XML BENCHMARKS PUT TO THE TEST

Ullas Nambiar, Zo′e Lacroix

Arizona State University
(zoe.lacroix, mallu)@asu.edu

St′ephane Bressan, Mong Li Lee, Yingguang Li

National University of Singapore
(steph, leeml, liyinggu)@comp.nus.edu.sg

Abstract: *The effectiveness of existing XML query languages has been studied by many who focused on the comparison of linguistic features, implicitly reflecting the fact that most XML tools exist only on paper. In this paper, with a focus on efficiency and concreteness, we propose a pragmatic first step toward the systematic benchmarking of XML query processing platforms. We begin by identifying the necessary functionalities an XML data management system should support. We review existing approaches for managing XML data and the query processing capabilities of these approaches. We then compare three XML query benchmarks XMach-1, XMark and XOO7 and discuss the applicability, strengths and limitations of these benchmarks. We highlight the bias of these benchmarks towards the data centric view of XML and motivate our selection of XOO7 to extend with document centric queries. We complete XOO7 to capture the information retrieval capabilities of XML management systems. Finally we summarize our contributions and discuss future directions.*

1. Introduction

Introduced as a schema-less, self-describing data representation language, eXtended Markup Language (XML) quickly emerged as the standard for information interchange on the Web. XML was introduced as a subset of SGML (Standardized General Markup Language) designed 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. XML's development was not furthered directly by the mainstream database community, yet database researchers actively participated in developing standards around XML, particularly query languages for XML. This led to the development of query languages such as XML-QL [16], LOREL [3], XSL [5], XQL [25] and others. These languages designed by the database community are biased toward the data centric view of XML. But XML was developed primarily as a document markup language that would be more powerful than HTML yet less complex than SGML. This leads us to question whether the query languages capture the whole essence and power of semistructured data representation. 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 and data centric characteristics of XML.

The semistructured nature of XML poses many challenges, query optimization being one of them. A naive evaluation of a query would entail traversing the entire document/database to retrieve the required information, which can be prohibitive in terms of cost and time. The edge-labeled data model of XML provides opportunities for issuing path queries that harness the power of structural recursion requiring multiple scans of the database in the absence of efficient indexing structures. Hence researchers have focused on developing indexing schemes for semistructured data and XML in particular in recent years. Dataguides [19], T-indexes [22] and Reversed Dataguides [21] have been proposed to index semistructured data. ToXin [24] has been recently proposed for indexing XML data. All these indexing schemes are suitable for some facet of XML data usage. Hence it is difficult to categorize any one scheme or a subset of the above schemes as being ideal for query processing needs. Yet it goes without saying that any XML query system that uses these schemes would do better query optimization and processing, at least for certain types of XML queries, compared to naive evaluation strategies.

At this juncture a user intending to setup a XML based data interchange/storage system would have to decide which of the XML query languages/processing systems to base her system on. What indexing schemes to use, which schema specifications to follow etc. In the absence of *standardization among standards*, a benchmark across which various XML query processing systems can be tested and compared could greatly help the user. This benchmark should be useful to not just assess the power of query languages but rather the XML management systems[1] which use these query languages and related optimization techniques.

The complexity in developing a generic benchmark comes from the myriad and yet unknown domains and scenarios, in which an XML query processing system could be present. Stand alone XML databases, E-commerce systems, web catalogs, data integration systems are but some of the applications where such query processing could be useful. As can be easily judged none of the above mentioned domains seem to be similar in expressive power and usage. Hence metrics like performance and cost vary from system to system and therefore developing a generic benchmark at this juncture would be highly premature. An application specific benchmark e.g. for e-commerce systems, multi-user web repositories etc. would be too narrow and limited in use. The need is for a benchmark which could be used to test systems usable in a number of scenarios and yet not too generic to be unable to define any good and precise metric for evaluation.

In this paper, we identify desirable characteristics of an XML query processing system. Our motivation is to develop a domain-specific benchmark for comparing querying capabilities of XML database systems. In particular, we are interested in determining a benchmark for comparing XML query systems that store and use data from local database/document collections. Our focus is on the query processing power of systems under test. The benchmark should allow users to compare existing systems and also give desirable properties for future XML query processing systems. These characteristics would further the research in query optimization in these systems. XMach-1 [7] and XMark [27] are benchmarks already proposed for XML data stores. We perform a comparative study of these benchmarks with our benchmark XOO7 [10] in terms of functionalities covered. Although storage and retrieval techniques of underlying data are

[1] Throughout the paper we 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.

of high importance while considering a system we do not propose any metric to measure the efficiency of storage and retrieval.

Section 2 addresses the expected functionalities of XML query languages. We address the need for a benchmark for current XML data systems in section 3 and list the XML manipulation capabilities. We perform a comparative study of XML data benchmarks in section 4. In section 5 we extend XOO7 by adding queries covering document oriented XML data processing functionalities. We conclude in section 6 by highlighting our contributions and the possible extensions of this work.

Id	**Description**
R1	Query all data types and collections of possibly multiple XML documents.
R2	Allow data-oriented and 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.

Table 1: Functionalities of XML Query Languages

2. XML Query Functionalities

The performance of the implementations of query languages for XML depends strongly on their expressive power: the functionalities they provide. Indeed, some of the expected functionalities may affect significantly the efficiency of the system. Many languages claim to be XML query languages, however their functionalities vary dramatically. Some languages such as LOREL or XSU offer the functionalities provided by traditional data oriented query languages such as SQL. Others focus on XML integration and restructuring with additional data-oriented functionalities such as join, nesting and aggregation for XML-QL or partial or none of these data-oriented functionalities for XSL and XQL. More recently, languages such as Quilt [15] and XQuery extend the data-oriented approach with functionalities to handle XML documents.

The design of a benchmark for XML query languages shall address the performance issues connected to the characteristics of XML query languages, thus their functionalities. XML query language functionalities were addressed in a comparative analysis of XML query Languages [8] and listed as ”must have” in the requirements [17] published by the W3C XML Query Language working group. Table 1 enumerates all these requirements.

An XML query language should support the manipulation and extraction of data from multiple documents (R1), by accessing and combining different parts within documents (R9), querying the DTD, XML Schema (R1) or along paths (R13), by using data types (R1) or evaluating conditions over textual elements (R5). XML queries should support implicit order (order of elements within the XML document) as well as explicit order (order defined in the schema)(R2). Complex Data models can be defined using the XML data model, in par with this, a XML query language should therefore be able to work with differing data models (R4) all of which would have a common origin. Since XML allows semistructured data representation, NULL values may be present. A missing element may or may not be easily represented as NULL valued element but vice versa may be TRUE and hence NULL value manipulation will take on additional complexity (R7). Support for quantification and negation in queries (R6) is needed.

XML can capture structured information and hence a XML query language should have the expressibility of a structured query language like SQL for relational databases. Therefore such a language should support various types of join operations (R9), aggregation (R10), sorting (R11). Unlike XML, relational model disregards the order. Hence sorting and aggregation increase in complexity when order and document structure need to be preserved in some form (R17). The language must be capable of generating new XML structures and transforming one XML structure to another (R18). Since queries can be along paths and paths can consist of recursive calls to itself or sub paths, structural recursion should be supported (R20). A query on a database may change the underlying data. Hence the query language should provide methods for updating the underlying database (R15).

3. Benchmarking an XML query processor

XML is poised to take over as the lingua franca of the Web, primarily as a data exchange format. This does give rise to mind boggling visions of large and connected data processing systems storing and manipulating the data in XML format. At the time of writing, scores of vendors are making their products XML aware. XML stores and management systems are being offered by most data management vendors. At a time when success/failure of a business is closely related to IT and information management, proliferation of tools to manage data raises issues of evaluating their reliability and performance. A benchmark to identify the important performance parameters for these various systems under varying levels of load and differing environments has thus become a necessity.

3.1 XML Management Systems

Broadly speaking, one can classify XML documents into two categories: data-centric and document-centric. Sales orders, scientific data, employee records etc. are examples of data-centric XML documents. Similar to traditional databases, the physical structure especially order of sibling elements is unimportant in such documents. On the other hand, document-centric documents are highly unstructured, have mixed content and their physical structure is important to ascertain the information contained in them. Web pages, and marketing brochures are examples of such documents. A relational or object-oriented database is suitable for storing and retrieving the data-centric XML documents. Native XML databases or content management systems are better suited for storing and retrieving document-oriented XML data.

From the above discussion we divide current XML management systems into XML-Enabled databases and Native XML databases. XML-Enabled databases (usually relational) contain extensions (either model- or template-driven) for transferring data between XML documents and themselves and are generally designed to store and retrieve data-centric documents. Based on storage techniques used, native XML databases fall into two categories: Text-based and Model-based storage. Text-based systems store the entire document in text form and provide limited storage/retrieval capabilities similar to traditional databases. Model-based databases store binary model of the document such as DOM (Document Object Model) in an existing or custom data store. There are three main differences between a Native XML database and an XML-Enabled database:

- Native XML database preserves physical structure e.g. CDATA sections,comments, PIs, DTDs, etc. Theoretically XML-enabled database can do so too, but practice tells a different story so far.
- Schemaless data can be stored in native XML databases. Attempts to learn schema have been made, to try and alleviate this problem for XML-enabled databases.
- XPath, DOM etc., XML related API is a must for accessing data in native XML. On the other hand, XML-enabled systems can offer direct access to the data, such as through ODBC.

For a detailed classification of XML management products refer to [9].

3.2 Issues in current management systems

Existing XML databases or more specifically XML-aware database systems are built on top of the relational or object-relational model. Kweelt [26], a proposed implementation of Quilt [15], uses flat files to store the data and hence works with the document centric nature of XML. LORE and XSU, transform and store the XML data in a database. The latter systems use XML to project and publish the data stored in the databases. However, XML data is ordered unlike relational data. Hence for systems claiming to be XML-aware an obvious drawback is dealing with the implicit order of the data. Order information is lost while converting from XML to relational. Thus any view generated by such systems will have a pseudo-order of the elements. On the other hand, with most of the systems having a database backend, it is but natural to expect such systems to display similar processing capabilities for XML queries e.g. aggregates, nested queries, count, grouping etc. Relational databases have efficient storage and retrieval techniques and can make use of various indexing mechanisms to evaluate a query. But these systems are not really able to optimize queries that exploit the document centric properties of XML data like text search, implicit and explicit order management etc. Mechanisms for efficient storage and retrieval are difficult to implement given the computational complexity for XML data manipulation. Hence data systems that use flat files/XML documents are most certain to end up doing a naive execution of the XML queries. Also native XML databases may show varying performance based on the storage method used. Native databases using text-based storage strategies are able to return entire documents in text form. Model-based databases on the other hand can return fragments of documents or combine fragments from different documents. Again which model of native database is faster, is situation specific. An ideal native database is one which would display characteristics of both models. But these systems are certain to maintain the true order of the data. Thus we are faced with systems that solve the data centric or the document centric view in isolation. We believe a true XML database system should be able to give good performance in the presence of both data oriented and information retrieval type queries. Also queries that exploit both features simultaneously are also feasible. Since existing (traditional) database systems are data centric we started developing XOO7 to evaluate such systems and we then propose extensions to XOO7 to include evaluation of document centric query processing capabilities.

4 Benchmarks for XML Management Systems: A Comparison

Semistructured data models and query languages have been studied widely in [1], [12]. In [18] several storage strategies and mapping schemes for XML data using a relational database are explored. Domain-specific database benchmarks for OLTP (TPC-C), decision support (TPC-H, TPC-R, APB-1), information retrieval, spatial data management (Sequoia) etc. are available [20]. In this paper we concentrate solely on the query processing capabilities of the systems we test.
XMach-1, XMark and XOO7 are the three benchmarks currently available that test XML systems for their query processing abilities. For detailed information about the data and schema used by these benchmarks refer [23].

ID	Description	Comments	Coverage
Q1	Get document with given URL.	Return a complete document with the original structure.	R1
Q2	Get doc-id from documents containing a given phrase.	Text retrieval query. The phrase is chosen from the phrase list.	R5
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).	R2
Q4	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.	R2
Q5	Get doc-id and id of parent element of author element with a given content.	Find chapters of a given author. Query across all DTDs or test documents.	R4
Q6	Get doc-id and insert date from documents having a given author (document attribute).	Join Operation.	R9
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.	R10
Q8	Get doc-id from the last 100 inserted documents having an author attribute.	Needs count, sort and join operations and accesses metadata.	R10
M1	Insert document with given URL.	The loader generates a document and URL and sends them to the HTTP server.	R15
M2	Delete a document with given doc-id.	A robot requests deletion.	R15

Table 2: Queries specified in XMach-1 Benchmark.

4.1 XMach-1

XMach-1 tests multi-user features provided by the systems. The benchmark is modeled for a web application using XML data. It evaluates standard and non-standard linguistic features such as insertion, deletion, querying URL and aggregate operations. Although the proposed workload and queries are interesting, the benchmark has not been applied and no results exist. Table 2 lists the queries provided as part of the benchmark. We have grouped the queries into 4 groups based on common characteristics they capture. Group I contains simple selection and projection queries with comparisons on element values. The values may be of type data or text. Queries under Group II require the systems to use the element order to extract results. Query Q6 performs a join operation where maintaining the order of data is crucial. Aggregate function processing and use of metadata information is necessitated by queries of Group III. Queries M1 and M2 are grouped together as Group IV since they perform insert and update. It is interesting to note that insert/update/delete operation semantics are yet to be identified fully for XML data model. The last column of Table 2 points to the XML functionalities listed in Table 1.

4.2 XMark

Xmark developed under the XML benchmark project at CWI, is a very recent benchmark proposed for XML data stores. The benchmark consists of an application scenario which models an Internet auction site and 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. Once again attributes have been omitted to simplify the diagrams. Table 3 shows the functionalities covered by queries given in XMark. We group these queries under 5 groups. Queries under Group I are path queries exploiting schema information. Specifically Q1 tests string manipulation ability, Q6 tests efficient path query processing, Q15 and Q16 tests systems ability to ascertain existence of paths while generating results. Group II makes use of order of elements to test efficiency in processing array lookup and test processing queries. Group III checks performance in presence of IDREFS. Group IV uses value based joins to test how good are the optimization techniques being used. Group V contains queries that require aggregate handling, sort operations and reconstruction/merging of parts of documents.

ID	Description	Comments	Coverage
Q1	Return the name of the person with ID 'person0' registered in North America.	Checking ability to handle strings with a fully specified path.	R1
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 problem determining the first element.	R2
Q3	Return IDs of all open auctions whose current increase is at least twice as high as initial.	More complex evaluation of array lookup.	R2
Q4	List reserves of those open auctions where a certain person issued bid before another person.	Querying tag values capturing document orientation of XML.	R4
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.	R2
Q6	How many items are listed on all continents ?	Test efficiency in handling path	R4

		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 relational system would handle this using foreign keys.	R13
Q9	List the names of persons and the names of items they bought in Europe (Joins person, closed_auction, item)	Same as Q8.	R13
Q10	List all persons according to their interest. Use French markup in the result.	Grouping, restructuring and rewriting. Storage efficiency checked.	R10
Q11	For each person, list the number of items on sale whose price does not exceed 0.02% of his income.	Value based joins. Authors feel this query is a candidate for optimizations.	R9
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.	R9
Q13	List names of items registered in Australia along with their descriptions.	Test ability of database to reconstruct portions of XML document.	
Q14	Return the names of all items whose description contains the word 'gold'.	Text search narrowed by combining the query on content and structure.	R2, R5
Q15	Print the keywords in emphasis in annotations of closed auctions.	Attempt to quantify completely specified paths. Query checks for existence of path.	R8
Q16	Return the IDs of those auctions that have one or more keywords in emphasis.	As above.	R8
Q17	Which persons don't 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.	User defined functions checked.	
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. The execution engine may produce an sorted result from the data.	R10
Q20	Group customers by their income and output the cardinality of each group.	A processor 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 3: XMark Benchmark Queries.

4.3 XOO7

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 attempts to provide a framework for handling semistructured data, it encompasses most of the modeling features of complex object models [2, 4]. There are straightforward correspondences between the object-oriented schemas and instances and XML DTDs and data. XOO7 was designed keeping in mind these similarities in data model of XML and object-oriented approach. XOO7 is an adaptation of the OO7 Benchmark [13]. Section 5 gives further motivations to our approach of designing a benchmark based on OO7, with a detailed analysis of the similarities between XML and object-oriented data model.

ID	Description	Coverage
Group I		
Q1	Randomly generate 5 numbers in the range of AtomicPart's MyID. Then return the AtomicPart's MyIDs according to the 5 numbers	R1, R2
Q4	Randomly generate 5 titles for Documents then return Document's MyIDs by lookup on these titles.	R1, R2
Group II		
Q2	Select 1% of the latest AtomicParts via buildDate return the MyIDs.	R4
Q3	Select 10% of the latest AtomicParts via buildDate return the MyIDs.	R4
Q7	Select all of the AtomicParts and return the MyIDs.	R4, R8
Group III		
Q5	Find the MyID of a CompositePart if it is later BaseAssembly it is using.	R1, R2
Q6	Find the MyID of a CompositePart (repeatedly) once there is a BaseAssembly or ComplexAssembly it is using with a buildDate more than it is.	R1, R2
Q8	Join AtomicParts and Documents on AtomicParts docId and Documents MyID.	R9
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.	R9
Q9	Randomly generate two phrases among all phrases in Documents. Select these documents containing 2 phrases.	R5
Q10	Repeat query 1 but replace duplicated elements using IDREF.	R13
Q12	Select all AtomicParts with corresponding CompositeParts as their sub-elements.	R1, R2
Q13	Select all ComplexAssemblies with type "type008".	R1, R2

Table 4: XOO7 queries.

OO7 provides a comprehensive evaluation of object-oriented database management system performance. The main OODBMS and storage managers have been benchmarked against OO7: E/Exodus, Objectivity/DB, and Ontos. Table 4 outlines the list of queries provided as part of XOO7 currently. We group the queries into 3 main groups and also provide a set of additional queries that cover all the 3 groups. Queries under group I are simple select/project queries using path expressions and schema information. Group II queries generate results requiring explicit order (similar to ORDER BY). Also Q7 uses universal quantification to test systems efficiency in handling quantification. Group III increases the complexity of generating ordered results by including joins. We mapped the OO7 schema and instances into a DTD and the corresponding XML data sets. We evaluated the performance of query processing facilities of: LORE, a special-purpose (or semi structured) system university prototype; Kweelt [26], an open source university prototype that works on ASCII XML data files; and a commercial object-relational database system (ORDBMS[2]) that provides a simple but limited mapping of XML data into object-relational data. The characteristics we measure are response time for different queries and classes of queries, time to load the data, and the space required to store the data. Details of the tests are available in [10].

All the above benchmarks cover an average of 5 to 8 functionalities listed in Table 1. While XMark has 20 query challenges, both XOO7 and XMach-1 have 8 benchmark queries. In addition, XMach-1 has 2 queries to test updates. We note that query Q8 in XMach-1 tests several operations: count, sort, join and existential, making it hard to analyze the experiment result and ascertain which feature leads to the given performance.

[2] We are withholding the name of the commercial system we tested given the sensitivity of our experimental results.

System Characteristics	**Xmach-1**	**Xmark**	**XOO7**	**XOO7-Extended**
Selection Queries	√	√	√	√
Projection Queries	√	√	√	√
Reduction: Remove a selected element	√	√	√	√
Restructuring: Reorder sub elements	√	√	√	√
Construction: Output new structure	√	√	√	√
Remote Execution: Data and Evaluator on different machines				
Preserve Implicit Order				√
Exploit Schema	√	√	√	√
Schemaless Document Manipulations				√
XLink and XPointer Manipulations				
Streaming Data Processing: Schema generation on the fly				
Transaction Processing				
Text Search	√	√	√	√
View Processing				
Stored Procedures and Triggers				
User Defined Functions		√		√
Aggregate Manipulations				√
Update Element/Database				
Delete Element/Database	√			
Append Data	√			
Extract results based on Similarity Criterion				
Quality based retrieval: Top-K results				

Table 5: Comparing Benchmarks over XML system characteristics.

4.4 Evaluating Benchmarks: Desired XML query system properties

Table 5 presents the capabilities of XML management systems that need to be captured by the benchmarks and what current benchmarks cover. It also lists the coverage provided by extended version of XOO7. Table 1 on the other hand lists desirable properties of a XML query language. Needless to say many of these properties do overlap. From Table 5 it is clear that all the three benchmarks cater to the data-centric view of XML management systems. As can be seen these benchmarks cover data-centric properties generally found in traditional database systems like selection, projection, reduction and mandatory schema requirement. BBut then characteristics such as transaction processing, view manipulation, aggregation and insert/updates/delete operations are not covered. We do wish to point out that some of these operations like deletes/updates are not yet clearly defined under the XML query model and hence are futuristic in nature. Yet XMach-1 clearly covers delete and insert operations. Some document-centric coverage is provided in terms of "text search", yet a complete document query mechanism is not tested. XML documents may be developed by multiple users collaboratively. Hence schema changes, document updates are more than necessary to be handled. This highlights the need for triggers that could get activated when a user (superuser) updates the schema. Hence this strengthens our goal to develop a benchmark that tests an XML data management system for both the data and document-centric capabilities of XML queries.

5 Extending XOO7 to Document-centric queries

XML syntax is suited for semistructured data. Yet XML and semistructured data have subtle differences. Without dwelling on the details of XML, a simple abstraction of XML is a labeled ordered tree [28]. A tree representation of XML and semi structured data is interchangeable but a graph structure of both models has differences. Semistructured 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 XML model will show its similarity to the object-oriented data model. XML is probably most similar 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 a lot on the particular DTDs or Schemas used to define the structure of an XML document. The object-oriented data model is similar to both XML and semistructured 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.

In fact XML is less natural in representing Relational databases (RDBMS). Individual tables can be directly represented literally, but with far more information about the data than actual RDBMS's do. Similarly representing relational query results involving joins, grouping, sorting etc. in XML is straight forward and is the most widely practiced use of XML in existing data management systems. But the core of an RDBMS is its relations. In particular, the set of constraints that exist between tables, and that are enforced by the RDBMS are what makes an RDBMS useful and powerful. It is surely possible to represent a constraint set in XML for purposes of communicating it, but XML has no inherent mechanism for enforcing constraints of this sort (DTDs and Schemas are constraints of a sort, but in a different and more limited way). A data model cannot be present without constraints or rather without the ability to enforce the constraints. Also characteristics of RDBMS like fixed record lengths, compact storage formats etc. designed to improve reliability and performance cannot be easily mimicked in XML. In fact the standard API for XML proposed by W3C called DOM uses the Document Object Model [6] for XML documents. The Resource Description Framework used for describing metadata for XML also has object-oriented flavour [11].

Thus while developing the benchmark we based our decisions on two facts. First, the benchmark is for XML query systems using XML data and documents stored locally. Second, XML data model shows high degree of similarity to object-oriented model. Hence we decided to take XOO7 - a benchmark we designed on top of the OO7 benchmark (designed to test performance of OODBMS) and extend it. Our decision to extend XOO7 is also motivated by Table 5 from where we can see that XOO7 does capture many of the data-centric features of any XML

Id	Description	Comments	Coverage
Group I			
Q14	Find ComplexAssembly with AtomicParts having ID greater than particular value	Query using path expression and comparison on non-textual value.	R1, R4
Q15	Return 1st and 2nd occurrence of CompositePart after finding any part having connection element with length >= 5.	Testing document support and maintenance of order.	R1, R2
Q18	Find the 1st and 2nd ComplexAssembly followinga any ComplexAssembly with length of component greater than 5. Format results as first-and-second.	Ability to generate new data conforming to the data model.	R2
Q24	For each ComplexAssembly count number of documents.	Use of cardinality of Document.	R1
Group II			
Q16	For all atomic parts display their parent ids.	Test how effectively relations are identified and manipulated.	R8
Q17	Join AtomicPart and Documents having common docId and MyId respectively. Perform ordered and unordered join.	Compare ability to perform a relational style join. Both ordered and unordered sets of data considered.	R9
Q19	Sort CompostiePart in descending order where buildDate is within a year from currentdate.	Use of environment values and sorting of result. Possibility of optimization.	R11
Q23	Return Module which have Documents containing “special part” as text.	A text search query. A simple case with equality.	R5
Q26	Find all ComplexAssembly of not type “type008”.	Robustness in presence of Negation.	R8
Group III			
Q20	Return MyId of parent element of any element with MyID=someId.	Tests how the data is interpreted and represented internally in the system.	R13
Q21	Return all AtomicPart of ComplexAssembly with type “type008”.	To check how structural recursion is handled.	R13.
Q22	Return all CompositePart having Connection elements with length greater than Avg(lenth) for all ComplexAssembly.	Aggregate function handling. Implicit order maintenance is implied.	R10
Group IV			
Q25	For ComplexAssembly of type “type008” give ‘Result’ containing Id of ComplexAssembly and Text equal to concatenation of all Title Elements in ComplexAssembly.	Generate a result and make use of User Defined Functions to concatenate the titles.	R18
Q27	List all ComplexAssembly of type “type008” Order their BaseAssembly in ascending order of the builddate of CompositeParts.	Implicit oder of ComplexAssemblies need to be maintained. Complexity in reordering of BaseAssembly.	R11, R17 R18

Table 6: New Queries added to XOO7.

management system. We do not change the schema of XOO7 (refer [23]) but merely add further queries to augment the functionality of the benchmark to capture the all feasible (within the current XML query model) characteristics identified in Table 5. As can be seen from Table 1 and Table 4, the queries generated from the OO7 model do not cover a substantial number of the XML query functionalities we identified earlier. Also in Section 3 we identified that although XOO7 initially focuses on the data centric query capabilities, it is imperative that, the benchmark be extended to include queries to test document processing capabilities. Towards this end, we provide 4 Groups of new queries for XOO7 (see Table 6). Group I tests the level of document-centric support given by systems and tests ability of maintaining order of data. Queries Q14, Q15, Q23 are document centric and test text value processing and order manipulation power of the system under test. Group II consists of queries that combine both data and document processing abilities of XML query languages. They test ability to perform text/keyword search, efficient negation handling and sorting of results. Group III tests the ability to support aggregate functions and efficient handling of structural recursion. Group IV queries test support for User Defined Functions and complexity in processing queries requiring both implicit and explicit order maintenance. Specifically Q17 and Q20 are mixed mode queries and generate results that require relational joins to be performed while preserving order and aggregate function processing with implicit order. Q11 tests systems ability to efficiently interact with its environment while processing queries. Q25 uses User Defined Functions, an ability most traditional databases provide. Q27 brings out the complexity involved in having to maintain both implicit order (order present in data) and the explicit order (specified by query) while generating results.

While designing the queries, we assume a document ordered representation of XOO7 data. We do acknowledge that this assumption may tilt balance towards native XML databases for certain types of queries. The complexity involved in satisfying this assumption on existing management systems (given most use a RDBMS or ORDBMS as backend) has to be empirically evaluated and forms part of our future work. The extension we provide is a first attempt at making the benchmark complete. For results of comparisons of existing XML management systems using the XOO7 benchmark refer [10]. We do realize that at the moment we are assuming single-user systems. In reality multi-user database systems are highly prevalent and widely used. Next we plan to extend XOO7 to include multi-user querying capabilities, querying in presence of schema information and other aspects. We also plan to empirically evaluate the new set of queries we provide and compare existing XML database systems thereby determining the feasibility and classifying power of these queries.

6 Conclusion and Future Work

XML is becoming ubiquitous. Increasing number of of-the-shelf management systems for XML are becoming available. To check whether these systems provide what XML, XML related technologies e.g. XPath, Xpointer etc. and XML query can deliver a benchmark becomes inevitable. In this paper, we first identify the desirable XML query characteristics. Then we briefly review existing XML management systems and provide a list of desirable characteristics for such systems. We also introduce existing XML query benchmarks which are XMach-1, XMark and XOO7 and compare them in terms of aforementioned capabilities of XML query and management systems. We point out that the existing benchmarks only test the data-centric XML processing capabilities of systems and extend XOO7 to include document-centric capabilities. We extend XOO7 [10] with 14 new queries to capture both document-centric query processing features and also additional data-centric features. This extension is by no means complete and we do intend to improve XOO7 further. The XOO7 benchmark is based on single user operations. In order to test how systems scale we intend to extend XOO7 to test platforms with multi-users. Also we intend to test some data management systems on the extended XOO7 and hope to get more insights for further development of the benchmark. Since different XML management systems use various computing environments we intend to look at issues of portability of our benchmark from system to system.

References

[1] S. Abiteboul. Object Database Support for Digital Libraries. In First European Conference on Research and Advanced Technology for Digital Libraries, Pisa, Italy, September 1997. Invited paper.

[2] S. Abiteboul and S. Grumbach. COL: A Logic-Based Language for Complex Objects. EDBT, pages 271--293, 1988.

[3] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J.L. Wiener. The Lorel Query Language for Semistructured Data. Journal on Digital Libraries, 1997. ftp://db.stanford.edu/pub/papers/lorel96.ps.

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

[5] S. Adler, A. Berglund, J. Caruso, S. Deach, P. Grosso, E. Gutentag, A. Milowski, S. Parnell, J. Richman, and S. Zilles. Extensible Stylesheet Language (XSL). W3C, 2000. working draft available at http://www.w3.org/TR/xsl.

[6] V. Apparao, S. Byrne, M. Champion, S. Isaacs, I. Jacobs, A. Le Hors, G. Nicol, J. Robie, R. Sutor, C. Wilson, and L. Wood. Document Object Model. W3C, 1998. available at http://www.w3.org/TR/REC-DOM-Level-1.

[7] T. Bohme and E. Rahm. XMach-1: A Benchmark for XML Data Management, 2000. Available at http://dbs.uni-leipzig.de/projekte/XML/XmlBenchmarking.html.

[8] A. Bonifati and S. Ceri. Comparative analysis of five xml query languages. SIGMOD Record, 29(1):68--79, 2000.

[9] R. Bourett. Xml database products, May 2001. available at http://www.rpbourret.com/xml/XMLDatabaseProds.htm/.

[10] S. Bressan, G. Dobbie, Z. Lacroix, M. L. Lee, Y. Guangli, U. Nambiar, and B. Wadhwa. XOO7: Applying OO7 Benchmark to XML Query Processing Tools. CS Department Technical Report TRB6/01, National University of Singapore. 2001.

[11] D. Brickley and R.V. Guha. Resource Description Framework (RDF) Schema Specification 1.0. W3C, 2000. candidate recommendation -- available at http://www.w3.org/TR/rdf- schema.

[12] P. Buneman. Semistructured Data. In Proc. ACM Symp. on Principles of Database Systems, Tucson, 1997. Invited tutorial.

[13] M.J. Carey, D.J. DeWitt, and J.F. Naughton. The OO7 benchmark. ACM SIGMOD Conference, pages 12--21, 1993.

[14] D. Chamberlin, D. Florescu, J. Robie, J.Sim'eon, and M. Stefanescu. XQuery: A Query Language for XML. W3C, 2000. available at http://www.w3.org/TR/xmlquery.

[15] D. Chamberlin, J. Robie, and D. Florescu. Quilt: An XML Query Language for Heterogeneous Data Sources. In Proceedings of the Workshop WebDB (in conjunction with ACM SIGMOD), Dallas, TX, 2000.

[16] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. XML-QL: a query language for XML. In preparation - Available at http://www.w3.org/TR/NOTE-xml-ql/, 1998.

[17] P. Fankhauser, M. Marchiori, and J. Robie. XML Query Requirements. W3C, 2000. available at http://www.w3.org/TR/xmlquery-req.

[18] D. Florescu and D. Kossman. A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database, May 1999. Report 3680 INRIA, France.

[19] R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In Proc. of Intl. Conf. on Very Large Data Bases, Delphi, Greece, August 1997. to appear.

[20] J. Gray. The Benchmark Handbook: For Database and Transaction Processing Systems. Morgan Kaufmann, 2nd edition, 1993.

[21] H. Liefke and D. Suciu. XMill: An Efficient Compressor for XML Data. In Proc. ACM SIGMOD Symp. on the Management of Data, Dallas, Texas, 2000.

[22] T. Milo and D. Suciu. Index Structures for Path Expressions. ICDT, 1997.

[23] U. Nambiar, Z. Lacroix, S. Bressan, M. L. Lee, and Y. Li. Benchmarking XML Management Systems: The XOO7 Way. Technical Report TR-01-005, Dept of Computer Science, Arizona State University. 2001.

[24] F. Rizzolo and A. Mendelzon. Indexing XML Data with ToXin. In proceedings of Fourth International Workshop on the Web and Databases (WebDb2001), May 2001.

[25] J. Robie, J. Lapp, and D. Schach. XML Query Language (XQL). In Proc. of the Query Languages workshop, Cambridge, MA, December 1998. available at http://www.w3.org/TandS/QL/QL98/pp/xql.html.

[26] A. Sahuguet. KWEELT, the Making -of: Mistakes Made and Lessons Learned, 2000. Technical Report, Dept of Computer Science, University of Pennsylvania.

[27] A. R. Schmidt, F. Waas, M. L. Kerste, D. Florescu, I. Manolescu, M. J. Carey, and R. Busse. The XML Benchmark Project. Technical Report INS-R0103, April 2001.

[28] V. Vianu. A Web Odyssey: from Codd to XML. Proceedings of 20th Symposium on Principles of Database Systems(PODS 2001), May 2001.