Why and How to Benchmark XML Databases

Albrecht Schmidt[1,2]
albrecht@cwi.nl

Florian Waas[2]
florianw@microsoft.com

Martin Kersten[1]
mk@cwi.nl

Daniela Florescu[3]
danaf@propel.com

Michael J. Carey[3]
mike.carey@propel.com

Ioana Manolescu[4]
ioana.manolescu@inria.fr

Ralph Busse[5]
busse@darmstadt.gmd.de

Abstract

Benchmarks belong to the very standard repertory of tools deployed in database development. Assessing the capabilities of a system, analyzing actual and potential bottlenecks, and, naturally, comparing the pros and cons of different systems architectures have become indispensable tasks as databases management systems grow in complexity and capacity. In the course of the development of XML databases the need for a benchmark framework has become more and more evident: a great many different ways to store XML data have been suggested in the past, each with its genuine advantages, disadvantages and consequences that propagate through the layers of a complex database system and need to be carefully considered. The different storage schemes render the query characteristics of the data variably different. However, no conclusive methodology for assessing these differences is available to date.

In this paper, we outline desiderata for a benchmark for XML databases drawing from our own experience of developing an XML repository, involvement in the definition of the standard query language, and experience with standard benchmarks for relational databases.

1 Introduction

XML and databases seem like an odd couple: two very different concepts driven by two very different communities with different expectations and requirements. Yet, an increasing demand for consistent and reliable ways to manage XML data [2, 3] suggests the marriage of the two. By that, we do not mean "management of XML data on top of a relational database" as it is often interpreted but a general coupling of the two worlds. For storing XML, numerous techniques have been suggested and the discussion whether conventional database technology, be it relational or extended relational, is up to the challenge is still in full swing.

From an application point of view, the discussion whether XML is syntax or data model looks slightly different: in order to assume the role of a true data exchange format in which both industry and research want XML to be seen, an XML database has to deliver on the performance demands of its key applications: web services, B2B, and e-Commerce scenarios to name just a few. Most of them require on-line, often interactive processing.

Throughout the history of the development of relational database technology, benchmarks served primarily as a scale to assess and compare new techniques and system components. On the other side, many innovations in query processing and storage management were achieved by trying to boost benchmark figures (*e.g.* [10, 11]); thus, benchmarking does not only mean that we measure the state of the art but is also a constant incentive for further development.

In this paper, we postulate desiderata for a general purpose benchmark for XML databases; we take both into account, the state of the art as well as recent developments.

First of all, let's have a look at the state of the technology: (1) Currently, there is no commonly agreed on notion of the functionality that an XML database can be expected to comply with. Therefore, we will go with the frequently used definition of an XML database, which refers to a system that stores and manages XML documents.[6] More crisp are the definitions of various query languages, XQuery [5], the most prominent of which is currently being standardized. (2) On the other hand, many applications currently under construction or being deployed already specify a number of requirements XML databases will have to meet.

However, before we go into a detailed analysis, it might be helpful to briefly survey the complementary development of relational systems and benchmarks in order to identify reasonable criteria and to avoid obvious mistakes. What can be carried over from more traditional

[1]CWI, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands
[2]Microsoft Corp., One Microsoft Way, Redmond, USA
[3]Propel Inc., San Jose, USA
[4]INRIA Rocquencourt, France
[5]GMD-IPSI, Dolivostr. 15, 64293 Darmstadt, Germany

[6]A number of similar definitions can be found at [3].

benchmarks like those defined by TPC, SAP, PeopleSoft amongst others? What can be learned?

Early benchmarks have been strictly geared toward testing database functionality on a very general level. The early Wisconsin Benchmark [12] for example consists of a number of queries which test the performance of small numbers of join operations. While helping to determine the bottlenecks in join processing, the benchmark does not take particular application scenarios into account beyond the obvious motivation that joins rank among the most costly operations thus have to be especially taken care of.

As database functionality evolved, implementations of different systems converged, and performance figures for simple queries became increasingly indistinguishable, the embedding of a database system into an application scenario became more and more important. TPC-H/R [16], one of the most important industry benchmarks, is modeled after the general requirements a data warehousing application poses to a database system.

Finally, the latest development are application specific benchmarks like the SAP and the PeopleSoft benchmark series, where performance characteristics for running one proprietary application are used to assess a system. In contrast to the early general purpose benchmarks, these kind of application benchmarks no longer evaluate databases as isolated units but as an integrated back-end in an application scenario, and assume that customers purchase the database solely for this purpose.

This evolution may seem like a linear development with databases changing from general purpose repositories to becoming part of an individual, complex application. However, this development has also seen failures and set backs, most notably TPC-W. The TPC-W benchmark is widely agreed to provide only little insight. Modeling many components of a internet commerce application, the interaction between components and different scenarios become opaque and hard to analyze. Unlike the application benchmarks mentioned above, the TPC-W scenario is kept overly generic, and thus, instead of exactly matching one app, it does not match any. While tuning a database system for SAP's or PeopleSoft's benchmark is of direct use for customers who directly deploy the database – often solely – in this scenario, the gains from high TPC-W numbers remains questionable.

In short, benchmarks have to match (1) the technology available at the time and (2) the application scenarios used in production; given the current developments, defining a general purpose benchmark is an important first step to conclusive assessments of XML databases.

In this paper, we review the conceptual idiosyncrasies of storing and retrieving of XML data and try to identify components and operations that a reasonable benchmark should cover. We also scrutinize what lessons, learned from query processing in relational databases, can directly be carried over.

2 Preliminaries

The question "how to assess query performance" is obviously preceded by the more basic question "what operations on an XML document are conceivable and reasonable". With its XML Query Working Group, the W3C addressed this issue with experts from both the database and the document communities. The scenarios and use cases [4] developed were directly taken from research input but also results of surveys of web services and application scenarios that did not yet deploy XML but could benefit from it. The outcome of this process became available in two phases: first, an algebra that tried to formalize operations was released and later, a draft for a query language was formulated.

It appears essential when staking out a benchmark, to review these two products as they set the stage for any performance evaluation by defining the set of operations available.

2.1 Query Algebra

The algebra, specified in a Haskell-like functional programming language [8], defined and illustrated by means of use cases a set of operations that appear meaningful to perform on XML documents. Due to the visible database background of the group, most operations suggested are very familiar from the world of relational database systems. Operations include filters, joins on values or along referential key constraints, grouping operations etc.

In contrast to their relational relatives, these operators obviously have to maintain order. In the notation of the algebra, this is implied by using *for*-loop like constructs which suggest iterating over an ordered list of data. These loops are only defining the variable binding and a system which can encode order through, say, an additional attribute at the object itself may take advantage of using operations that are not order preserving, *e.g.*, hash joins.

With respect to the definition of a benchmark, the algebra immediately provides two requirements. On the one hand, we certainly want to test most if not all operations suggested. On the other hand, the issue of treating order should be a the central aspect in the tests—the more if the XML database in question is implemented on a relational database system.

2.2 Query Languages

While the algebra mainly served to outline the basic operations that would have to be supported by a language it does not specify how those operations should be exposed in a query language. At the beginning of the standardization process, several query languages were in use, the most popular of which are XQL [17] and XPath [6]. Those languages might be considered obsolete in the light of the standardized query language XQuery – XQuery

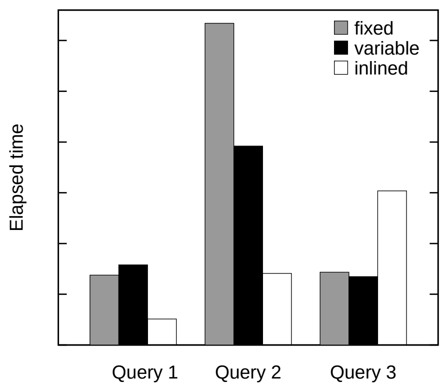

Figure 1: Query times of select benchmark queries on a relational database system using three different mappings of XML to relational structures

subsumes XPath and is more powerful than XQL. However, a sizable user base currently deploys alternative XML query languages including several proprietary ones and to the best of our knowledge no commercial implementation of XQuery is available at the moment. Since XQuery has been being standardized by the W3C, with all major manufacturer fully supporting the development, XQuery is clearly the query language of choice for formulating the benchmark queries.

In order to be able to test also systems which do not provide full XQuery support, be it for reasons of integrating legacy applications and data, be it that the application scenario the XML database has been developed for simply does not require the additional expressiveness of XQuery, it seems highly advisable to define *query groups*, *i.e.,* a classification and bundling of the individual queries. Often, simple queries can be formulated using only XPath primitives. A second dimension along which we will define query groups is of course the functionality. That way, the functionality of a system can be categorized and assessed conclusively and transparently.

The XQuery standardization efforts have come a long way already, yet, XQuery is by no means complete. So far merely read-only query scenarios have been addressed; inserts, updates and XML specific derivatives thereof have, at the time of writing, not been dealt with. To organize the benchmark in query groups also helps in this situation as groups can be enhanced with versioning to reflect the current standard. Releasing a new version of a particular query group to incorporate new developments does not invalidate old results.

3 Motivating Examples

For XML data the physical break-down of data is of significant impact and decidedly more important than it is in the relational world. The decision how to store a document does not only depend on performance and redundancy considerations but also on external knowledge of the document in the form of constraints. In the simplest case, there is no external knowledge and one has to use a generic mapping like one of the binary mappings described for instance in [9, 18]. If there is external knowledge, be it in the form of constraints like a DTD or even something more expressive like a schema language [14] then one may decide to use a more advantageous mapping for example by inlining $1 : 1$ relationships into larger relations or mapping generic string values to richer data types to save space and avoid coercions at query execution time [20]. In an object-oriented scenario, *i.e.,* the primary access pattern is navigational iteration over DOM like syntax trees [21], it is often worthwhile to map the document structure to a set of class definitions. In this case, external knowledge can be exploited by regrouping objects into disk pages according to different access scenarios. Depending on the application scenario there likely are more optimizations to favor the query profile of the application. A number of mappings have been presented recently (*e.g.,* [7, 9, 13, 18, 20]). However, it seems two design principles prevail: binary decomposition, *i.e.,* grouping ancestor relationships in one or more relations, and inlined representation, *i.e.,* storing $1 : 1$ relationships between parent tags and their element and attribute children in one relation while $1 : n$ relationships are mapped to two relations which share a common key [20]. While the former excel through their conceptual simplicity, the latter try to retain a relational flair by mimicking the translation rules (*i.e.,* fragmenting the document along $1 : n$ parent-child relationships) between ER-models and relational schema.

Closely related to the issue of the physical representation is that of query optimization. Over the past two decades, there have been significant advances in optimizing SQL queries; in practice rule-base and cost-based optimization techniques are used, both of which are tightly coupled to the underlying physical data model. It would be highly desirable to re-use large parts if not all of this knowledge.

To illustrate these points, we implemented different mapping models atop a relational database system and compared the query times for three queries taken from *The XML Benchmark Project* [19]. Figure 1 shows the resulting query times: The first column in each group represents the time for a *fixed*, document-independent binary storage model such as [15], *i.e.,* all parent-child relationships are stored in a single relation resulting in many self-joins during query execution. The second column is the performance of the same data mapped to a *variable*, document-dependent relational module such as [18], *i.e.,* all parent-child relationships of a specific type are clustered in a separate relation. The third column in each group eventually shows the performance of an *inlined multi-attribute* mapping in the spirit of [20]; here relations are larger and can have many more attributes than there are per relation in the binary case. From the plot, it is evident, that each of the models has its advantages and can outrun all other alternatives in certain situations. However, to understand the issues and bottlenecks encountered in this simple experiment we need to have a closer look at the processing of the queries:

Query 1 is a very basic point query specified by a path expression. On the inlined model this results to a scan of one relation returning a single tuple; on the other models a number of joins or self-joins are required to materialize the path expression. On the one hand, the inlined model trades in data volume for joins with respect to the binary models, on the other hand, in the variable model a larger number of tables has to be accessed.

Query 2 specifies a join depending on the document order. Here the multi-attribute model benefits from the fact that fewer joins need to be executed than in the binary case. As to the other two models, the implicit clustering of data as achieved by the variable model results in better statistics as compared to the fixed model. Therefore, the the variable model is significantly more amenable for the optimizer than the fixed model.

Query 3 finally underlines the differences in numbers of columns per table. The multi-attribute model suffers from the data volume when a large number of joins is executed where only few columns are required for the result. Joins are much less costly to process on both the fixed and the variable decomposition schemes.

The example provides a good impression of the extent and nature of the performance issues a benchmark has to address. Besides the relatively obvious question of the physical data break-down, a number of other aspects have severe impact on the query performance. Concerning the physical design, the usage of indexes and surrogate OIDs where additional information like order is encoded in the OID can speed up querying on the expense of rendering updates or inserts overly expensive. An area which has been largely neglected so far is the optimization of XML specific queries. Besides issues like in Query 2 where the physical storage scheme prevents extracting of meaningful statistics relational optimization techniques are also oblivious of XML specifics like constraints imposed by the hierarchical structure of the document.

4 Challenges

Above, we explained some of the effects that caused significant differences with respect to performance characteristics though the physical storage format differed only slightly. Certainly, a general performance assessment cannot scrutinize differences on as fine a level of granularity as we hinted at. Rather we identify 10 general challenges which aim at providing a comprehensive and conclusive performance analysis covering all performance critical aspects of processing of XML.

1. *Bulk loading.* The importance of bulk loading data has been repeatedly emphasized when assessing database performance in general. In XML databases, bulk loading currently assumes an even more important role as no insert/update operations are available and most systems support insert only on a document level. Due to the fact that different models imply different levels of granularity when shredding the document this seemingly simple operation may entail severe costs to setup and maintain indices or constraints.

2. *Reconstruction.* Also know as round-tripping, reconstructing the original document is the counterpart of bulk loading. Though being an operation simple to specify, round-tripping is not a common but still necessary operation in most scenarios. Reconstruction reveals the price of achieving loss-less storage of the document, i.e. the trade-off between efficient indexing and preserving the full semantics of the document.

3. *Path traversals.* Specifying paths arguably is one of the most basic and natural operations on structured documents. Not only useful as a stand alone operation, path expressions are but ubiquitous part of almost all complex operations. Efficient path traversals often bring about a trade-off with respect to redundancy, data volume or degree of fragmentation.

4. *Casting.* XML is essentially text and as such queries frequently demand casting to other elementary data types like integers, floats or even user-defined types. String operations are notoriously expensive.

5. *Missing elements.* The semi-structured nature of XML in general brings about a highly heterogeneous structure of records that under some mappings results in many NULL values. Apart from strategies to store NULL values in a compact way, we also need efficient methods to query for NULLs as they often represent spots of interest.

6. *Ordered access.* Order is *the* omni-present feature when querying XML and affects all aspects of data management. Obviously, sticking with order preserving implementations may become a severe bottleneck. Rather, sophisticated and flexible treatment of the document order should ensure that it well integrated into the optimization process up to a degree that it is ignored when not needed.

7. *References.* References are an important modeling primitive almost comparable to referential constraints in relational databases. Technically speaking, chasing references requires efficient access

methods supporting random access across the document rather than navigational access.

8. *Joins.* Join operations on values, *i.e.* content, have been seen as a critical aspect early XML query languages lacked. Particularly data-centric applications require the combination of data based on values. The difficulties, bottlenecks, and challenges posed by joins are well-known from relational database systems.

9. *Construction of large results.* As opposed to round-tripping, construction of large results refers to assembling large new documents from the data stored. Many application areas demand the ability to handle large data volumes.

10. *Containment, full-text search.* Containment and full-text search are elementary operations when querying XML. The problem and its intrinsic difficulties are well-known from other application domains, hence, it hardly needs further motivation.

A benchmark for XML databases needs to address these 10 points. Given an application scenario, most of the challenges can be directly expressed as a single query, some however offer a larger degree of freedom and it seems advisable to address them with several, differently parameterized queries.

5 Other Quality Parameters

The above challenges are clearly focused on the performance of the actual XML database as motivated in Section 1. From an application programmer's perspective, other quality parameters of an XML database are visible and at first sight, it may seem natural to include those. They can be divided into two groups, infrastructure issues and total cost of ownership, we list them here for completeness and underline afterward why we believe they should not be included in the list of challenges.

Infrastructure. Especially if front-end and back-end of the database are not tightly coupled, communication costs may dominate and obscure the performance characteristics of the query processor. Some of these issues have been addressed in XMach-1 [1], a mainly system focused benchmark. Issues include:

Access protocols, like HTTP, OLE DB, ODMG, ODBC, native APIs*etc.*, including their noise and transmission costs, often determine the degree of usability of a solution; *Result representations*, like DOM, SAX, serialized XML, or proprietary structures should be in line with application requirements; *Responsiveness versus completeness*, *i.e.*, availability of the first or the complete query result, including the influence of lazy evaluation and the availability of cursors can have a great impact on performance of production systems; The *expressiveness of the query language*, *e.g.*, the missing restructuring capability of XPath queries, often determines whether a query engine fits a given application scenario; Lastly, *data throughput* in multi-user application scenarios;

Total Cost of Ownership. With increasing complexity of software systems, the total cost of ownership becomes more and more important as it usually dominates the costs of the software itself. With respect to XML processing, we can identify a number of issues:

Installation effort: Does the product work out of the box or does it require extensive multi-stage preparation and installation? *Generality support:* Is it possible to store documents with and without schema information? What price does one have to pay for lack of modeling? *Consistency support:* Is it possible to validate incoming documents against a schema or other constraints? *Preparation effort:* What mapping and schema definition tasks are necessary before the document can be imported (cf. eXcelon vs. Tamino)? *Training:* Does the Software require extensive training for technical staff or does it integrate into existing infrastructure? *Interaction paradigm:* Does the product provide a stand-alone document management system with tools for direct interaction, or is it rather an enabling technology for front-end applications? *Updates:* Does the system provide fine-grained update functionality or is it restricted to replacing complete documents?

In the introduction we already outlined why not to benchmark infrastructure issues or total cost of ownership: A benchmark has to reflect primarily the state of the art. Given the current developments, we believe a benchmark that addresses the performance challenges we listed above provides the most valuable input for the advancement of XML processing technology. Benchmarks which include other aspects provide less accurate and insightful a feedback than we envision.

6 Conclusion

The wide choice of architectures and environments makes it difficult to decide which XML Query Processor and which infrastructure fit best a given application. From a system analysis point of view, the noise introduced by the infrastructure into which the query engine is embedded often obscures the performance of the core components and makes fine-tuning and architectural improvements hard to realize. Therefore, the XML Benchmark Project, in which we gathered our experiences, aims at providing tools to analyze and improve query processors and make sense of their performance characteristics. We have motivated the need for an XML benchmark by pointing out which parts of systems need consideration and adaptation: while the front-end query language is in the process of standardization, there are many performance critical issues in the design of the physical storage schema, the physical algebra, query optimizer and execution. The desiderata listed in this paper are intended to serve as a basis for a comprehensive performance analysis of XML databases.

Acknowledgments. The authors would like to thank Stefan Manegold and Niels Nes for their comments on an early draft of this paper. The first author would also like to thank Michael Rys for his valuable input.

References

[1] T. Böhme and E. Rahm. XMach-1: A Benchmark for XML Data Management. In *Proceedings of BTW2001*, Oldenburg, 2001. Springer, Berlin.

[2] R. Bourett. XML Database Products. `http://www.rpbourret.com/xml/XMLDatabaseProds.htm`, 2000.

[3] R. Bourret. XML and Databases. `http://www.rpbourret.com/xml/XMLAndDatabases.htm`, 2000.

[4] D. Chamberlin, P. Fankhauser, M. Marchiori, and J. Robie. XML Query Use Cases. `http://www.w3.org/TR/xmlquery-use-cases`, 2001.

[5] D. Chamberlin, D. Florescu, J. Robie, J. Siméon, and M. Stefanescu. XQuery: A Query Language for XML, February 2001. `http://www.w3.org/TR/xquery`.

[6] J. Clark and S. DeRose. XML Path Language (XPath), Version 1.0, November 1999. W3C Recommendation, `http://www.w3.org/TR/xpath`.

[7] A. Deutsch, M. Fernandez, and D. Suciu. Storing Semistructured Data with STORED. In *Proc. of the ACM SIGMOD Int'l. Conf. on Management of Data*, pages 431–442, Philadephia, PA, USA, 1999.

[8] P. Fankhauser, M. Fernández, A. Malhotra, M. Rys, J. Siméon, and P. Wadler. XQuery 1.0 Formal Semantics, June 2001. W3C Working Draft, `http://www.w3.org/TR/query-semantics`.

[9] D. Florescu and D. Kossmann. Storing and Querying XML Data using an RDMBS. *IEEE Data Engineering Bulletin*, 22(3):27–34, 1999.

[10] C. Galindo-Legaria and M. Joshi. Orthogonal Optimization of Subqueries and Aggregation. In *Proc. of the ACM SIGMOD Int'l. Conf. on Management of Data*, pages 571–581, Santa Barbara, CA, USA, May 2001.

[11] J. Goldstein and P. Larson. Qoptimizing Queries Using Materialized Views: A Practical, Scalable Solution. In *Proc. of the ACM SIGMOD Int'l. Conf. on Management of Data*, pages 331–342, Santa Barbara, CA, USA, May 2001.

[12] J. Gray. Database and Transaction Processing Performance Handbook. `http://www.benchmarkresources.com/handbook`, 1993.

[13] C. Kanne and G. Moerkotte. Efficient Storage of XML Data. In *Proceedings of the 16th International Conference on Data Engineering*, page 198, 2000.

[14] D. Lee and W. W. Chu. Comparative Analysis of Six XML Schema Languages. *ACM SIGMOD Record*, 29(3):76–87, 2000.

[15] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A Database Management System for Semistructured Data. *ACM SIGMOD Record*, 26(3), 1997.

[16] M. Poess and C. Floyd. New TPC Benchmarks for Decision Support and Web Commerce. *ACM SIGMOD Record*, 29(4):64–71, December 2000.

[17] J. Robie, J. Lapp, and D. Schach. XML Query Language (XQL). In *QL'98 – The Query Languages Workshop*, Boston, MA, USA, December 1998.

[18] A. Schmidt, M. Kersten, M. Windhouwer, and F. Waas. Efficient Relational Storage and Retrieval of XML Documents. In *International Workshop on the Web and Databases*, pages 47–52, Dallas, TX, USA, 2000.

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

[20] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. In *Proc. of the Int'l. Conf. on Very Large Data Bases*, pages 302–314, Edinburgh, UK, 1999.

[21] W3C. Document Object Model (DOM). `http://www.w3.org/DOM/`.