Bachelorarbeit

zum Erwerb des akademischen Abschlusses

Bachelor of Science in Business Informatics

zum Thema

Benchmarking the DB2 XML and Text Extender

eingereicht an der	Wirtschafts- und Sozialwissenschaftlichen Fakultät der Universität Rostock
vorgelegt von:	Tim F. Rieger
Matrikel-Nr.:	099202731
Studiengang:	Business Informatics
Bearbeitungszeitraum:	3 Monate
Betreuer:	Dr. Holger Meyer
2. Gutachter:	Prof. Dr. Andreas Heuer
Lehrstuhl:	Datenbank- und Informationssysteme

Bremen, den 04.10.2002

Contents

List of Abbreviations

BAPCo - *B*usiness *A*pplication *P*erformance *Co*uncil

DBS - *D*ata*b*ase *S*ystem

CLOB - *C*haracter *L*arge *Ob*ject

DBMS - *D*ata*b*ase *M*anagement *S*ystem

DAD - *D*ocument *A*ccess *D*efinition

DTD - *D*ocument *T*ype *D*efinition

HTTP - *H*yper *T*ext *T*ransfer *P*rotocol

LAN - *L*ocal *A*rea *N*etwork

OO - *o*bject-*o*riented

OODBS - *o*bject-*o*riented *d*atabase *s*ystems

OS - *O*perating *S*ystem

PC - *P*ersonal *C*omputer

RNG - *R*andom *N*umber *G*enerator

RDBS - *R*elational *D*atabase *S*ystem

SPEC - *S*tandard *P*erformance *E*valuation *C*orporation

SUT - *S*ystem *u*nder *T*est

TPC - *T*ransaction *P*rocessing *P*erformance *C*ouncil

UDF - *U*ser *D*efined *F*unction

W3C - *W*orld *W*ide *W*eb *C*onsortium

XMach-1 - *X*ML Data *Ma*nagement ben*ch*mark, version *1*

List of Abbreviations

Xmark	- *X*ML Bench*mark*
XML	- E*x*tensible *M*arkup *L*anguage
XMLMS	- *XML M*anagement *S*ystem
Xqps	- *X*ML *q*ueries *p*er *s*econd

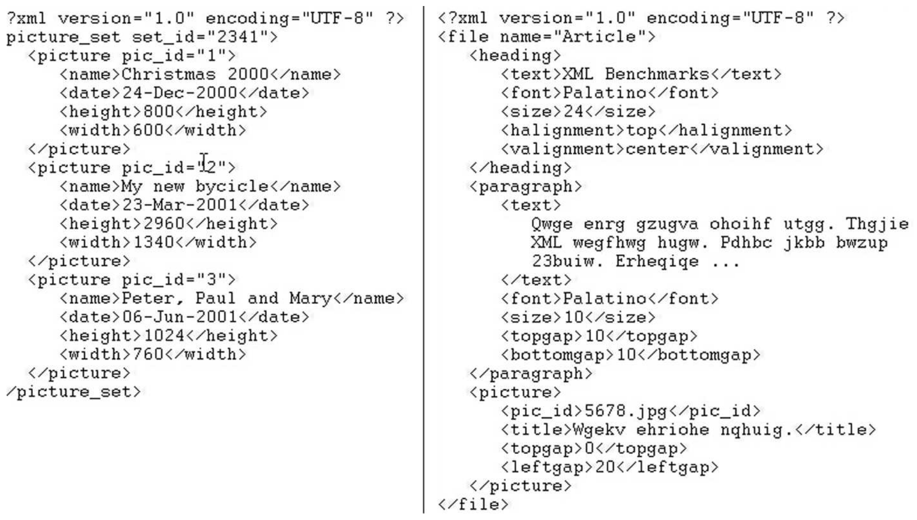
1 Introduction

XML is a document definition language that is currently experiencing a boom. Because of its self-describing nature, it is especially popular as a data exchange format in web-environments. Applications of XML pop up everywhere and more and more XML data is generated. This leads to the demand for efficient storage means. About every database vendor has a product to satisfy this need.

In this thesis the difference between XML data and conventional relational data is explained which is the reason why special management systems for XML are needed. Then the necessity for XML benchmarks is explained and criteria for judging benchmarks are introduced. Currently several different benchmarks for XML database systems are under development. This thesis has a closer look at three XML benchmarks and discusses their particularities and suitability for benchmarking XML management systems. A report on the experiences made when trying to use one of the benchmarks to test IBM's Universal Database DB2 complements this paper.

2 Particularities of XML Data

Basically only two categories of XML documents exist: data-centric and document-centric ones. Data-centric documents hold all of their information in their tags. Therefore these can easily be converted to conventional relational databases by mapping their elements to attributes in tables. Examples of data-centric XML documents are addressbooks, picture albums or music catalogs. The order inside these documents is unimportant. Document-centric XML files also carry information in their structure; the order and nesting of elements is very important. It is difficult to map these documents to (object-) relational database systems because simply copying the data is not enough. Examples of document-centric XML documents are web-pages[1] or XML based file formats like the ones of the OpenOffice suite. Figure 2.1 illustrates the difference of data-centric and document-centric XML documents with examples for a picture album and a file format for word processors.



```
?xml version="1.0" encoding="UTF-8" ?>
picture_set set_id="2341">
   <picture pic_id="1">
      <name>Christmas 2000</name>
      <date>24-Dec-2000</date>
      <height>800</height>
      <width>600</width>
   </picture>
   <picture pic_id="2">
      <name>My new bycicle</name>
      <date>23-Mar-2001</date>
      <height>2960</height>
      <width>1340</width>
   </picture>
   <picture pic_id="3">
      <name>Peter, Paul and Mary</name>
      <date>06-Jun-2001</date>
      <height>1024</height>
      <width>760</width>
   </picture>
/picture_set>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<file name="Article">
   <heading>
      <text>XML Benchmarks</text>
      <font>Palatino</font>
      <size>24</size>
      <halignment>top</halignment>
      <valignment>center</valignment>
   </heading>
   <paragraph>
      <text>
         Qwge enrg gzugva ohoihf utgg. Thgjie
         XML wegfhwg hugw. Pdhbc jkbb bwzup
         23buiw. Erheqiqe ...
      </text>
      <font>Palatino</font>
      <size>10</size>
      <topgap>10</topgap>
      <bottomgap>10</bottomgap>
   </paragraph>
   <picture>
      <pic_id>5678.jpg</pic_id>
      <title>Wgekv ehriohe nqhuig.</title>
      <topgap>0</topgap>
      <leftgap>20</leftgap>
   </picture>
</file>
```

Figure 2.1: Examples of data-centric and document-centric XML files.

[1]HTML and XHTML have been defined in XML.

As different kinds of XML documents exist, also different approaches for XML management systems (XMLMS) have emerged. Native XML database systems take a straight-forward approach and do not try to convert the data into relational data. Instead they have their own method of storing XML files, so they do not have a problem with document-centric XML documents. However they cannot utilize conventional optimization techniques due to their different storage method. XML-enabled or -aware database systems build up on top of a (object-) relational database systems. They map XML data to their tables. Because they can use their internal optimization, good results can be achieved when querying data-centric XML data. However, these systems usually perform poor when confronted with document-centric XML data as it is unknown to their data model.

3 Evaluation of Current XML Benchmarks

With the increasing amount of XML data being generated the necessity of efficient storage systems increases. More and more XML Management Systems (XMLMS) are pushed into the market by various software companies. Also, a lot of academic research is done on different ways to manage XML documents.

Benchmarks are the only means to measure the performance of these systems. They help the developer to judge if the last change in source code resulted in better performance or did the contrary. Customers can judge for themselves if the XMLMS they are planning can stick up to their demands. By allowing direct comparison between different systems, benchmarks induce constant improvement of the available systems and competition between different developers for the benefit of the customer.

However, since the design of the first benchmarks there has been argument about their independence and relevance for practical use. Early benchmarks lacked acceptance because they were tailored by vendors for their own machines resp. software. Therefore neutral consortia[1] were founded to develop standard benchmarks, some of these are the Transaction Processing Performance Council (TCP), the Standard Performance Evaluation Corporation (SPEC) or the Business Application Performance Consortium (BAPCo).

A benchmark must be domain-specific to give relevant results. SPEC, for example, develops benchmarks for the workstation domain whereas BAPCo defines benchmarks for personal computers (PC) which simulate a typical office environment. A university will not care about the BAPCo results when buying a new mainframe, because that is not the domain in which the system will be used. In his book [GRA], Gray has defined four criteria for domain-specific benchmarks to be useful:

1. **Relevance:** It must measure the performance of typical operations for the specific domain. Performance should be distinguished between peak performance and the price/performance ratio.

[1]Of course, these were founded and are operated by vendors.

2. **Portability:** Implementation should be possible on different systems and architectures.

3. **Scalability:** The benchmark should be equally applicable to small and large computer systems.

4. **Simplicity:** To be accepted, the benchmark needs to be comprehensible.

In the next sections three current XML benchmarks will be introduced. First there will be an introduction on the idea behind the benchmark and its developers, followed by deeper look at the benchmarks structure and available implementations. Each section will conclude with a look at experiences with the concerning benchmark and a discussion of its correspondence with Gray's criteria.

It has become necessary to develop XML benchmarks because the established benchmarks do not cover all of the features of XML or of its query language proposals (e.g. document-centric data, implicit ordering of XML data, creation of XML structures, . . .).

3.1 Xmark

Xmark stands for "*XML* Bench*mark*" and has been developed at the CWI [SCH02b], which is the Research Institute for Mathematics and Computer Science in the Netherlands. The reason for its development was that none of the existing benchmarks cover all of XML's potential, as its authors Schmidt et al. believe. The main sources of information on the benchmark were the publications by its authors [SCH02a, SCH01a, SCH01b] and the project's homepage [SCH02b].

3.1.1 Concept

The scenario that Xmark proposes models an Internet auction site. Still the authors stress that by carefully selecting different their queries the benchmark is not only relevant for XMLMS in this domain. The XML data on which the benchmark works is contained in a single document. A tree representation of most elements can be seen in figure 3.1. It is a straight-forward approach and grasps all of the necessary entities one would expect for an auction site like items to be sold, open and closed auctions, people (buyers and sellers), product categories etc. The only unusual thing is probably their arrangement in a XML-typical tree structure and not in an entity relationship model. As all of the data will be contained in a single document, 'site' will be the root element for the whole database.

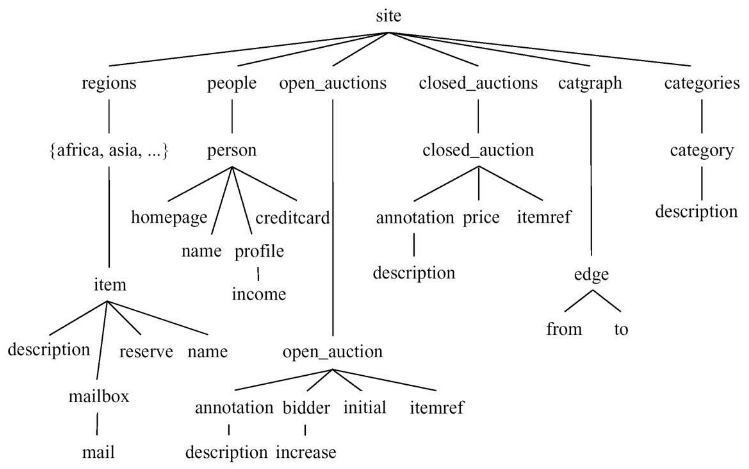


Figure 3.1: Element relationship between most elements of Xmark [SCH01b]

To generate realistic text for the database, Xmark uses the 17,000 most common words in the plays of Shakespeare. A document type definition (DTD) file is supplied for XMLMS that can utilize this information. In order to support different database sizes, a scaling factor is introduced. A scaling factor of 1.0 will result in a database of 100 MB; factors of 0.1, 10 and 100 would result in a file size of 10 MB, 1 GB and 10 GB respectively. Xmark neither measures the time it takes the database to load the XML document nor does it consider data manipulation operations. The chosen is metric milliseconds per query.

3.1.2 Benchmark Operations

The queries are formulated in XQuery [XQE]. Altogether there are 20 different queries which the authors considered necessary to create a benchmark that is relevant to many different fields of application. Tables 3.2 and 3.4 list all of the queries along with comments on their intention.

3.1.3 Implementation and Experiences

By the time of writing, the data generation tool for the benchmark was available on the benchmark's homepage [SCH02c]. It is written in C, so it can be compiled and used on every system with a C compiler. To make it platform-independent it does not rely on supplied random number generators (RNG) but instead comes with its own RNG routine. The queries are available for download, too, but they are only useful if the tested system is already capable of executing XQuery code. What is left to do, is writing the necessary routines to measure the execution time of queries. Since these are different for every XMLMS, this will be left to the benchmark user.

So far, Xmark has been used to benchmark Monet XML [MON] which is a research DBS at the CWI.

3.1.4 Rating of Benchmark

Although it appears as if Xmark was only suitable to benchmark Internet auction sites, Schmidt et al. tried to achieve relevance for many domains by including relatively many queries. The clear description of the queries targeted features helps users to pick out the results which are important for their specific demands. People who use XML data in multi-user environments should keep in mind that the benchmark only support single-user mode.

Portability is achieved by supplying C code that brings it out RNG and formulating queries verbally and in XQuery which will probably become the standard XML query language.

ID	Description	Comment
Q1	Return the name of the person with ID 'person0' registered in North America.	Checking ability to handle strings with a fully specified path.
Q2	Return the initial increases of all open auctions.	Evaluate cost of array lookups. Query on the order of data. A relational backend may have a problem determining the first element.
Q3	Return IDs of all open auctions.	More complex evaluation of array lookup.
Q4	List reserves of those open auctions where a certain person issued a bid before another person.	Querying tag values capturing document orientation of XML.
Q5	How many sold items cost more than 40?	Check how well a DBMS performs since XML model is document oriented. Checks for typing in XML.
Q6	How many items are listed on all continents?	Test efficiency in handling path expressions.
Q7	How many pieces of prose are in our database?	Query is answerable using cardinality of relations. Testing implementations.
Q8	List the names of persons and the number of items they bought.	Check efficiency in processing IDREF's. (Note that a relational system would handle this using foreign keys.)
Q9	List the names of persons and the names of items they bought in Europe. (Joins person, closed_auction and item.)	Same as Q8.
Q10	List all persons according to their interest. Use French markup in the result.	Grouping, restrcuturing and rewriting. Storage efficiency checked.

Table 3.2: Xmark query operations (Pt. 1) [BRE01c, SCH01b]

ID	Description	Comment
Q11	For each person, list the number of items for sale whose price does not exceed 0.02% of the person's income.	Value based joins.
Q12	For a richer than average person, list the number of items currently on sale whose price does not exceed 0.02% of the person's income.	Same as above.
Q13	List names of items registered in Australia along with their description.	Test ability of database to reconstruct portions of XML documents.
Q14	Return the names of all items whose description contains the word 'gold'.	Text search narrowed by combining the query on content and structure.
Q15	Print the keyword in emphasis in annotations of closed auctions.	Attempt to quantify the cost of long path traversals. Query checks for existence of path.
Q16	Return the IDs of those auctions that have one or more keywords in emphasis.	Same as above.
Q17	Which persons do not have a homepage?	Determine processing quality in presence of optional parameters.
Q18	Convert the currency of the reserve of all open auctions to another currency.	Checks User Defined Functions (UDF's).
Q19	Give an alphabetically ordered list of all items along with their location.	Query uses SORTBY, which might lead to a SQL-ish ORDER BY and GROUP BY because of lack of schema. This query requires a sort on generic data.
Q20	Group customers by their income and output the cardinality of each group.	Computes a simple aggregation by assigning each person to a category. The aggregation is semistructured because income information is not mandatory.

Table 3.4: Xmark query operations (Pt. 2) [BRE01c, SCH01b]

Due to the scaling factor, Xmark can create and benchmark databases of different sizes. Hence it fulfills Gray's scalability requirement.

Judging whether this benchmark is simple is not easy. On the one side it possesses 20 queries which not little but on the other side these queries are very comprehensible due to the vividness of the scenario. Therefore it can be stated that Xmark is simple enough for users who are acquainted with the matter.

Apart from the little flaw of the missing multi-user simulation which is important to many users, the benchmark can very well be regarded as useful according to Gray's criteria.

3.2 XOO7

This benchmark has been developed by researchers from the National University of Singapore and Arizona State University. It is based on the OO7 benchmark [CAR] because its developers believed that the XML data model has a lot of similarity to the object-oriented model. The OO7 benchmark was developed for benchmarking object-oriented databases (hence the "OO" in the name) simulating a technical documentation. Taking an already existing database benchmark was only a first step. Some adaptions had to be made to meet the demands of an XML benchmark. Most of the information on XOO7 stems from the publications by its authors [BRE02, BRE01a, BRE01b, BRE01c, BRE01d, BRExxb] and the project's homepage [BRExxa].

3.2.1 Concept

Bressan et al. - the developers of XOO7 - are of the opinion that the object-oriented model (OO-model) and XML have much similarity. Classes (or schemata) and instances from the OO-model corresponded directly to DTD's and XML data in the XML model. Therefore it was decided to take a successful benchmark for object-oriented database systems (OODBS) and adapt it, so that can be used for XMLMS. The adaption was accomplished by mapping the OO7 schema and instances onto a DTD and corresponding XML data sets.

Figure 3.2 illustrates the corresponding DTD. Other than XMach-1 that was designed to model a multi-user web-environment, XOO7's developers favor a benchmark for comparing querying capabilities. Multi-user support is not regarded, instead the focus is on query processing power. Querying does not include data manipulation which means that XOO7 is not concerned with XML data insertion, update or deletion. Besides the response time for queries, the benchmark also regards the time it takes to load the benchmarks data sets and the disk space they consume in the different XMLMS.

3.2.2 Benchmark Operations

The OO7 benchmark included eight queries. These were included in XOO7 and complemented by five additional queries, as can be seen in figure 3.6. Group I covers simple selection/projection queries. Queries in group II require explicit order to generate results and group III needs joins to get ordered results. The additional queries cover all three groups.

ID	Description
Group I	
Q1	Randomly generate five numbers in the range of AtomicPart's MyID. Then return the AtomicPart's MyIDs according to the five numbers.
Q4	Randomly generate five titles for Documents. Then return Document's MyIDs by lookup on these titles.
Group II	
Q2	Select 1% of the latest AtomicParts via buildDate and return the MyIDs.
Q3	Select 10% of the latest AtomicParts via buildDate and return the MyIDs.
Q7	Select all of the AtomicParts and return the MyIDs.
Group III	
Q5	Find the MyID of a CompositePart if it is more recent than the BaseAssembly it is using.
Q6	Find the MyID of a CompositePart (recursively) if it is more recent than the BaseAssembly or the ComplexAssembly it uses.
Q8	Join AtomicParts and Documents on the docId of AtomicPart and the MyID of Document.
Additional	
Q11	Select all BaseAssemblies from one XML database where it has the same MyID and type attributes as the other BaseAssemblies but with later buildDate.
Q9	Randomly generate two phrases from all phrases in Documents. Select the documents containing these two phrases.
Q10	Repeat Q1 but replace duplicated elements using IDREF.
Q12	Select all AtomicParts with corresponding CompositeParts as their sub-elements.
Q13	Select all ComplexAssemblies with type "type008".

Table 3.6: XOO7 query operations [BRE01c]

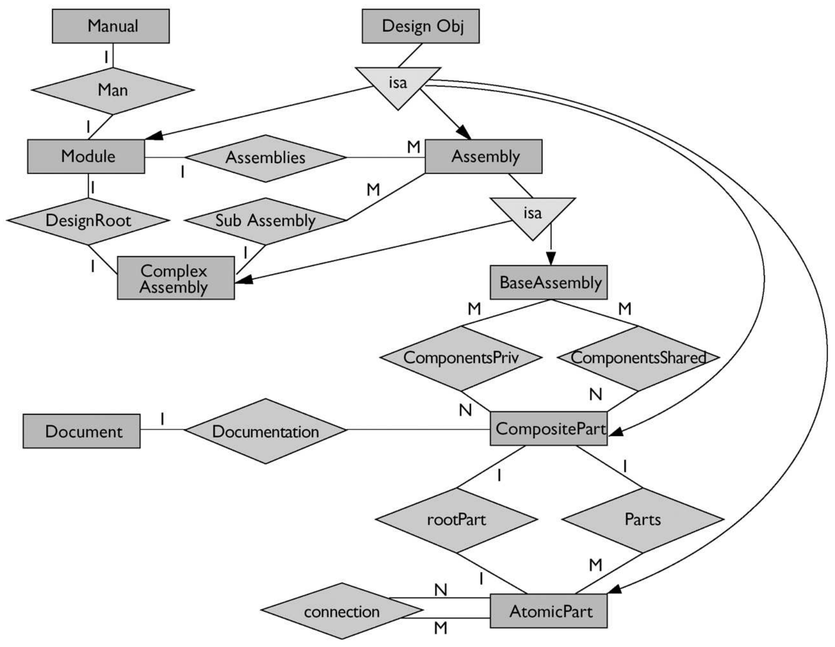


Figure 3.2: DTD of the XOO7 benchmark [BRE02]

In order to cover "all feasible queries within the current XML query model [BRE01c]" 14 more queries were added to XOO7 resulting in a total of 27 queries. The new version is called *XOO7-Extended*. Listing all of these queries in this thesis would not significantly increase the understanding of XOO7. Instead the features of the added queries will shortly be explained.

The queries of the extended version are again arranged in four groups. Group I evaluates the document-centric support of the XMLMS. Group II consists of queries for testing text/keyword search, negation handling and sorting of results. Aggregate functions and efficient handling of structural recursion are covered by group III. Finally group IV is concerned with User Defined Functions (UDF's) and queries that need explicit and implicit order maintenance.

3.2.3 Implementation and Experiences

XOO7 must have already been implemented, because in [BRE01d] and [BRE02] its authors report their experiences when benchmarking different systems. The implementation supports data sets of three sizes: 12.8 MB, 8.4 MB and 4.2 MB. To run the benchmark, native implementations had to be written for every XMLMS since there was no common XML query language. However, no publicly available implementation of the benchmark could be found at the time of writing.

The benchmarked systems are Kweelt, Xena, Lore and a not specified commercial XPath implementation. Kweelt [SAH] is a native XMLMS that works directly on the XML files. Xena [XEN] is a XML-enabled DBS developed at the University of Singapore. It is actually a front-end to the relational database system (RDBS) MySQL [MYS] that maps XML documents to tables. Lore [GOL] is a semistructured DBS that has been adapted to handle XML data by creating *data guides*. Data guides are a summary of all paths in the database starting from the root.

An extensive discussion of the results can be found in [BRE02]. Summing them up, it can be stated that XML-enabled DBS have the best performance when relational queries are processed due to their internal optimization while native XMLMS are better in processing navigational and document queries with which XML-enabled DBS are not familiar.

3.2.4 Rating of Benchmark

The authors of the benchmark did not do the work of creating a clear scenario for their benchmark. Instead they refer to a well-known benchmark and state that it is problematic to support a certain use-case due to the many applications of XML. XOO7 claims to be representative for cases where extensive queries that exhaust XML's potential to the fullest are done on XML documents. It must be noted that the benchmark lacks multi-user simulation. Users must decide for themselves, if this benchmark has relevance for their domain.

XOO7's portability would increase a lot if a standard XML query language was supported by all tested systems. Because there is no publicly available implementation, it cannot be verified if the implementation is OS-independent.

By providing three different sizes of data sets XOO7 achieves some kind of scalability. However, these are so small compared to Xmark that there suitability can be doubted for industry-strength scenarios.

Compared to the original OO7, this benchmark can hardly be called "simple" anymore. 27 queries are a lot and allow for many interpretations of the results. It is just the question whether prospective users would not rather want a crisp and clear answer (even it is less accurate). However, the authors have arguments for choosing these queries and it cannot be ruled out that 27 queries are the closest one can get to simplicity without loosing relevance.

Because of the foregone points, calling XOO7 useful in the way that Gray defines it (see chapter 3) is problematic.

3.3 XMach-1

XMach-1, which stands for “XML Data Management Benchmark, version1”, was developed at the University of Leipzig, Germany in 2000. It was published in 2001 as the *first* XML database benchmark [RAH02]. During its development its authors Rahm and Böhme focused on scalability, multi-user simulation and the evaluation of the entire data management system. The main sources of information on the benchmark were the publications [RAH01a, RAH01b, RAH01c, RAH01d, RAH02] and the project’s homepage [XMA].

3.3.1 Concept

A reason for XML’s popularity is its usefulness as universal data exchange format. Since large amounts of XML data occur in web environments, the benchmark is based on a web application. Unlike other benchmarks, XMach-1 does not only measure the performance of the DBS, but instead defines a system consisting of four components: XML database, application servers, loaders and browsers (see figure 3.3). It has therefore the most complex architecture of all benchmarks in this paper which makes it also the hardest to implement.

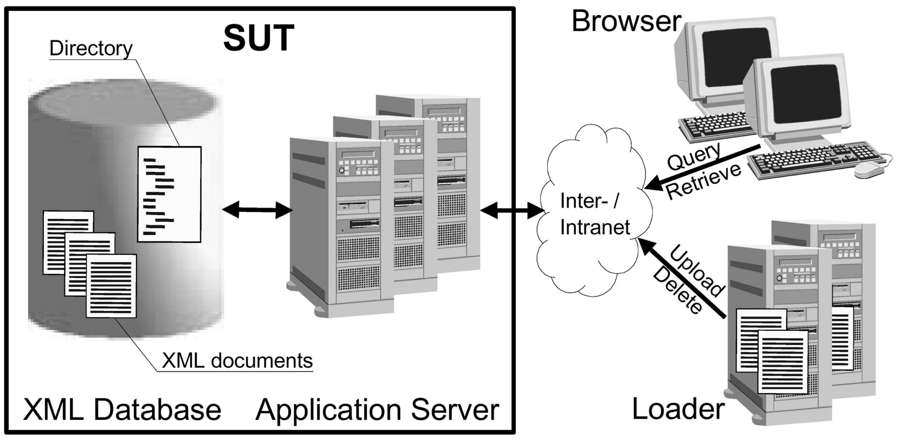


Figure 3.3: Components of XMach-1 Architecture [RAH01b]

The core of the system is, of course, the XML database. It manages the XML data and supplies it to clients. Because the benchmark simulates a web-application, clients cannot directly connect to the database but must use the interface provided by the application servers. Database and application servers together are called the *System under Test* (SUT). Clients connect to the SUT through the

Internet using Hyper Text Transfer Protocol (HTTP) connections. They are distinguished between *browsers* and *loaders*. Browser clients can only query data while loader clients can insert, update and delete data from the database.

The database contains different kinds of XML data. There is a directory document that keeps track of all referenced XML documents in the database. Documents need not to be stored in the database. The loaders could for example search the web for XML documents and index them in the directory document of the database (similar to robots of a search engine). Users of the simulated environment can use loaders to upload their XML documents to the database. These local documents will also be referenced in the directory document. The directory document itself is an XML file and contains information on the path of the referenced document, its internal identification code, insertion time and the name of the loader.

Besides the directory document storing references to XML documents, the database also contains entire XML documents. These are generated by the benchmark using the 10,000 most common English words and distributing them according to Zipf's Law [ZIP]. The benchmark can be run with schema-based documents, i.e. documents have a document type definition (DTD), or without. In the schema-based case the benchmark will generate different DTD's and XML files referring to these. The number of DTD's scales with the database's size because it is unrealistic to believe that a database contains only XML documents with the same DTD. In the schema-less case, the benchmark still generates the same XML documents but no DTD files are generated and the documents do not refer to a DTD. Rahm and Böhme suggest four different database sizes with 10,000, 100,000, 1 million and 10 million XML documents (plus the directory document).

3.3.2 Benchmark Operations

The workload of the XMach-1 benchmark is composed of eleven different operations: eight query operations and three data manipulation operations. Tables 3.8 and 3.10 list all of the operations and comment on these.

Of course, the benchmark defines the percentage that each operation should have of the total workload during a benchmark run. This makes the benchmark repeatable and allows to directly compare different SUT's[2]. Table 3.11 explains the distribution of the different operations.

[2]A different SUT could also mean "same DBS, different application server", e.g. if a database wants to find out which application server harmonizes best with its product.

ID	Description	Comment
Q1	Get document with given URL.	Return a complete document (complex hierarchy with ordering preserved).
Q2	Get doc_id and URL from documents containing a given phrase in paragraph elements.	Text retrieval query. The phrase is chosen from the phrase list. Join needed to get URL for qualifying documents.
Q3	Start with the first element which name starts with 'chapter' and recursively follow first element which name start with 'section'. Return each of the 'section' elements.	Simulates navigating a document tree using sequential operators.
Q4	Return flat list of head elements which are children of elements whose names start with 'section' from a document given by doc_id.	Restructuring operation simulating creation of a table of contents.
Q5	Get document name (last path element in directory structure) from all documents which are below a given URL fragment.	Browse directory structure. Operation on structured unordered data.
Q6	Get doc_id and id of parent element of author element with given content.	Find chapters of a given author. This test efficiency of index implementation.
Q7	Get doc_id from documents which are referenced by at least four other documents.	Get important documents. Needs some kind of group by and count operation.
Q8	Get doc_id from the last 100 updated documents having an author attribute.	Needs count, sort, join and existential operations and accesses metadata.

Table 3.8: XMach-1 query operations [RAH01c]

ID	Description	Comment
M1	Insert document with given URL.	The loader generates a document and URL and sends them to the HTTP server.
M2	Delete a document with given doc_id.	A robot requests deletion, e.g. because the corresponding original document no longer exists on the web.
M3	Update URL and update_time for a given doc_id.	Update directory entry.

Table 3.10: XMach-1 data manipulation operations [RAH01c]

Operation ID	Percentage
Q1	≤30
Q2	≥20
Q3, Q4, Q5, Q6	≥10 each
Q7, Q8	≥4 each
M1	≥0.75
M2	≥0.25
M3	≥1

Table 3.11: XMach-1 operation mix composition [RAH01c]

As can be seen, operation Q1 is the main operation of the benchmark. It is used to determine the throughput which is measured in *XML queries per second* (Xqps). It is also the only one that has a maximum limit to its percentage. The other operations are used to find out how well the throughput of the SUT responds to stress. The system must maintain a high Xqps-rate while executing the other operations. There is a response time limit for all operations Q1 - Q8 and M3 of 3 seconds, so that the candidate cannot keep up his throughput by delaying other operations until infinity. Only operations M1 and M2 (insert and delete) are regarded as not time-critical by XMach-1's developers and have a response time limit of 20 seconds. Only a small share of the operations manipulate the XML data ($\geq 2\%$). Since insert operations (M1) occur three times more often than delete operations (M2), the size of the database will increase during the execution of the benchmark. Rahm and Böhme recommend a benchmark execution time of at least one hour.

3.3.3 Implementation and Experiences

There is a publicly available implementation of benchmark written in Java [RAH01d]. It consists of document generator and an execution framework depending on an application server for Java servlets. The execution framework is made of by the clients which can be instantiated in varying numbers. A servlet is responsible for taking requests from the clients and translating them for the database system.

To achieve comparable results on different SUT's the benchmark has to be run with the same number of clients and the same database size. According to [RAH01b], XMach-1 has been used to benchmark different XMLMS, but it could not be determined which these were.

3.3.4 Rating of Benchmark

According to Gray's criteria (see chapter 3), XMach-1 is portable, scalable and simple. It is portable because it has been implemented in Java. Scalability is achieved by supporting different numbers of clients and database sizes. It is comprehensive and the basic ideas can be grasped quickly.

Only the point “relevance” must be left open for discussion. There are too many opinions on what a XMLMS should achieve to decide that this benchmark is relevant for XML applications[3]. Especially the small number of queries might not deliver analyses fine-grained enough for some users.

Still XMach-1 is a very interesting benchmark. Although not defined in any of the current XML query language drafts like XQuery, it includes XML data manipulation. These operations can be implemented - just not with XQuery and similar languages. While Xmark and XOO7 ignore data manipulation, not every XML database is read-only and hence it is more realistic to include this matter in the benchmark. Also, XMach-1 is the only benchmark where each operation has a prescribed share of the workload mix, whereas the other benchmarks execute every operation the same number of times. For many users (especially of the targeted domain) XMach-1 can be a useful benchmark because of its assessment of the whole system which will give results close to real-life.

[3]Of course, this does not imply that XMach-1 is not relevant.

4 Benchmarking DB2 with XMach-1

DB2 is a XML enabled databases, i.e. it has existed before and programmers added some features to let it handle XML data. The nature of XML makes it hard for relational database systems like DB2 to handle it easily. Therefore it would be interesting to know more about its performance in this field. When evaluating the three benchmarks, it was decided to use XMach-1 for benchmarking DB2. For several reasons it seemed to be fit best:

- It is the only one to simulate a multi-user environment. Thus it does not only measure the efficiency of the tested system concerning certain operations, but also how well the system scales. Since the majority of databases are being used in multi-user environments like LAN's or even the Internet, this feature is very important to prospective users.
- A complete benchmark framework was already available. For XOO7 no software could be found. The XMark team provided software for generating the XML data and the benchmark's queries but no software to measure the time per operation was provided.
- XMach-1 has been implemented completely in Java making it machine and OS independent. After adding the DB2 specific code, it should be possible to benchmark DB2 on different systems (thus achieving good reusability).

Although a framework for the benchmark was already provided, still several adjustments had to be made. First, a loader had to be written to load the XML data of the benchmark into the DBS. Second, the queries needed to be implemented. There is no generic way of implementing them, because every DBS has its own approach of handling XML data.

4.1 The System under Test

The "system under test" (SUT) was a Pentium II PC at 400MHz running SuSE Linux 8.0. IBM's DB2 was available in version 7.1 for Linux and the XML

Extender was version 7.1.2. The application server was Tomcat 4.0.3 from the Apache Project.

4.2 Management of XML Data with DB2

IBM's database system DB2 cannot manage XML data by itself. It requires an add-on called *XML Extender* [IBM]. The XML Extender basically knows two methods for storing XML data: *XML Column* and *XML Collection*.

XML Column is simpler and more straightforward. It stores entire XML documents in columns (see figure 4.1 for a graphical representation). The advantage of this method is that it requires only little preparation, the XML file stays intact and can be extracted easily again. The downside of this method is that extra tables, so called *side tables*, are needed for indexed elements which can also be seen in figure 4.1. Also it is not recommended for composing new XML documents from stored data. XML Column should be used for the purpose of archiving entire XML documents which are unlikely to change [IBM00a].

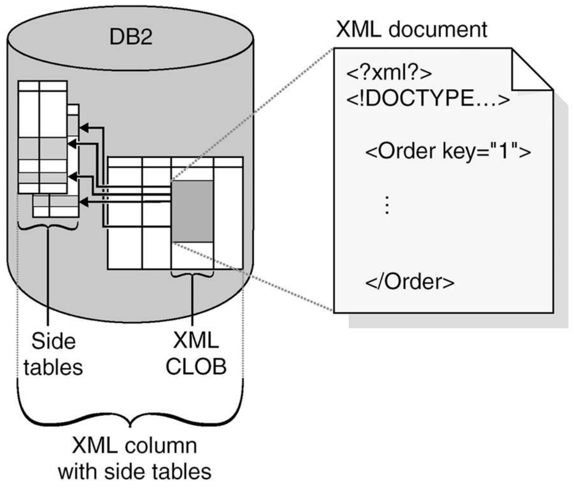


Figure 4.1: Illustration of the *XML Column* storage method [IBM00a]

XML Collection is a more sophisticated storage method. XML documents are decomposed and their content is mapped to relational tables instead of storing documents as a whole. This way all classical database optimization techniques can be used when working with the XML data, resulting in a better performance than with XML Column. The mapping of the document to relational tables is done with *Document Access Definition* (DAD) files. These are special XML files that describe how each element of the XML source file should be stored. In general, XML files have a *Document Type Definition* (DTD) that describe their structure. Because different XML structures must be mapped differently, there needs to be a DAD file for every XML document's DTD if the document should be stored using XML Collection. See figure 4.2 for a graphical representation of the mapping.

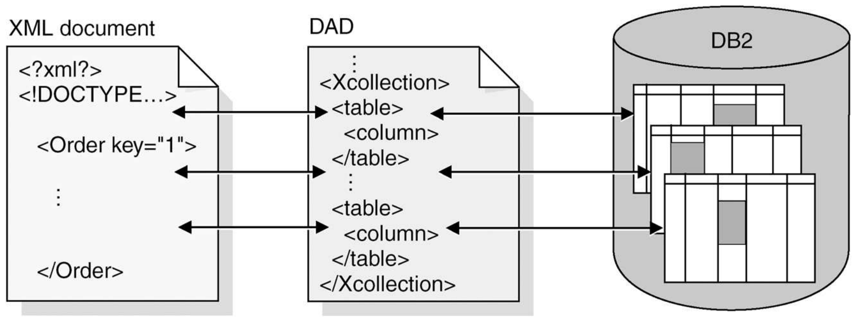


Figure 4.2: Illustration of the *XML Collection* storage method [IBM00a]

XML Collection is suitable for cases where the user wants to create different XML documents from the XML data he gathered. If the structure of the XML data is not important, XML Collection should be preferred over XML Column. Also, if the XML data will be updated a lot, XML Collection should be used for fast update operations. No side tables are required for indexing data in XML Collections. It is sufficient to put a regular index on the appropriate column [IBM00a].

4.3 Problems with DB2's XPath Implementation

The query operations of the XMach-1 benchmark have been formulated in XQuery (see the appendix or [XQE]). This was a wise decision, for at the moment it seems that XQuery will become *the* XML query language used by all future products. The downside of choosing XQuery is that currently very few products contain XQuery implementations. This is also true for the version 7.1.2 of the XML Extender that was used.

Fortunately the XML Extender supports XPath [XPA] expressions. XPath allows to navigate in XML documents using paths similar to the ones in filesystems. The path expressions of XPath have been integrated into XQuery when it was specified by the World Wide Web Consortium [W3C]. So XPath is a subset of XQuery. Therefore it should have been possible to reformulate XMach-1's queries using XPath and user defined functions (UDF's) in DB2.

It turned out that the XML Extender does not support all of the XPath syntax. Table 4.2 lists the supported syntax.

Syntax	Description
`/`	Represents the XML *root* element.
`/tag1`	Represents the element *tag1* under *root*.
`/tag1/tag2/.../tagn`	Represents an element with the name *tagn* as the child of the descending chain from *root*, *tag1*, *tag2*, through *tagn-1*.
`//tagn`	Represents any element with the name *tagn*, where the double slashes denote zero or more arbitrary tags.
`/tag1//tagn`	Represents any element with the name *tagn*, a child of an element with name *tag1* under *root*, where double slashes denote zero or more arbitrary tags.
`/tag1/tag2/@attr1`	Represents the attribute *attr1* of an element *tag2*, which is a child of element *tag1* under *root*.
`/tag1/tag2[@attr1="5"]`	Represents an element with the name *tag2* whose attribute *attr1* has the value 5. *tag2* is a child of an element with name *tag1* under *root*.
`/tag1/tag2[@attr1="5"]/.../tagn`	Represents an element with the name *tagn*, which is a child of the descending chain from *root*, *tag1*, *tag2*, through *tagn-1*, where the attribute *attr1* of *tag2* has the value 5.

Table 4.2: XPath syntax supported by DB2 [IBM00a]

Before beginning with the implementation of XMach-1, a series of test queries was executed. To do so, a table called `files_tab` with the following properties was created:

```
create table files_tab
(filename varchar(255) not null,
DTD varchar(255),
content db2xml.XMLClob compact not logged,
primary key (filename))
```

Column `filename` held the name of the XML file, column `DTD` held the name of the DTD file for the XML file and column `content` held the entire XML file as character large object (CLOB). This relates to the storage technique called *XML Column* by IBM [IBM00a]. It was chosen to quickly get a working database. Into this table 1000 well-formed XML files were inserted. These had been created with the XMach-1 Java framework [RAH01d].

A first try was to extract the entire XML file with the file name `0001_doc_1_test.xml` using the following SQL statement:

```
select content
from files_tab
where filename='0001_doc_1_test.xml'
```

The query was successful and DB2 returned the XML files content.

Queries 2, 5, 6, 7 and 8 of the XMach-1 benchmark work with XML files without actually knowing their name. They specify the file using a criterion like a given phrase in a paragraph element. Next should be tested, whether statements like these were possible. The aim was to retrieve a file with a given *doc_id* attribute:

```
select filename
from files_tab
where db2xml.extractVarchar(content,
"/*/@doc_id")='d1'
```

DB2 returned the following error message:

```
SQL0206N "/*/@doc_id" is not valid in the context
where it is used.  SQLSTATE=42703,
```

where the SQLSTATE message 42703 means: “An undefined column, attribute, or parameter name was detected [IBM00c].” The XPath expression “`/*/@doc_id`” represents the value of the attribute *doc_id* of any element which is a child of the document root. Apparently the XML Extender does not understand the wildcard expression * as it also does not show up in table 4.2.

Even when the name of the element and the filename were specified, the XML Extender did not return the expected result. The statement

```
select db2xml.extractVarchar(content,
"/document1/@doc_id")
from files_tab
where filename='0001_doc_1_test.xml'
```

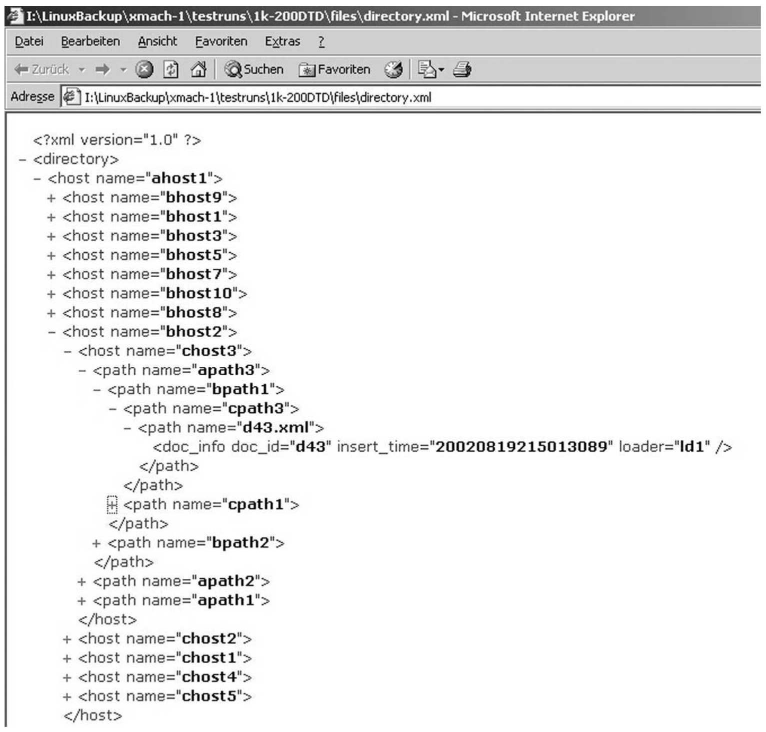
resulted in SQLSTATE 42703 again:

```
SQL0206N "/document1/@doc_id" is not valid in the
context where it is used.  SQLSTATE=42703
```

The next statement tried to query a more complex path. It has been taken from query 1 of the XMach-1 benchmark [RAH01a] and is according to table 4.2 supported by the XML Extender.

```
select db2xml.extractVarchar(content, "/directory/
host[@name='ahost1']/host[@name='bhost2']/
host[@name='chost3']/path[@name='apath3']/
path[@name='bpath1']/path[@name='cpath3']/
path[@name='d43.xml']/doc_info/@doc_id")
from files_tab
where filename='directory.xml'
```

The path represents the value of the attribute *doc_id* of the document `d43.xml` which has the URL `/ahost1.bhost2.chost3/apath3/bpath1/cpath3/d43.xml` in the document `directory.xml`. Figure 4.3 will prove this path correct. Hence the value one would expect as a result would be 'd43'. DB2 does not


I:\LinuxBackup\xmach-1\testruns\1k-200DTD\files\directory.xml - Microsoft Internet Explorer
Datei
Bearbeiten
Ansicht
Favoriten
Extras
?
Zurück
Suchen
Favoriten
Adresse
I:\LinuxBackup\xmach-1\testruns\1k-200DTD\files\directory.xml

```
  <?xml version="1.0" ?>
- <directory>
  - <host name="ahost1">
    + <host name="bhost9">
    + <host name="bhost1">
    + <host name="bhost3">
    + <host name="bhost5">
    + <host name="bhost7">
    + <host name="bhost10">
    + <host name="bhost8">
    - <host name="bhost2">
      - <host name="chost3">
        - <path name="apath3">
          - <path name="bpath1">
            - <path name="cpath3">
              - <path name="d43.xml">
                  <doc_info doc_id="d43" insert_time="20020819215013089" loader="ld1" />
                </path>
              </path>
            + <path name="cpath1">
            </path>
          + <path name="bpath2">
          </path>
        + <path name="apath2">
        + <path name="apath1">
        </host>
      + <host name="chost2">
      + <host name="chost1">
      + <host name="chost4">
      + <host name="chost5">
      </host>
```

Figure 4.3: Screenshot of `directory.xml`

give the expected answer. Instead it returns the following error message, saying that it only accepts paths with at most 128 characters.

```
SQL0107N The name "/directory/host[@name='ahost1']/
host[@name='bhost2']/host[@name='c" is too long.  The
maximum length is "128".  SQLSTATE=42622
```

Paths this short will not allow navigation in complex XML documents. Of course, one could argue that it is possible to split up huge XML files to different tables using the storage technique called *XML Collection* by IBM [IBM00a]. In the case of XMach-1 this is not an option, because the amount of DTD files grows with the size of the database generated by the XMach-1 framework [RAH01c]. Every DTD file requires a matching DAD file for mapping XML data to side tables. For example, the document generator already created 41 DTD files for the smallest possible benchmark database with only 1000 XML documents. Since there is currently no automatic way of generating DAD files, it is not possible within defensible time to use *XML Collection* to store the XML documents.

Because the path of the previous statement was too long, it was tried next to at least receive the value of the attribute *name* of the child elements of the element *host* whose name is 'bhost2'. (Basically, that means shortening the path to a length that is acceptable for DB2.)

```
select db2xml.extractVarchar(content, "/directory/
host[@name='ahost1']/host[@name='bhost2']/host/@name")
from files_tab
where filename='directory.xml'
```

DB2 returned:

```
SQL0206N
"/directory/host[@name='ahost1']/host[@name='bhost2']/
host/@name" is not valid in the context where it is
used.  SQLSTATE=42703
```

The actual reason why DB2 failed here, is that it expects to find atomic values when extracting XML data with XPath statements. As can be seen in figure 4.3, the path would have yielded several different *host* elements with different *name* attributes. DB2 expects only one value per column for every tuple. Everything else would violate the First Normal Form (1NF), compare [HEU].

Besides some weaknesses in the current XPath implementation, it is this restriction that hinder XMach-1's reference implementation [RAH01d] from benchmarking DB2. The Syntax of XQuery allows the usage of XPath expressions that yield a set of values [XQE]. XMach-1 uses this feature in queries 2, 4, 5, 6, 7 and 8. Since this much needed feature was not available, it was unrealistic to believe that the XMach-1 benchmark could be implemented with version 7.1.2 of the XML Extender. The new version 8.1 of the XML Extender now supports XQuery (see [IBM] for details), but at the time of writing this thesis, it was not available yet. Therefore the idea of using XMach-1 to benchmark DB2 had to abandoned[1].

4.4 Available XQuery Implementations

Since XMach-1 relies on XQuery to run, the next idea was to find a XQuery implementation and use it instead of DB2 for the benchmark. It was agreed to choose open-source software to make it easy on the budget.

Qexo [BOT02a] was chosen, which is an XQuery implementation for Kawa[2] [BOT02b]. It has been released under the GNU license (see [GNU] for details) and is “the only open-source XQuery implementation currently available” [BOT02a]. Qexo is written in Java, it accepts XQuery code on the command-line as short statements or in files as complete XQuery programs. It compiles XQuery code to Java bytecode and executes it.

This concept is very interesting, because Qexo can be easily integrated into other software, e.g. as a servlet. Also, it works directly on XML files thus making the user independent from expensive XML enabled DBS or XMLMS. It would have been nice to test Qexo with XMach-1 to find out how well it performs. Unfortunately it is still work in progress and not all of the features needed by XMach-1 are already supported. For example, at the moment it cannot navigate through directories of XML files as needed by query 5 and it does not support the `SORTBY` functionality needed by query 8.

[1]Since the XML Extender did not provide the anticipated ability, the IBM DB2 Text Extender for text search and analysis also could not be used for the benchmark.

[2]Kawa is a Java-based system for executing Scheme code. However, Kawa and the programming language Scheme are not important in the context of this thesis.

5 Conclusion and Outlook

In this thesis the necessity for special XML benchmarks has been laid out. The differences between native and XML-enabled DBS was explained. Three current XML benchmarks have been introduced and examined. According to Gray's criteria for benchmarks their usefulness has been rated.

The original objective to benchmark IBM's DB2 with one of the benchmarks could not be accomplished. Due to the problems analyzed in section 4.3, DB2 with version 7.1.2 of the XML Extender could not be benchmarked against the XQuery implementation of XMach-1. Version 8.1 of the XML Extender with support for XQuery will hopefully allow to use XMach-1 for benchmarking DB2 in the future.

To conclude, this thesis will give suggestions on the deployment of the three discussed benchmarks when benchmarking XMLMS's.

Different interest groups have different expectations from XML benchmarks and also benchmarks are not equally suitable for every user. For example, Xmark differs from XOO7 in that it is quite easy to understand and that the necessary implementation is for the greatest part already available. XOO7 in turn consists of more queries and so far there have been no attempts to publish its implementation. Still both address the same user groups. For once, these are database vendors and researchers who want to examine certain aspects of their system. These benchmarks are very advisable for system analysis, to find bottlenecks in the current implementation and to validate improvements of the system. But also end-users might have an interest in using these. If they have very specific requirements, these benchmarks could help to find out if a certain XMLMS fulfills these.

XMach-1 is not as detailed as the other benchmarks as it has less operations in its workload. So, maybe it does not include the specific features some users need. Because it is the only benchmark to regard multi-user capabilities and the cooperation of all components in the system under test (DBS and application server), XMach-1 is more suitable for people who want to deploy a XMLMS in a multi-user environment similar to the one modeled. Users that rely on the performance of a XMLMS to actually work with it, will probably find this benchmark more appealing as it is closer to the real world. It does not find out what the system could do in every aspect that could be imagined for XML usage, but it does give an idea of how the benchmarked system might behave under stress in a production environment.

In the future, new versions of the examined benchmarks could eliminate some of todays shortcomings. For example, Xmark's queries are still under development [SCH01b] and the XOO7 team has announced to work on multi-user support and increase the portability of the system [BRE01c]. Data manipulation is also a big issue that might be included as XQuery's development furthers. None of the benchmark project's is being considered finished and they are still up for discussion in the research community, so one should watch out for future developments.

Bibliography

[BRE02] Bressan, S. et al.: "Current Approaches to XML Management", IEEE Internet Computing, Los Alamos, IEEE Computer Society, July 2002

[BRE01a] Bressan, S. et al.: "The XOO7 XML Management System Benchmark", National University of Singapore Computer Science Department technical report TR21/00, November 2001

[BRE01b] Bressan, S. et al.: "XOO7: Applying OO7 Benchmark to XML Query Processing Tools", Proceedings of 10th International Conference on Information and Knowledge Management (CIKM-2001), New York, ACM Press, 2001

[BRE01c] Bressan, S. et al.: "XML Benchmarks Put to the Test", Proceedings of Third International Conference on Information Integration and Web-based Applications & Services (IIWAS), Linz, Austria, September 2001

[BRE01d] Bressan, S. et al.: "Benchmarking XML Management Systems: The XOO7 Way", Arizona State University, Dept. of Computer Science Technical Report TR-01-005, July 2001

[BRExxa] Bressan, S. et al.: "The XOO7 Benchmark" on the Internet <http://www.comp.nus.edu.sg/~ebh/XOO7.html> as of: date could not be determined

[BRExxb] Bressan, S. et al.: "Efficient XML Data Management: An Analysis", on the Internet <http://www.comp.nus.edu.sg/~ebh/XOO7/download/ECWEB02.pdf> as of: date could not be determined

[BOT02a] Bothner, P.: “Qexo: The GNU Kawa implementation of XQuery”, on the Internet <www.gnu.org/software/ qexo> as of: 28-Jul-2002

[BOT02b] Bothner, P.: “Kawa, the Java-based Scheme system”, on the Internet <http://www.gnu.org/software/kawa> as of: 12-Jun-2002

Bibliography

[CAR] Carey, M.J.; DeWitt, D.J.; Naughton, J.F.: "The OO7 Benchmark", Proceedings SIGMOD, New York, ACM Press, 1993

[COD] Codd, E.F.: "A Relational Model of Data for Large Shared Data Banks", Communications of the ACM, vol. 13, no. 6, 1970

[GNU] “Homepage of the GNU Project”, on the Internet <www.gnu.org>

[GOL] Goldman, R.; McHugh, J.; Widom, J.: “From Semistructured Data to XML: Migrating the Lore Data Model and Query Language”, Proceedings Workshop on Web and Databases (WebDB99), New York, ACM Press, 1999

[GRA] Gray, J.: “The Benchmark Handbook”, 2nd edition, San Mateo, Morgan Kaufmann, 1993

[HEU] Heuer, A.; Saake, G.: "Datenbanke: Konzepte und Sprachen", 2nd edition, Bonn, mitp, 2000

[IBM] "IBM DB2 XML Extender Homepage", on the Internet <http://www-3.ibm.com/software/data/db2/extenders/xmlext/>

[IBM00a] "IBM DB2: XML Extender Administration and Programming", version 7, IBM, 2000

[IBM00b] "IBM DB2: Administration Guide", version 7, vol. 2, IBM, 2000

[IBM00c] “IBM DB2 Message Reference”, vol. 2, IBM, 2000

[MON] Kersten, M.; Boncz, P.: "Monet Database", on the Internet <http://monetdb.cwi.nl> as of: 28-Apr-1998

[MYS] "MySQL Homepage", on the Internet <http://www.mysql.com>

[RAH02] Rahm, E., Böhme, T.: "XMach-1: A Multi-User Benchmark for XML Data Management", Proceedings VLDB workshop Efficiency and Effectiveness of XML Tools and Techniques (EEXTT2002), Hongkong, 2002

[RAH01a] Rahm, E., Böhme, T.: "XMach-1 Appendix: Queries" on the Internet <http://dbs.uni-leipzig.de/de/projekte/XML/XMach-1_queries.html> as of: 23-Oct-2001

[RAH01b] Rahm, E.; Böhme, T.: "Benchmarking XML Database Systems - First Experiences", Position Paper, Ninth International Workshop on High Performance Transaction Systems (HPTS), Pacific Grove, California, 2001

[RAH01c] Rahm, E.; Böhme, T.: "XMach-1: A Benchmark for XML Data Management", Proceedings of German Database Conference BTW2001, Berlin, Springer, 2001

[RAH01d] Rahm, E.; Böhme, T.: "Xmach-1 Reference Implementation" on the Internet <http://dbs.uni-leipzig.de/de/projekte/XML/xmach_1_20010518.jar> as of: 18-May-2001

[SAH] Sahuguet, A.: “Kweelt: More Than Just ’Yet Another Framework to Query XML’!”, Proceedings SIGMOD, New York, ACM Press, May 2001

[SCH02a] Schmidt, A.R. et al.: "XMark: A Benchmark for XML Data Management" on the Internet <http://www.cwi.nl/htbin/ins1/publications?request=pdf&key=ScWaKeCaMaBu:VLDB:02> as of: August 2002

[SCH02b] Schmidt, A.R. et al.: "XMark – An XML Benchmark Project" on the Internet <http://monetdb.cwi.nl/xml/> as of: 19-Jul-2002

[SCH02c] Schmidt, A.R. et al.: "The Data Generation Tool – xmlgen" on the Internet <http://monetdb.cwi.nl/xml/downloads.html> as of: 19-Jul-2002

[SCH01a] Schmidt, A.R. et al.: "Why and How to Benchmark XML Databases" on the Internet <http://www.cwi.nl/htbin/ins1/publications?request=pdf&key=ScWaKeFlCaMaBu:SIGMODREC:01> as of: September 2001

[SCH01b] Schmidt, A.R. et al.: "The XML Benchmark Project" on the Internet <http://www.cwi.nl/htbin/ins1/publications?request=pdf&key=ScWaKeFlMaCaBu:TR-CWI:01> as of: April 2001

[W3C] “W3C Homepage”, World Wide Web Consortium, on the Internet <http://www.w3c.org>

[XEN] "Xena – XML sEcurity eNforcement Architecture", on the Internet <http://xena1.ddns.comp.nus.edu.sg:8080/Xena> as of: date could not be determined

[XMA] "XMach-1 homepage", on the Internet <http://dbs.uni-leipzig.de/de/projekte/XML/XmlBenchmarking.html> as of: 02-Sep-2002

[XPA] Clark J.; DeRose S.: "XML Path Language (Xpath) version 1.0", on the Internet <http://www.w3c.org/TR/xpath> as of: 16-Nov-1999

[XQE] Boag S. et al.: "Xquery 1.0: An XML Query Language", on the Internet <http://www.w3.org/TR/xquery> as of: 16-Aug-2002

[ZIP] Zipf, G.K.: “The Psychobiology of Language”, Boston, Houghton Mifflin, 1935

Appendix

XMach-1 Queries formulated in XQuery

The following queries have been taken from [RAH01a].

Q1

Description: Get document with given URL.
Parameter: URL = /ahost1.bhost2.chost3/001_loader1.xml

```
LET $a :=
/directory/host[@name="ahost1"]/host[@name="bhost2"]
/host[@name="chost3"]/path[@name="001_loader1.xml"]
/doc_info/@doc_id,
$b := /*[@doc_id = $a]
RETURN $b
```

Q2

Description: Get doc_id from documents containing a given phrase in paragraph elements.
Parameter: phrase = "test pattern"

```
FOR $a IN /*[@doc_id]
WHERE SOME $p IN $a//paragraph SATISFIES contains($p, "test
pattern")
RETURN distinct($a/@doc_id)
```

The query was slightly changed compared to the specification document in order to focus on more specific tasks.

Q3

Description: Start with first chapter element and recursively follow first section element. Return last section element.
Parameter: doc_id = "d1" suffix = "10"

```
DEFINE FUNCTION deepestFirstSection(ELEMENT $e) RETURNS
ELEMENT
{ IF (empty($e/section10)) THEN $e
  ELSE deepestFirstSection($e/section10[1])
}
deepestFirstSection(/document10[@doc_id="d1"]/chapter10[1]
/section10[1])
```

The query was slightly changed compared to the specification document in order to focus on more specific tasks.

Q4

Description: Return flat list of head elements which are children of section elements.
Parameter: doc_id = "d1" suffix = "10"

```
FOR $a IN /document10[@doc_id="d1"]//section10/head10
RETURN $a
```

The query was slightly changed compared to the specification document in order to focus on more specific tasks.

Q5

Description: Get document name (last path element in directory structure) from all documents which are below a given URL fragment.
Parameter: URL = "/ahost1.bhost2.chost3/"

```
FOR $a IN
/directory/host[@name="ahost1"]/host[@name="bhost2"]
/host[@name="chost3"]//path[doc_info]
RETURN $a/@name
```

Q6

Description: Get doc_id and id of parent element of author element with given content.
Parameter: Author = "Karl May"

```
FOR $a IN /*[@doc_id]
WHERE $a//*/author="Karl May"
RETURN <document> <doc_id>{ string($a/@doc_id) }</doc_id>
{ FOR $e IN $a//*[author="Karl May"]
  RETURN <parent_id>{string($e/@id)}</parent_id>
} </document>
```

Q7

Description: Get doc_id from documents which are referenced by at least four other documents.

```
NAMESPACE xlink='http://www.w3.org/1999/xlink'
FOR $refID IN distinct(/*//link/@xlink:href)
LET $refDocs :=
distinct(/*[.//link/@xlink:href=$refID]/@doc_id)
WHERE count($refDocs) >= 4
RETURN <docid>{string($refID)}</docid>
```

Q8

Description: Get doc_id from the last 100 updated documents having an author attribute.

```
LET $b := (FOR $a IN /directory//doc_info[@update_time]
           SORTBY (./@update_time DESCENDING)
           WHERE not(empty(/*[@doc_id=$a/@doc_id]/@author))
           RETURN <docid>{string($a/@doc_id)}</docid> )
RETURN $b[1 TO 100]
```

Eidesstattliche Versicherung

Ich versichere eidesstattlich durch eigenhändige Unterschrift, daß ich die Arbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen sind, habe ich als solche kenntlich gemacht. Ich weiß, daß bei Abgabe einer falschen Versicherung die Prüfung als nicht bestanden zu gelten hat.

Bremen, den 04.10.2002

Tim F. Rieger