ON ANALYZING THE DATABASE PERFORMANCE FOR DIFFERENT CLASSES OF XML DOCUMENTS BASED ON THE USED STORAGE APPROACH

Hagen Höpfner and Jörg Schad and Essam Mansour
International University Bruchsal, Campus 3, 76646 Bruchsal
hoepfner@acm.org, s9jgscha@stud.uni-saarland.de, essam.mansour@ieee.org

Keywords: XML Data Management, XML Performance

Abstract: With the increasing popularity of XML data also the need for permanent XML document storage grows. As of today there exist number of different XML storage alternatives ranging from XML enabled relational database systems over a new class of hybrid database systems providing native storage for XML and relational data to pure native XML systems. This paper examines how these different storage approaches perform in respect to the different classes of XML data by devising a new benchmark HYBE with special consideration to certain features of hybrid database systems. First results indicate that hybrid database systems can deliver performance which is (almost) equivalent to native XML database systems making them the optimal choice for small to mid-size companies with the need for both XML and relational data storage.

1 INTRODUCTION

In recent years the amount and usage of XML data grew rapidly and today it is used for many purposes such as data transfer, configuration files or to store information. So, the safe, efficient and reliable storage for such documents becomes more and more important. Until a few years ago there existed - besides classical file systems - only two options for storing XML documents: either native XML databases (e.g. Apache Xindice or Tamino (Schöning, 2003)) or XML enabled relational databases providing an XML data type. In the second case XML documents are either stored as a character large object (clobbing) or shredded into relational database tables (object relational mapping). Recently large database vendors such as Oracle or IBM developed another alternative, the so called hybrid database systems. They store both, relational data and XML data natively by providing two separate storage systems. However, all alternatives have certain advantages and drawbacks. Choosing the appropriate technique depends on the application and is a non trivial task. Even though there exist a number of XML benchmarks they are usually focused on a specific application domain (e.g., financial data) and so far have not considered the advantages of hybrid systems. Therefore, we here aim at devising a benchmark considering different characteristics of projects while explicitly including the features of hybrid systems.

Paper structure: Section 2 briefly describes related work. Section 3 discusses the nature of XML data. Section 4 explains the storage alternatives. Section 5 outlines our newly devised benchmark HYBE. Section 6 shows our preliminary results for the different domains and storage approaches considered.

2 RELATED WORK

As of today there already exists a range of benchmarks for XML database systems and also a range of performance analysis has been performed so far.

XBench (Yao et al., 2004) is a family of XML benchmarks. It generates various types of XML documents (data-oriented and varying in their size) and simulates different applications by inserting and reading data using XQuery. The *Michigan Benchmark* (Runapongsa et al., 2006) runs 45 different queries (loading, inserting, deleting and updating) on one large XML document with a recursive structure. It mostly measures the performance of the implementation but is not suited to measure across different systems involving relational databases and SQL queries. *TPox* (Nicola et al., 2007) is a domain specific benchmark for the financial sector using FIXML (Cover, 2002). It contains a content generator for different document sizes. The database operations tests include read (XQuery and SQL/XML), insert, delete and update queries. The systems simulates up to 1,000,000 users to test multiuser performance. *XMach-1* (Böhme and Rahm, 2001) is an XQuery based benchmark for native XML databases and XML-enabled databases focusing on b2b applications. It simulates a Web application for handling text files consisting mostly of several document-oriented XML files. However, as the meta data is handled in a data-oriented fashion, XMach-1 is not restricted to document-oriented files only. *XMark* (Schmidt et al., 2001) models the database in the back end of an Internet auction platform. It provides a document generator for different document types mainly having data-oriented characteristics but descriptive text features include some document oriented features as well. XMark uses only retrieval queries and does not include any database updates. As Xmark is executed as a single user application the execution time of each query is measured individually but thereby focuses on the query processing and not the entire database system. *X007* (Li et al., 2001) is the XML extension of the 007 benchmark for object-oriented database systems (Carey et al., 1993). X007 does not model a specific application. It provides one rather generic but a customizable (size, complexity, depth) complex XML file that is fairly regular. So, X007 focuses on data-oriented features. Issued queries are a means of retrieving data and do not include update queries.

(Lu et al., 2005; Nicola and Rodrigues, 2006) analyze the performance with regard to the storage models and different data requirements. (Nicola and Rodrigues, 2006) focuses on IBM DB2 and compares its pure XML storage models against shredding or clobbing. Those studies indicate that clobbing is best performing if XML documents are always considered as a whole. Shredding should be used for fixed schema data as it is often the case with data-oriented documents. Native XML storage seems to be the best choice for document-oriented documents or if the schemata evolve. This view is also supported by (Serna and Gerrikagoitia, 2005).

3 NATURE OF XML DATA

Different usage scenarios for XML data require different types of XML documents. They are classified into *document-oriented* and *data-oriented* (DuCharme, 2004). Some authors also mix these categories as semistructured XML documents (Bourret, 2005). This differentiation is quite important as each class poses different requirements to a database and generally different database systems might be adequate for each class. For example, previous research has shown that data-oriented XML documents can be stored efficiently in relational database systems while document-oriented XML documents should be stored in a native manner (Bourret, 2005). This paper shows how hybrid database systems fit into this picture.

Data-oriented XML documents usually focuses on data for machine processing. Examples include flight orders or stock quotes. Data-orientation is often used to transport certain information in a predefined format. This usually results in a relatively flat element hierarchy and regular structure that conforms to simple schema information. As this schema information can be defined precisely for data-oriented XML data and does not change frequently, it can be easily mapped into relational database systems.

The focus of *Document-oriented XML documents* is more similar to a document in the usual sense. Documents often contain much free text. At this, usually the entire document is of interest. Often they are made for human consumptions; examples include books and advertisements (Bourret, 2005). It is usually less regularly structured. Consequently it is less precise and frequent schema updates prohibit (or at least hinder) a mapping into relations. Hence, native XML database systems are often used for storing document-oriented XML documents.

Semistructured XML documents mix the two other classes (Bourret, 2005). Here part of the document is data-oriented while some subparts might be document-oriented. Access to the data-oriented segments is still possible for machines while the document-oriented part might contain additional information for human readers.

4 XML DATA STORAGE

As mentioned in the previous sections, there exist various alternatives for storing, retrieving and maintaining XML documents. A straight forward solution would be to treat the data as files and storing them in the normal file system. Collections of different XML documents could be achieved by a folder hierarchy. Unfortunately this approach is accompanied by several drawbacks such as a complicated search for partial documents or a non appropriate access control.

4.1 Clobbing

Another simple solution is to store the XML documents as simple streams of characters using a relational database system. For this purpose either a `varchar` column could be utilized. Some database system such as Oracle and DB2 provide an explicit character large objects (clob) type for storing larger character streams[1]. Therefore, the data is outsourced from the columns itself and just referenced. As clobbing interprets XML documents unparsed it faces similar problems as the file system storage; even while insertion and retrieval of full documents are simple and fast, searching or accessing individual elements often requires parsing of the data.

In order to make those operations more feasible, database developers utilized another idea from relational databases: indexing (Nicola and Rodrigues, 2006). Those indexes or side tables, as called in DB2, capture part of the structure of the XML document and thereby make lookup and search over those elements independent from re-parsing the data. Unfortunately, those tables require parsing on document insertion. According to (Nicola and Rodrigues, 2006) this slows down insertions by a factor 1.3. Indexing also requires additional storage space. In the extreme case the entire document is indexed[2], but this requires at least double the storage space of the original data.

Update queries (if supported at all) are usually very slow as often the entire document has to reinserted into the database (cf. (Nicola and Rodrigues, 2006; Chen et al., 2007)). When comparing the different XML characteristics we do not expect much difference for the non-indexed clobbing as there the schema or characteristics or not fully considered. For the indexed clobbing we expect the data-oriented XML documents to deliver better performance as the index is built based on schema information which is more regular for data-oriented XML documents.

Despite those issues clobbing is regularly used for storing XML Data especially in cases where only entire documents are of relevance. Also with clobbing the original form is preserved which might be required by company or legal policies.

4.2 Shredding

Shredding is also referred to as object-relational mapping or structured storage of XML documents. It is the practice of converting XML documents into a relational data format. This approach exploits the existing object-relational database technology and provides an interface for XML on top. Most commercial database systems support shredding. Unfortunately, it has some drawbacks as well. It requires a complex table structure to represent one to many relationships. A similar problem occurs with nested and/or recursive XML structures (Nicola and van der Linden, 2005, p. 2). Also it usually requires a fixed schema to derive the XML to relational mapping which is hard to modify later. Queries usually involve many time consuming joins to collect all requested data. This directly influence the performance.

Shredding requires more storage space due to the representation of sparse XML documents (not all subelements are presented). However, shredding simple data-oriented XML documents can sometimes even reduce the storage consumptions as tags are not repeated. So, shredding is a suitable storage model for small, fixed-schema XML documents that result in a simple relational table mapping. However, it generally reduces the flexibility of the XML data format and results in worse performance when compared to other approaches.

In this setting support for XQuery or XPath queries requires those queries to be rewritten as SQL queries on the underlying tables. The result then has to be re-transformed to XML which is often done, e.g. in Oracle 11g (Lee, 2007, p. 16 ff.), via SQL/XML functionality.

4.3 Native XML Storage

As there exists no formal definition for "Native XML Databases" (XML DB) the term is often used for marketing reason for very different products. It was probably coined by the German Software AG for their first release of the Tamino XML Server in 1999 (Schöning, 2003). A widely used and cited definition is the one by the XMLDB organization. It is based on the (logical) model for an XML document – as opposed to the data in that document – and stores and retrieves documents according to that model (Bourret, 2005). At a minimum, the model must include elements, attributes, PCDATA, and document order. Examples of such models are the XPath data model, the XML Infoset, and the models implied by the DOM and the events in SAX 1.0. An XML DB has an XML document as its fundamental unit of (logical) storage, just as a relational database has a row in a table as its fundamental unit of (logical) storage. However, it is not required to have any particular underlying physical storage model. For example, it can be built on

[1]`varchar` usually supports data up to 3–5 KB while `clob` columns in Oracle and DB2 support up to 2 GB of data (Chen et al., 2007).

[2]This would be equivalent to the document being stored in using an object relational mapping.

a relational, hierarchical, or object-oriented database, or use a proprietary storage format such as indexed, compressed files. A native XML database can store XML documents without any limitation to the underlying structure and allows performant access to this XML structure.

Many of those systems use a hierarchical data model so for example Tamino (Schöning, 2003) or DB2 (Nicola and van der Linden, 2005). Therefore, the XML documents are parsed and the data is stored according to the underlying structural model. This natural representation makes it suitable for both document- and data-oriented XML documents and provides flexibility regarding the schema information as this is most often derived from the data itself.

As the derivation of such model requires a parsed data representation insertion times are often slower as compared to clobbing without indexing (Nicola and Rodrigues, 2006, p. 4), but this approach results in superior query performance.

4.4 Hybrid XML Storage

Hybrid XML storage does not define another approach for storing XML documents. It is done comparably to native XML data storage but actually introduces two separate storage engines into a database system: One for relational data and another for native XML storage. This results in a combination, which is invisible to the user. Often they support different models specifying how the XML document should be stored for different needs. Oracle, for example, supports three different storage models for different needs. The "Hybrid Idea" was first introduced by Oracle with their 10g release[3] and IBM with DB2 9. As of today hybrid systems are also available from other vendors (e.g., Microsoft SQL Server 2007).

The hybrid database architecture is illustrated in Figure 1. It usually provides distinct interfaces for XML and for relational data. These are then compiled into one common query format which can then be executed against both the relational and XML storage. There is no translation of XQuery into SQL as in the case of shredding. Both data formats can be be queried via both interfaces (e.g., XQuery on relational data and SQL on XML documents). It is also possible to join XML documents and relational data (Nicola and van der Linden, 2005).

The underlying storage is often based on hierarchical models to match the XML structure. Here often the same set of features is supported as in native XML databases. The XML data type can be used as a normal data type in table column definitions but the XML documents are stored separately in a postparsed representation. Consequently, a parsing at insertion time is required. However, there is no need to parse the data at the point of query execution.

[3]Even though we would not consider the storage concept of 10g to be native as it utilizes shredding.

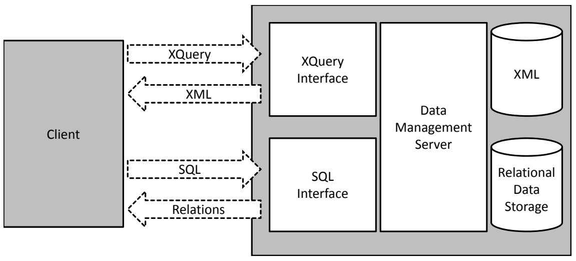

Figure 1: General Hybrid Database Architecture

5 BECHMARKING

To evaluate the performance of the different XML data storage approaches we devised a new benchmark called HyBe. It explicitly includes hybrid systems in the performance analysis and supports several query alternatives such as XPath, XQuery, SQL/XML, XUpdate, W3C XQuery Update Facility. HyBe consists of two parts: 1) a feature list considering the general functionality of the storage approach or database system and 2) a performance analysis with a fixed set of queries measuring the performance. The considered features were: maximal complexity of XML documents, support for schema information, support for different query languages, support for updating queries, and support for combining XML and relational data.

The query sets used for performance analyses comprised: inserting 100 documents into the database (sequentially and batch), retrieving a full document specified by an ID, retrieving a specified value via XPath and XQuery expressions (indexed and non-indexed) and via SQL/XML, updating a single specified value using XUpdate and W3C Update Facility, and queries that combine relational and XML data. For a more detailed description of the features and query sets of HyBe please consider (Schad, 2008).

Due to the differences in database performance and requirements of different XML document categories we considered the two major classes in our benchmark described in Section 3: data- and document-oriented XML documents.

The data-oriented documents were generated by using Toxgene (Barbosa et al., 2002). The generated documents are highly regular and the corresponding schema information can simply be derived automatically as the element hierarchy is flat and does not change between different documents. Each document has an average size of 25 to 50 Byte and a maximum element depth of four.

For document-orientation we used the XML generator of XMark. Still we were not looking for one very large file containing all the data but rather split it into 500 smaller documents. Each resulting document had a size between 150 and 700 Byte and an element depth of less than ten elements. Furthermore, some schema variation (when derived automatically) is present as not all elements are mandatory, but we kept the schema variation intentionally low to keep the results interpretable. However, we assume a performance advantage with more evolving schemata for hybrid and native systems.

6 PRELIMINARY RESULTS

As the target of this research was to compare the performance of storage alternatives rather than the performance of systems, we decided to use only a single DBMS. We choose IBM's DB2 9.5 as it supports clobbing, shredding, and the hybrid storage.

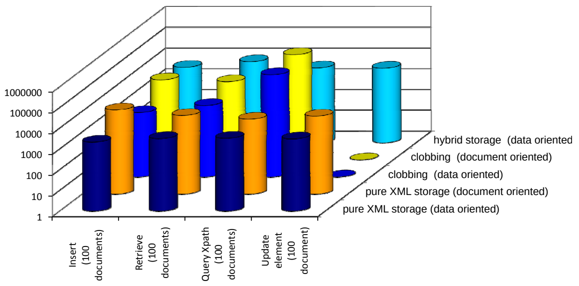

Figure 2: Average time per operation per storage alternative

We ran each query of our query set (compare Section 5) 100 times and in order to check for and incorporate speedup effects over time (i.e., caused by DBMS optimization and buffering as it is done in real world applications) we repeated each run. These resulting average times are shown in Figure 2.

To compare the different storage approaches two different views were used: first the different queries were compared for each storage option visualizing the different characteristics proposed above. As a next step we compared the different storage options for each query visualizing the performance characteristic for each query type. This is especially interesting when it known before hand that the main operation for XML data in a certain project will be inserting.

The analysis of different queries per storage option shows that the hybrid XML storage has performance characteristics comparable to native XML storage (c.f., (Schad, 2008)). It performs better for data-oriented documents (less complex parsing) but still reasonably well (especially retrieval) for document-oriented documents. The execution times differed around a factor of 3 to 5 when the entire document was considered. For node specific operations the factor was roughly 1.2. Remember, the document-oriented data was a factor of 40 larger, which is a common ratio in real world applications as well.

The XML Extender Shredding storage could not handle our document-oriented documents as the mapping probably resulted in too many table columns. Therefore, we could not compare execution times of data and document-oriented data. Still insertion times were relatively long as they require a parsed representation and creation of a (possibly) complex table structure and full document retrieval consumes the most time when compared to the other alternatives. This results from number of join operations which are required. Still for a number of small data-oriented documents it performed reasonably well.

For document insertion we could confirm that non-indexed clobbing is fasted if concerned with full documents insertion and retrieval. It is about 1.5 to 2 times faster than the hybrid XML storage and almost 3.5 times faster than the XML Extender. Still considering the effort of parsing XML data, this gap is still considerably well and is the case where we require parsing in the clobbing setting as well it actually performs worse than the hybrid XML solution. The execution times for full document retrieval also left a similar picture. However, the differences were a little lower (between 1.3 and 1.5) when compared to the hybrid storage but significantly higher for schredding (around 2 to 3) as here a full document retrieval requires a number of join operations. However, without indexes XML specific queries such as XQuery consume a lot of time. Due to the XML parsing at query time XPath queries to a certain node was a factor of more than 20 times slower when compared to the hybrid and shredded storage. Here the hybrid storage performs best followed by shredding (still around two times slower). Both approaches can naturally access individual nodes and therefore the difference between document and data-oriented documents is here quite low. Updating of individual nodes shows the same picture as a parsed representation is required, too.

7 CONCLUSIONS

In this paper we presented a performance comparison of the currently existing storage alternatives for XML data in databases. We firstly described the characteristics of the most important XML document classes that build the basis of HYBE. To the best of our knowledge, HYBE is the first benchmark that evaluates the hybrid storage approach. It was then used for relatively measuring the differences between the storage alternatives regarding different query types.

Our results indicate that it is important to consider the XML storage approach individually for each project. A main distinction is whether the database has to "understand" the data or can just consider them as one large file. Other important points to consider are, e.g., read vs. write or the update frequency.

Clobbing and shredding are limited to certain characteristics and more or less suitable for data-oriented documents. Hybrid systems are a good choice for document-oriented documents. Also hybrid systems offer often more flexibility as they support a wider range of features especially in cases where XML and relational data must be combined.

As the focus of this study was the comparison of different approaches it is interesting to compare different implementations of those approaches by different products such as Oracle's XML DB, IBM's DB2 and Microsoft's SQL Server. Furthermore, we plan to extend the query set and extend the XML data used (e.g., different document sizes). Another issue is the evolution of XML schemata when performing update queries. Finally, we plan to observe index related performances including speed-up and time for index creation. Here we assume that those results will vary significantly across different storage options.

REFERENCES

Barbosa, D., Mendelzon, A. O., Keenleyside, J., and Lyons, K. (2002). ToXgene: An extensible template-based data generator for XML. In *WebDB 2002 Proc.*, pages 49–54. ACM.

Böhme, T. and Rahm, E. (2001). XMach-1: A Benchmark for XML Data Management. In *BTW 2001 Proc.*, pages 264–273. Springer.

Bourret, R. (2005). XML and Databases. online article. `http://www.rpbourret.com/xml/XMLAndDatabases.htm`.

Carey, M. J., DeWitt, D. J., and Naughton, J. F. (1993). The 007 Benchmark. *ACM SIGMOD Record*, 22(2):12–21.

Chen, W.-J., Sammartino, A., Goutev, D., Hendricks, F., Komi, I., Wei, M.-P., and Ahuja, R. (2007). *DB2 9 pureXML Guide*. IBM redbooks. IBM.

Cover, R. (2002). FIXML - A Markup Language for the FIX Application Message Layer. *Cover pages*. `xml.coverpages.org/fixml.html`.

DuCharme, B. (2004). Documents vs. Data, Schemas vs. Schemas. *XML 2004*, pages 1554–4648.

Lee, G. (2007). *Oracle Database 11g XML DB Technical Overview*. Oracle Corportion.

Li, Y. G., Bressan, S., Dobbie, G., Lacroix, Z., Lee, M. L., Nambiar, U., and Wadhwa, B. (2001). XOO7: applying OO7 benchmark to XML query processing tool. In *CIKM 2001*, pages 167–174.

Lu, H., Yu, J., Wang, G., Zheng, S., Jiang, H., Yu, G., and Zhou, A. (2005). What makes the differences: benchmarking XML database implementations. *TOIT*, 5(1):154–194.

Nicola, M., Kogan, I., and Schiefer, B. (2007). An XML transaction processing benchmark. In *SIGMOD 2007*, pages 937–948. ACM.

Nicola, M. and Rodrigues, V. (2006). A performance comparison of DB2 9 pureXML and CLOB or shredded XML storage.

Nicola, M. and van der Linden, B. (2005). Native XML support in DB2 universal database. In *VLDB 2005*, pages 1164–1174.

Runapongsa, K., Patel, J. M., Jagadish, H. V., Chen, Y., and Al-Khalifa, S. (2006). The Michigan benchmark: towards XML query performance diagnostics. *Information Systems*, 31(2):73–97.

Schad, J. (2008). XML-Document Management in Databases — A Performance Evaluation for Hybrid Database Systems. Bachelor thesis, IU in Germany, School of IT, Bruchsal, Germany.

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

Schöning, H. (2003). Tamino-Software AG's Native XML Server. In *XML Data Management*, chapter 2. Addison-Wesley.

Serna, A. and Gerrikagoitia, J. K. (2005). David & Goliath: A Comparison Of XML-Enabled and native XML Data Management Techniques. *XML Journal*. `xml.sys-con.com/node/104980`.

Yao, B. B., Özsu, M. T., and Khandelwal, N. (2004). XBench Benchmark and Performance Testing of XML DBMSs. In *ICDE 2004*, pages 621–632.