The XOO7 XML Management System Benchmark

STÉPHANE BRESSAN, MONG LI LEE, YING GUANG LI

National University of Singapore

{steph, leeml, liyg}@comp.nus.edu.sg

ZOÉ LACROIX, ULLAS NAMBIAR

Arizona State University

{zoe.lacroix, mallu}@asu.edu

Abstract

As XML becomes the standard for electronic data interchange, it is necessary to design benchmarks to provide for the comparative performance analysis of XML management systems (XMLMS). In this work, we propose XOO7, a benchmark for XMLMS. The XOO7 benchmark is an XML version of the OO7 benchmark [11] enriched with relational, document and navigational queries that are specific and critical for XML databases. We show that the benchmark meets the four criteria, namely, relevance, portability, scalability and simplicity. We implement XOO7 and illustrate its applicability by using it to evaluate the performance of four XML management systems.

Key Words

XOO7, Performance Analysis, XML Benchmark, Query Processing

1 Introduction

Introduced as a schema-less, self-describing data representation language, XML quickly emerged as the standard for information interchange for the Web [10]. XML was designed as a subset of SGML (Standardized General Markup Language) to solve the problems faced by the document community. XML

augments HTML (Hyper Text Markup Language) by allowing data to carry its meaning and not just presentation details. The semi-structured nature of XML has prompted database researchers to actively participate in developing standards centered on XML, in particular, query languages and query processing tools for XML. These languages and tools can be classified into two categories namely (i) Languages and tools designed with a document focus such as XQL [27] and XPath [17] and (ii) Those designed with a database focus e.g. LOREL [3] and XML-QL [20]. Recently, XQuery [14] has been published by the World Wide Web Consortium as a candidate for a standard query language for XML, combining both document-centric and data-centric characteristics of XML.

Many Internet systems today use a database to store and manipulate the data. The need to efficiently store, manage and interchange large amounts of data is ever increasing. A variety of XML management systems (XMLMS) are becoming available. Current XMLMS can be divided into two categories: XML-Enabled databases and Native XML databases. XML-Enabled databases, typically relational databases, such as DB2 XML Extender from IBM, Informix, Microsoft SQL Server 2000, Oracle 8i & 9i [16], provide extensions for transferring data between XML documents and themselves. Such systems are generally designed to store and retrieve data-centric XML documents. On the other hand, Native XML databases such as Kweelt [28], IPEDO, Tamino, 4SuiteServer, DBDOM, dbXML [6] etc, either store the entire XML document in text form with limited retrieval capabilities or store a binary model (e.g. Document Object Model) of the document in an existing or custom data store.

As we expect new Web based applications for e-commerce to require XML query processing facilities, a user intending to set up an XML based data interchange or storage system would be faced with the question of which of the XML query languages or processing systems to base his/her system on. With so many proposals and tools, a benchmark across which various XML query processing systems can be tested and compared would enable end-users to choose the tool most suitable for their application in terms of its features and performance. This benchmark should be useful to not only assess the power of various XML query languages but also to evaluate the efficiency of the XML management systems

which use these query languages. Throughout the paper, we shall interchangeably use "XML management systems" and "XML Query Processing Systems" to refer to data management tools that store/produce/manipulate XML data and provide a data retrieving mechanism based on a known XML query language.

In this paper, we present XOO7 – a benchmark to evaluate the performance of XML management systems. The XOO7 benchmark is an XML version of the OO7 benchmark [11] enriched with document and navigational queries that are specific and critical for XML databases. When developing the benchmark we based our decisions on three facts.

- The benchmark is for XML query systems using XML data and documents stored locally.
- XML data model shows a high degree of similarity to the object-oriented model.
- The benchmark should adhere to the four criteria laid out in the Benchmark Handbook [24], namely, relevance, portability, scalability and simplicity.

The rationale underlying both the design of XML, XML query languages, and the object-oriented data model and query languages is the need for richer structure for the flexible modeling and querying of complex data. Although XML also attempts to provide a framework for handling semi-structured data, it encompasses most of the modeling features of complex object models [2, 4]. The OO7 benchmark [11] provides a comprehensive evaluation of object-oriented database management system (OODBMS) performance, including E/Exodus, Objectivity/DB, and Ontos. Hence we decided to take OO7 - a well-established benchmark designed to test performance of object-oriented database management system (OODBMS) and extend it for XML. However, we observe that the OO7 benchmark queries are mostly data-centric and do not cover a substantial number of the XML query functionalities identified by the W3C Query Language Working Group [13]. It is therefore imperative to design queries that test document processing capabilities. Towards this end, we provide two new groups of queries in XOO7. Finally, with a focus on efficiency and concreteness, we implement XOO7 and show the

applicability of the XOO7 benchmark by using it to evaluate the performance of four different query processing platforms for XML.

The rest of the paper is organized as follows. Section 2 presents the benchmark specifications. The XOO7 data model and queries are described in Section 3. Section 4 gives the results of the experiments to evaluate the different XML management systems. Section 5 discusses related work and compares XOO7 to two other proposed benchmarks for XML. Finally, we conclude our work in Section 6 and highlight the possible extensions to this work.

2 Benchmark Specification

The well-known Benchmark Handbook by Jim Gray [24] has laid down the following four key criteria for a domain-specific benchmark.

- Relevance
- Portability
- Scalability
- Simplicity

Various domain-specific benchmarks have been developed because no single metric can measure the performance of computer systems on all applications [24]. For instance, the Wisconsin benchmark [5] is widely used to test the performance of relational query systems on simple relational operators, the AS^3AP benchmark [32] provides a more complete evaluation of relational database systems by incorporating features such as testing utility functions, mix batch and interactive queries, and multi-user tests, the Set Query benchmark [26] evaluates the ability of systems to process complex queries which are typical in decision-support applications and data mining applications, TPC-D for online transaction processing (OLTP), TPC-H, TPC-R, APB-1 [25] for decision support, information retrieval, Sequoia [30] for spatial data management, OO7 [11] for object-oriented databases, Bucky [12] for object-relational databases, and the most recent TPC-W benchmark [31] for e-commerce. All the benchmarks satisfy the important criterias of being relevant to their domain, portable, simple, and scalable.

The design of a benchmark for evaluating XML management systems is a non-trivial task. The *eXtended Markup Language* (XML) is a self-describing data representation language that has emerged as the standard for electronic information interchange. Thus its potential use covers a variety of complex usage scenarios. Our objective is not to propose an all-powerful benchmark measuring all aspects of XML usage. Rather we focus on the query processing aspect of XML. In fact, for the sake of fairness and simplicity, we propose that the XML query processing tools be tested in the simplest possible set-up, that is, with locally stored data and in a single machine / single user environment.

In the following subsections, we will discuss the various issues raised when designing a benchmark for XML management systems. This involves designing a benchmark data set and the corresponding benchmark queries that adhere to the four benchmark design criteria. We note that these criteria are inter-related, often to the extent of being conflicting, and affect the ability of a benchmark to adequately capture the performance of the systems under evaluation. For example, if the test data of an XML management system benchmark is too simple (*Simplicity*), it will not be able to capture the ability of XML to represent complex structures (*Relevance*). On the other hand, if the schema of the XML data is very complex (*Relevance*), then some XML systems may not have the capability to store the data properly (*Portability*) and it may be difficult to change the size of the data (*Scalability*).

2.1 Benchmark Database

The structure of the benchmark dataset must be complex enough in order to illustrate clearly the potential of XML data representation. XML is radically different from relational data model. In particular, XML provides an implicit ordering of its data since it has been designed as a subset of SGML to represent documents. In addition, XML also provides references, deep nesting as well as hyperlinks. These features are similar to those found in object-oriented models that we will elaborate in the next section.

A benchmark for XML management is expected to capture most if not all the above characteristics. That is, in addition to the traditional object-oriented complex object modeling features, the benchmark should also consider the document aspect (the implicit order of elements) and navigation (references). In addition, the semantics and structure of an XML schema should be easily understandable. We found that a good starting point to design a benchmark for XML is to adapt an existing benchmark for object-oriented data management systems and enhance it with specific XML features. For example, the OO7 benchmark provides a simple yet structurally rich application domain, which can be easily extended to XML.

In order to evaluate the scalability of a system, a benchmark should provide data sets of varying sizes. Since XML data can be represented as a tree, the depth and width of the tree should be adjustable. This can be achieved as follows:

- The depth of a tree can be controlled by varying the number of repetitions of recursive elements.
- The width of the tree can be adjusted by varying the cardinality of some elements.

Since XML itself is platform and application independent, portability is not a major issue here. We observe that unlike XML elements, the ordering of attributes in XML is optional. Hence, an XML management system can return the results of a query without preserving the attribute order.

2.2 Benchmark Queries

Bonifati and Ceri address the functionalities of XML query languages by carrying out a comparative analysis of the major XML query languages in [7]. The W3C XML Query Language Working Group has also published a list of "*must have*" requirements for XML query languages [13]. Table 1 presents these requirements. Since element order is another important functionality [21], we have added it into list of functionalities as R21. The performance of the implementations of query languages for XML depends greatly on their expressive power, that is, the functionalities they provide.

We observe that the requirements enumerated by the W3C demand that an XML query language should provide data-centric, document-centric and navigational capabilities. XML can capture structured information and the query language should have the ability to express a structured query language similar to SQL for relational databases. Such data-centric capabilities or *Relational queries* include various types of join operations (R9), aggregation (R10), sorting (R11), etc. Queries that use the implicit and explicit order of elements in a XML document are classified as *Document queries*. Such document-centric capabilities are required when order and document structure need to be preserved in some form (R17, R21). Queries that require traversal of XML document structure using references/links as supported by XLink/XPointer specification [18, 19] are called *Navigational queries* (R13, R20).

Id	**Description**
R1	Query all data types and collections of possibly multiple XML documents.
R2	Allow data-oriented, document-oriented and mixed queries.
R3	Accept streaming data.
R4	Support operations on various data models.
R5	Allow conditions/constraints on text elements.
R6	Support for hierarchical and sequence queries.
R7	Manipulate NULL values.
R8	Support quantifiers (∃,∀, and ~) in queries.
R9	Allow queries that combine different parts of document(s).
R10	Support for aggregation.
R11	Able to generate sorted results.
R12	Support composition of operations.
R13	Allow navigation (reference traversals).
R14	Able to use environment information as part of queries e.g. current date, time etc.
R15	Able to support XML updates if data model allows.
R16	Support for type coercion.
R17	Preserve the structure of the documents.
R18	Transform and create XML structures.
R19	Support ID creation.
R20	Structural recursion.
R21	Element ordering.

Table 1: Desired functionalities of XML Query Languages

Some XMLMS systems may have limited query-processing capabilities and it is possible that many benchmark queries cannot be executed on such systems. For example, XPath only supports the *count* function but not the *avg* (average) function. A benchmark query that tests the performance of both *count* and *avg*

functions cannot be evaluated on systems that implement XPath. This problem can be resolved by having separate benchmark queries, each testing a different type of aggregation. Furthermore, depending on the application, users may only want to test a subset of the functionalities covered by some queries. For example, some applications may never need to update the database; while others do not need to restructure the retrieved results. Hence, it is important to distribute the various functionalities into different queries so that users can always choose the queries according to the functionalities they need. Separating the functionalities also facilitate the analysis of the experiment results since it will be very clear which feature is being tested. Finally, the benchmark queries should allow the range of values of selected attributes to be varied in order to control the percentage of data retrieved by queries, that is, the selectivity of queries.

3 The XOO7 Benchmark

The XML syntax is suited for semi-structured data. Yet XML and semi-structured data have subtle differences [1, 8]. Without going into the details of XML, a simple abstraction of XML is a labeled ordered tree. A tree representation of XML and semi-structured data is interchangeable but a graph structure of both models have differences. Semi-structured data model is based on unordered collections, while XML is ordered. Unique identifiers can be associated with elements in XML. References to such elements can be made by other elements in the XML document.

A close observation of the XML model will show its similarity to the object-oriented data model. Both XML and the object-oriented data model provide a rich structure for modeling and querying of complex data/objects. While XML is able to handle semi-structured data, it supports most of the features of complex object models. XML is probably closer to object-oriented data model in as much as it also consists of nodes, and nodes can contain heterogeneous data. On the other hand, just how heterogeneous nodes are depends on the DTDs or Schemas used to define the structure of an XML document. The object-oriented data model is similar to both XML and semi-structured data model with respect to representation of objects or entities using trees. Similar to XML we can assign

object identities or 'oids' to objects if these have to be referenced by other objects. An object identifier can become part of a namespace and can refer other objects across the Web. This is similar to the notion of Namespaces in XML. Compared to object-oriented data model the XML data model does not have classes, methods and inheritance; instead it has element types and attributes which are similar to classes and attributes in object-oriented model.

Thus in developing a benchmark for XML, we decided to use the well-established OO7 benchmark that has been designed for object-oriented database management system as the starting point. The XOO7 benchmark is an XML version of the OO7 benchmark with new elements and queries designed to test the features that are unique in XML.

3.1 XOO7 Database

The basic data structure in our proposed XOO7 benchmark comes from the OO7 benchmark. Figure 1 shows the conceptual schema of the database modeled using the ER diagram. We have translated this schema into the corresponding DTD as shown in Figure 2. We note that there may be other ways to translate the ER diagram to a DTD, which are beyond the scope of this paper. Nevertheless we outline our main translation decisions.

Since XML does not cater for ISA relationship, we have to preprocess the inheritance of attributes and relationships. This transformation is common to many OO7 implementations. We choose <Module*i*> as the root element of the XML document. Although XML is supposed to be portable and independent of any platform, but there are systems that designate special functions to certain attributes. For example, LORE uses the name of the root element to identify an XML document, but the majority of XMLMS use the file name as the identifier. In XOO7, we assign a unique root name to each XML file. In this way, the XML data can be stored and queried in most systems.

The <Module*i*> element contains two sub-elements, <Manual> that contains a large amount of text data, and <ComplexAssembly> which is a recursive element.

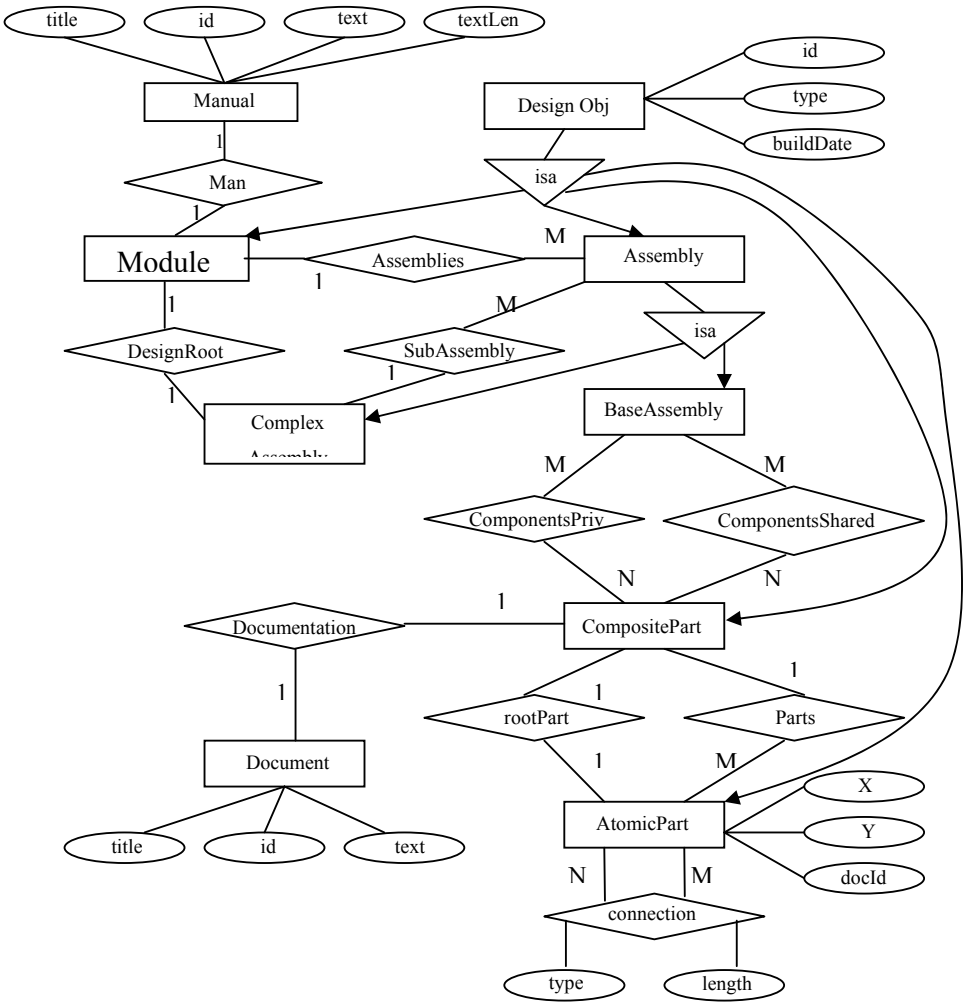

Figure 1: Entity-relationship diagram for the OO7 benchmark.

Each <Manual> element has three attributes, MyID (integer), title (string), and textLen (integer). The sub-elements of <ComplexAssembly> can be <ComplexAssembly> or <BaseAssembly>. The sub-element of <BaseAssembly> is <CompositePart>. Each <CompositePart> has a unique sub-element <Document> and a number of <Connection> sub-elements.

We create two <para> elements in <Document> to cater for the document-centric aspect of XML. In fact, the <Document> element provides for a liberal use of free-form text that is "marked up" with elements. <Document> contains two attributes, MyID (integer) and title (string). There is no attribute in <para>. The <Connection> element contains two attributes, type (string) and length (integer). There are two <AtomicPart> elements in <Connection>. Among these elements, Module, ComplexAssembly, BaseAssembly, CompositePart and AtomicPart are

Design Objects, so they inherit the three attributes from design objects. The attributes are MyID (integer), type (string), and buildDate (integer) (check the figure).

```
<!DOCTYPE Module>
<!ELEMENT Module      (Manual, ComplexAssembly)>
<!ATTLIST Module      MyID           NMTOKEN #REQUIRED
                      type           CDATA          #REQUIRED
                      buildDate      NMTOKEN        #REQUIRED>
<!ELEMENT Manual      (#PCDATA)>
<!ATTLIST Manual      MyID           NMTOKEN        #REQUIRED
                      title          CDATA          #REQUIRED
                      textLen        NMTOKEN        #REQUIRED>
<!ELEMENT ComplexAssembly (ComplexAssembly+ | BaseAssembly+)>
<!ATTLIST ComplexAssembly
                      MyID           NMTOKEN        #REQUIRED
                      type           CDATA          #REQUIRED
                      buildDate      NMTOKEN        #REQUIRED>
<!ELEMENT BaseAssembly (CompositePart+)>
<!ATTLIST BaseAssembly
                      MyID           NMTOKEN        #REQUIRED
                      type           CDATA          #REQUIRED
                      buildDate      NMTOKEN        #REQUIRED>
<!ELEMENT CompositePart (Document, Connection+)>
<!ATTLIST CompositePart
                      MyID           NMTOKEN        #REQUIRED
                      type           CDATA          #REQUIRED
                      buildDate      NMTOKEN        #REQUIRED>
<!ELEMENT Document    #PCDATA|(para+)>
<!ATTLIST Document    MyID           NMTOKEN        #REQUIRED
                      title          CDATA          #REQUIRED>
<!ELEMENT para        (#PCDATA)>
<!ELEMENT Connection (AtomicPart, AtomicPart)>
<!ATTLIST Connection
                      type           CDATA          #REQUIRED
                      length  NMTOKEN        #REQUIRED>
<!ELEMENT AtomicPart EMPTY>
<!ATTLIST AtomicPart
                      MyID           NMTOKEN        #REQUIRED
                      type           CDATA          #REQUIRED
                      buildDate      NMTOKEN        #REQUIRED
                      x              NMTOKEN        #REQUIRED
                      y              NMTOKEN        #REQUIRED
                      docId          NMTOKEN        #REQUIRED>
```

Figure 2: DTD for XML data in XOO7 Benchmark

Since the Connection object in the OO7 database contains attributes, we need to map this object to an XML element <Connection> in order to preserve the

attributes. We omitted some objects especially for object-oriented data, like *assembly* and *designed-object.* These two objects are to show the inheritance relation in an object-oriented database. We chose to push down the attribute inherited from parent objects to the child objects that have been changed to XML elements in XOO7. Eventually, the XOO7 DTD comprises only *contains* relationship from parent nodes to child nodes and *belongs-to* relationship from child nodes to parent nodes. This makes the overall structure of the benchmark is straightforward.

The structure of the XOO7 dataset is both complex and comprehensive. It involves recursive elements, attributes, text data, numerical data, document-centric data etc. We provide the DTD with the XML files that can be easily changed to XML Schema using tools like *XMLSpy*[33] so that both schema-based and schema-less system can store and query the data.

Similar to OO7, the XOO7 benchmark allows datasets of varying size to be generated: small, medium, and large. Table 2 summarizes the parameters and their corresponding values that are used to control the size of the XML data.

Parameters	**Small**	**Medium**	**Large**
NumAtomicPerComp	20	200	200
NumConnPerAtomic	3, 6, 9	3, 6, 9	3, 6, 9
DocumentSize (bytes)	500	1000	1000
ManualSize (bytes)	2000	4000	4000
NumCompPerModule	50	50	500
NumAssmPerAssm	3	3	3
NumAssmLevels	5	5	7
NumComPerAssm	3	3	3
NumModules	1	1	1

Table 2: The XOO7 database parameters

We would like to highlight some of the important differences in the XOO7 benchmark database parameters from those of the OO7 benchmark:

- While there are seven levels of assemblies in the OO7 benchmark, we can only use up to five levels in the small and medium databases because most

existing XML tools have limitations in the volume of data they can manipulate.

- There are ten modules in the large database in the OO7 benchmark. In XOO7, this is fixed at one because we have chosen the module element as the root of the XML document.
- Since the XML dataset can be represented as a tree, the data size can be changed in two directions: depth wise and breadth wise. The depth of the tree can be varied by changing the value of *NumAssmLevels,* while the breadth of the tree can be controlled by the value of *NumAtomicPerComp* or *NumCompPerModule*. Furthermore, users can include different amounts of text data according to their application by changing the size of *Document* and *Manual*.

3.2 XOO7 Queries

As mentioned in the previous section, XML queries can be divided into three groups: traditional database queries, navigational queries, and document queries. Since there are recursive elements and elements at different levels of the tree in an XML file, this would involve queries on recursive data (ComplexAssembly) and queries on sub-elements (AtomicPart in CompositePart). We translate the traditional database queries from the OO7 benchmark and add additional queries to test the unique functionalities of XML.

The original OO7 benchmark consists of eight queries. We divide them into three groups as shown in Table 3. Group I involves lookups; Group II involves range queries; and Group III is composed of join queries. Q1 remains unchanged since it evaluates exact match. The <Document> element has been changed to incorporate the document-centric aspect of XML, hence we modify Q4 to return the first <para> element. This allows us to test element order preservation.

Since Q2, Q3 and Q7 differ only in the selectivity, we have combined them into one query and allow the user to tune the percentage of data to retrieve. Q5 and Q8 remain unchanged. Q5 evaluates the parent-child relationships while Q8 tests join

operations. Since ComplexAssembly is a recursive element, we modify Q6 to test the evaluation of regular path expression.

Group	ID	Description
I	Q1	Exact match lookup. Generate 5 random numbers for AtomicPart's MyID. Return the AtomicParts according to the 5 numbers.
	Q4	Path lookup. Generate 5 random titles for Document. Return the Documents according to the 5 titles.
II	Q2	Select 1% of AtomicParts (with a buildDate after 1990).
	Q3	Select 10% of AtomicParts (with a buildDate after 1900).
	Q7	Select all AtomicParts.
III	Q5	Single-level "make". Find the CompositePart if it is more recent than the BaseAssembly it uses.
	Q6	Multi-level "make". Find the CompositePart (recursively) if it is more recent than the BaseAssembly or the ComplexAssembly it uses.
	Q8	Ad hoc join. Join AtomicPart and Document on the docId of AtomicPart and the MyID of Document.

Table 3: The eight queries in the OO7 benchmark

As can be seen from Table 1 and Table 3, the queries generated from the OO7 benchmark do not cover a substantial number of the XML query functionalities we identified earlier. In fact, the majority of the queries focus on the data centric query capabilities. It is therefore imperative that the benchmark be extended to include queries to test document processing capabilities. Table 4 lists the complete set of queries in the XOO7 benchmark together with comments and coverage. We divide the queries into three groups. Group I comprises of traditional database queries. Most of these functionalities are already tested in many existing database benchmarks. Queries in Group II are navigational queries. They are designed because of the similarity between XML data and semi-structured data [1, 8]. These queries test how traversal is done in the XML tree. Group III comprises of document queries. These queries test the level of document-centric support given by systems and tests ability of maintaining order of data.

These three groups of queries are relevant to the characteristics of XML data and cover most of the functionalities. All of the queries are simple and each query covers only a few functionalities. The majority of the queries are supported in many XMLMS, which makes them very portable. Users can always choose a subset of queries to test on the features required in their applications. In the next section, we will demonstrate how we use the XOO7 benchmark to compare four different XML management systems.

Group	ID	Description	Comments	Coverage
I	Q1	Randomly generate 5 numbers in the range of AtomicPart's MyID. Then return the AtomicPart according to the 5 numbers	Simple selection. Number comparison is required.	R1, R2
	Q2	Randomly generate 5 titles for Documents then return the first paragraph of the Document by lookup on these titles.	String comparison and element ordering are required in this query.	R1, R2
	Q3	Select 5% of AtomicParts via buildDate (in a certain period).	Range query. Users can change the selectivity.	R4
	Q4	Join AtomicParts and Documents on AtomicParts docId and Documents MyID.	Test *join* operation. It is commonly tested in many benchmarks.	R9
	Q5	Randomly generate two phrases among all phrases in Documents. Select those documents containing the 2 phrases.	Text data handling. *Contain*[1]*s*-like functions are required.	R5
	Q6	Repeat query 1 but replace duplicated elements using their IDREF.	This is to test the ability to reconstruct a new structure.	R1, R18
	Q7	For each BaseAssembly count the number of documents.	Test *count* aggregate function.	R9, R10
	Q8	Sort CompositePart in descending order where buildDate is within a year from current date.	Test sorting and use of environment information.	R11, R14
	Q9	Return all BaseAssembly of type "type008" without any child nodes.	Users only require a single level of the XML document without child nodes included.	R18
II	Q10	Find the CompositePart if it is later than BaseAssembly it is using (comparing the buildDate attribute).	Compare attributes in parent nodes and child nodes. It tests the basic relation in XML data.	R13
	Q11	Select all BaseAssemblies from one XML database where it has the same "type" attributes as the BaseAssemblies in another database but with later buildDate.	Selection from multiple XML documents. It performs string and number comparison as well.	R9
	Q12	Select all AtomicParts with corresponding CompositeParts as their sub-elements.	Users may have various choices to store the same data, so they always require switching the parent nodes with the child nodes.	R18
	Q13	Select all ComplexAssemblies with type "type008".	Note ComplexAssembly is a recursive element, so it tests on regular path expression. An XMLMS should not browse every node.	R20
	Q14	Find BaseAssembly of not type "type008".	Robustness in presence of negation.	R8
	Q15	Return all Connection elements with length greater than Avg(length) within the same composite part without child nodes.	Avg function and group-by-like functions are required in this query.	R9, R10
	Q16	For CompositePart of type "type008", give 'Result' containing ID of CompositePart and Document.	Part of information of an element is required in this query. Test data transformation.	R17, R18
III	Q17	Among the first 5 Connections of each CompositePart, select those with length greater than "len".	Test element order preservation. It is a document query.	R9, R21
	Q18	For each CompositePart, select the first 5 Connections with length greater than "len".	Similar as above. It is to check if some optimization is done.	R9, R21

Table 4: Queries in the XOO7 benchmark

[1] *Contains* is a function defined in the XQuery specification. It checks if a string occurs in anther string.

4 Performance Study

We implement the XOO7 benchmark and demonstrate its applicability by using it to evaluate four XML query processing platforms: LORE, Kweelt, XENA, and a commercial product, we call it DOM-XPath[2]. The experiments are run on a SunOS 5.7 Unix system (333MHz), with 256MB RAM and 3GB disk space. The C++ implementation of XOO7 is available at http://www.comp.nus.edu.sg/~ebh/XOO7.html.

LORE [3], developed at Stanford University, is one of the earliest systems designed to store and query semi-structured data. It has subsequently been extended to query XML documents [22]. While LORE supports many needed features, it fails to support some important aggregate and update functions.

Kweelt [28], an example of a native XMLMS, was designed and implemented at the University of Pennsylvania. It stores data in flat files and hence favours the document-centric nature of XML. Its query language is based on Quilt [15], which in turn leverages the XPath standard.

Most of the existing XMLMS are XML-Enabled and built on top of relational or object-relational systems. They are used to publish data in XML and allow XML queries to be translated into SQL statements. XENA [34] is an XML-Enabled data management system developed at the National University of Singapore. It is implemented on top of MySQL database system. Based on the XML schema, XENA automatically stores the XML data into relational tables and retrieves data from the tables by converting an XPath query into several SQL queries.

DOM-XPath is a commercial XPath implementation using the DOM parser. The system we tested is the trial version of the product. DOM-XPath uses a DOM parser to load files into memory. It requires the whole document to be held in memory while it is being loaded. The developers claim that DOM-XPath meets all the requirements from XPath specification.

[2] We have not used the actual name of the product at the request of the developers

Our decision to choose the above XMLMS was primarily motivated by the easy availability of their source code and the detailed documentation on their implementations.

4.1 Data Conversion

In this set of experiments, we compare the time and space required for converting the XOO7 datasets to the format used by each of the XML management system. We use datasets of three sizes: 4.2MB, 8.4MB and 12.8 MB. The space utilization is determined from the amount of hard disk space used by each system for the various databases in the benchmark. Figure 3 shows the space and time requirements of the various XML data management systems we test.

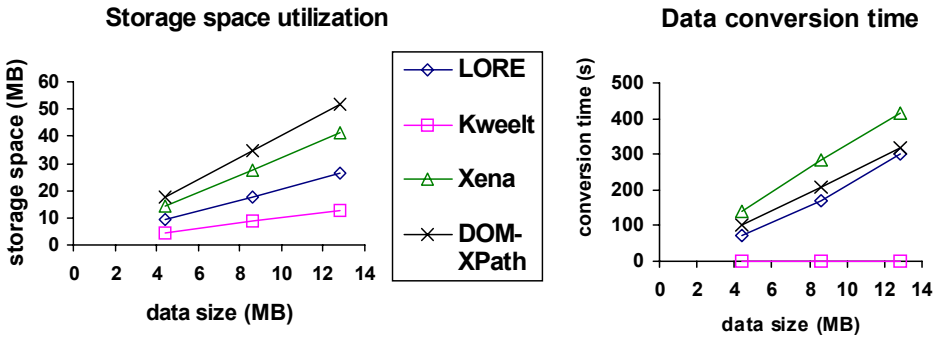

Figure 3: Storage space utilization and data conversion time

LORE creates Dataguides [23] for the datasets to facilitate efficient query processing. Dataguides are a concise and accurate summary of all paths in the database that start from the root which are much larger in size compared to the original XOO7 dataset. Hence LORE requires almost double the space of the original XML dataset.

Not surprisingly, Kweelt is most efficient compared to other systems in terms of space usage. Kweelt queries the ASCII file directly and does not need to convert the XML data into another format. The storage space it needs is therefore the same as the size of the original XML data.

XENA stores XML data in MySQL tables. Although the conversion from XML to relational tables results in removing the many tags around the XML data, XENA has to generate a number of relational tables in order to capture properly the structure of the XML data. In fact, XENA creates two groups of tables. The first group is based on the XML schema with one table per XML element; the second group of tables contains meta-data for the management of the base tables. Hence XENA requires almost three times the space of the actual XML data.

DOM-XPath provides tools to convert XML data into its internal format. The system creates three binary files for an XML dataset. One of the files is a proprietary database that preserves the Native XML structure by storing the entire document tree of the dataset. As shown in Figure 3, DOM-XPath takes up the most space compared to the other three XML management systems.

Since Kweelt processes directly the raw XML data, no extra time is needed for data conversion. XENA takes the largest amount of time to convert the XML model to a relational data model. LORE requires time for generating Dataguides and we assume the DOM-XPath is also generating indexes, a task that takes up time.

4.2 Query Response Time

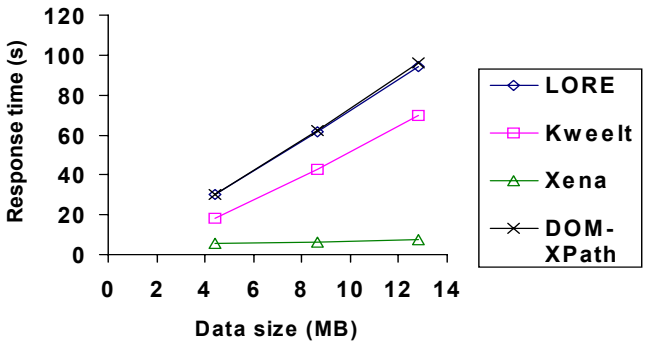

Figure 4: Response time results for relational queries.

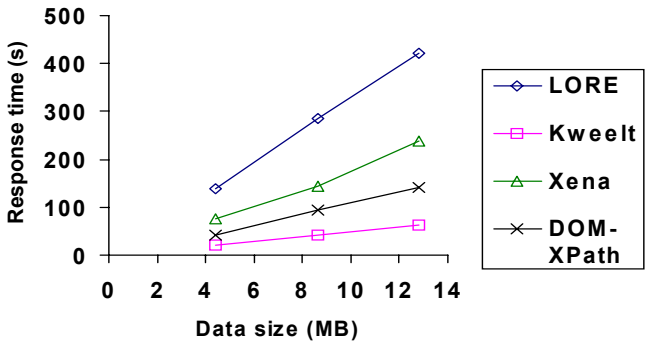

Figure 5: Response time results for navigational queries.

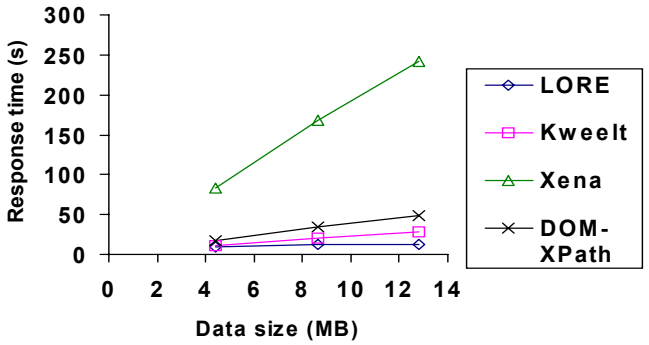

Figure 6: Response time results for document-oriented queries.

In this set of experiments, we run the 18 XOO7 benchmark queries on the four systems. Each query is run 10 times and the average response time of the queries are recorded. Having divided the XOO7 queries into three groups, Relational (traditional databasean), Document and Navigational queries, we aggregate the response times for the queries in each group and compute the average response time. Note that there are some queries are not supported in all the systems because the function has not been implemented, like sort in DOM-XPath and XENA. Figure 4 shows the performance of the four systems for relational queries. The performance of the systems for navigational and document queries are given in Figures 5 and 6 respectively.

For the relational queries, XENA, the XML-enabled database system, leverages the query processing power of the relational database engine (MySQL) and yields the best response time. In addition, its performance for processing this group of queries is the least affected by the increase of the data size. On the other hand,

LORE and DOM-XPath perform equally bad (their graphs overlap in Figure 4). The response time of LORE is not as good as expected. We suspect that this may be caused by the data type coercion implemented in LORE. From the available documentation, it seems that LORE checks the data type very frequently, leading to bad performance even for relational queries. Kweelt and DOM-XPath are XML-Native implementations. They store elements belonging to the same entities in the DTD discontinuously, which subsequently requires more fault pages during data retrieval.

For navigational queries, XENA requires larger response times as the query target contains lower level elements in the XML tree. This is because XENA requires more time to reconstruct the XML data tree from the various relational tables. Its performance also significantly degrades with increase in data size. LORE yields particularly bad performance, most likely due to the fact that the OEM data model is unable to handle parent-child relations efficiently. The primary reason for the poor performance of XENA and LORE is that these two systems store the XML data according to entities. On the other hand, Kweelt and DOM-XPath store the XML data natively, thus capturing the relations between parent nodes and child nodes in the original data. From the experiment result, we see that Kweelt is able to process navigational queries most efficiently.

XENA proves to be inadequate for processing document queries. This result is expected since XENA fetches out all the target elements first and then selects those elements that satisfy the conditions according to the index in the preserved order. Kweelt and the XPath implementation perform relatively better for those queries whose results contain multi-level nodes. LORE performs the best in these queries mostly because of its ability to use Dataguides as shortcuts to extract the relevant elements directly.

To conclude, the XML-enabled relational database system performs the best when processing simple relational queries. The native-XML implementation Kweelt processes more efficiently navigational queries and document queries when it performs poorly on relational ones.

5 Related Work

In this section we discuss and compare three benchmarks currently available that test XML systems for their query processing abilities: XMach-1 [9], XMark [29] and XOO7. We will first describe XMach-1 and XMark before comparing them with XOO7.

5.1 The XMach-1 Benchmark

The XMach-1 benchmark [13] is a multi-user benchmark designed for B2B applications. The benchmark models a web application using XML as follows: XML manipulations are performed through an XML-based directory system sitting on top of XML files. These XML files constitute the data sets. It assumes that size of the data files exchanged will be small (1-14 KB).

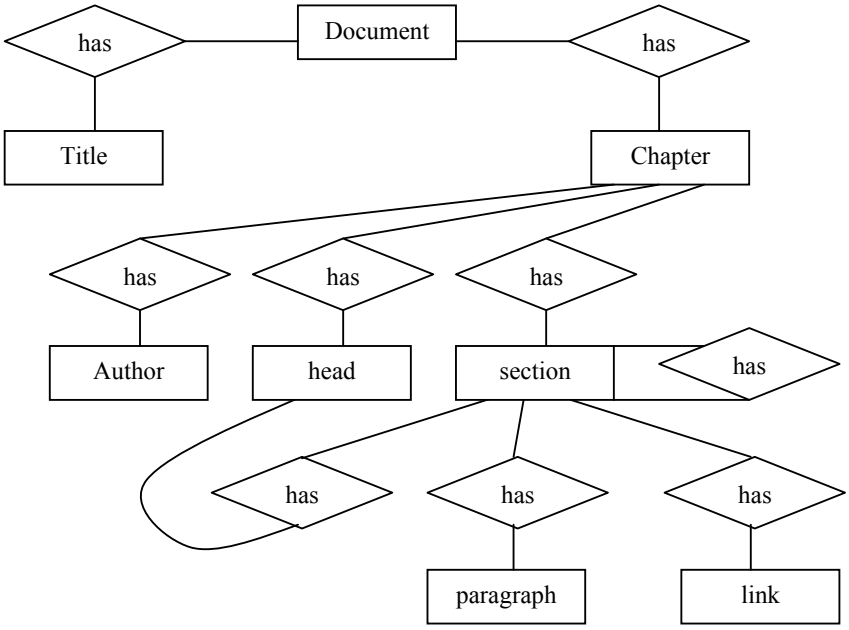

Figure 7: ERD of XMach-1 database.

Figure 7 shows the ER diagram of XMach-1 database. Attributes have been omitted to simplify the diagram. Each of the XML file simulates an article with elements such as title, chapter, section, paragraph etc. The element section is a recursive element. The text data are from taken from natural language text. Users can vary an XML file size by modifying the number of the article elements. The size of the database is controlled by varying the number of XML files.

Xmach-1 evaluates standard and non-standard linguistic features such as insertion, deletion, querying URL, and aggregate operations. The benchmark consists of eight queries and two update operations as shown in Table 5. We divide the queries into four groups according to the common characteristics they capture. Group I contains simple selection and projection queries with comparisons on element or attribute values. Group II queries require the systems to use the element order to extract results. Group III tests on aggregation functions and use of metadata information. Group IV comprises of update operations. We note that while the proposed workload and queries are interesting, the benchmark has not been applied and no performance results is available.

Group	ID	Description	Comment
I	Q1	Get document with given URL.	Return a complete document (complex hierarchy with original ordering preserved).
	Q2	Get doc_id from documents containing a given phrase.	Text retrieval query. The phrase is chosen from the phrase list.
	Q5	Get doc_id and id of parent element of author element with given content.	Find chapters of a given author. Query across all DTDs/text documents.
II	Q3	Return leaf in tree structure of a document given by doc_id following first child in each node starting with document root.	Simulates exploring a document with unknown structure (path traversal).
	Q4	Get document name (last path element in directory structure) from all documents that are below a given URL fragment.	Browse directory structure. Operation on structured unordered data.
	Q6	Get doc_id and insert date from documents having a given author (document attribute).	Join operation.
III	Q7	Get doc_id from documents that 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 inserted documents having an author attribute.	Needs count, sort and join operations and accesses metadata.
IV	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 does no longer exist in the web.

Table 5: Queries specified in the XMach-1 Benchmark.

5.2 The XMark Benchmark

The XMark benchmark consists of an application scenario that models an Internet auction site. The ER diagram of the database is shown in Figure 8. We have

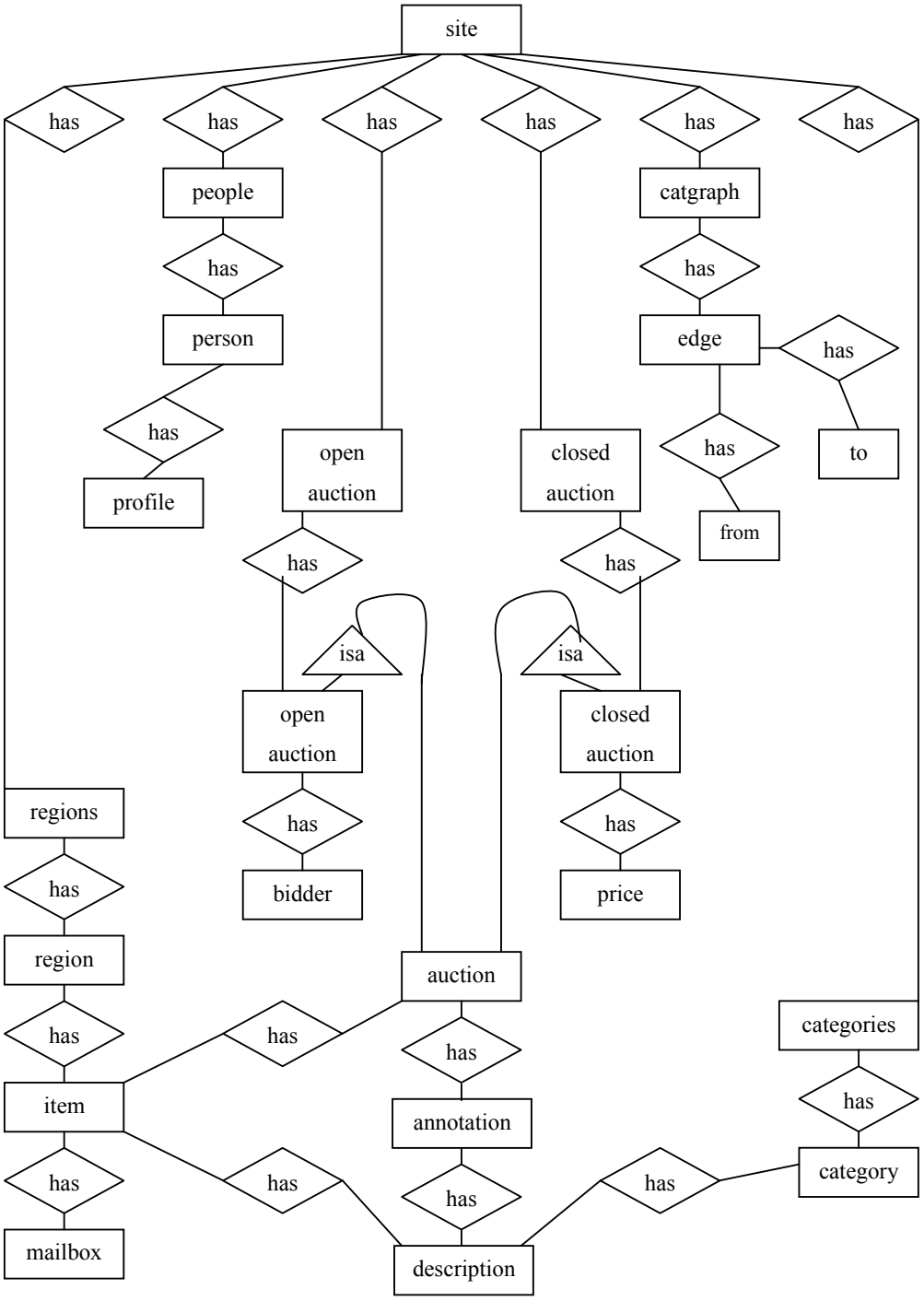

Figure 8: ERD of XMark database.

omitted the attributes to simplify the diagram. The main entities are: item, person, open auction, close auction, and category. Items are the objects that are on for sale or sold already; person entities contain sub-elements like name, email address, phone number etc.; open auctions are auctions in progress; close auctions are the finished auctions; categories feature a name and a description. The XMark benchmark enriches the references in the data, like the item IDREF in an auction

element and the item's ID in an item element. The text data used are the 17000 most frequently occurring words of Shakespeare's plays. The standard data size is 100MB with scaling factor 1.0 and users can change the data size by 10 times from the standard data (the initial data) each time.

Group	ID	Description	Comment
I	Q1	Return the name of the person with ID 'person0' registered in North America.	Checking ability to handle strings with a fully specified path.
	Q5	How many sold items cost more than 40.	Check how good a DBMS performs since XML model is document oriented. Checks for typing in XML.
	Q14	Return the names of all items whose description contains the word 'gold'.	Text search but narrowed by combining the query on content and structure.
II	Q2	Return the initial increases of all open auctions.	Evaluate cost of array lookups. Authors site Authors site that a relational backend may have problems determining the first element. Essentially query is about order of data which relational systems lack.
	Q3	Return IDs of all open auctions whose current increase is at least twice as high as initial.	More complex evaluation of array lookup.
	Q4	List reserves of those open auctions where a certain person issued bid before another person.	Querying tag values capturing document orientation of XML.
	Q11	For each person, list the number of items currently on sale whose price does not exceed 0.02% of person's income.	Value based joins. Authors feel this query is a candidate for optimizations.
	Q12	For each richer-than-average person, list the number of items currently on sale whose price does not exceed 0.02% of the person's income.	As above
	Q17	Which persons don't have a homepage?	Determine processing quality in presence of optional parameters.
III	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 implementation.
	Q8	List the names of persons and the number of items they bought.	Check efficiency in processing IDREFs. Note a pure relational system would handle this situation using foreign keys.
	Q9	List the names of persons and the names of items they bought in Europe. (joins person, closed auction, item)	As above
	Q10	List all persons according to their interest use French markup in the result.	Grouping, restructuring and rewriting. Storage efficiency checked.
	Q13	List names of items registered in Australia along with their descriptions	Test ability of database to reconstruct portions of XML document.
	Q15	Print the keywords in emphasis in annotations of closed auctions.	Attempt to quantify completely specified paths. Query checks for existence of path.
	Q16	Return the IDs of those auctions that have one or more keywords in emphasis.	As above.
IV	Q18	Convert the currency of the reserve of all open auctions to another currency.	User defined functions checked.
	Q19	Give an alphabetically ordered list of all items along with their location.	Query uses SORTBY, which might lead to an SQL-ish ORDER By and GROUP BY because of lack of schema.
	Q20	Group customers by their income and output the cardinality of each group.	The processor will have to identify that all the subparts differ only in values given to attribute and predicates used. A profile should be visited only once.

Table 6: Queries specified in the XMark Benchmark

XMark provides 20 XQuery challenges designed to cover the essentials of XML query processing. These queries have been evaluated on an internal research prototype, Monet XML, to give a first baseline. Table 6 shows the XMark queries and our comments. Based on the query functionality, we divide the queries into four groups. Group I contains simple relational queries with comparison on various types of data values. Queries in Group II are document queries which preserves element order. Q11 and Q12 are queries to test join operation. Group III contains navigational queries. Q8 and Q9 check the navigational performance in presence of ID and IDREF. Q15 and Q16 test long path traversals. Queries in Group IV require aggregate handling and sort operations.

5.3 Comparison

The XOO7, XMach-1, and XMark benchmarks have been designed to investigate the performance of different XMLMS. In terms of database description, XMach-1 is the most straightforward. It models a document, which is easy for users to understand. XMark simulate an auction scenario, which is a very specialized interest area, containing elements and attribute that may be difficult for users to understand. XOO7 adapts and extends the OO7 database on describing a Module, which can be understood easily.

XOO7 and XMach-1 have similar database structure. Both of them contain recursive elements, *section* in XMach-1 and *ComplexAssembly* in XOO7. However, the structure of XMach-1 is not deeply nested compared to XOO7. Although the XMark database does not contain recursive elements, it is nevertheless more complex than both XOO7 and XMach-1, containing many elements.

All the three benchmark datasets can be scaled by varying the number of elements in the XML files. In fact, XOO7 allows users to change the file size both depth-wise and breadth-wise. In addition, XMach-1 controls the database size by changing the number of XML files.

Coverage	XMach-1	XMark	XOO7
R1	√	√	√
R2	√	√	√
R3			
R4	√	√	√
R5	√	√	√
R6			
R7		√	
R8		√	√
R9	√	√	√
R10	√	√	√
R11		√	√
R12			
R13		√	√
R14			√
R15	√		
R16			
R17		√	√
R18		√	√
R19			
R20			√
R21		√	√

Table 7 Comparing benchmarks over XML query language functionalities.

To compare the queries proposed in the three benchmarks, we construct a table to show the coverage of the queries. From Table 7, we can see that the majority of the XML query language functionalities have been covered by the XMark and XOO7 queries.

We observe that the queries in the XOO7 and XMach-1 benchmarks are straightforward and each query covers only a few functionalities. Hence, even if an XMLMS does not support some features, most of queries are still executable on the system. In contrast, XMark proposes many complex queries that cover more than one functionalities. It is quite possible that many queries in XMark may not be executable because one or two of the tested features have not been implemented in the system. It is clear that XMach-1 and XOO7 have better portability.

6 Conclusion

In this paper, we have presented the XOO7 benchmark for evaluating the query processing capabilities for XML management systems. We used the well-known OO7 benchmark for object-oriented database systems as the starting point, and tailored the original database to capture the unique features in XML, namely, the document aspect. We have designed the benchmark queries to substantially cover the XML query functionalities. As a result, the XOO7 benchmark comprises of three categories of queries: the traditional database queries, navigational queries, and document queries. We compared XOO7 to two existing XML benchmarks and showed that XOO7 conforms better to the four benchmark specification criteria: relevance, portability, scalability and simplicity. Finally, we implemented XOO7 and demonstrate its applicability by using it to evaluate the performance of four different XML management systems. For future work, we plan to extend XOO7 to involve multi-user query capabilities.

References

[1] S. Abiteboul, P. Buneman, and D. Suciu. Data on the Web: From Relations to Semistructured Data and XML, Morgan Kaufman Publishers, 2000.

[2] S. Abiteboul and S. Grumbach. COL: A Logic-Based Language for Complex Objects. *In Proc. of Int. Conf. On Extending Database Technology (EDBT)*, pp 271-293, 1988.

[3] S. Abiteboul, D. Quass, J. McHug, J. Widom, and J. Wiener. The LOREL Query Language for Semistructured Data. *International Journal on Digital Libraries*, 1(1):68, April 1997.

[4] S. Abiteboul and M. Scholl. From Simple to Sophisticate Languages for Complex Objects. *Data Engineering Bulletin* 11(3), pp 15-22, 1988.

[5] D. Bitton, D. J. DeWitt and C. Turbyfill. Benchmarking database systems: A systematic approach. *Proceedings of the Very Large Database Conference*, 1983.

[6] R. Bourret. XML databae products, May 2001.
http://www.rpbourret.com/xml/XMLDatabaseProd.htm/.

[7] A. Bonifati and S. Ceri. Comparative Analysis of Five XML Query Languages. *SIGMOD Record*, 29(1): 68-79, 2000.

[8] P. Buneman. Semistructured Data. In Proc. of Symposium on Principles of Database Systems (PODS), pp 117-121, 1997.

[9] T. Bohme and E. Rahm. XMach-1: A Benchmark for XML Data Management. http://dbs.uni-leipzig.de/projekte/XML/XmlBenchmarking.html

[10] T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. Extensible Markup Language (XML) 1.0 (Second Edition), 2000.
http://www.w3.org/TR/2000/REC-xml-20001006/.

[11] M. J. Carey, D. J. DeWitt, and J. F. Naughton. The OO7 benchmark. *ACM SIGMOD Int. Conf. On Management of Data*, pp. 12-21, Washington, 1993.

[12] M. J. Carey, D. J. DeWitt, J. F. Naughton, et al. The Bucky Object-Relational Benchmark. *Proc. ACM SIGMOD Conference*, pp.135-146, Tucson, AZ, 1997

[13] D. Chamberlin, P. Fankhauser, M. Marchiori, and J. Robie. XML Query Requirements. W3C Working Draft 15 August 2000.

http://www.w3.org/TR/xmlquery-req.

[14] D. Chamberlin, D. Florescu, J. Robie, and J. Sim. XQuery: A Query Language for XML, 2000. http://www.w3.org/TR/xmlquery/.

[15] D. Chamberlin, J. Robie, and D. Florescu. Quilt: An XML Query Language for Heterogeneous Data Sources. *ACM SIGMOD Workshop on Web and Databases (WebDB'00)*, Dallas, 2000.

[16] B. Chang, M. Scardina, K. Karun, S. Kiritzov, I. Macky, A. Novoselsky, and N. Ramakrishnan. ORACLE XML Handbook (184-190), 2000.

[17] J. Clark and S. DeRose. *XML Path Language (XPath)*. W3C, 1999. http://www.w3.org/TR/xpath.

[18] S. DeRose, R. Daniel, and E. Maler. *XML Pointer Lanuage (XPointer)*. W3C, 1999. http://www.w3.org/TR/WD-xptr.

[19] S. DeRose, E. Maler, D. Orchard, and B. Trafford. *XML Linking Language (XLink)*. W3C, 2000. http://www.w3.org/TR/xlink.

[20] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, D. Suciu. XML-QL: A Query Language for XML. http://www.w3.org/TR/NOTE-xml-ql/.

[21] M. Fernandez, J. Simeon, P. Wadler. XML Query Languages: Experiences and Exemplars, 2000.

http://www-db.research.bell-labs.com/user/simeon/xquery.html

[22] R. Goldman, J. McHugh, and J. Widom. From Semistructured Data to XML: Migrating the Lore Data Model and Query Language, *ACM SIGMOD Workshop on Web and Databases (WebDB'99)*, 1999.

[23] R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. *Proc. of Intl. Conf. on Very Large data Bases*, Delphi, Greece, August 1997. In press.

[24] J. Gray. The Benchmark Handbook: For Database and Transaction Processing Systems, 2nd Edition, Morgan Kaufmann Publishers, Inc., 1993.

[25] OLAP Council: APB-1 OLAP Benchmark Release II.

http://www.olapcouncil.org/research/bmarkly.htm, 1998.

[26] P. E. O'Neil. Database Performance Measurement. In the *Computer Science and Engineering Handbook*, CRC Press, 1997: 1078-1092.

[27] J. Robie, J. Lapp, D. Schach. XML Query Language (XQL), 1998. http://www.w3.org/TandS/QL/QL98/pp/xql.html

[28] A. Sahuguet, Kweelt: More than just "yet another framework to query XML!", SIGMOD demonstration session, Santa Barbara, California, May 2001, pp 605. http://db.cis.upenn.edu/Kweelt/.

[29] A. R. Schmidt, F. Waas, M. L. Kersten, D. Florescu, I. Manolescu, M. J. Carey, R. Busse. The XML Benchmark Project. Technical Report INS-R0103, CWI, Amsterdam, The Netherlands, April 2001.

[30] M. Stonebraker, J. Frew, K. Gardels, J. Meredith. The Sequoia 2000 Benchmark. *SIGMOD Conference*, pp. 2-11, 1993.

[31] Transaction Processing Performance Council. http://www.tpc.org/, 2000.

[32] C. Turbyfill, C. Orji, D. Bitton. ASAP: A comparative relational database benchmark. *Proceedings Compcon 1989*.

[33] XML Spy Company. http://www.xmlspy.com, 2001.

[34] Y. Wang and K. L. Tan. A Scalable XML Access Control System. *10th World Wide Web Conference*, May 2001.