Comparing XML Processing Performance in Middleware and Database: A Case Study

Marcus Paradies, Susan Malaika
IBM Watson
19 Skyline Drive
Hawthorne, NY 10532, USA
{mparadi,malaika}@us.ibm.com

Matthias Nicola
IBM Silicon Valley Lab
555 Bailey Avenue
San Jose, CA 95123, USA
mnicola@us.ibm.com

Kevin Xie
IBM Toronto Lab
8200 Warden
Markham, Canada
kxie@ca.ibm.com

ABSTRACT
XML processing is at the core of many middleware systems. In recent years XML databases have become widely available. This article identifies three common XML processing use cases, and compares their performance when XML manipulation is performed in an XML database with equivalent XML manipulation implemented through middleware application code. The article concludes with guidelines for XML processing placement and identifies areas for further study.

Categories and Subject Descriptors
C.4 [PERFORMANCE OF SYSTEMS]: *Design studies*

General Terms
Measurement, Performance, Design, Experimentation, Languages

Keywords
XML, SQL/XML, XQuery, Database, Middle Tier, Performance

1. INTRODUCTION
Beyond XML as a message format, there is a continuous need to store XML permanently: sometimes for auditing and compliance reasons, sometimes because XML is a more flexible and suitable data format than a rigid relational database schema, and sometimes because using XML can simplify applications and improve their efficiency. Storing XML "intact" can often be more efficient than shredding and reconstructing XML to/from relational tables. In response to this trend, the major database vendors have added XML capabilities to their products [3][4][8] and several XML-only databases have emerged [2]. XPath and XQuery have been standardized as XML query languages. Additionally, the SQL standard has been extended with an XML data type and XML-specific functions, collectively known as SQL/XML [1]. SQL/XML allows XPath and XQuery to be embedded in SQL statement and is commonly used in XML database applications [5]. Additionally, databases offer XML indexing and update capabilities to support efficient search and manipulation of XML.

Middleware'10, Nov 29 – Dec 3, 2010, Bangalore, India.

When XML messages are stored in a database system, application designers often have a choice whether to perform XML manipulation in the application code (frequently using DOM, SAX, or XSLT) or in the database server with XQuery or SQL/XML. For example, extracting element values from an XML document, splitting a large XML document into smaller XML pieces, or modifying existing XML documents are common operations. They can be performed either in the middle-tier application code or as part of database insert, update, and query statements. In this paper we investigate this trade-off in terms of performance and development cost (lines of code).

Section 2 describes three common XML use cases that we call *extract*, *split* and *modify*. Section 2 also explains how the use cases are implemented with database and middleware technology, respectively. Performance measurements for each use case are presented in section 3. Finally, sections 4 and 5 summarize the results, suggest guidelines for XML processing placement based on the results, and indicate directions for future work.

In this paper, we use the following terms:

- **middle-tier software** or **middle-tier** to refer to software that typically executes in middleware or application servers
- **database software** or **database** to refer to software and processing that typically executes inside a database system

2. THREE XML PROCESSING USE CASES
The use cases and measurements investigated in this paper reuse parts of the TPoX (Transaction Processing over XML) benchmark [6]. TPoX is an application-level XML benchmark based on a financial application scenario. TPoX defines several XML document collections, XML queries and updates, and provides an XML data generator as well as a workload driver. The workload driver is a Java application that simulates and measures one or multiple concurrent users performing XML processing.

Our use cases operate on 50,000 "custacc" documents from TPoX, each document describing a customer with his personal information and all of his one or more investments accounts. Each use case is implemented in two ways, (1) using SQL/XML to push the XML manipulation to a DB2 9.7 database instance, and (2) in Java with a DOM or SAX parser in the middle-tier. Both options read or write XML to/from the database, only the location and implementation of the XML manipulation differs. The SQL/XML implementation for each use case, performs the XML

manipulation as part of the database read or write operations. The equivalent middle-tier implementation of each use case reads and writes full document from/to the database, but XML specific operations happen inclusively in the middle-tier code.

The three use cases and their sub-cases investigated in this paper are summarized in Table 1.

Table 1. XML Processing Use Cases

Use Cases	Names	Descriptions
1.1	Extract5	Extract a small number of values from each randomly selected XML document (approx 5% of the element values) from the 50,000 documents.
1.2	Extract50	Extract a medium number of values (50%).
1.3	Extract100	Extract most values (almost 100%).
2.1	Split500	Split a large XML document containing 500 concatenated XML fragments into individual XML documents. Concatenated XML documents are often used for download or FTP, but are not useful for individual processing.
2.2	Split2500	Split a document containing 2500 fragments.
2.3	Split5000	Split a document containing 5000 fragments.
3	Modify	Insert, change, and delete XML elements in each randomly selected XML document from the collection of 50,000 documents.

The following sub-section describes the use cases in more detail. Each use case description consists of two parts:

- The use case as implemented in the middle-tier, e.g., with DOM or SAX in a stand-alone XML parser
- The use case as implemented in database, e.g., with SQL/XML inside a database

Further details on the use case implementation, including code, will be made available at https://www.ibm.com/developerworks/wikis/display/ettk/MiddleTier [11].

2.1 Use Case 1: Extract

The first use case extracts values from stored XML documents and populates them into Java objects. The use case contains three extractions: **Extract5**, **Extract50**, and **Extract100** as described in Table 1.

The **middle-tier** implementation reads a binary large object (BLOB) containing XML from the database and parses it with a SAX parser in the application (Figure 1). During the parsing process XML node values are extracted and stored in Java objects.

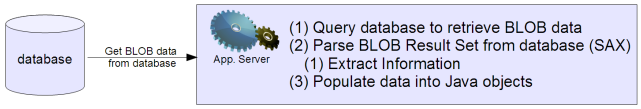

Figure 1: Implementation of Use Case 1 in the Middle-Tier

For the **database** implementation, the XMLTABLE function defined in the SQL/XML standard [1] is used to extract XML values into a relational result set. The result set is fetched and populated into Java objects. The following is the SQL/XML statement for **Extract5**:

```
SELECT T.*  FROM custacc_xml ,
XMLTABLE(XMLNAMESPACES(DEFAULT 'http://tpox-
benchmark.com/custacc'),'$DOC/Customer'
COLUMNS
firstname VARCHAR(20) PATH 'Name/FirstName',
lastname  VARCHAR(20) PATH 'Name/LastName',
dayOfBirth DATE     PATH 'DateOfBirth') AS T
WHERE CUSTID = ?
```

Both the middle-tier and the SQL/XML tests randomly pick customer IDs to select XML documents from the database to perform element extraction.

2.2 Use Case 2: Split

The second use case splits a large XML document into smaller documents (individual customer documents). We test three different sizes, **Split500**, **Split2500**, and **Split5000** as described in Table 1. Splitting large XML batch files is a common scenario in business applications. The purpose is to break a batch of business objects into individual XML documents.

To split a XML batch file in the **middle-tier**, the XML file is read from the file system and parsed into a DOM tree. The DOM tree is split into smaller chunks of XML data and new DOM trees are created. Each new DOM tree contains one `Customer` element (Figure 2). Each DOM tree is serialized into a Java Byte stream and stored as an XML document into the database.

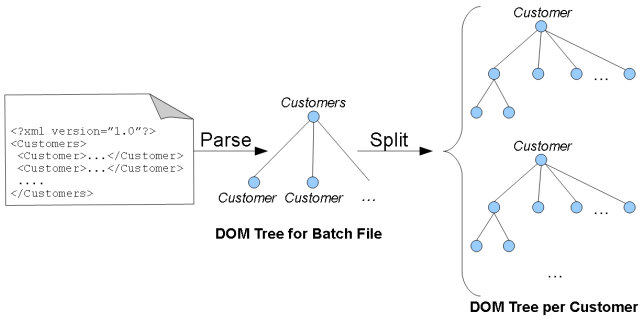

Figure 2: Splitting an XML document

For the **database** implementation of this use case, the entire XML batch file is sent as a binary stream to the database as a parameter of an SQL/XML INSERT statement. The binary stream is parsed by the database and then split into smaller documents using the XMLTABLE function [7]. The split XML documents are inserted into an XML column. All this happens in the following statement:

```
INSERT INTO custacc_xml(DOC)
 SELECT X.frag
 FROM XMLTABLE( XMLNAMESPACES(DEFAULT
       'http://tpox-benchmark.com/custacc'),
     '$batch/Customers/Customer'
        PASSING cast(? as xml) as "batch"
   COLUMNS frag XML PATH 'document{.}') AS X
```

2.3 Use Case 3: Modify

The third use case inserts, updates, and deletes individual elements within a stored XML document. The **middle-tier** implementation of this use case reads a random XML document from a BLOB column in the database, parses the document into a DOM, and modifies the DOM by inserting, changing, or deleting an element as desired. Then the modified document is serialized

and written back to the database, replacing the original document with the modified one. The **database** implementation of this use case sends a SQL/XML Update statement with XQuery Update expression [10] to the database. For example, the following update identifies a random document via its WHERE clause and then applies the XQuery Update in the SET clause to replace the value of an "Email" element in the document.

```
UPDATE custacc SET doc =
 XMLQUERY('declare default element namespace
 "http://tpox-benchmark.com/custacc";
    copy $c := $DOC
    modify do replace value of
 $c/Customer/Addresses/EmailAddresses/Email[1]
            with "newemail@newdomain.com"
    return $c')
WHERE CUSTID = ?
```

3. PERFORMANCE RESULTS

All tests were performed on a x86 single core machine running Windows XP with 1.66 GHz CPU speed and 2 GB of main memory. To simplify the monitoring of the overall CPU consumption, both the Java application code and the DB2 database server were running on the same machine.

3.1 Performance of Use Case 1: Extract

Figure 3 shows the average throughput of the three variants of use case 1 (**Extract5**, **Extract50**, **Extract100**) over a period of 10 minutes. Test runs longer than 10 minutes did not produce significantly different results. The SQL/XML-based value extraction in the database achieves notably higher throughput (in transactions per second) than middle-tier SAX parsing, especially when the number of extracted values is small. The reason is that a database with native XML capabilities, such as DB2, stores XML in a parsed format and can perform XML value extraction without renewed XML parsing. This benefit diminishes when many or most of the values are extracted from a document, because in that case multiple SQL statements and very wide result sets need to be processed.

All tests for use case 1 were performed with a single user. Both the SQL/XML (database) and the SAX (middle-tier) tests used approximately 70% of the CPU capacity of the system.

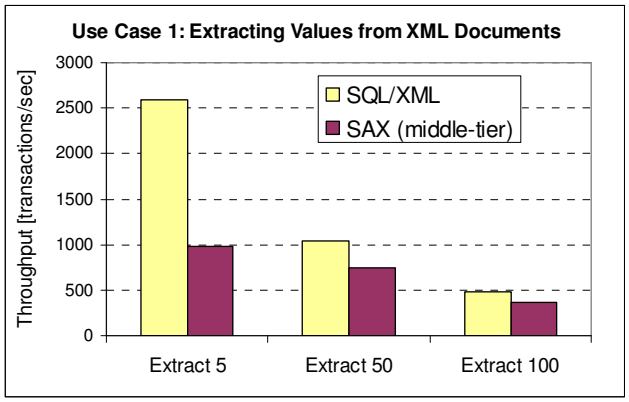

Figure 3: Extraction in the middle-tier vs. database

Figure 4 shows that the throughput for all variants of use case 1 were stable over the 10 minute runs. Each data point represents the average throughput within an interval of 60 seconds.

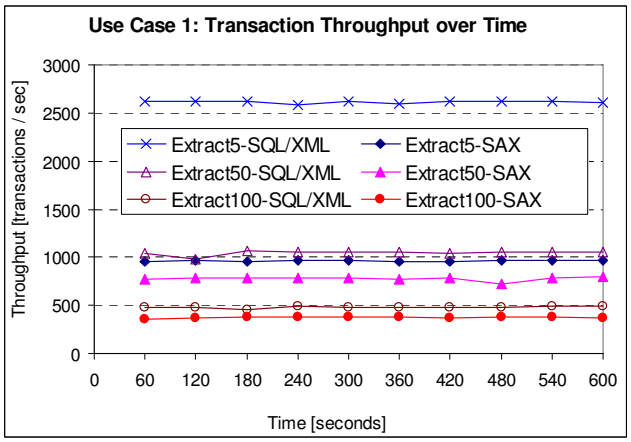

Figure 4: Stability of Throughput in Use Case 1

3.2 Performance of Use Case 2: Split

The performance of use case 2 is measured in terms of the elapsed time (Figure 5) and CPU consumption (Figure 6) for splitting the large XML file into smaller documents and inserting them into the database. Splitting a small batch file (**Split500**) took about the same time with DOM and SQL/XML, but the CPU consumption of the DOM-based middle-tier implementation was almost twice as high as for the SQL/XML processing. For the database, there is only one SQL statement, whereas with middle-tier there are many SQL inserts, e.g., 500 inserts for SPLIT500, which increases the elapsed time because of the increased interactions between the processes in the middle-tier and database.

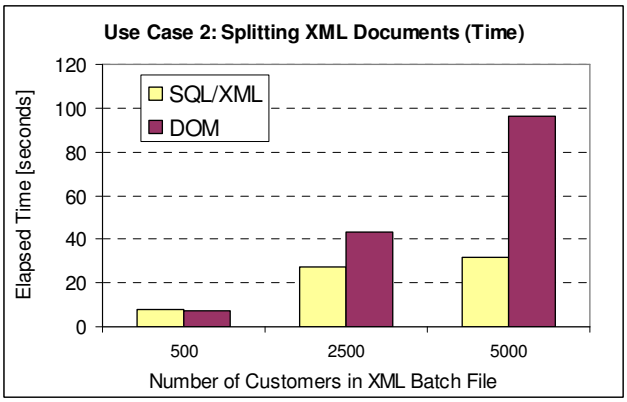

Figure 5: XML Splitting in the middle-tier vs. database

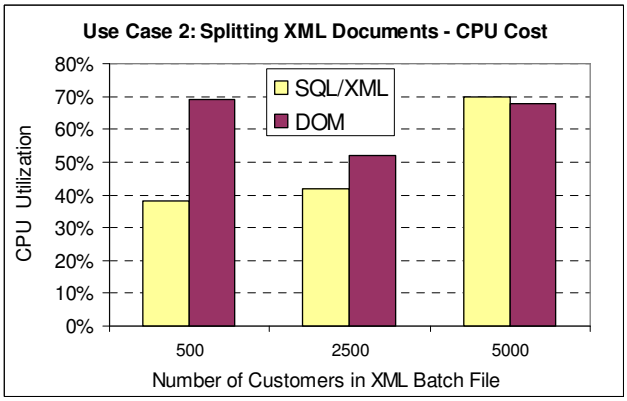

Figure 6: XML Splitting in the middle-tier vs. database

The larger the XML batch file the bigger the elapsed time benefit of SQL/XML over the middle-tier. For a large batch file of 5000 customers, SQL/XML was 3x faster at about the same CPU utilization as the DOM implementation.

3.3 Performance of Use Case 3: Modify

The performance of use case 3 is measured in transactions (updates) per second when one or multiple concurrent users perform changes on XML documents. In our tests, each concurrent user selects a random document for update, so the probability of excessive lock contention is low.

Figure 7 shows the update throughput averaged over a period of 10 minutes. For a single user, the XML modify performance is 2.25x higher in the SQL/XML database implementation than in the DOM-based middle-tier implementation. This advantage increases to 2.5x for 5 user, and to 2.6x for 10 and 15 users.

There are two reasons why the SQL/XML updates perform better than the middle-tier updates. First, the SQL/XML updates can modify the XML documents in the parsed storage format of the database without incurring renewed XML parsing. Second, SQL/XML updates can modify a stored XML document without sending it from the database server to the client and back.

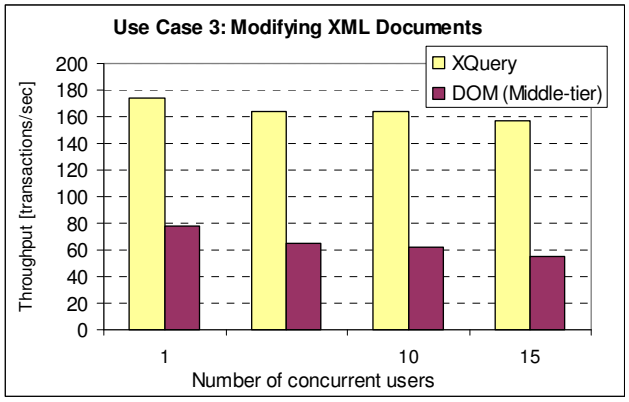

Figure 7: Modifying XML in the middle-tier vs. database

3.4 Development Cost

Figure 8 shows the number of lines of code required to implement each of the use cases in either Java for the middle-tier or in SQL/XML for database-side processing. The number of lines of code is a common metric to estimate software development cost, number of coding errors, and software maintenance cost over time.

For the database implementation we counted the lines of code in the SQL/XML query, update, and insert statements that we used, as well as the Java code that is required to prepare and submit these statements to the database, retrieve any result set, and bind the result set values to Java objects. For the middle-tier we counted the lines of code in the SQL statements that get and put full XML document from and to the database plus all Java code for SAX- or DOM-based parsing and manipulation of the XML documents and for binding extracted values to Java objects.

For use cases 1.1 through 1.3, i.e. **Extract5**, **Extract50**, and **Extract100**, the middle-tier implementation required 10x as many lines of code as the SQL/XML-based implementation. For use cases 2 and 3 this ratio is 13x and 9x, respectively. A big factor is that the middle-tier implementation uses *procedural code* to traverse and manipulate XML documents explicitly. In contrast, SQL/XML consists of SQL and XQuery, which are *declarative* languages and provide more powerful constructs to query and modify XML documents.

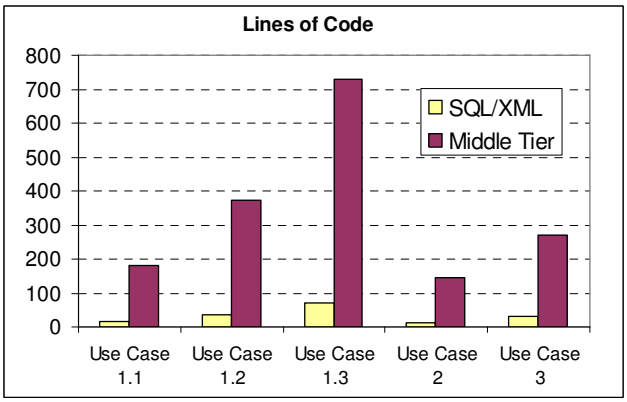

Figure 8: Lines of Code to implement Use cases

4. RESULTS SUMMARY AND GUIDELINES

In all XML processing use cases that we investigated, a SQL/XML-based database implementation outperforms a middle-tier implementation with traditional DOM and SAX parsers for XML. In use case 1, extracting values from XML documents and assigning them to Java objects is up to 3x faster when the extraction happens with SQL/XML rather than SAX. In use case 2, splitting large XML documents into smaller XML fragments with SQL/XML is up to 3x faster than with DOM-based implementation. In use case 3, modifying existing XML documents with SQL/XML and XQuery Updates in the database is 2.2x to 2.6x faster than performing the equivalent operations with DOM in middle-tier code. At the same time, the SQL/XML implementations require 10x to 13x less code than the Java-based middle-tier implementations.

These results confirm our observations in several real-world XML applications where companies have gained significant performance benefits from moving XML manipulation from the middle-tier to the database [5]. Also, offloading XML processing from the middle-tier to a database with native XML capabilities does not imply that the CPU consumption is simply moved from one layer of the IT architecture to another. Using XML functionality in a database rather than in application typically reduce the overall resource consumption, reduces hardware requirements, improves end-to-end performance in terms of response times and throughput, and decreases the cost of developing and maintaining XML-based applications.

From these findings we deduce the following suggestions for XML processing placement.

XML processing should happen as much as possible in the database layer when:

- XML documents are stored in a database for audit, archiving, or mass-query purposes anyway. In that case, using XQuery and SQL/XML capabilities in database read and write operations is advantageous, as shown in this paper.
- the values in the XML documents need to be accessed more than once. In this case an XML database provides a performance benefit through its "parse once, read many" model, XML is parsed only once upon insert and then never again for any subsequent queries or updates.

In contrast, XML processing should happen in the middle-tier when

- the XML is not stored but transient, and processed only "on the fly"
- values inside the XML documents needs to be accessed only once in the lifetime of the document

When the same XML data is being presented in different ways for different clients and can be processed nicely through XSLT, it can be helpful to perform the XSLT processing in the client itself (such as a browser), to strike a balance between client and server CPU consumption.

5. CONCLUSION AND FUTURE WORK

This paper has shown that the XML capabilities in today's databases, such as native XML storage and support for XQuery and SQL/XML, provide powerful and efficient support for XML processing. The results in this paper have shown that XML processing in the database can be several times faster than in the middle-tier. Hence, pushing XML manipulation from the application code down into the database can be an effective way to improve XML-based applications and service-oriented architectures. Along with the performance benefit comes a reduction of application code, often at the order of 10x or more. This reduces the cost to develop new and maintain existing applications.

There are several areas for further study. First, we intend to run use cases 1 and 2 in a concurrent multi-user context, much like use case 3. It will be also be interesting to conduct the same experiments with a database other than DB2, because different XML databases might provide different levels of performance benefits. Likewise it will be fruitful to evaluate other XML processing techniques in the middle-tier such as a StaX parser or novel XQuery and XPath capabilities in application servers. For example, the Websphere XML Feature Pack [9] supports XPath and XQuery in the middle-tier. And finally, another important use case to investigate are XML transformations. Traditionally such transformations are done with XSLT which often requires XML parsing. However, certain transformations can be implemented in XQuery which a native XML database can typically process without XML parsing.

6. REFERENCES

[1] Eisenberg, A., Melton, J. "Advancements in SQL/XML", SIGMOD Record, 33 (2), 2004

[2] Holstege, M.: "Big, Fast, XQuery: Enabling Content Applications", IEEE Data Engineering Bulletin, Vol. 31 No. 4, 2008. http://sites.computer.org/debull/A08dec/marklogic.pdf

[3] Murthy, R. et al.: "Towards an enterprise XML architecture", SIGMOD 2005.

[4] Nicola, M., van der Linden, B.: "Native Support XML in DB2 Universal Database", 31st International Conference on Very Large Databases, VLDB 2005.

[5] Nicola, M.: "Lessons Learned from DB2 pureXML Applications A Practitioner's Perspective", 7th International Database Symposium, XSYM 2010.

[6] Nicola, M. et al.: "An XML Transaction Processing Benchmark", ACM SIGMOD Conference 2007. http://tpox.sourceforge.net/

[7] Rodriguez, V., Nicola, M.: "XMLTABLE By Example, Part 2: Common scenarios for using XMLTABLE with DB2", IBM developerWorks, 2007. http://www.ibm.com/developerworks/data/library/techarticle/dm-0709nicola/

[8] Rys, M.: "XML and Relational Database Management Systems: Inside Microsoft SQL Server", SIGMOD 2005.

[9] Spyker, A., van der Linden, B.: "Programming XML across multiple tiers", IBM developerWorks, 2010. http://www.ibm.com/developerworks/xml/library/x-xmlfeat1/index.html

[10] XQuery Update Facility, http://www.w3.org/TR/2006/WD-xqupdate-20060711/

[11] Code samples of the tests in this paper will become available here: https://www.ibm.com/developerworks/wikis/display/ettk/MiddleTier