An XML Transaction Processing Benchmark

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

Irina Kogan
IBM Toronto Lab
8200 Warden Avenue
Markham, ON, Canada
ikogan@ca.ibm.com

Berni Schiefer
IBM Toronto Lab
8200 Warden Avenue
Markham, ON, Canada
schiefer@ca.ibm.com

ABSTRACT
XML database functionality has been emerging in "XML-only" databases as well as in the major relational database products. Yet, there is no industry standard XML database benchmark to evaluate alternative implementations. The research community has proposed several benchmarks which are all useful in their respective scope, such as evaluating XQuery processors. However, they do not aim to evaluate a database system in its entirety and do not represent all relevant characteristics of a real-world XML application. Often they only define read-only single-user tests on a single XML document. We have developed an application-oriented and domain-specific benchmark called "Transaction Processing over XML" (TPoX). It exercises all aspects of XML databases, including storage, indexing, logging, transaction processing, and concurrency control. Based on our analysis of real XML applications, TPoX simulates a financial multi-user workload with XML data conforming to the FIXML standard. In this paper we describe TPoX and present early performance results. We also make its implementation publicly available.

Categories and Subject Descriptors
H.2.4 [**Database Management**]: Systems–*transaction processing*.

General Terms
Measurement, Performance, Design. Experimentation

Keywords
XML, Database, Benchmark, XQuery, SQL/XML, TPoX

1 INTRODUCTION
Comprehensive and efficient support for XML data management is an increasingly important requirement for database systems. This includes storage, manipulation, search and retrieval of XML data while guaranteeing transactional consistency, recoverability, high availability, performance and usability. All major relational database systems offer some form of XML support [18][21][22] and there is also a variety of XML-only databases such as Tamino, X-Hive, Ipedo, Xyleme, Neocore, and others [6]. Additionally, there are open-source and research implementations of XML databases and XQuery processors, including Galax, MonetDB, eXist, Saxon and Timber [6][15]. However, an industry standard XML database benchmark to compare different systems is missing. Neither the Transaction Processing Council (TPC, *tpc.org*) nor the Standard Performance Evaluation Corporation (SPEC, *spec.org*) have announced plans to develop and standardize an XML database benchmark.

At the same time there is increasing demand for and adoption of XML database technology in commercial enterprises in virtually every industry sector, including finance and banking, insurance, government, retail, health care, and manufacturing. For many IT decision makers this raises the question of how to compare XML databases. Performance is always among the most critical criteria, and, therefore, an XML database benchmark is required.

In addition to feature-specific micro-benchmarks that focus on core XML processing operations, application-level workloads are important to assess the performance of an entire system as a whole. The research community has proposed various XQuery and XML database benchmarks, e.g. XMach-1 [4], XMark [24][25], XPathMark [9], XOO7 [7], XBench [30], MBench [23], and MemBeR [3][16]. Some are predominantly application-oriented, such as XMach-1 and XBench, while others are designed as abstract micro-benchmarks, e.g. MBench and MemBeR. XMark, XPathMark and X007 can be viewed as a blend because their data and queries represent a fictitious application scenario but they also try to exercise all relevant aspects of the XQuery and XPath languages. Further analysis and comparison of the benchmarks can be found in [1][5][17]. Additionally, [15] describes the execution and results of five of the benchmarks on six open-source XQuery processors/databases.

Except for XMach-1, all of these benchmarks focus predominantly on XQuery processing rather than on evaluating a complete database system. Indeed, most of the benchmarks define queries only, despite real-world requirements for insert, update and delete operations. Many of them are also designed as single-user tests on a single large XML document. Such tests can be very valuable to investigate design alternatives and optimizations in an XQuery processing engine. However, these benchmarks are not sufficient to stress all performance-relevant components of a full-fledged XML database system and concurrent user activity.

XMach-1 distinguishes itself from the other benchmarks since it defines a read/write multi-user workload over many small XML documents. It also includes a basic form of XML document variability. With these characteristics, XMach-1 pursued some of the same goals as TPoX. However, there are several significant differences between TPoX and XMach-1, which we discuss in Section 2.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGMOD'07, June 12–14, 2007, Beijing, China.
Copyright 2007 ACM 978-1-59593-686-8/07/0006...$5.00.

We find that none of the existing benchmarks matches our goal to comprehensively represent a realistic application scenario. We carefully looked at real-world XML applications, including and beyond those characterized in [19], and devised a list of requirements for an application-oriented XML database benchmark. Subsequently we designed and implemented TPoX as an attempt to meet these requirements. We are making TPoX available as open source at http://tpox.sourceforge.net/ [27] and invite any interested groups to use it and to provide feedback or contributions to extend and improve the benchmark.

Based on our analysis of XML applications, we describe a set of benchmark requirements in Section 2 and compare them to the existing benchmarks. Section 3 describes the design and implementation of TPoX as an attempt to meet these requirements. Section 4 presents early results of TPoX on a commercial database system. Section 5 concludes with future directions.

2 XML BENCHMARK REQUIREMENTS

In this section we discuss our XML database benchmark requirements relative to the existing benchmarks. We do realize that some of the benchmarks were never intended to meet many of these requirements or were restricted by the available language standards and implementations at their time of definition. The following paragraphs in this section describe each of our criteria and its importance to a benchmark definition. This discussion is summarized in Table 1.

Data- vs. document oriented. XML applications can roughly be classified as predominantly data-oriented or predominantly document-oriented. For example, order or sales processing is typically data-oriented while the management of contracts, emails or news articles is document-oriented. In hybrid applications with characteristics of both, one of the two areas often determines overall performance more than the other. Data- and document-centric XML applications often differ significantly in the size and structure of their XML documents and their prevalent operations. For example, document-centric applications often handle larger documents or use full-text search operations much more than data-centric applications. Thus, we believe that two separate XML benchmarks are required, just as there are separate benchmarks for OLTP (TPC-C) and Decision Support (TPC-H/R) for relational databases. TPoX models a data-centric scenario while XMach-1 is predominantly document-centric.

Very large number of small documents. We see that many data-centric XML applications deal with millions to billions of relatively small XML documents, i.e. less than 100KB, often even less than 20KB [10][11][19]. Only XBench and XMach-1 define multi-document tests, over 2.5M and 10M documents, respectively, but they only touch the low end of what we think is the required scale. TPoX is designed to scale from millions to billions of XML documents.

Document and schema variability. XML is often used for its flexibility, i.e. to make applications more resilient to diversity and change in message formats, business forms, and other types of documents. This is a critical property to capture in a benchmark, which so far has only been addressed by XMach-1. XMach-1 uses many different DTDs with 2 – 100 instance documents per DTD. The DTDs differ only in an integer number that is appended to most element names. Although this is a simple yet effective approach, we find it difficult to match this approach to data diversity in real XML applications. In TPoX we address data variability through the use of a complex real-world XML Schema (FIXML, *Financial Information eXchange Markup Language*). FIXML defines thousands of optional elements and attributes but only a very small subset appears in any given instance document. Additionally, we are investigating how to introduce "unpredictable" XML Schema changes into a benchmark implementation.

Multi-user tests are imperative. *System under test (SUT) is the database*. No real-world XML database application serves a single user. XMach-1 includes multi-user tests, but defines the SUT to consist of database server *plus* application server. This is an

Table 1: Comparison of XML Benchmarks (extended from [5])

	XMach-1	**XMark**	**XPathMark**	**XBench**	**XOO7**	**MBench**	**MemBeR**	**TPoX**
Main data focus	doc-centric	data-centric	data-centric	doc & data	data-centric	data-centric	data-centric	data-centric
Evaluation scope	DBMS + AS	query processor	XPath processor	entire DBMS	query processor	micro, query processor	micro, query processor	entire DBMS
# user	single/multi	single	single	single	single	single	single	single/multi
# server	≥ 1	1	1	1	1	1	1	≥ 1
# documents used	10k – 10M	1	1 of 2	1 - 2.5M	1	1	1 of 3	3M – 360B
Document sizes	2 – 100KB	10MB – 10GB	10MB – 10GB	1KB – 10GB	4MB – 1GB	50MB – 50GB	11MB	2 – 25 KB
DB size	156MB – 152GB	10MB – 10GB	10MB – 10GB	10MB – 10GB	4MB – 1GB	50MB – 50GB	11MB	10GB – 1PB
# schemas/DTDs	#DTDs = #docs/20	1	2	1	1	9	3	3 (43 XSD files)
# element types	4*#DTDs +7	74	74+	~200	9	2	19	~3600
Metric	throughput	response time	response time	response time	response time	response time	response time	throughput, resp.time
Schema validation	optional	no	no	no	no	no	no	yes
Namespaces	no	no	yes	no	no	no	no	yes
# queries	8	20	57	67	23	49	17	7+
# value updates	1	0	0	0	0	0	0	2
# structural updates	0	0	0	0	0	5	0	4
# doc deletes	1	0	0	0	0	0	0	2
# doc inserts	1	0	0	0	0	1	0	2
# joins across 2+ document types	0	0	0	1	0	0	0	3+
Data diversity	yes (tag names)	no	no	no	no	no	no	yes
Schema evolution	no	no	no	no	no	no	no	no

important scenario to cover, but the application server impacts overall system performance significantly, which makes an isolated assessment of database performance very difficult. In particular, XMach-1 allows queries to be processed by application code in the application server. With today's XML database technology supporting XQuery and SQL/XML processing, this is no longer needed.

Read/Write workload. A read-only workload (XMark, X007, XBench, XPathMark, MemBeR) ignores critical aspects of XML data management, such as concurrency control, logging, and index maintenance. Without writes, an unrealistic number of indexes and materialized views can be defined to optimize query performance without any associated cost or performance penalty. In fact, we have seen such questionable use of a read-only XML benchmark in a commercial setting.

XMach-1 includes full document insert and delete operations as well as value updates that modify attribute values but do not change a document's structure. These inserts/updates/deletes constitute only 2% of the XMach-1 workload mix, which we consider too low to stress all database system components. We observe that financial transaction processing databases often have 10% to 50% writes in their workload, especially if many reads are satisfied by application layer caches. TPoX defines a mixed workload of 30% writes and 70% reads.

XMach-1 also allows queries to see stale data since committed updates are allowed to take up to 30 seconds before they have to be reflected in query results. This can be reasonable for some web applications but is not acceptable for the financial scenario in TPoX or most any of the financial XML applications we have analyzed. Unlike XMach-1, MBench includes structural updates but is a micro-benchmark with a single very artificial XML document.

Multiple document types and joins. We see that an increasing number of XML applications use more than one document type as well as joins to combine them. Only XBench includes such a join (and only one).

Namespaces. We require the use of multiple namespaces in XML Schemas, instance data, and queries. This is common in real-world applications. Storage, indexing and query processing of namespaces can be subject to performance optimization and must not be ignored. Except XPathMark, none of the existing benchmarks uses namespaces.

Schema validation is required in many (but not all) XML applications and can significantly affect performance. It also produces type annotations for XML nodes which can impact XQuery processing performance. The existing benchmarks allow schema validation, but none of them requires it as a mandatory operation.

In the next section we describe how TPoX tries to address these requirements.

3 THE TPoX BENCHMARK

In [12], Gray states that database benchmarks should be domain-specific and relevant to a certain type of application, portable across platforms, scalable and simple. Portability, simplicity, and scalability are technical requirements that need proper attention during benchmark design and implementation. Relevance to a specific application domain is equally important and is the starting point for our TPoX design.

3.1 The TPoX Application Domain

Businesses in every industry are embracing XML for vertical applications. We see a particularly high XML adoption rate in the financial industry. The world's leading financial companies have developed more than a dozen XML vocabularies to standardize their industry's data processing [28]. FpML, FIXML, SwiftML, IFX, MISMO, OFX, and XBRL are among the most popular. FIXML 4.4 is an industry-standard XML Schema for trade-related messages such as trade capture reports, buy/sell orders, and many others [10]. The FIX (Financial Information eXchange) protocol is used by more than 150 leading financial companies worldwide. The XML version, FIXML, has been developed to improve extensibility, application layer independence, message validation, and robustness. FIXML also enables straight-through processing, which reduces operating costs and improves the quality and timeliness of information [8]. Many of the major U.S. exchanges and clearinghouses are starting to roll out FIXML projects, e.g. at the Chicago Mercantile Exchange [8] and the Options Clearing Corporation [20], which is one of the world's largest equity clearing firms.

The FIXML Schema 4.4 consists of 41 schema documents [10]. It contains more than 1300 type definitions and more than 3600 elements and attributes. The vast majority of those are optional and only a small subset of them are present in any given instance document. Typical FIXML applications perform message-based transaction processing involving large numbers of small XML documents. All these characteristics are very similar to many other financial applications and their XML Schemas, such as OFX and FpML. The OFX (Open Financial Exchange) schema consists of 59 XSD files, ~2500 elements and ~800 type definitions. The FpML (Financial products Markup Language) has 21 XSD files, 1730 elements and attributes, and 600 type definitions. Like FIXML, OFX and FpML are adopted by major financial institutions world-wide [11].

Supporting financial companies in their adoption of XML has helped us understand their data and processing characteristics. For example, we have worked with multiple brokerage and securities processing companies on storing and querying FpML, FIXML and other financial data in XML format. We decided to design a benchmark that is relevant to this application domain and reflects the data and query patterns that we have seen. Our benchmark simulates an online trading scenario and uses FIXML to model some of its data.

TPoX is purposefully simplified, yet still representative in terms of documents, transactions, and usage of XML Schemas. Another important aspect of TPoX is the flexibility and extensibility of its implementation. The data distributions, transactions, workload composition, data and multi-user scaling, commit frequencies, think times, etc. are all controlled by configuration parameters. To propose a reference workload, we have chosen specific parameter

values in this first version of TPoX. However, any parameter can be changed anytime, which makes TPoX a versatile performance test harness for XML databases [27].

3.2 TPoX Data and XML Schemas

Figure 1 shows TPoX's main logical data entities. Customers have one or more accounts. For each account, one or more orders are executed. Each order buys or sells shares of exactly one security. A security is a stock, bond or mutual fund. Each account contains one or more holdings. A holding, also called position, is a certain number of shares of a particular security in an account. Each security typically has many orders and holdings across the customers' accounts.

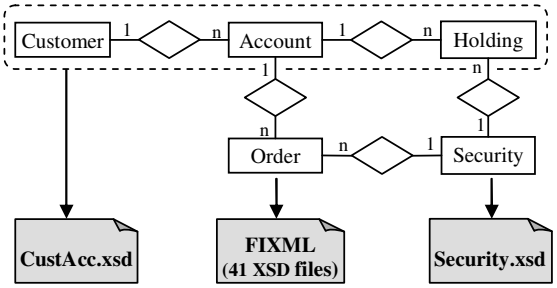

Figure 1: TPoX Entities and XML Schemas

TPoX's data entities are represented by three XML schemas. Orders are represented using the FIXML 4.4 schema. Then there is one XML document per customer that includes personal data and information about all of his accounts and holdings, i.e. account and holding data are inlined with the customer data ("CustAcc"). This is an intentional design choice to reward technological progress for updates and concurrency control on a sub-document level.

A fixed number of 20833 security documents represents the vast majority of US-traded stocks, bonds and funds with real ticker symbols, real fund families, etc. Security documents range from 3KB to 10KB because stock and fund descriptions have relatively large text values of variable size. This allows for some text operations despite the data-centric focus of TPoX. CustAcc documents are between 4KB and 20KB in size. Orders are 1KB to 2KB and characterized by many attributes and a high ratio of nodes to data. All TPoX documents contain namespaces.

The three document collections are interrelated, e.g. Order documents contain security symbols and account numbers that exist in the Security and the CustAcc documents, respectively. We use the Toxgene data generator [26] to produce instance documents for all three schemas. We developed code to partition and parallelize the data generation in a configurable number of concurrent Toxgene sessions. This improves the performance and scalability of the data generation while preserving referential consistency across all generated documents. The generated documents are spread over a configurable number of directories to avoid deterioration of file system performance when millions of files are placed into a single directory.

The TPoX data generation is defined in Toxgene templates and characterized by various uniform and non-uniform distributions. They control element occurrences, appearance of optional elements and attributes, as well as data values within and across documents. For example, the number of accounts per customer is drawn from a normal distribution with min=1, max=7, mean=1, and variance=2, expressing that customers with one or two accounts are a lot more common than customers with many accounts. Similar concepts of "data skew" are applied to the number of occurrences of phone numbers, middle names, holdings and other variable data items. A customer's PIN, password and either social security number or taxpayer ID are stored encrypted as Base64Binary type values. All Security documents contain a name, symbol, price, etc. Additional sub-trees do or do not occur depending on the type of the Security, e.g. element "FundInformation" vs. "StockInformation". For more details on the XML Schemas and data generation, see [27].

Table 2: TPoX Scaling

	Scale	Total #documents	~Raw size	Min. #users
XS	1	3,620,833	10GB	10
S	10	36,020,833	100GB	100
M	10^2	360,020,833	1TB	1,000
L	10^3	3,600,020,833	10TB	10,000
XL	10^4	36,000,020,833	100TB	100,000
XXL	10^5	360,000,020,833	1PB	1,000,000

In TPoX, the database can be scaled from extra small (XS) to extra-extra large (XXL) by increasing the number of Order and CustAcc documents (Table 2). We use an average of five orders per customer, e.g. 3,000,000 Order and 600,000 CustAcc documents at scale factor XS. Smaller or intermediate scale factor are possible, e.g. for functional testing. Multi-user tests simulate at least as many concurrent users as Gigabytes of raw data used.

The proposed scaling of the TPoX database goes far beyond the size of the existing XML benchmarks to match the expected future growth of real XML applications. We scale the TPoX database in steps (like TPC-H) rather than with a continuous function (like TPC-C). Since competitive XML database technology can be quite different for small vs. large databases, it may not be meaningful to use a continuous scaling function to compare systems that are intended for different size databases.

3.3 TPoX Transactions and Workload

The TPoX benchmark is executed in two stages. *Stage 1* performs concurrent inserts to populate the database and maintain all desired indexes at the same time. There is no separate stage for building indexes. The number of documents and required degree of concurrency ("number of users") is defined in Table 2. *Stage 2* performs a multi-user read/write workload on the populated database, with 70% queries and 30% write operations (inserts, updates and deletes combined). Both stages are executed by the workload driver described in Section 3.4. Optionally, an additional *read-only stage* can be executed (100% queries) in which all queries have equal weight.

The mixed workload consists of 17 transactions. Each performs one or more of the following operations: insert, delete, structural update, value update or query. Queries are expressed in XQuery and can be executed as-is or embedded in SQL, e.g. through the use of SQL/XML functions, such as XMLQUERY and XMLEXISTS. The number of queries in TPoX is lower than in

many of the existing benchmarks because it is not our goal to exercise every feature of the XQuery language. This is better done by micro-benchmarks. Instead, the objective of TPoX is to distil the most performance-relevant aspects of a real application scenario into a concise workload. For comparison, the TPC-C benchmark is highly relevant with only five different transactions.

Table 3 describes the business meaning of the TPoX transactions and their relative weight in the workload. The exact query and update statements are provided in the Appendix. Table 4 maps the transactions to database operations. Additional candidate queries for the TPoX scenario are offered in [27], including 3-way joins.

Table 3: Business Descriptions of TPoX Transactions

I1	Customer places a new order (insert order document)	7%
I2	Add a new customer (insert CustAcc document)	1%
D1	An order is cancelled or archived (delete order doc)	7%
D2	Remove a customer (delete CustAcc document)	1%
U1	Close an existing customer's account	1%
U2	Open a new account for an existing customer	1%
U3	Update the price of a security	3%
U4	Update the status of an order	3%
U5	Execute a "buy" order of a given security & account: 1. If shares already exist, increase the quantity; otherwise, add a new holding 2. Replace account balances and values dates 3. Abort if the max. number of holdings is exceeded	3%
U6	Execute a "sell" order (opposite of U5)	3%
Q1	Retrieve an order for a given order id	10%
Q2	Retrieve a security for a given ticker symbol	10%
Q3	Get a customer's personal data, construct profile doc.	10%
Q4	Search securities based on 4 predicates and return specific elements of interest	10%
Q5	Construct an account summary and statement	10%
Q6	Retrieve the price of a certain security	10%
Q7	Get a customer's most expensive order	10%

Table 4: Mapping of Transactions to XML Operations

XML Database Operation	TPoX Transactions
Full document insert/delete	I1, I2 / D1, D2
Full document retrieval	Q2
Element/attribute value update	U3, U4, U5, U6
Subtree insert	U2, U5
Subtree delete	U1, U6
Subtree replace	U5, U6
Element construction	Q3, Q4, Q5, Q6, U2, U5, U6
Predicate evaluation	Q1-Q7, U1-U6, D1, D2
*, // processing	Q4
Join across document types	Q7, U5, U6
Aggregation	Q7
Arithmetic on XML values	Q7, U5, U6
Schema validation required	I2, U2, U4

3.4 TPoX Performance Metrics

The primary performance metric of the benchmark is TTPS (*TPoX Transactions Per Second*) which is the throughput of the multi-user read/write workload (*stage 2*) at a given scale factor. Additionally we recommend that two secondary metrics, TIPS and TQPS, are reported for the same scale factor. TIPS is the throughput of the insert workload (*stage 1*), TQPS is the throughput of the optional read-only workload. TTPS, TIPS and TQPS must be the throughput as reported by the TPoX workload driver for the respective part of the benchmark. The steady-state measurement interval should be at least 1 hour.

Table 5: TPoX Performance Metrics

Primary metric:	TTPS (*TPoX Transactions Per Second*)
Secondary metrics:	TIPS (*TPoX Inserts Per Second*)
	TQPS (*TPoX Queries Per Second*)

The system under test (SUT) includes the database system, the operating system, the workload driver, and the hardware of the database server including storage and all auxiliary components. If the workload driver runs on a separate client machine instead of the server then the client and the network are also part of the SUT.

It is not permitted to change the configuration or any tuning parameters of any part of the SUT between the stages of the benchmark. Another requirement is that insert, update and delete operations are immediately reflected in subsequent query results.

3.5 The TPoX Workload Driver

The TPoX workload driver is used for all stages of the benchmark execution. This driver is a lightweight Java application that spawns 1 to *n* concurrent threads. Each thread simulates a user that connects via JDBC to the database and submits a stream of transactions without think times. Each stream is a weighted random sequence of transactions picked from Table 3. Each transaction is assigned a weight that determines the transaction's percentage in the workload mix.

At run time, the workload driver replaces parameter markers in the transactions with concrete values drawn from configurable random distributions or lists of input values. In the transactions, parameter markers can be denoted by question marks, as would be typical for SQL and SQL/XML statements. Alternatively, the workload driver can also detect and use numbered parameter markers denoted by a vertical bar (pipe), i.e. |1, |2, |3 etc.. See [27] for further details.

For example, order IDs for **Q1**, **U4** and **D1** are drawn randomly from the total range of order IDs in the database population, including new orders from **I1**. The transactions, their weights and the eligible input for each parameter marker is described in a *workload description file* which is input to the workload driver.

Figure 2 shows a sample workload description file for a workload consisting of **Q1** (75%) and **Q2** (25%). The two queries are stored in files `get_order.xqr` and `get_security.xqr`, and have only one parameter marker each. **Q1** uses order ids that are uniformly distributed between 103282 and 15103281. **Q2** looks for securities based on ticker symbols chosen randomly from the specified input file. Multiple parameters per transaction are also supported.

```
numOfTransactions = 2

t1 = queries/get_order.xqr
p1|1 = uniform | 103282 - 15103281

t2 = queries/get_security.xqr
p2|1 = file | input/security_symbols.txt

w1 = 75
w2 = 25
```

Figure 2: Sample Workload Description File

Parameter markers in insert statements are fed from pre-generated documents that reside in a pool of directories. This pool of input documents is shared by all concurrent user threads under a synchronized document counter. During the mixed read/write workload (stage 2), the workload driver also uses this counter to dynamically increase the range of documents eligible for subsequent update and delete operations.

In our tests we found that neither the synchronization nor the I/O to input files had a significant impact on the workload performance if adequate I/O bandwidth is provided. The CPU consumption of the workload driver is sufficiently low so that it can run on the same machine as the database system under test. It can also run on a separate client machine if desired.

Figure 3 shows the command line options of the workload driver. The only required parameter is -w to provide a workload description. All other parameters are optional or use reasonable default values if not specified [27]. The execution can be limited either by total run time (-ti) or by the number of transactions that each concurrent user executes (-tr). The former is used for the mixed read/write workload in *stage 2*, the latter for *stage 1* where a specific number of insert transactions is required to correctly populate the database.

```
usage: WorkloadDriver [-h] [-v level] [-dbs DBMS] [-d
database] [-id dbuser] [-pw passwd] [-ht host] [-pt port]
[-pd sec] [-pc n] [-cl level] [-m MB] [-s seed] [-ti sec]
[-r sec] [-tr #txns] [-fto seconds] [-tt milliseconds]
[-cc #txns] [-u #users] [-w filename]

 -h                    help
 -v  <level>           verbosity (2=highest, 0=lowest)
 -dbs <db system>      database system (eg. DB2)
 -d  <database>        database name
 -id <user id>         database user id
 -pw <password>        database password
 -ht <host>            host where DB resides
 -pt <port>            database port
 -pd <#seconds>        stats print delay
 -pc <n>               compute the n-th percentiles
 -cl <conf. level>     compute confidence intervals
 -m  <MB>              memory for -pc and -cl computations
 -s  <seed>            seed to produce random numbers
 -ti <seconds>         benchmark test duration
 -r  <seconds>         ramp-up time in seconds
 -tr <#transactions>   limits #txns executed per user
 -fto <seconds>        forced time out when -tr is used
 -tt <milliseconds>    think time (in ms) for each user
 -cc <#transactions>   commit count (default: 1)
 -u  <#users>          number of concurrent threads/users
 -w  <file name>       workload description file
```

Figure 3: Workload Driver Options

For *stage 2*, a ramp-up period can be specified (-r) that precedes the measurement interval to reach a steady state of transaction throughput. The workload driver reports the average, minimum and maximum response time for each transaction type as well as the throughput in transactions per minute. The response time includes the time to fetch all query results from the database. For each transaction time percentiles and confidence intervals can be computed for their response times, using –pc and –cl respectively. Depending on the verbosity level (-v), the driver can also report the response times, throughput and number of completed transactions separately for each concurrent user. Query result sets (or sequences) can be written to files, if desired.

During a benchmark run, the performance numbers can be emitted every *n* seconds (-pd) to allow analysis of performance behaviour over time. For example, this helps determining whether insert performance is constant as the tables grow (*stage 1*), or how long of a ramp-up time is needed to reach a steady-state (*stage 2*).

3.6 TPoX Extensibility

The TPoX benchmark was designed for multi-user execution of a mixed read/write workload. However, the TPoX implementation also allows query-only, write-only and single-user tests simply by changing the input to the workload driver. In its simplest form of operation, the workload driver can read a directory with queries (one text file per query) and execute each of them a given number of times (-tr) against a database. For example, it is trivial to use our workload driver to run the XMark benchmark [24] in single- as well as multi-user mode.

The TPoX workload we have defined uses no think time between transactions and commits immediately after every transaction. Nevertheless, for testing purposes a think time (-tt) or less frequent commits (-cc) can be specified. Each transaction in our workload consists of a single statement or query expression. Yet the workload driver is also capable of executing multi-statement transactions. Although the workload driver was designed for TPoX, it can run *any* SQL or XQuery workload against a database. We have deployed it successfully on several versions of Linux and Windows as well as on AIX.

We have used the workload driver extensively on DB2 9 [18], but it can be run with minimal changes (if any) against any database that supports JDBC (-dbs). In fact, a university that has tested an early version of TPoX has successfully used it on a database other than DB2. Due to the modular design, the majority of the workload driver logic is independent from a specific database or specific database API. Hence, it is not hard to extend the driver for use with databases or XQuery engines that do not support JDBC.

Since the TPoX data generation is based on templates rather than hard-coded, it is easy to change value distributions, probabilities of element/attribute occurrences, or the ratio of orders to customers. In [27] we provide detailed documentation for the data generation and the workload driver, including source code, to allow easy adoption and modifications of TPoX.

4 EARLY TPoX RESULTS

In this section we discuss a select subset of early results and experiences with TPoX for different scale factors and different OS and hardware configurations. The first one is a 100GB TPoX that we ran on AIX [14]. The second is a 50GB TPoX test that Intel® conducted on Linux to exercise their dual-core Intel® Xeon® 7100 Series processors [13]. In both benchmarks the database system was DB2 9.

DB2 9 provides pureXML™ technology, which means that XML data is stored and processed as type-annotated trees [18]. To query XML data, DB2 supports SQL/XML and XQuery through a single *hybrid* query compiler and processing engine. Additionally, path-specific XML indexes can be defined on attributes or elements to speed up predicate evaluation and joins. The usage of XML Schemas is optional in DB2. For maximum flexibility, different XML documents in the same XML column can be associated with different schemas, if desired.

For the 100GB tests (scale factor "S"), we used a medium size IBM System p5 560Q server with eight 1.5GHz CPUs and 32GB of memory. The operating system was AIX 5L v5.3 TL04. The

storage subsystem was an IBM TotalStorage DS8100, attached to the server with 4 fiber channels. On the DS8100 we used a total of 64 disks, each 73 GB and 15000 RPM, for the raw input data, DB2 tables and indexes, transaction log, and database backups.

The XML data was generated and stored in an AIX enhanced journaled file system (JFS2). The default block size in a JFS2 file system is 4096 bytes. This leads to internal fragmentation and waste of storage space when 30 million Order files between 1KB and 2KB are stored. Hence, we recreated the JFS2 file system with a bock size of 512 bytes. During the population of the database tables (Stage 1), each input file is read exactly once and never again. This means that file system caching has no benefit and is pure overhead. Hence, we mounted this file system with the -o cio option to enable concurrent I/O and prevent caching.

We configured DB2 to use a 16KB page size for all tables and indexes, and we created three simple tables with a single XML column each:

```
create table custacc ( cadoc XML )
create table order ( odoc XML )
create table security ( sdoc XML )
```

Subsequently, we defined 24 indexes on these empty tables; ten on `custacc`, five on `order`, and nine on `security`. The index definitions are provided in [27]. Before populating the database, we increased the database buffer pool to half the physical memory (16GB), which is common practice. Then we enabled DB2's self-tuning memory manager to allow autonomous and dynamic resizing of the buffer pool, package cache, lock list and other memory areas based on the workload characteristics. No configuration changes whatsoever were made between stage 1 and the subsequent query and mixed workloads.

The performance results of the concurrent inserts in stage 1 are summarized in Table 6. Due to the low and odd number of security documents, we used only 83 concurrent users to insert them (83 * 251 = 20,833). The CustAcc documents were inserted at an average rate of 1550 inserts per second (TIPS). The Order documents are significantly smaller and have fewer indexes defined on them. Hence, a much higher average insert rate of 5320 TIPS was achieved. Both, CustAcc and Order data were inserted and indexed at approximately 30GB/hour. The –pd option of the workload driver reports the current throughput every *n* seconds and allowed us to confirm that the insert rate remained relatively stable as the order table and indexes grew from zero to 30 million XML documents (Figure 4).

Table 6: Stage 1 - Insert Performance (AIX, 100GB)

Table	Documents inserted	TIPS	Concurrent users
custacc	6,000,000	1550	100
order	30,000,000	5320	100
security	20,833	1226	83
TOTAL	**36,020,833**	**3770**	

After populating the database, we performed a series of multi-user query tests using the seven queries with equal weights (Q1 through Q7 from Table 3). We executed this workload for 25, 50, 75, 100, 125, and 150 concurrent users, each time for 1 hour. Figure 5 shows the query throughput (left y-axis) as well as the CPU utilization and I/O wait in percent (right y-axis).

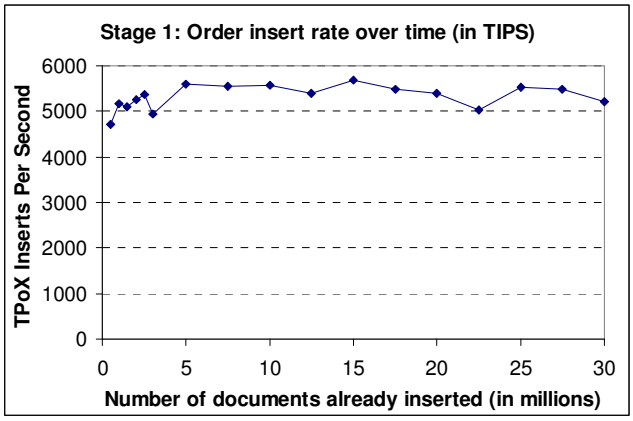

Figure 4: FIXML Order Insert Performance (AIX, 100GB)

The query throughput increased with the number of users as the CPUs were better utilized. Eventually the throughput gradually levelled off as CPU utilization converged towards 100%. The best throughput was achieved with 150 users and reached 5480 TQPS with a CPU utilization of 96%. Increasing the number of users to 175 did not produce a significantly higher throughput since the machine's CPU capacity was already exhausted.

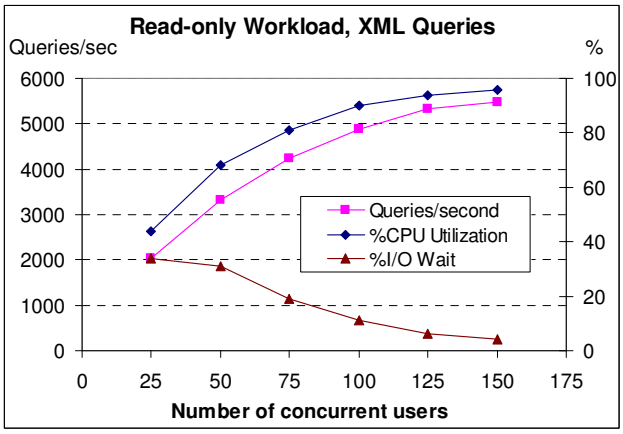

Figure 5: TQPS - Query throughput (AIX, 100GB)

Intel's TPoX tests with 50GB of raw data were performed on an Intel® Xeon® MP Server with four 3.4 GHz Intel® Xeon® 7140M processors and 16GB of main memory. Disk space was provided by two IBM® N5500 storage systems, each housing 56 hard drives. This setup was the database server for DB2 9.

Additionally, a secondary Intel® Xeon® 7000 server was configured as a client machine to run the workload driver. The operating system on both the client and server was Novell SUSE Linux Enterprise Server 9 SP3.

The set of database tables and indexes were the same as for the 100GB tests on AIX. In addition to the 20,833 securities, 3 million CustAcc and 15 million Order documents were inserted by 100 concurrent users in stage 1. Subsequently, the mixed workload (stage 2) was executed. At the time of these TPoX tests, the updates in our mixed workload were simpler than the ones listed in Table 3 and shown in the Appendix. Due to the lack of an XQuery Update implementation, we only used two "dummy" updates, one for the CustAcc and one for the Security table. Both

```
*** SYSTEM WORKLOAD STATISTICS ***

Tr.# Name              Type  Count    %-age  Total Time(s) Min Time(s) Max Time(s) Avg Time(s)
1    get_order           Q  1351493  10.01  64029.54      0.00        1.16        0.05
2    get_security        Q  1350373  10.00  7056.80       0.00        0.44        0.01
3    customer_profile    Q  1349868   9.99  39587.92      0.00        1.32        0.03
4    search_securities   Q  1350538  10.00  5558.96       0.00        0.42        0.00
5    account_summary     Q  1351407  10.01  45769.59      0.00        1.47        0.03
6    get_security_price  Q  1350714  10.00  6748.53       0.00        0.45        0.00
7    customer_max_order  Q  1349904   9.99  242146.45     0.00        2.99        0.18
8    updcustacc          U  405627    3.00  54336.42      0.00        2.71        0.13
9    updsecurity         U  405852    3.00  6232.79       0.00        1.46        0.02
10   delcustacc          D  269966    3.00  31785.72      0.00        2.58        0.12
11   delorder            D  1350009   9.99  107624.55     0.00        1.56        0.08
12   insertcustacc       I  269877    2.00  20668.00      0.01        2.98        0.08
13   insertorder         I  1351224  10.00  87656.86      0.00        14.44       0.06

*** SYSTEM THROUGHPUT *** The throughput is 225114 transactions per minute (3751.90 per second).
```

Figure 7: Workload Driver Output (Stage 2, 50GB TPoX, Linux, 200 users)

updates simply read a document and replaced it with itself. Also, schema validation was not yet used at the time of these tests.

The mixed workload was scaled from 25 to 200 concurrent users. Figure 7 shows the actual output of the workload driver for the mixed workload with 200 users. The throughput result was 3751.90 TTPS (225115 transactions/minute) which is also the right-most data point in Figure 6. For each transaction, the workload driver reports the number of executions, the min, max and average response times and the total cumulative elapsed time across all executions by all concurrent users. The latest version of the driver at [27] optionally also computes percentiles and confidence intervals for the response times. This feature was not available at the time we conducted the reported measurements. We added this to the workload driver based on reviewer comments.

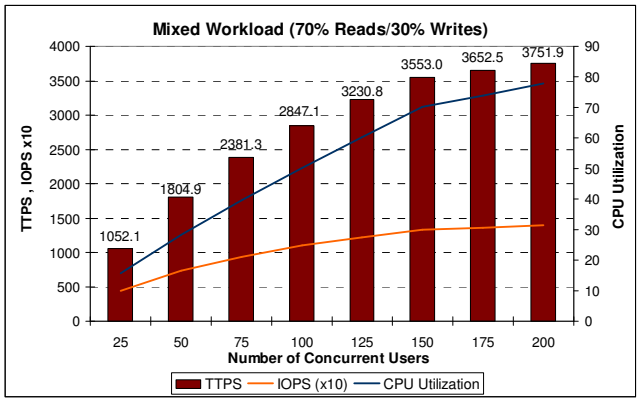

Figure 6: TTPS Read/write throughput (Linux, 50GB)

The performance and scalability of the read/write workload in Figure 6 indicated good scaling with respect to CPU utilization. Throughput and CPU utilization increased for larger number of users but started to flatten out at around 150 users. At this point, the I/O subsystem was near capacity performing approximately 13000 I/O operations per second (IOPS). The slight performance increase above 150 users was a result of even better buffer pool hit ratios.

Further details on the 50GB Linux and 100GB AIX test scenarios can be found in [13] and [14], respectively. Additionally, the flexibility of the TPoX framework allowed for numerous test variations which helped us to better understand the performance behavior of the database system and hardware configurations. For example, the insert and mixed workloads allowed us to examine the critical impact of XML index maintenance, logging, and free space management on the performance of these tests. We also ran tests with additional candidate queries [27] and obtained valuable insights into XML join processing and XML index usage. We emphasize that TPoX does not necessarily require a big hardware setup. Small TPoX runs on a low-cost desktop computer have provided us with very useful results too.

5 SUMMARY & FUTURE WORK

In this paper we described the design and implementation of the TPoX XML Database Benchmark. TPoX is an application-oriented benchmark which models a financial XML processing scenario. The TPoX schemas, data and transactions are based on our experiences with financial institutions that have implemented or evaluated XML data management solutions. TPoX involves XML documents conforming to the FIXML industry standard XML Schema.

The TPoX benchmark distinguishes itself from existing XML database benchmarks in its application domain and its focus on multi-user read/write tests with high scalability using very large numbers of small XML documents. To the best of our knowledge, TPoX is also the first XML database benchmark which includes complex updates based on the XQuery Update Facility [29].

Our contributions include the analysis of existing benchmarks, their comparison to real-world XML applications, the design and implementation of TPoX to fill the identified gaps, and the description of TPoX experiences and results for two scale factors on two different platforms. Additionally we are making TPoX available as an open source project [27] and invite interested parties to use and extend TPoX, e.g. by defining more queries and updates.

We have identified a number of desirable improvements for TPoX. Some of them are already work in progress. For more flexibility it would be useful to make the data generation "resumable", i.e. able to continue where a previous data generation ended while still maintaining data consistency across the "old" and the "new" documents. Another interesting topic to pursue is efficient data generation "on-the-fly" to avoid the pre-generation of XML documents altogether. The workload driver currently replaces parameter markers with values from uniform distributions of numbers or from hard-coded input files. It would be useful to also

support random distributions for date and timestamp values instead of using long enumerated value lists.

The current XML data for TPoX contains a number of interestingly skewed value distributions many of which are not yet exploited by the queries we proposed. We believe that the TPoX data is fertile soil for many more interesting query and update workloads. Topics to explore include heavier analytical queries, batch jobs which are common in various financial applications, and a full-text search workload. The later would require a variation of the data generation templates to include larger text fields in order and/or customers documents, such as comments. Since schema changes over time are a reality in many XML applications, we believe that it is important to include XML schema evolution in a benchmark such that the database system can not predict and prepare for the changes before the start of the workload. This requires further investigation.

REFERENCES

[1] L. Afanasiev and M. Marx: "An analysis of the current XQuery benchmarks", *Experimental Evaluation of Data Management Systems* (EXPDB), 2006.

[2] L. Afanasiev and M. Marx: "XCheck – An Automated XQuery Benchmark Tool", 2005, http://ilps.science.uva.nl/Resources/XCheck/index.html

[3] L. Afanasiev, I. Manolescu and P. Michiels: "MemBeR: A Micro-benchmark Repository for XQuery", *XML Symposium (XSym)* 2005.

[4] T. Böhme, E. Rahm: XMach-1: "A Benchmark for XML Data Management", *Proceedings of German database conference BTW2001*, pp 264-273, March 2001.

[5] T. Böhme et al: "Multi-User Evaluation of XML Data Management Systems with XMach-1", LNCS Vol. 2590, 2003.

[6] R.Bourret: "XML Database Products", http://www.rpbourret.com/xml/XMLDatabaseProds.htm

[7] S. Bressan, G. Dobbie, Z. Lacroix, M. L. Lee, Y. G. Li, U. Nambiar and B. Wadhwa: "XOO7: Applying OO7 Benchmark to XML Query Processing Tools", *International Conference on Information and Knowledge Management (CIKM),* November 2001.

[8] Chicago Mercantile Exchange: "*The Business Case for FIXML*", 2004, http://www.cme.com/files/BusinessCaseFIXML.ppt http://www.cme.com/clearing/cm/stan/fixml6615.html

[9] M. Franceschet: "XPathMark - An XPath benchmark for XMark generated data", *International XML Database Symposium (XSYM)*, pp.129-143, 2005.

[10] The Financial Information eXchange Protocol, FIXML 4.4 Schema Specification 20040109, Revision1 2006-10-06 http://www.fixprotocol.org/specifications/fix4.4fixml

[11] FpML, http://www.fpml.org/participants/

[12] J. Gray: The Benchmark Handbook. Morgan Kaufmann, San Mateo, CA, 1993.

[13] Intel Corporation, Whitepaper: „*DB2 9 pureXML Scalability on Intel Xeon Processor MP Platforms Using IBM N Series Storage*", http://www.intel.com/cd/ids/developer/asmo-na/eng/dc/mobile/technologies/326328.htm, Dec 2006.

[14] I. Kogan, M. Nicola, B. Schiefer: "*DB2 9 XML performance characteristics*", Technical Report, developerWorks, June 2006. http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0606schiefer/

[15] S. Manegold: "An Empirical Evaluation of XQuery Processors", in *Experimental Evaluation of Data Management Systems* (EXPDB), 2006.

[16] I. Manolescu, C. Miachon and P. Michiels: "Towards micro-benchmarking XQuery", *Experimental Evaluation of Data Management Systems (EXPDB)*, 2006.

[17] U. Nambiar, Z. Lacroix, S. Bressan, M. L. Lee, and Y. G. Li: "XML Benchmarks put to the test", *3rd International Conference on Information Integration and Web-based Applications & Services* (IIWAS), September, 2001.

[18] Nicola, M., Van der Linden, B.: Native XML Support in DB2 Universal Database, *International Conference on Very Large Data Bases (VLDB),* 2005.

[19] Nicola, M., John, J.: XML Parsing: A Threat to Database Performance, *12th Intl. Conference on Information and Knowledge Management*, CIKM 2003.

[20] The Options Clearing Corporation: "Data Distribution Service and Inbound FIXML", http://www.theocc.com/products/dds_ref_materials.jsp

[21] Oracle XML DB 10g www.oracle.com/technology/tech/xml/xmldb

[22] Pat et al.: "Indexing XML Data Stored in a Relational Database", *International Conference on Very Large Data Bases (VLDB),* 2004.

[23] K. Runapongsa, J. M. Patel, H. V. Jagadish, Y. Chen, and S. Al-Khalifa: "The Michigan Benchmark: Towards XML Query Performance Diagnostics", *Proceedings of the 29th VLDB Conference*, 2003.

[24] A. Schmidt, F. Waas, M. L. Kersten, M. J. Carey, I. Manolescu and R. Busse: "XMark: A Benchmark for XML Data Management", *International Conference on Very Large Data Bases (VLDB)*, pp 974-985, August 2002.

[25] A. Schmidt, F. Waas, S. Manegold, M. L. Kersten: „A Look Back on the XML Benchmark Project", *Intelligent Search on XML*, Vol. 2818 of LNCS/LNAI, pp 263-278, 2003.

[26] *Toxgene Data Generator,* http://www.alphaworks.ibm.com/tech/toxgene

[27] Transaction Processing over XML (TPoX), XML Database Benchmark, to http://tpox.sourceforge.net/, Jan 2007.

[28] XML on Wall Street, *Financial XML Projects*, http://lighthouse-partners.com/xml

[29] XQuery Update Facility, W3C Working Draft, July 2006, http://www.w3.org/TR/xqupdate/

[30] B. Yao, M. T. Özsu, and J. Keenleyside: "XBench - A Family of Benchmarks for XML DBMSs", *EEXTT 2002 and DiWeb 2002, LNCS Vol. 2590*, pp. 162-164.

APPENDIX: QUERIES AND UPDATES

For the benchmark execution, parameter markers are used instead of the literal values in the predicates (shown in bold). The input function for a collection of documents can vary depending on the database system used. For DB2 9 it is *db2-fn:xmlcolumn*.

Q1: get_order

```
declare namespace o="http://www.fixprotocol.org/FIXML-4-4";
for $ord in db2-fn:xmlcolumn("ORDER.ODOC")/o:FIXML
where $ord/o:Order/@ID="103415"
return $ord/o:Order
```

Q2: get_security

```
declare default element namespace "http://tpox-
benchmark.com/security";
for $s in db2-fn:xmlcolumn("SECURITY.SDOC")/Security
where $s/Symbol= "SFDBX"
return $s
```

Q3: customer_profile

```
declare default element namespace "http://tpox-
benchmark.com/custacc";
for $cust in db2-fn:xmlcolumn("CUSTACC.CADOC")/Customer
where $cust/@id=2009
return
        <Customer_Profile CUSTOMERID="{$cust/@id}">
                {$cust/Name}
                {$cust/DateOfBirth}
                {$cust/Gender}
                {$cust/Nationality}
                {$cust/CountryOfResidence}
                {$cust/Languages}
                {$cust/Addresses}
                {$cust/EmailAddresses}
        </Customer_Profile>
```

Q4: search_securities

```
declare default element namespace "http://tpox-
benchmark.com/security";
for $sec in db2-fn:xmlcolumn("SECURITY.SDOC")/Security
where
   $sec/SecurityInformation/*/Sector= "Energy" and
   $sec/PE[. >= 30 and  . < 35] and
   $sec/Yield > 4.5
return
        <Security>
                {$sec/Symbol}
                {$sec/Name}
                {$sec/SecurityType}
                {$sec/SecurityInformation//Sector}
                {$sec/PE}
                {$sec/Yield}
        </Security>
```

Q5: account_summary

```
declare default element namespace "http://tpox-
benchmark.com/custacc";
for $cust in db2-fn:xmlcolumn("CUSTACC.CADOC")/Customer[
                                                @id=1011]
return
<Customer>
        {$cust/@id}
        {$cust/Name}
        <Customer_Securities>
        { for $account in $cust/Accounts/Account
          return
        <Account ACCOUNT_ID="{$account/@id}"
           BALANCE="{$account/Balance/OnlineActualBal}">
            <Securities>
                {$account/Holdings/Position/Name}
            </Securities>
        </Account>}
        </Customer_Securities>
</Customer>
```

Q6: get_security_price

```
declare namespace s="http://tpox-benchmark.com/security";
for $s in db2-fn:xmlcolumn("SECURITY.SDOC")/s:Security
where $s/s:Symbol= "SFDBX"
return     <print>The open price of the security
"{$s/s:Name/text()}" is {$s/s:Price/s:PriceToday/s:Open/text()}
dollars</print>
```

Q7: customer_max_order

```
declare default element namespace
"http://www.fixprotocol.org/FIXML-4-4";
declare namespace c="http://tpox-benchmark.com/custacc";
let $orderprice :=
for $ord in db2-fn:xmlcolumn("ORDER.ODOC")/FIXML/Order
for $cust in db2-fn:xmlcolumn("CUSTACC.CADOC")/c:Customer[
        @id=1011 and
        c:Accounts/c:Account/@id=$ord/@Acct/fn:string(.)  ]
return $ord/OrdQty/@Cash
return max($orderprice)
```

We have implemented each query in both pure XQuery and SQL/XML notation. Here is **Q1** in SQL/XML notation with parameter markers:

Q1: get_order_sqlxml

```
SELECT XMLQUERY
 ('declare namespace o="http://www.fixprotocol.org/FIXML-4-4";
  for $ord in $odoc/o:FIXML
  return $ord/o:Order' PASSING odoc AS "odoc")
FROM order
WHERE XMLEXISTS
 ('declare namespace o="http://www.fixprotocol.org/FIXML-4-4";
  $odoc/o:FIXML/o:Order[@ID=$id]'
  PASSING odoc AS "odoc", cast (? as varchar(10)) as "id")
```

Updates:

U1: close_account

```
UPDATE custacc
SET cadoc = XMLQUERY(' declare default element namespace"
http://tpox-benchmark.com/custacc";
  transform
    copy $c := $doc
    modify
(: don't delete the account if the customer has only one account :)
      if (count($c/Customer/Accounts/Account) >= 2)
      then do delete
            $c/Customer/Accounts/Account[@id="104138966"]
      else ()
  return  $c'       PASSING cadoc AS "doc")
WHERE XMLEXISTS ('declare default element namespace
"http://tpox-benchmark.com/custacc";
$cadoc/Customer/Accounts/Account[@id="104138966"]'
PASSING cadoc AS "cadoc")
```

U2: open_account

```
UPDATE custacc
SET cadoc = XMLQUERY('declare default element namespace
"http://tpox-benchmark.com/custacc";
  transform
    copy $c := $doc
    modify
(: don't add the account if it exceeds the max of seven accounts :)
      if (count($c/Customer/Accounts/Account) < 7)
      then do insert
              <Account id="104138966">
                <Category>9</Category>
                 <Currency>YEN</Currency>
                 <OpeningDate>2006-11-19</OpeningDate>
                 (…)                  (: shortened for brevity. See [27]. :)
              </Account>
           into $c/Customer/Accounts
      else ()
  return $c'
  PASSING cadoc AS "doc")
WHERE XMLEXISTS ('declare default element namespace
"http://tpox-benchmark.com/custacc";
$cadoc/Customer[@id=1011]' PASSING cadoc AS "cadoc")
```

U3: price_change

```
UPDATE security
SET sdoc = XMLQUERY('declare default element namespace
"http://tpox-benchmark.com/security";
  transform
    copy $secdoc := $doc
    modify
        let $price := $secdoc/Security/Price
        let $newlasttrade := $price/PriceToday/Open*0.95
        return
            (do replace value of $price/LastTrade with $newlasttrade,
             do replace value of $price/Ask with $newlasttrade*1.01,
             do replace value of $price/Bid with $newlasttrade*0.99)
      return  $secdoc'       PASSING sdoc AS "doc")
WHERE XMLEXISTS('declare default element namespace
"http://tpox-benchmark.com/security";
$sdoc/Security[Symbol="VIVAX"]'
PASSING sdoc AS "sdoc")
```

U4: order_status

```
UPDATE order
SET odoc = XMLQUERY('declare default element namespace
"http://www.fixprotocol.org/FIXML-4-4";
  transform
    copy $o := $doc
    modify (
      do replace value of $o/FIXML/Order/@SolFlag with "N",
      do replace value of $o/FIXML/Order/Instrmt/@Src with "C")
    return  $o'       PASSING odoc AS "doc")
WHERE XMLEXISTS ('declare default element namespace
"http://www.fixprotocol.org/FIXML-4-4";
$doc/FIXML/Order[@ID="103415"]'
PASSING odoc AS "doc")
```

U5: buy_security

```
UPDATE custacc
SET cadoc = XMLQUERY('declare default element namespace
"http://tpox-benchmark.com/custacc"; declare namespace s =
"http://tpox-benchmark.com/security";
  transform
    copy $c := $doc
    modify
let $acct := $c/Customer/Accounts/Account[@id="104138966"]
let $actualbalance := $acct/Balance/OnlineActualBal
let $clearedbalance := $acct/Balance/OnlineClearedBal
let $workingbalance := $acct/Balance/WorkingBalance
let $mvaluedate := $acct/gValueDate/mValueDate[last()]
let $currentdate := fn:current-date()
(: need to get security information :)
let $sec := db2-fn:xmlcolumn("SECURITY.SDOC")/s:Security[
                                                    s:Symbol="OIIM"]
return (
(: don't buy if it exceeds the maximum of 10 holdings per acc :)
    if (count($acct/Holdings/Position)=10 and
    count($acct/Holdings/Position[Symbol="OIIM"])=0)
then ()
else (
(: add new position if no shares of this security exist in this acc :)
if (count($acct/Holdings/Position[Symbol="OIIM"])=0) then
      do insert
         <Position>
              <Symbol>{$sec/s:Symbol/text()}</Symbol>
              <Name>{$sec/s:Name/text()}</Name>
              <Type>{$sec/s:SecurityType/text()}</Type>
              <Quantity>50</Quantity>
         </Position>
      into $acct/Holdings
(: if shares of this security existed in this acc, add the new shares:)
else
      do replace value of
         $acct/Holdings/Position[Symbol="OIIM"]/Quantity with
         $acct/Holdings/Position[Symbol="OIIM"]/Quantity + 50,
(:  now update the account balances and timestamp :)
do replace value of $acct/LastUpdate with fn:current-dateTime(),
do replace value of $actualbalance with
                        $actualbalance+(50*$sec/s:Price/s:Ask),
do replace value of $clearedbalance with
                        $clearedbalance+(50*$sec/s:Price/s:Ask),
```

```
do replace value of $workingbalance with
                    $workingbalance+(50*$sec/s:Price/s:Ask),
do replace $mvaluedate with
      <mValueDate>
          <ValueDate>{$currentdate} </ValueDate>
          <CreditMovement>156882.77 </CreditMovement>
          <ValueDatedBal>45736.85 </ValueDatedBal>
      </mValueDate>  )   )
return  $c'
     PASSING cadoc AS "doc")
WHERE XMLEXISTS ('declare default element namespace
"http://tpox-benchmark.com/custacc";
$cadoc/Customer/Accounts/Account[@id="104138966"]'
PASSING cadoc AS "cadoc")
```

U6: sell_security

```
UPDATE custacc
SET cadoc = XMLQUERY('declare default element namespace
"http://tpox-benchmark.com/custacc"; declare namespace s=
"http://tpox-benchmark.com/security";
  transform
    copy $c := $doc
    modify
let $acct := $c/Customer/Accounts/Account[@id="104138966"]
let $actualbalance := $acct/Balance/OnlineActualBal
let $clearedbalance := $acct/Balance/OnlineClearedBal
let $workingbalance := $acct/Balance/WorkingBalance
let $mvaluedate :=$acct/gValueDate/mValueDate[last()]
let $currentdate := fn:current-date()
(: need to get security information :)
let $sec := db2-fn:xmlcolumn("SECURITY.SDOC")/s:Security[
          s:Symbol=$acct/Holdings/Position[1]/Symbol/fn:string(.)]
return (
 (: don't sell if it depletes the last position in this account :)
if (count($acct/Holdings/Position) < 2 and
$acct/Holdings/Position[1]/Quantity <= 50) then ()
else (
(: delete the position if the sell redeems all shares held :)
if ($acct/Holdings/Position[1]/Quantity <= 50) then
   do delete $acct/Holdings/Position[1]
(: otherwise subtract the sold shares :)
else
   do replace value of $acct/Holdings/Position[1]/Quantity
                    with $acct/Holdings/Position[1]/Quantity - 50,
(:  now update the account balances and timestamp :)
do replace value of $acct/LastUpdate with fn:current-dateTime(),
do replace value of $actualbalance with
                    $actualbalance - (50*$sec/s:Price/s:Bid),
do replace value of $clearedbalance with
                    $clearedbalance - (50*$sec/s:Price/s:Bid),
do replace value of $workingbalance with
                    $workingbalance - (50*$sec/s:Price/s:Bid),
do replace $mvaluedate with
      <mValueDate>
          <ValueDate>{$currentdate} </ValueDate>
          <CreditMovement>156882.77 </CreditMovement>
          <ValueDatedBal>45736.85</ValueDatedBal>
      </mValueDate>  )   )

return  $c'     PASSING cadoc AS "doc")
WHERE XMLEXISTS ('declare default element namespace
"http://tpox-benchmark.com/custacc";
$cadoc/Customer/Accounts/Account[@id="104138966"]'
PASSING cadoc AS "cadoc")
```

The latest versions of the TPoX transactions, data and workload driver are available at http://tpox.sourceforge.net/ .