Current Approaches to XML Management

XML management systems vary widely in their expressive power and query-processing efficiency, and users should choose the XMLMS that best meets their needs.

Ullas Nambiar and Zoé Lacroix
Arizona State University

Stéphane Bressan, Mong Li Lee, and Yingguang Li
National University of Singapore

The Extensible Markup Language has become the standard for information interchange on the Web. Developed primarily as a document markup language more powerful than HTML yet less complex than SGML, XML does not require content to adhere to structural rules. XML gives a single, human-readable syntax for representing data, including data in relational format. Hence XML appeals to both the document and the database communities.

Early developers of XML content storage tools, who came from the database community, regarded XML as yet another data format for adapting relational and sometimes object-relational data-processing tools. While this use of XML is acceptable, it does not harness XML's full power. XML is inherently semistructured. However, documents subscribing to the *data-centric* view of XML are highly structured and can be represented equivalently in tables or in XML with document type definitions (DTDs) or XML schema specifications (see the sidebar, "Related W3C Documents," for this and other XML specifications). As in traditional relational databases, sibling element order is unimportant in such documents.

We refer to documents with implicitly ordered XML content as *document-centric*. The file's element order (as siblings in a tree-like representation) conveys its implicit order, whereas a document attribute or tag expresses an explicit order. Although it is easy to express explicit order in relational databases, capturing the implicit order while converting a document-centric XML document into a relational database is a problem. Besides the implicit order, document-centric XML documents allow little or no structure, deep nesting, and hyperlinked components. Tables can represent implicit order, nesting, and hyperlinks but only with costly time and space transformations.

This article studies the data- and document-centric uses of *XML management systems* (XMLMS). We want to provide XML data users with a guideline for choosing the data management system that best meets their needs. Because the systems we test are first-generation

 1089-7801/02/$17.00©2002 IEEE

Table 1. XML query language functionalities, or requirements, by query type supported.

Relational	Document	Navigational
Support all data types under XML schema specification	Allow conditions/constraints on text elements	Allow navigation (reference traversals)
Process collections of XML documents and data types	Support type coercion	Support hierarchical and sequence queries
Accept streaming data	Preserve document structure	
Manipulate NULL values	Support identity creation and preservation	
Support quantifiers ($\forall$, $\exists$, and $\sim$) in queries	Support structural recursion	
Allow queries that combine different document parts	Support documents with and without schema information	
Support aggregation		
Sort results		
Support operation composition		
Allow use of environmental information in queries		
Allow operations on document schema if available		
Support updates if data model allows		
Transform and create XML structures		

approaches, we suggest a hypothetical design for a useful XML database that could use all the expressive power of XML and XML query languages.

Expressive Power vs. Efficiency

In 1996, the World Wide Web Consortium (W3C) formed a working group to define a markup language with SGML's power and extensibility but HTML's simplicity. The team stripped away all the complex parts of SGML to create XML, a versatile yet easy-to-use markup language.

XML overcomes the shortcomings of both SGML and HTML. SGML parsers require strict adherence to the DTD, making it too complex for everyday use, such as for Web publishing. HTML parsers make no such demands, allowing multiple interpretations of the same data, but thereby making it difficult to add semantics to HTML data.

In 1998, the W3C approved version 1.0 of the XML specification, and several working groups began to identify and characterize multiple XML uses, among them querying XML documents. The W3C XML Query Language working group lists desirable functionalities, presented in Table 1, as "must-have" requirements. These functionalisties were also identified and analyzed by academic teams (see 1 for example).[1] Despite the dramatic differences between data- and document-centric approaches, the W3C XML Query Language working group states that an XML query language must provide data-centric, document-centric, and navigational capabilities, thus capturing completely the XML data model expressiveness.

This polarized view of XML generates issues regarding the functionality or expressive power of existing XML query languages. To compare the expressive power of XML query languages, we classify XML queries with respect to the functionalities listed in Table 1.

XML queries with an expressive power similar to that of Datalog,[2] the database logic, for relational models are *relational queries*. Queries using the implicit and explicit order of XML elements are *document queries,* while queries that require traversing the XML document structure using references or links as supported by XLink or Xpointer are *navigational queries*.

Using a data- or document-centric view of XML when developing a query language affects not only the language's expressive power, but also its performance. Data-centric query languages cannot exploit XML's implicit order. Relational systems can, however, efficiently process most data-centric queries. Unordered XML data requires the least processing time. Indeed, because unordered data is similar to relational data, we can use traditional optimized relational approaches to process it. An XMLMS that uses XML's data-centric characteristics might provide less expressive power in terms of queries supported than a system using a document-centric approach, but it is likely to perform better. Fully ordered data require preserving the XML document's logical structure. Naïve approaches to processing queries against fully ordered data require loading the entire document into main memory and creating a tree structure of the document. Processing time is high for fully ordered XML content.

If processing efficiency were the only criteria, we would favor unordered XML data representation and use relational approaches to efficiently solve queries. But not all XML queries can be

expressed against unordered data. While a system can use an ordered data representation to represent unordered data, the converse is not possible. Tools that use the fully ordered representation or the document-centric view of XML can represent all types of XML content whereas those using the unordered or data-centric view of XML can only represent efficiently structured and relational-like data. We can make a similar analysis for nested or hyperlinked documents versus flat data.

The expressive power and processing efficiency of XML management systems depends on the underlying data representation.

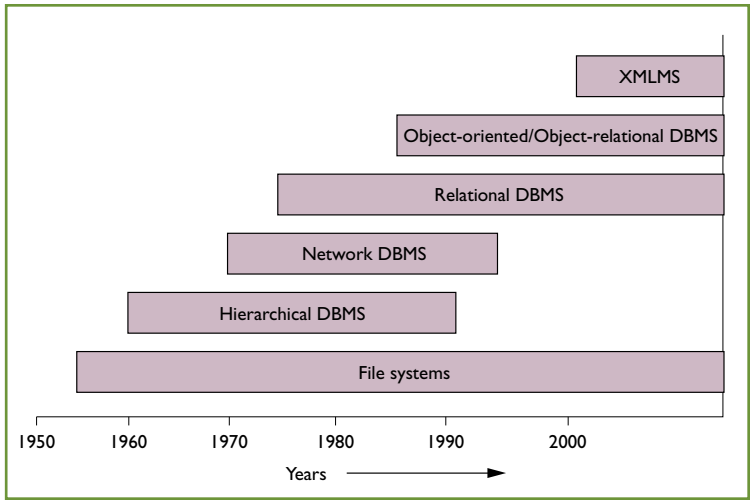

Figure 1. Timeline of database management system evolution. In flat file systems, data structure was irrelevant. For XML data management systems, however, knowing the data structure is essential.

Data Management Systems

Data management systems control data acquisition, analysis, translation, storage, and retrieval. Although these functions are explicit, the type of data they represent is less clear. Data management systems were motivated by developers' desire for

- redundancy control,
- the ability to answer queries against data,
- data protection from system failure,
- efficient data sharing among multiple users,
- data security and integrity, and
- separation of applications from data.

Data management systems evolved from flat files to current object-oriented or object-relational database management systems, which will further evolve to the XML management systems of the future.

From Flat Files to XML

Figure 1 follows the evolution of traditional data management systems to current and future systems.

The *flat file systems* of the mid- to late-1950s followed a simple design: text sequences formed records, and data structure held no importance. These systems suffered from uncontrolled data redundancy, data inconsistency, low productivity, and high maintenance costs, however, and researchers began looking for ways to exploit the structure of stored data.

Hierarchical databases were the first improvement over flat files. Although they seemed intuitive, because their design depended on the data structure, they were inherently inflexible: Any changes to the data structure required changes to applications using the data. To overcome hierarchical system shortcomings, developers created *network databases* by adding links between structures. To use a network database effectively, however, you have to know how it is structured and how to traverse the structure.

In the early 1970s, IBM Research's Edgar F. Codd proposed the revolutionary *relational database*.[3] Representation in relational databases is flexible, simple, and application-independent. The relational model can represent a large variety of data; however, it does not adapt well to semistructured data with complex nesting.

XML's introduction as a data representation and exchange medium prompted its adoption among relational data model proponents. XML data structure can be complex, however, and to efficiently process XML data, users must know its structure.

XML Database Systems

The relational database community's focus on XML as another data format for relational and object-relational data-processing tools led to the development of query languages such as XML-QL,[4] Lorel, and XML Query Language (XQL).[5] These languages are biased toward the data-centric view of XML, which requires data to be fully structured but unordered. In essence, they use XML to publish structured data in a platform- and application-independent manner. XML thus merely provides "syntactic sugaring" to the underlying relational data. While this data-centric use of XML is acceptable, it does not harness XML's full power.

The document-centric use of XML relies not only on the data representation expressed through markups, but also on the data's component ordering. Many systems that subscribe to the document-centric view use XPath, a language designed to identify parts of XML documents, to query XML data. Recently, the W3C published XQuery, which

combines XML's data- and document-centric characteristics, as a candidate standard query language for XML.

We can thus divide current XML management systems into *XML-enabled databases* and *native XML databases*.[6] XML-enabled databases, which are usually relational, contain model- or template-driven extensions for transferring data to and from XML documents and are generally designed for data-centric documents.

There are three main differences between XML-enabled databases and native XML databases:

- A native XML database preserves physical structure — CDATA sections, comments, programming instructions, DTDs, and so on. Theoretically, an XML-enabled database can do so as well, but practice tells a different story.
- Native XML databases can store schemaless data. We could use techniques to identify structure in raw documents to be stored in XML-enabled systems; however, such techniques are rather limited.
- XPath, DOM, or similar XML-related APIs are required to access data in native XML systems. XML-enabled systems, on the other hand, offer direct access to the data through open-standard APIs, such as open-database connectivity (ODBC).

The sidebar, "Current XML Management Products," describes some XML-enabled and native XML database products. Ronald Bourret provides a detailed classification of XML management products.[6]

Taking the Middle Ground

Unlike relational data, XML data is ordered. An obvious drawback for systems claiming to be XML-aware is that they must deal with the implicit order of XML data. Converting from XML data to relational data causes order information to be lost. Therefore, any view generated by XML-enabled systems presents a pseudo-order of the elements. On the other hand, most of these systems have database backends, so they should provide SQL-like processing capabilities for XML queries — for example, aggregating, nesting, and grouping.

Relational databases have efficient storage and retrieval techniques and can evaluate queries using various indexing mechanisms. But these systems cannot adequately express or optimize queries that exploit XML data's document-centric properties, such as text search and implicit and explicit order management. The computational complexity of XML data manipulation makes relational database indexing mechanisms difficult to implement. Flat file data systems containing XML documents are almost certain to perform naïve execution of XML queries. Moreover, the performance of native XML databases varies with the storage method used. These systems are certain to maintain the data's true order, however.

Thus, available systems solve the data-centric or document-centric view in isolation. We believe an XML management system should perform well for both data-oriented and information-retrieval type queries, as well as queries that exploit both features simultaneously.

Most XML-enabled systems are built on top of commercial implementations of relational databases, which are tuned for performance. Therefore, we realize that any comparison we make at this stage might be inefficient. To get a true picture of native XMLMS potential, we must wait for systems like ToX (see the sidebar, "Current XML Management Products").

Selecting an XMLMS

To deal with the expanding volume of XML data on the Web, XML data users will have to select an XMLMS that fits their needs. Users having data-centric needs — for example, legacy system users — will favor an XML-enabled data management system, whereas users who wish to store and manipulate documents will choose a native XMLMS. To give XML data users a guideline for selecting the XMLMS that best suits their needs, we use the XO07 benchmark,[7] which we derived from 007,[8] a benchmark developed for object-oriented databases, to classify four contemporary XML management systems based on the type of queries they process efficiently.

System Comparison

We tested the expressive power and processing efficiency of four data management systems: the Lightweight Object Repository (Lore), Kweelt, Xena, and a commercial XPath implementation that we call DOM-XPath (the product's developers declined our request to use its real name). Here, we analyze the results to identify how XML-enabled and native-XML management systems differ in expressive power and processing efficiency.

First, we compare the time and disk space required by various data management systems to convert XML data into their proprietary format. Then we compare the processing efficiency of these systems in terms of the response time for

Current XML Management Products

Many XML management systems exist. We categorize them as native or XML-enabled systems, based on characteristics such as whether they are data- or document-centric, whether they preserve an XML document's structure, and what type of access they allow users.

Native and Near-Native Systems

Kweelt,[1] a proposed implementation of Quilt,[2] is a native XML management system that stores data in flat files and favors XML's document-centric nature. ToX, a more advanced native XMLMS, supports multiple data storage methods and uses indexing to improve query processing efficiency.[3] Various implementations of XQuery, such as Galax (db.bell-labs.com/galax) and Quip (developer.softwareag.com/tamino/quip/) are native XML implementations. The Tamino XML server (www.softwareag.com/tamino) is a commercial native XMLMS.

Lore is not exactly native since it is a semistructured data management system revised to handle XML documents.[4,5] For Lore, the most difficult aspect of converting XML data was tackling its implicit order. Developers of the Araneus project, which originally represented Web sites, attempted a design similar to Lore's.[6] Araneus has the flavor of a traditional object-oriented model extended to flexibly model heterogeneities and irregularities with union types and untyped links. When converting data to XML, it exploits DTDs and uses lists to represent the data's implicit order (possibly nested collections of tuples).

XML-Enabled Systems

Most existing XML data management systems are XML-enabled and built on top of relational or object-relational systems. With these systems, users can publish data in XML and translate XML queries into SQL statements.

Xena, designed at the National University of Singapore as an XML-enabled data management system using XPath, is implemented on top of the MySQL database system.[7] It automatically stores the XML data in tables according to the XML schema. Xena then retrieves data from tables by converting an XPath query into several SQL queries, using XML schema.

Oracle9i is a well-known commercial relational/object-relational database management system that supports XML processing. XSU, which is bundled with Oracle9i, can transform data retrieved from object-relational tables or views into XML.[8] XSU provides tools to map the XML content into relational tables. Complex structured XML data, however, might have to be restructured for mapping.

Microsoft's SQL Server 2000 (www.microsoft.com/sql/default.asp) is also XML-enabled. SQLXML (www.sqlxml.org), which is bundled with SQL Server 2000, lets developers map XML files into relational tables, create XML views of existing relational data, query relational data with XPath, and generate XML results for SQL queries.

References

1. A. Sahuguet, "Kweelt: More Than Just 'Yet Another Framework to Query XML!'" *Proc. SIGMOD*, ACM Press, New York, May 2001.
2. D. Chamberlin, J. Robie, and D. Florescu, "Quilt: An XML Query Language for Heterogeneous Data Sources," *Proc. Workshop on Web and Databases* (WebDB 00), ACM Press, New York, May 2000.
3. D. Barbosa, A. Barta, and A. Mendelzon, "ToX — The Toronto XML Engine," *Proc. Int'l Workshop on Information Integration on the Web*, 2001; www.cos.ufrj.br/wiiw/papers/09-Denilson_Barbosa(10).pdf.
4. S. Abiteboul et al., "The Lorel Query Language for Semistructured Data," *J. Digital Libraries*, vol. 1, no. 1, 1997, pp. 68-88.
5. R. Goldman, J. McHugh, and J. Widom, "From Semistructured Data to XML: Migrating the Lore Data Model and Query Language," *Proc. Workshop on Web and Databases* (WebDB 99), ACM Press, New York, 1999.
6. G. Mecca, P. Merialdo, and P. Atzeni, "Araneus in the Era of XML," *IEEE Data Eng. Bulletin*, vol. 22, no. 3, Sept. 1999, pp. 19-26.
7. Y. Wang and K. Tan, "A Scalable XML Access Control System," *Proc. 10th World Wide Web Conf.*, May 2001; www10.org/cdrom/posters/1096.pdf.
8. B. Chang et al., *Oracle XML Handbook*, Oracle Press, Berkeley, Calif., 2000.

relational, document, and navigational queries. For the experiments, we used a 333-MHz system running SunOS 5.7 with 256-Mbytes RAM.

We derived the data sets for the XOO7 benchmark specification from data sets provided by OO7. Because XML does not cater to "isa" relationships (implicit one-to-one relationships introduced in ER models to capture the correspondences of entities from one subclass to its super-class), we preprocessed attribute and relationship inheritance. This transformation is common to many OO7 implementations.

The data sets contain elements associated with parts, atomic composite entities that can be combined in assemblies, or assemblies. Figure 2 (next page) gives a detailed representation of the XOO7 DTD. The document root is the *module*. Each element belonging to `Design-Object` has three attributes: `MyID` (an integer), `type` (a string), and `buildDate` (an integer). `Complex-assembly` is a recursive element. We use `manual` and `document` to simulate long strings. `AtomicPart` uses `docId` to reference the `MyID`s of document elements. Thus the data sets generated by XOO7 essentially capture all the important features of XML representation. The element-attribute structure is similar to RDBMS relations, while the inheritance and recursion capture inherent document characteristics. The data sets are available in three sizes: large (12.8 Mbytes), medium (8.4 Mbytes), and small (4.2 Mbytes). Earlier work describes the data sets and the DTD in detail.[12]

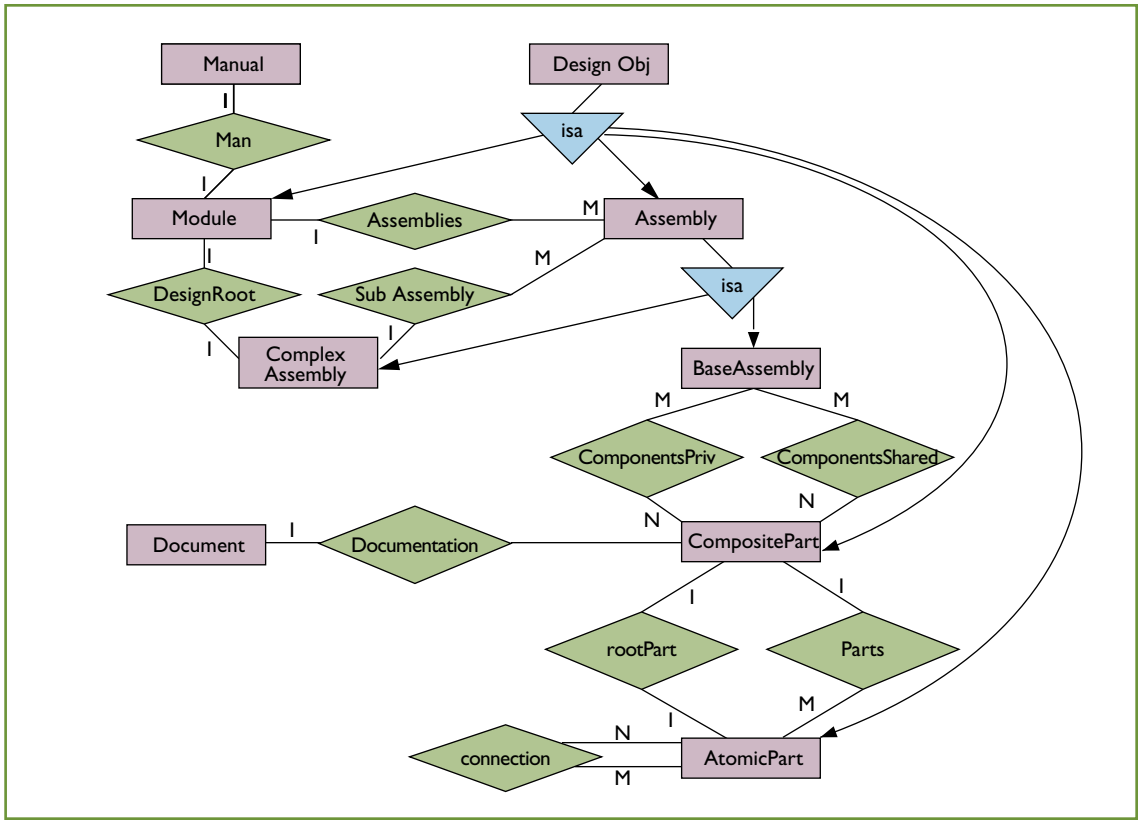

Figure 2. The XOO7 document-type definition. Data sets contain elements associated with parts or assemblies. Purple rectangles represent entity sets, blue triangles, isa relationships, and green lozenges, relationships.

Time and Space Requirements

We converted the data sets to the format used by each of the XML data management systems, recording the time and space required to do so. *Space utilization* is the amount of hard disk space used by each system for the various databases in the benchmark. Figure 3 compares the space and time requirements.

Kweelt queries the ASCII files directly so it does not need to convert the XML data into another format. The required storage space is thus the same size as the initial XML data set and the time required for the conversion is null.

Xena converts the input files to a relational representation based on the data's XML schema and stores it in MySQL tables. Although this conversion drops the redundant tags around the XML data, Xena must generate a number of relational tables to properly represent the XML data. In fact, Xena creates two groups of tables: the first is based on the XML schema with one table per entity; the other manages the first group of tables. Consequently, Xena requires almost three times the amount of space used by the actual XML data after conversion and significantly affects the time needed for conversion.

Lore creates *data guides* — a concise and accurate summary of all paths in the database that start from the root — to facilitate query processing.[10] Therefore, Lore requires almost twice the amount of space as the actual XML data. The conversion is less costly in time than the expensive process performed by Xena.

The XPath commercial implementation, DOM-XPath, provides tools to convert XML data into its internal format. It creates three binary files for each XML data set. One of the files is a proprietary database that preserves the native XML structure by storing the data set's entire document tree, thereby occupying much more space than the original XML data set.

The time needed to perform the translation increases with respect to the size of the data set while remaining more efficient than Xena and Lore. Not surprisingly, Kweelt is the most efficient in terms of space. Because Kweelt directly processes raw XML data, it requires no time for data conversion. As expected, Xena, an XML-enabled database system, requires the most time to convert from the XML model to a relational model. Lore

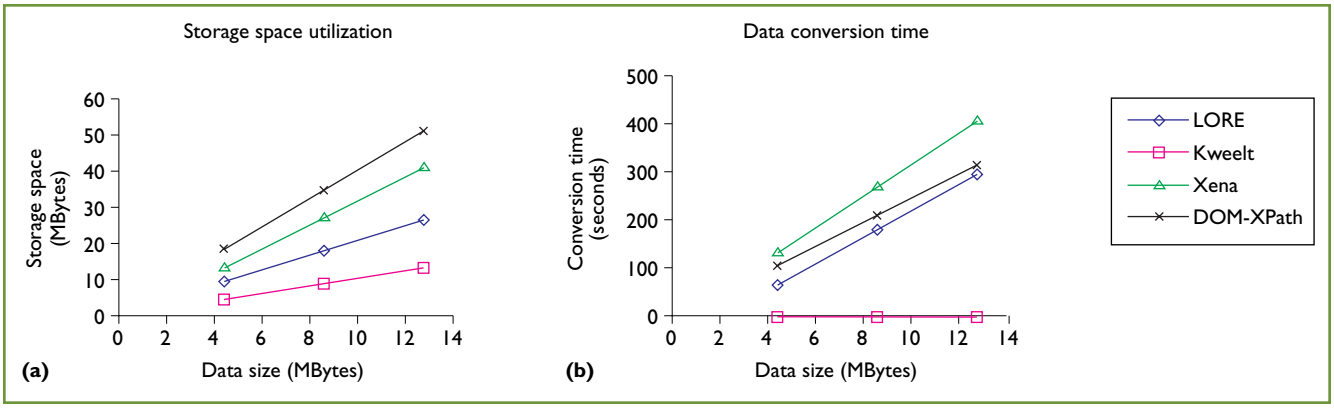

Figure 3. Storage space utilization and data conversion time for four XML management systems tested. Kweelt, a native-XML system that directly processes raw XML data, is the most efficient in terms of both space and time.

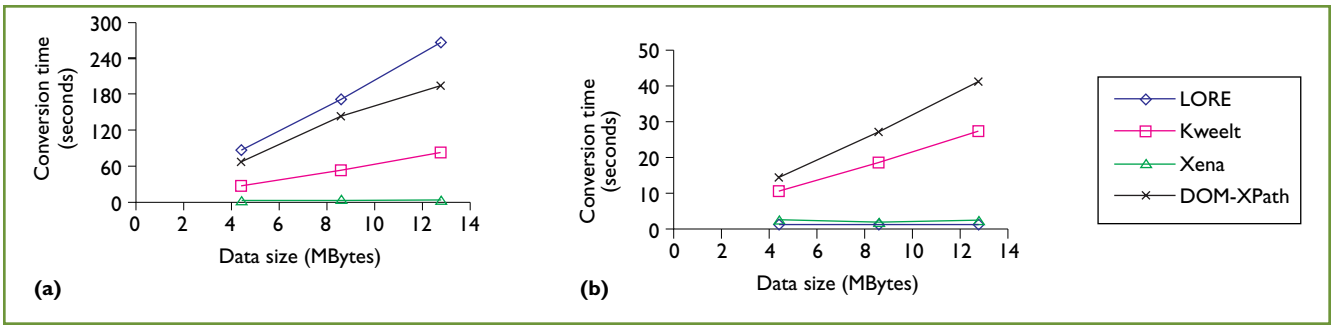

Figure 4. Response times for two relational queries. (a) Query 1 tests simple selection through number comparison; (b) Query 2 tests simple selection through string comparison.

requires time for generating data guides, and we assume the XPath implementation also generates indexes, a time-consuming task.

Response Time Analysis

There are 18 queries in the XOO7 benchmark. We divide them into three groups: relational queries, document queries, and navigational queries. We provide the complete list of XOO7 queries elsewhere.[12] Here we show the four systems' performance for selected representative queries from each group.

- *Query 1*. Randomly generate five numbers in the range of `AtomicPart`'s `MyID`. Then return the `AtomicPart` according to the five numbers.
- *Query 2*. Randomly generate five document titles, then return each document's first paragraph by looking up the titles.

Figure 4 shows the results for relational queries 1 and 2. Query 1 tests simple selection using number comparison while query 2 uses string comparison. For both queries, Xena gives the best response time by leveraging the power of its backend relational database. Lore shows interesting results: It is the slowest to execute query 1, but it executes query 2 relatively quickly. This is because Lore supports data type coercion, which is efficient for string comparison but not for other data type comparisons.

Both Kweelt and DOM-XPath use DOM. However, it seems that Kweelt performs better than DOM-XPath. There are two possible explanations for this. First, the two systems use different DOM parsers. Second, because DOM-XPath is a commercial product, it tries to handle issues such as updates and access control, which introduce additional processing overhead. As a research prototype, Kweelt concentrates solely on query processing. DOM-XPath processes relational queries better than Kweelt. We suspect the DOM parser used by DOM-Xpath performs better for relational queries. Moreover, as a university research prototype, Kweelt might not have the best implementation possible. In general, Xena, the XML-enabled XMLMS, yields the best performance for relational queries. Lore's performance varies depending on whether data type coercion is required. The two native XML management systems, Kweelt and DOM-XPath, show average performance.

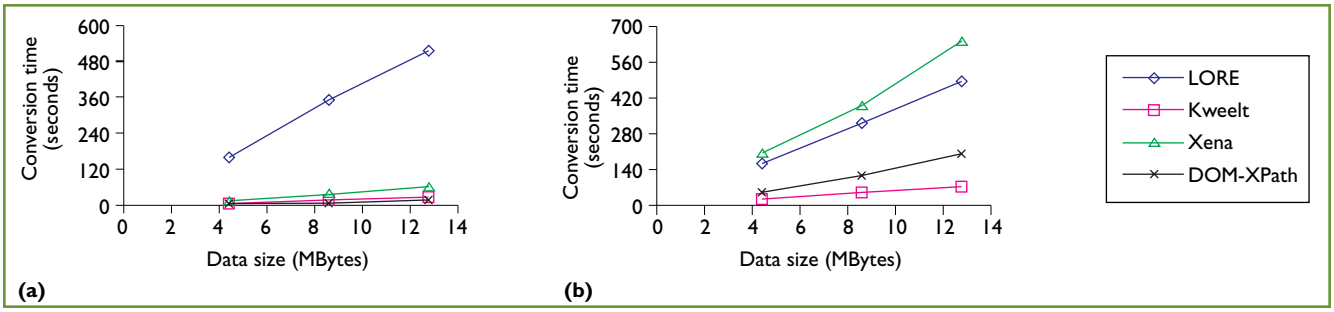

Figure 5. Response times for two navigational queries. (a) Query 10 tests processing efficiency for parent-child relations; (b) Query 14 tests system efficiency while crossing the XML data structure in the presence of negation.

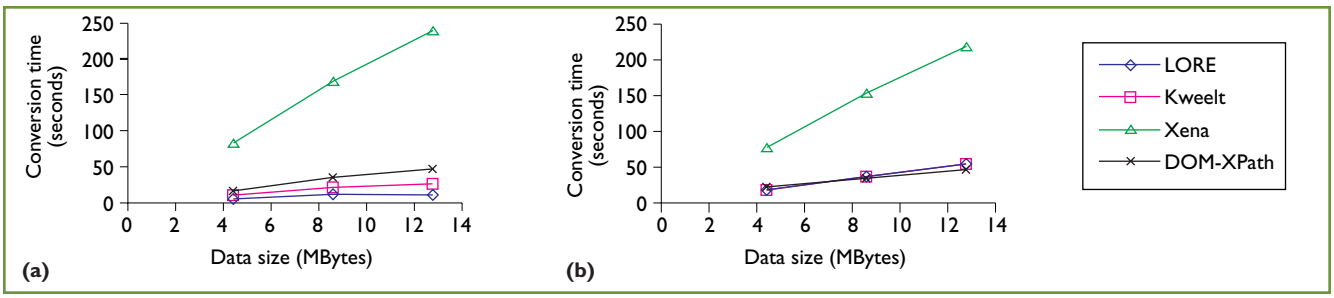

Figure 6. Response times for two document queries. (a) Query 17 tests element order where the ordering pre-exists; (b) Query 18 tests element order where no ordering exists.

- *Query 10*. Find the `CompositePart` if it is more current than `BaseAssembly`.
- *Query 14*. Find `BaseAssembly` of not-type "type008."

Figure 5 shows the results for navigational queries 10 and 14 of the XOO7 benchmark. Query 10 tests the processing efficiency for parent-child relations. Query 14 checks system efficiency in traversing the XML data structure in the presence of negation. Lore performs poorly for both queries. We suspect that the object exchange model (OEM) data model is unable to handle parent-child relations efficiently and requires more time to preserve the original data structure. Xena processes query 10 faster than Lore but not query 14. This is because Xena stores an XML file by breaking it into elements, thus losing the parent-child relations. To track parent-child relations, Xena requires each element to record the paths to its ancestors.

The selectivity factor in query 10 is 10 percent, and the result size of query 14 is also almost 10 percent. Xena's response time increases 10 times from query 10 to query 14. Kweelt and DOM-XPath store the XML data in its original native form, which results in good performance.

To summarize, Lore and Xena, the XML-enabled systems, perform poorly for navigational queries. Both systems store the XML data according to its entities, or elements. Kweelt and DOM-XPath store the XML data in its original form, thus preserving the parent-child relationships for navigational queries.

- *Query 17*. Among the first five connections of each `CompositePart`, select those with length greater than "len."
- *Query 18*. For each `CompositePart`, select the first five connections with length greater than "len."

Figure 6 shows the results for two document queries: queries 17 and 18 of the XOO7 benchmark. These two queries test the element order in an XML file. The two differ in that the ordering required by query 17 exists in the original database, while for query 18, an XMLMS must reconstruct the ordering.

Xena proves inadequate for processing document queries. We expected this as Xena fetches all the target elements first, and then selects the elements that satisfy the conditions according to the index in the preserved order. Lore performs best in these queries mostly because it uses data guides as shortcuts to extract the relevant elements directly. However, Lore is unable to process query 18 because it does not support ordering in an intermediate result (in this case, all connections with length greater

Related W3C Documents

Document Object Model (DOM), V. Apparao et al., 1998 • www.w3.org/TR/REC-DOM-Level-1.
Extensible Stylesheet Language (XSL) Version 1.0, S. Adler et al., 2000 • www.w3.org/TR/xsl.
XML Linking Language (XLink) Version 1.0, S. DeRose, E. Maler and D. Orchard, 2000 • www.w3.org/TR/xlink.
XML Path Language (XPath) Version 1.0, J. Clark and S. DeRose, 1999 • www.w3.org/TR/xpath.
XML Pointer Language (XPointer) Version 1.0, S. DeRose, R. Daniel and E. Maler, 1999 • www.w3.org/TR/WD-xptr.
Xquery 1.0: An XML Query Language, S. Boag et al., 2000 • www.w3.org/TR/xquery.
XML Query Requirements, D. Chamberlin et al., 2000 • www.w3.org/TR/xmlquery-req.
XML Schema Part 0: Primer, D. Fallside, 2001 • www.w3.org/TR/xmlschema-0.
XML Schema Part 1: Structures, H. Thompson et al., 2001 • www.w3.org/TR/xmlschema-1.
XML Schema Part 2: Datatypes, P. Biron and A. Malhotra, 2001 • www.w3.org/TR/xmlschema-2.

than "len"). Kweelt and the DOM-XPath perform relatively better for the queries whose results contain multilevel nodes. They check the element order using the information in the DOM tree.

The experiments run with the XOO7 benchmark confirm our analysis of data- and document-centric views. The XML-enabled relational database system performs best when processing simple relational queries. The native-XML implementation Kweelt, efficiently processes navigational and document queries, but performs poorly for relational queries.

References

1. A. Bonifati and S. Ceri, "Comparative Analysis of Five XML Query Languages," *SIGMod Record*, vol. 29, no. 1, 2000, pp. 68-79.
2. C.J. Date, *An Introduction to Database Systems,* Addison-Wesley, Reading, Mass., 1995.
3. E.F. Codd, "A Relational Model of Data for Large Shared Data Banks," *Comm. ACM*, vol. 13, no. 6, 1970, pp. 377-387.
4. A. Deutsch et al., "XML-QL: A Query Language for XML," *Proc. Query Languages Workshop*, W3C, 1998; www.w3.org/TandS/QL/QL98/pp.html.
5. J. Robie, J. Lapp, and D. Schach, "XML Query Language (XQL)," *Proc. Query Languages Workshop*, W3C, 1998; www.w3.org/TandS/QL/QL98/pp/xql.html.
6. R. Bouret, "XML Database Products," May 2001, www.rpbourret.com/xml/XMLDatabaseProds.htm.
7. S. Bressan et al., "XOO7: Applying OO7 Benchmark to XML Query Processing Tools," *Proc. 10th Int'l Conf. Information and Knowledge Management* (CIKM-2001), ACM Press, New York, 2001.
8. M.J. Carey, D.J. DeWitt, and J.F. Naughton, "The OO7 Benchmark," *Proc. SIGMOD*, ACM Press, New York, 1993, pp. 12-21.
9. S. Bressan et al., *The XOO7 XML Management System Benchmark*, National Univ. of Singapore Computer Science Dept. tech. report TR21/00, Nov. 2001.
10. R. Goldman and J. Widom, "Data Guides: Enabling Query Formulation and Optimization in Semistructured Databases," *Proc. Very Large Databases* (VLDB 97), Morgan Kaufmann, San Francisco, 1997, pp. 436-445.

Ullas Nambiar is pursuing a PhD in computer science at Arizona State University. His research interests include data integration over static and streaming sources, distributed mediation, and active content delivery services.

Zoé Lacroix is a research assistant professor in the mechanical and aerospace engineering department at Arizona State University, where her projects include electronic business hubs, biological database integration, optimization, semantics of Web query semantics, and the semantic Web. She received a PhD in computer science from the University of Paris XI. She has been involved in the World Wide Web Consortium's XML Query Language and XForms working groups.

Stéphane Bressan is a senior fellow in the computer science department of the National University of Singapore's School of Computing. He received a PhD in computer science from the Laboratoire D'informatique Fondamentale of the University of Lille, France. Bressan's main areas of research are information and knowledge management, Web applications and services, and the design and implementation of database management systems.

Mong Li Lee is an assistant professor in the School of Computing, National University of Singapore. She received a PhD in computer science from the National University of Singapore in 1999. Her research interests include database performance issues, and cleaning, integrating, and querying heterogeneous and semistructured data.

Yingguang Li is an MSc student at School of Computing, National University of Singapore. He is interested in XML query processing and storage strategies.

Readers can contact the authors at {mallu,zoe.lacroix}@asu.edu and {steph,leeml,liyg}@comp.nus.edu.sg.