the most popular benchmark is XMark – a single-application and single-user benchmark with a fixed set of queries and a single data characteristic, i.e. size in bytes. On the other hand, such a simple fixed benchmark enables one to test only one specific situation, however according to statistical analysis of real-world XML data [20] there are multiple types of XML data, i.e. multiple applications. And, naturally, new ones occur every day.

Consequently, we state the following two aims of FlexBench:

1. We want to support as much characteristics of the tested application as possible.
2. The benchmark system should still be simple and easy-to-use.

To fulfill the first condition the FlexBench involves three parts – a data generator, a schema generator and a query generator. In general the synthesis of a benchmark can start with any of the three generators. An overview of possible strategies can be seen in Table 2.

All of the existing benchmarks exploit a restricted combination of the second and third approach, where the schema and the queries are fixed and the data are synthesized according to the schema. In FlexBench we use the first approach – the XML documents are synthesized first, then the schemas and, finally, the queries on top of them. In our opinion it is the most user-friendly approach and, in addition, it enables us to exploit XML data characteristics[1] from a statistical analysis of real-world XML data [20]. Not only do their characteristics describe the data from various points of view and very precisely, but the strategy enables

[1] Due to space limitations we will not repeat the definitions of the characteristics. Most of them (such as depth or fan-out) are quite common, whereas the definitions of the less known ones (such as relational or DNA patterns) can be found in [20].

Strategy	**Description**
data → schema & queries	XML documents are synthesized/provided first. Schema and queries are then synthesized on the top of them. It is also possible to synthesize the queries on top of the schema (instead of the XML documents). The schema can be synthesized by an external tool as well, since many suitable ones already exist.
schema → data → queries	The schema is synthesized/provided first. XML documents valid against it are synthesized next (possibly by an external tool). Queries are then synthesized on top of the XML documents or the schema.
queries → schema → data	Queries are synthesized (or provided by a user) first. Then the respective XML documents and their schemas are created on top of them.

Table 1. Three possible strategies when synthesizing a benchmark

us to exploit the particular results for pre-setting FlexBench (see Section 3.4). Since most of these characteristics cover properties of XML documents, not their schemas, mainly because not all XML documents have a schema, when deciding whether to synthesize XML documents on top of their schemas or vice versa, this fact convinced us to synthesize the documents first.

On the other hand, supporting a wide set of data characteristics may lead to some conflicts. There are many dependencies between them and a user may specify contradicting properties. For instance, the size (in bytes) of a document and the number of its elements are correlated and particular values may be in conflict. A related important aspect is that the generator never respects the specified values accurately. For example, the output files usually have a little percentage of bytes higher (or lower) than a user originally specified. However, these deviations have a minimum impact on the quality of synthesized data and it is a common feature of most of the existing data generators.

3.1 Data Generator

FlexBench data generator supports basic data types of its parameters, i.e. data characteristics (e.g. strings, integers, floating point numbers, etc.) as well as advanced ones – discrete statistical distributions. The supported data characteristics are listed in Table 2 which involves their classification, data types and the most important aspect – conflicting parameters.

The first set of so-called *basic* parameters involves the output directory and the amount of documents to be synthesized. Naturally they have no conflict with the other parameters and they can be specified as a constant value.

The set of *structural* parameters describes the structure of the document tree. In particular its depth, fan-out (i.e. a kind of width), number of attributes and total size in bytes. All these characteristics can be specified as a statistical distribution. The important aspects are their conflicts with other parameters.

Type	**Parameter name**	**Conflicts with**	**Type**
Basic	Output directory	None	Constant
	Number of documents	None	Constant
Structural	Size (in bytes)	Number of elements, fan-out, depth, percentage of text, attribute values	Statistical distribution
	Fan-out	Depth, size	Statistical distribution
	Depth	Fan-out, size	Statistical distribution
	Number of attributes	Size, percentage of text	Statistical distribution
Textual	Percentage of text	Size	Constant
	Percentage of mixed-content elements	Percentage of text	Constant
	Depth of mixed-content	Depth	Statistical distribution
	Percentage of simple mixed-content elements	Percentage of text	Constant
Patterns	Percentage of pure recursions	Other recursions	Constant
	Percentage of trivial recursion	Other recursions	Constant
	Percentage of linear recursion	Other recursions	Constant
	Percentage of general recursion	Other recursions	Constant
	Percentage of DNA patterns	None	Constant
	Percentage of relational patterns	None	Constant
	Percentage of shallow relational patterns	None	Constant
Schema	Percentage of DTDs	None	Constant
	Percentage of XSDs	None	Constant

Table 2. Parameters of the benchmark generator and their relations

Naturally, the worst situation is for the size in bytes which is correlated with almost all other characteristics. On the other hand, it is the most common parameter in all the existing benchmarks and it is also the most natural parameter to be specified by a user. Almost the same information would be specified by the number of elements, however, for the previously specified reasons, we have decided to support the total size instead. As for the fan-out and depth which specify the "shape" of the tree, they are correlated mutually as well as with the size. Also note that instead of depth and fan-out, we could use the distribution of levels of the document tree. However, similarly to the previous case, our choice seems to be more natural and user-friendly. Finally, the number of attributes deals with the special type of nodes of the document tree which are correlated only with size and textual parameters.

The third set of data characteristics – the *textual* ones – do not influence the shape of the document tree but the particular textual values. These may occur in three situations: as attribute values, within textual contents of elements or within mixed contents of elements. The most important and natural parameter is undoubtedly the total percentage of text in the XML document which is correlated with size. On the other hand, the remaining parameters specify the number of mixed-content and simple-mixed content elements, as well as the depth of mixed-content elements, i.e. their complexity. Naturally, they are correlated with respective characteristics, i.e. percentage of text or depth of the document.

The fourth set of data characteristics involves various types of data *patterns* as were defined in [20]. Since recursion is related to element names, there are almost no conflicts with other data characteristics for all its four types; they may conflict only mutually. Similarly, since the remaining patterns, i.e. relational patterns, shallow relational patterns and DNA patterns, specify pre-defined subtrees of the synthesized tree, they are not in direct conflict with any of the other parameters as well.

The last set of characteristics involves the percentage of XML documents for which a schema (a DTD [11] or an XSD [7,22]) should be inferred. Typically this will be 0% or 100%, however for special applications, such as those dealing with schema inference or schema evolution, it may be useful to infer a schema only for a subset of the synthesized XML documents. Naturally, these parameters are in no conflict with others.

3.2 Schema Generator

According to the statistical analysis [20], XML schemas of XML documents are used quite often. As long as some XMLMSs use schemas as an information how to create the internal representation of XML documents, FlexBench involves a schema generator as well. However, for the sake of simplicity it involves a third-party implementation, since there exists a plenty of suitable solutions [19]. Hence, since this is not the key aspect of FlexBench, we will omit further details.

3.3 Query Generator

Having synthesized a set of XML documents and having inferred their XML schema if required, the next important component of FlexBench is a query generator. Since, in general, it can have on input any kind of XML data, it must be able to synthesize queries over them, so that we can get reasonable results. Hence, the question is what kind of queries it should synthesize and how.

FlexBench query generator is based on the idea of exploitation of a set of XQuery templates with empty parts that are filled with XML document names and respective element/attribute names according to the given XML data.

For instance, consider the following query template:

```
for $a in doc("input.xml")//elem
order by $a
return <result>{$a}</result>
```

Its instance is created as follows: The `input.xml` is replaced with name of a synthesized XML document and the element name `elem` with any of its elements.

The apparent problem is that we need to tight the data with the queries, i.e. to enable a user to specify which of the elements/attributes should be used in the templates and, hence, queried. Under a closer observation we can see that most of them are directly represented by data generator parameters. For example, if the user tends to synthesize many mixed-content elements, the query generator can "guess" that (s)he wants to synthesize queries involving these elements as well. In general, the user may specify which of the elements should be queried. However, since this approach is useful only in case the user is acquainted with the data in detail, the FlexBench data generator outputs elements with "interesting" features (such as the most common element, the mostly used element in recursion, the mostly used mixed-content element, the mostly used trivial element, etc.) of which a user may choose.

The second question is how to select the query templates. As we have mentioned, the choice of the particular queries depends on the type of the benchmark. Since FlexBench is an application-level benchmark, our aim is to support as many distinct types of queries as possible. In [5] the queries used in the existing benchmarks are divided into several categories as depicted in Table 3.

Category	**MBench**	**XMark**	**XOO7**	**XMach**	**XBench**
Core XPath	12	3	1	0	1
XPath 1.0	4	3	8	3	12
Navigational XPath 2.0	22	5	6	1	22
XPath 2.0	5	8	6	2	23
Sorting	1	1	1	1	9
Recursive functions	2	0	0	1	0
Intermediate results	0	0	0	0	0

Table 3. Categories and numbers of queries in existing benchmarks

In the following sections we will go through all the categories, define them briefly and discuss the respective characteristics. As we have mentioned, the FlexBench query synthesis is based on the idea of query templates and creating their instances suitable for the given data. Hence, the characteristics are related mainly to the amount of synthesized queries. For the space limitations we do not list all the currently supported templates – the whole set can be found in [23].

Core XPath Queries These queries test XMLMS performance when finding an XML element specified by the navigational part of the XPath [14]. The usage of position information, all functions and general comparisons are excluded.

FlexBench distinguishes between two special cases – the queried element at the first level and a nested element. From the point of view of the query generator it is useful to let the user specify how deep the queried element should occur. Hence, FlexBench enables one to specify the number of the queries and statistical distribution of levels of queried elements.

Text Queries Preserving the order of a text is one of the most important aspects of XMLMSs. The text queries test the performance of an XMLXS when searching for a text. They also belong to the category of Core XPath. FlexBench enables one to specify the number of these queries.

XPath 1.0 Queries When storing a document-centric XML document, an XMLMS must store the elements in their original order (otherwise the meaning of the text would be lost). That is why there exist ways to query the ordered access. In general, there are two possibilities:

- *Absolute order* – queries that retrieve elements according to their absolute position in the XML tree
- *Relative order* – queries that return elements according to their neighboring elements in the XML tree

FlexBench enables one to specify the number of absolute and relative order access queries. Besides these obvious parameters, more customizable absolute ordered queries are useful, e.g. a user can specify what index should be used.

Navigational XPath 2.0 Queries From navigational XPath 2.0 [6] the usage of position information and all aggregation and arithmetic functions are excluded. However, we include queries with `some` and `every` clauses to construct quantified expressions as well. A user is able to specify the amount of such queries; `some` and `every` versions of a query are synthesized randomly in rate 50:50.

XPath 2.0 Queries An instance of this category can be queries which apply an aggregation function on the data. Examples of such functions are a sum of some set of numbers, an average value, etc. The aggregation-function queries suppose that there exist similar elements which can be used as a source for the aggregation. FlexBench enables one to specify the number of particular aggregate-function queries.

Sorting Queries These queries use sorting of a set of the given attributes to get an ordered list of elements/attributes. FlexBench enables users to specify their amount.

Queries with Recursive Functions XQuery enables a user to define own functions, possibly recursive. Since such a function is highly related to the data, as a template we have chosen a simple recursive function so generic that it can be used on any XML document. It computes the depth of an XML document.

```
declare function local:depth ( $root as node()? ) as xs:integer?
  {
  if ($root/*)
  then max($root/*/local:depth(.)) + 1
  else 1
  };

local:depth(doc("input.xml"))
```

Similarly to the previous cases, FlexBench enables users to specify the number of synthesized recursive queries.

Queries with Intermediate Results Intermediate result queries contain `let` clause of XQuery. A user can specify the amount of such queries.

Currently, FlexBench supports a set of simple query templates for each of the previously described query class (see [23]). Naturally, there are much more complex constructs and queries that can be expressed in XQuery, such as various joins, complex user-defined functions, etc. As for the future work, we intent to involve them in FlexBench as well and to deal with their more precise binding with the synthesized data. Such complex queries will require looser XQuery templates and a kind of data analyzer capable of creating their correct instances. The current implementation of FlexBench is a preliminary version that enables one to demonstrate the basic advantages of the chosen approach. Hence, the templates were selected so that no complex data analysis was necessary.

3.4 Pre-defined Settings of Parameters

To fulfill the second aim of FlexBench stated at the beginning of this chapter, we provide a set of its predefined settings. Since we have created the set of supported data parameters on the basis of characteristics analyzed in [20], we can exploit the data categories and their real-world characteristics stated in the analysis as well. In particular, the XML data are divided into these six classes:

– *data-centric* documents, i.e. documents designed for database processing (e.g. database exports, lists of employees, lists of IMDb movies, etc.),
– *document-centric* documents, i.e. documents which were designed for human reading (e.g. Shakespeare's plays, XHTML documents, novels in XML, DocBook documents, etc.),
– documents for data *exchange*, e.g. medical information on patients and illnesses, etc.,

- *reports*, i.e. overviews or summaries of data (usually of database type),
- *research* documents, i.e. documents which contain special (scientific or technical) structures (e.g. protein sequences, DNA/RNA structures, etc.) and
- *semantic web* documents, i.e. RDF documents.

A user can use these pre-defined categories through command-line parameters. For instance, specifying:

```
java -jar flexbench.jar -data-exchange
```

is equivalent to specifying all the parameters like:

```
java -jar flexbench.jar NumberOfGeneratedFiles=9
                        PercentageOfTextGen=uniform(31,40)
                        PercentageOfFilesWithDTD=100 ...
```

4 Preliminary Experiments

To depict advantages of the proposal we have chosen basic typical use cases of FlexBench. We describe them and their results in this section.

4.1 Comparing XMLMSs

We start with a typical aim of XML benchmarking – to compare performance of several XMLMSs. We use the six pre-defined sets of parameters specified in Section 3.4 and three simple XMLMSs Qizx [3], Qexo [1] and Saxon [4].

First of all, the total execution times of all benchmark queries are depicted in Table 4.

	Qizx (s)	**Qexo (s)**	**Saxon (s)**
Data-centric	3.268	4.942	11.423
Document-centric	10.564	19.919 (but failed on text queries)	61.767
Exchange	8.116	15.438	18.290
Reports	55.324	failed	failed
Research	3.945	5.050	7.139
Semantic web	7.430	9.874	27.902

Table 4. Total execution time on six different types of applications

As we can see, Qizx and Qexo XMLMSs performed almost equally in data-centric, data exchange, research and semantic web scenarios, whereas Qizx sustained superior performance. Remaining two categories output more interesting results. Both Qexo and Saxon failed in case of reports – the cause of failure was low heap space available. (We will describe the scalability of FlexBench in Section 4.2 to determine maximum XML document size which Qexo can work with.) Saxon has the worst efficiency in general.

Interesting ones are also the document-centric results. It seems that Qizx is much faster than Qexo when querying document-centric documents. Moreover, Qexo does not support text queries because of function usage which it does not support. Note that such experiments could not be performed using any of the existing benchmarks, because they do not support so wide characteristics of the data to specify different applications.

To analyze the document-centric application more precisely we synthesized 55 absolute ordered queries, 70 aggregate function queries, 137 exact match queries, 12 intermediate result queries, 9 quantification queries, 14 recursive function queries, 31 relative order queries, 9 sorting queries and 72 text queries for the document-centric category. As we can see in Table 5, Qexo needs twice the time of Qizx and even fails in some of the cases. None of the categories shows extra deviation from this pattern, so we can conclude that Qexo has overall problems when working with the document-centric documents. On the other hand, Saxon has problems with basic queries (especially in exact match queries), but outperforms Qexo in more complex ones. And, finally, eXist [2] was added to depict how complex recursive function queries are.

Query category	**Number of**	**Average time for query (ms)**			
	of queries	**Saxon**	**Qizx**	**Qexo**	**eXist**
Core XPath (exact match) queries	137	328	25	67	405
XPath 1.0 (absolute and relative order) queries	86	157	26	52	250
Navigational XPath 2.0 (quantification) queries	9	109	64	failed	289
XPath 2.0 (aggregate function) queries	70	55	38	75	301
Sorting queries	9	873	437	failed	274
Recursive function queries	14	failed	failed	failed	1549
Intermediate result queries	12	170	162	234	413

Table 5. Comparing XMLMSs in query categories of document-centric benchmark

4.2 Scalability of Benchmark Generator

FlexBench capabilities can be also used to detect the limits of an XMLMS. As we have seen in Table 4, Qexo has some problems with large files and its default allocated heap space. We will keep its default setting of heap space considering Qexo as an unconfigurable black box and we will try to determine the approximate size of the XML file that Qexo can process successfully. Firstly, we synthesize XML documents with various sizes ranging from 1MB to 13MB.

Then we perform the respective tests. Finally, as we have seen from a log file, Qexo has problems with XML files bigger than 7MB.

Naturally, such testing can be done using any data generator. The advantage of FlexBench is that we can find the limits for distinct applications, i.e. data collections and operations.

4.3 Modifying Parameters

In the following tests we will show the advantage of the various parameters of FlexBench. We can use any of the parameters (or their combination) and selected type(s) of queries to study their impact on the selected XMLMSs. For illustration we choose influence of recursion and text queries on Saxon and Qizx.

Figure 1 shows how the percentage of recursion correlates with corresponding total execution times for text queries. As we can see, in both the selected cases the systems behave quite naturally – the more the percentage of recursion is, the higher the execution times are. However, Saxon overcomes Qizx in all three cases.

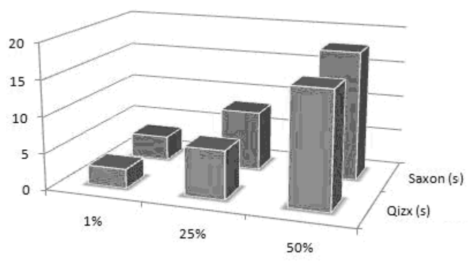

Fig. 1. Effect of percentage of recursion on text queries

4.4 Consistency of FlexBench Output

Last but not least, we will discuss the consistency of the results. One might argue why results over data and queries that are completely synthetic should be trustworthy. How big is the chance that the results will not be entirely different with the same parameters passed to FlexBench again? For the sake of result consistency, multiple tries were made when benchmarking. Each time the whole benchmark was re-synthesized and applied to Qizx. The results can be seen in Table 6.

As we can see, the average total execution time for synthesized document-centric benchmark is 27.413s and the standard deviation is 1.333s. In case of synthesized semantic web benchmark, the average total time is 30.509s and standard deviation is 5.190s. Considering total randomness of synthesized data and queries (even the amount of queries is more-or-less randomized) the results are convincing.

Total execution time	**Qizx**
Document-centric benchmark #1 (s)	26.592
Document-centric benchmark #2 (s)	29.871
Document-centric benchmark #3 (s)	25.948
Document-centric benchmark #4 (s)	27.358
Document-centric benchmark #5 (s)	27.297
Semantic web benchmark #1 (s)	30.117
Semantic web benchmark #2 (s)	27.550
Semantic web benchmark #3 (s)	28.579
Semantic web benchmark #4 (s)	33.579
Semantic web benchmark #5 (s)	32.718

Table 6. Stability of results of a randomly synthesized benchmark

5 Conclusion and Future Work

There are several major achievements of this paper. Firstly, the XML data generator supports numerous interesting parameters of XML documents, in fact so many that no third-party solution was suitable to be exploited and utilized. It is also a notable fact that the parameters were taken from available statistics about real-world XML data sets and, hence, the realistic results could be used for reasonable pre-setting. Secondly, the query generator can synthesize queries so that they can be applied on the given data. Every other XML benchmark has its query workload fixed and consequently, the respective data generators are highly restricted. Thus, in general, the main advantage of our approach is that we are much more flexible when synthesizing data, since we are not bound to any pre-defined queries.

Experimental results showed us how easy is to determine interesting insights about tested XML databases. Thanks to our different kinds of data and various queries we were able to show different behaviors of the benchmarked XMLMSs. This would not be possible with a fixed set of data and XQuery queries. Moreover, we have created pre-defined sets of benchmark parameters corresponding to the real use-cases of XML data.

Authors of [9] state that every benchmark should have the following four basic properties:

– *relevance*: FlexBench synthesizes XML queries that cover most constructs of XQuery.
– *portability*: FlexBench is implemented in Java and outputs portable XML format.
– *scalability*: FlexBench is scalable through lots of data/query parameters.
– *simplicity*: No parameter of FlexBench is mandatory and we provide a set of pre-defined data sets.

Another way to evaluate the quality of an XQuery workload was stated in [12]:

1. Is there a restriction on XML document structure?
2. Is there a database size and load volume scalability?
3. Is there a query type variability?
4. Is there an ad hoc and open interface for schema input and operation input?

FlexBench's answers are:

1. No, there are no restrictions. XML documents and their schemas are synthesized and a user can specify numerous parameters of the result.
2. Yes, there is. Moreover, every other parameter of synthesized XML documents is scalable as well (not only the size parameter).
3. Yes, there is. FlexBench supports more categories of queries than the rest of benchmarks.
4. If we consider FlexBench parameters as a form of a schema, then yes, FlexBench is also easily extensible by new parameters and new templates for synthesized queries.

Naturally, there is also a plenty of possible future improvements of FlexBench. Firstly, we intent to perform more elaborate experiments with various types of XMLMSs from XML-enabled to native XML ones and, especially, with commercial solutions. Secondly, we plan to extend the proposed idea as much as possible. In particular we will focus on the set of XML query templates that can be much wider and enable one to test various aspects of an XML application. As we have mentioned, this task opens a wide research area for creating reasonable instances of the templates. At the same time, we need to tighten the relation between the synthesized data and the queries so that the user can specify more precisely which items should be queried and how. Side but still important tasks involve implementation of a user-friendly interface for specifying the characteristics as well as a repository for their predefined settings. And, last but not least, we want to focus on benchmarking of stream processing [15], where the various characteristics of synthetic data can be widely exploited as well.

References

1. *Qexo – The GNU Kawa implementation of XQuery.* Kawa, 2007. `http://www.gnu.org/software/qexo/`.
2. *eXist-db: Open Source Native XML Database.* exist-db.org, 2008. `http://exist.sourceforge.net/`.
3. *Qizx/db.* Pixware, 2008. `http://www.xmlmind.com/qizx/`.
4. *Saxon: The XSLT and XQuery Processor.* SourceForge.net, 2008. `http://saxon.sourceforge.net/`.
5. L. Afanasiev and M. Marx. An Analysis of the Current XQuery Benchmarks. In *ExpDB'06: Proc. of the 1st Int. Workshop on Performance and Evaluation of Data Management Systems*, pages 9–20, Chicago, Illinois, USA, 2006. ACM.
6. A. Berglund, S. Boag, D. Chamberlin, M. F. Fernndez, M. Kay, J. Robie, and J. Simeon. *XML Path Language (XPath) 2.0.* W3C, January 2007.

7. P. V. Biron and A. Malhotra. *XML Schema Part 2: Datatypes (Second Edition).* W3C, October 2004.
8. S. Boag, D. Chamberlin, M. F. Fernndez, D. Florescu, J. Robie, and J. Simeon. *XQuery 1.0: An XML Query Language.* W3C, January 2007.
9. T. Bohme and E. Rahm. Benchmarking XML Database Systems – First Experiences. In *HPTS'01: Proc. of 9th Int. Workshop on High Performance Transaction Systems*, Pacific Grove, California, 2001.
10. T. Bohme and E. Rahm. *XMach-1: A Benchmark for XML Data Management.* Database Group Leipzig, 2002. `http://dbs.uni-leipzig.de/en/projekte/XML/XmlBenchmarking.html`.
11. T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. *Extensible Markup Language (XML) 1.0 (Fourth Edition).* W3C, September 2006.
12. S. Bressan, M.-L. Lee, Y. G. Li, Z. Lacroix, and U. Nambiar. The XOO7 Benchmark. In *Proc. of the VLDB'02 Workshop EEXTT and CAiSE'02 Workshop DTWeb – Revised Papers*, pages 146–147, London, UK, 2003. Springer-Verlag.
13. R. Busse, M. Carey, D. Florescu, M. Kersten, I. Manolescu, A. Schmidt, and F. Waas. *XMark – An XML Benchmark Project.* Centrum voor Wiskunde en Informatica (CWI), Amsterdam, 2003. `http://www.xml-benchmark.org/`.
14. J. Clark and S. DeRose. *XML Path Language (XPath) Version 1.0.* W3C, November 1999.
15. J. Dvorakova and F. Zavoral. Using Input Buffers for Streaming XSLT Processing. In *GlobeNet/DB'09: Proc. of the 1st Int. Conf. on Advances in Databases*, Guadeloupe, French Caribbean, 2009. IEEE.
16. K. Runapongsa et al. *The Michigan Benchmark.* Department of Electrical Engineering and Computer Science, The University of Michigan, 2006. `http://www.eecs.umich.edu/db/mbench/`.
17. S. Bressan et al. *The XOO7 Benchmark.* 2002. `http://www.comp.nus.edu.sg/~ebh/XOO7.html`.
18. M. Franceschet. *XPathMark.* University of Udine, Italy, 2005. `http://users.dimi.uniud.it/~massimo.franceschet/xpathmark/`.
19. I. Mlynkova. An Analysis of Approaches to XML Schema Inference. In *SITIS'08: Proc. of the 4th Int. Conf. on Signal-Image Technology and Internet-Based Systems*, Bali, Indonesia, 2008. IEEE.
20. I. Mlynkova, K. Toman, and J. Pokorny. Statistical Analysis of Real XML Data Collections. In *COMAD'06: Proc. of the 13th Int. Conf. on Management of Data*, pages 20–31, New Delhi, India, 2006. Tata McGraw-Hill Publishing Ltd.
21. M. Nicola, I. Kogan, R. Raghu, A. Gonzalez, M. Liu, B. Schiefer, and G. Xie. *Transaction Processing over XML (TPoX).* `http://tpox.sourceforge.net/`.
22. H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. *XML Schema Part 1: Structures (Second Edition).* W3C, October 2004.
23. M. Vranec and I. Mlynkova. *FlexBench: A Flexible XML Query Benchmark.* September 2008. `http://urtax.ms.mff.cuni.cz/~vranm3bm/dp/flexbench/`.
24. B. B. Yao and M. T. Ozsu. *XBench – A Family of Benchmarks for XML DBMSs.* University of Waterloo, School of Computer Science, Database Research Group, 2003. `http://se.uwaterloo.ca/~ddbms/projects/xbench/`.