Transaction Processing over XML (TPoX) Benchmark:

Workload Driver Overview and Usage

Version 1.2, June, 2008

Irina Kogan	ikogan@ca.ibm.com
Matthias Nicola	mnicola@us.ibm.com
Vitor Rodrigues	vrodrig@us.ibm.com
Mike Liu	mikezliu@ca.ibm.com
Kevin Xie	kxie@ca.ibm.com

http://tpox.sourceforge.net/

1 Overview 2
2 Installation Instructions 2
2.1 Installation on 64-bit AIX 2
2.2 Installation on Linux 3
2.3 Installation on Windows 3
3 Workload Driver Usage 4
4 Transactions and Transaction Templates 7
5 Workload Description File 7
5.1 Example 1 (files containing transactions, no parameters, no weights) 8
5.2 Example 2 (transaction weights) 8
5.3 Example 3 (directory containing transactions) 9
5.4 Example 4 (parameter markers for queries) 9
5.5 Example 5 (parameter markers for document and sub-document level inserts) 11
5.6 Example 6 (parameter markers for deletes) 13
5.7 Example 6 (parameter markers for updates) 14
5.8 Example 7 (parameter markers for workloads other than TPoX) 15
6 Program Output 16

1 Overview

- The TPoX workload driver (performance testing application) is implemented in Java. It is general enough to exercise concurrent database workloads other than TPoX.
- The driver spawns 1 to n parallel threads each of which simulates a database user that connects to the database and submits a mix of XQueries (or SQL/XML queries), inserts, updates, and deletes (all of which are referred to as "transactions").
- The transactions are picked from a set of predefined templates.
  - A transaction template is a query, insert, update or delete statement in which parameter markers are used instead of literal predicate values.
  - At run time, parameter markers in the transaction templates are replaced by actual values drawn from configurable random value distributions.
  - Configuration parameters are used to indicate the workload mix; i.e., each transaction is assigned a weight in the workload mix.
- Optionally, a seed for the random number generator can be specified to ensure fully reproducible benchmark runs.
- A benchmark run is limited either by a time limit or by the number of transactions that each thread ("user") executes.
- Performance metrics are collected and a report is produced (on screen / files):
  - Min, max, avg and total elapsed time for each transaction template
  - Total throughput in transactions per minute
  - Number of completed transactions per user
- The current version of the workload driver supports DB2 9, but can be adapted to support other database systems.
- The workload driver comes with an initial set of transactions. New transactions can easily be added.

2 Installation Instructions

Installation instructions are provided for 64-bit AIX, Linux, and Windows.

2.1 Installation on 64-bit AIX

The requirements are **AIX 64 bit** and **java 1.4.2+ for 64 bit**.

These are the installation instructions:

1. Unzip `TPoX.zip` into your home directory and change to the `TPoX/WorkloadDriver` directory:

   ```
   unzip TPoX.zip
   cd TPoX/WorkloadDriver
   ```

   Make sure that the variable `$HOME` points to your home.

2. In order to be able to run the workload driver, `$JAVADIR`, `$CLASSPATH`, `$LIBPATH` and `$LD_LIBRARY` need to be set properly.

   Here is how they can be set when DB2 is used. The settings for other database systems might be similar. The parts that are likely to be different are the ones in ***bold-italic***.

DB2 only:

```
export DBS=DB2

export DBS_HOME=$HOME/sqllib

export TPOX_HOME=$HOME/TPoX

export JAVADIR=$DBS_HOME/java/jdk64/jre/bin
export
CLASSPATH=$DBS_HOME/java/db2jcc.jar:$DBS_HOME/java/db2jcc_license_cisuz.j
ar:.:$TPOX_HOME/WorkloadDriver/plugins/commons-cli-
1.0.jar:$TPOX_HOME/WorkloadDriver/classes:$TPOX_HOME/$DBS/classes
export LIBPATH=$DBS_HOME/lib:.
export LD_LIBRARY_PATH=/usr/lib:/lib:$DB_HOME/lib:$LD_LIBRARY_PATH
```

Now you are ready to run the program. The help (-h) option shows which options are available:

```
java -classpath $CLASSPATH  WorkloadDriver -h
```

The TPoX/WorkloadDriver directory contains several sample *@run** scripts with suggested invocations of the workload driver and the "*properties*" directory which contains some workload description files that can be customized or just used as samples. They point you to the transactions that come with the TPoX package.

2.2 Installation on Linux

The installation on Linux is analogous to the installation on AIX.

DB2 only:
If you use Java that comes with DB2 (currently Java 1.5 for most platforms), please note that its location on Linux is slightly different from that on AIX:

```
export JAVADIR=$DB_HOME/java/jdk64/jre/bin
```

2.3 Installation on Windows

On Windows, it is usually enough to just set the `$CLASSPATH` (assuming that the right version of java is installed).

DB2 only:

```
set DBS=DB2

set DBS_HOME=C:\Program Files\IBM\SQLLIB

set TPoX_HOME=D:\TPoX

set
CLASSPATH=".;%DBS_HOME%\java\db2jcc.jar;%DBS_HOME%\java\db2jcc_license_cu
.jar;%TPoX_HOME%\WorkloadDriver\plugins\commons-cli-
1.0.jar;%TPoX_HOME%\WorkloadDriver\classes;%TPOX_HOME%\%DBS%\classes"
```

`echo %CLASSPATH%` will show the current class path. You can run the workload driver using that class path like this:

```
java -classpath <text produced by running echo %CLASSPATH%>
WorkloadDriver –h
```

Alternatively, you can set your class path permanently by right-clicking on `My Computer`, selecting `Properties`, choosing the `Advanced` tab and clicking on `Environment Variables` at the bottom. Go to `System Variables` and add the variable `CLASSPATH` containing the text from `echo %CLASSPATH%` above, but *without* quotation marks. (If the system CLASSPATH variable already exists, just edit or extend it so that the path is valid for other applications too.) Now "`java WorkloadDriver –h`" should work fine on your computer without using the `-classpath` option for the java command.

3 Workload Driver Usage

The workload driver accepts the following command line parameters.

```
usage: WorkloadDriver [-c comment] [-d database] [-cl l] [-fto # seconds]
       [-ht host] [-pc n] [-pd # seconds] [-sc schema] [-h] [-id user id] [-r #
       seconds] [-ti # seconds] [-s seed] [-m MB] [-dbs db system] [-cc #
       transactions] [-pt port] [-tr # transactions] [-v level] [-u # users]
      [-pw password] [-tt # milliseconds] [-w file name]
 -m <MB>                memory for –pc and –cl computations
 -v <level>             2-verbose output, 1-less verbose, 0-stats only
 -pc <n>                computes the n-th percentiles of response times
 -r <# seconds>         ramp-up time
 -c <comment>           comment about the workload
 -cc <# transactions>   commit count (per user)
 -cl <l>                computes the confidence intervals with confidence
                        level 'l'
 -d <database>          database name
 -dbs <db system>       database system
 -fto <forced time out> forced time out when #transactions is specified
 -h                     help
 -ht <host>             host where DB resides
 -id <user id>          valid user id
 -pd <# seconds>        statistics print delay between intervals
 -pt <port>             port to be used to connect to the DB
 -pw <password>         password of the user with the specified id
 -s <seed>              random seed for reproducibility purposes
 -sc <schema>           default schema for all database objects refered in
                        a workload
 -ti <# seconds>        limits execution time (in seconds), overrides -tr
 -tr <# transactions>   limits #transactions executed per user
 -tt <# milliseconds>   think time (in ms) for each user
 -u <# users>           number of concurrent threads/users
 -w <file name>         name of the file containing workload description
```

When –h is used for help, no other parameters need to be provided. Otherwise, the only required parameter is –w to provide the workload description file. This file defines the workload that the driver should execute (see section 5).

If no values are provided for the remaining parameters, the following defaults are used:

- "localhost" for host (if the database resides on a host different from where the workload driver is executed, use the IP address or the remote hostname, e.g. myserver.sam.com)
- "tpox" for database name
- "DB2" for database system
- 654321 for seed (generally, it can be any number of java long type)
- For DB2, the default database schema name is the user id used to make the database connection
- 0 for verbosity level
- 1 for the number of concurrent users
- 240 seconds for execution time limit, if neither execution time nor a limit for the number of transactions is specified; timing starts *after* all users have connected to the database.
- 0 milliseconds for think time (use think time if you would like to put every "user"/thread to sleep before it moves on to the next transaction)
- 0 seconds for the print delay between intervals for outputting statistics (use it if you would like the statistics to be printed every few seconds, as described for stats.txt in section 6).
- 0 seconds for the ramp-up period (use it if you would like to allow for some time for the system to warm up; more info is in section 6)
- 1 for the commit count (i.e., committing after each transaction by default; larger commit counts may improve performance)
- Forced time out is not used unless explicitly specified. Forced time out can be useful if the benchmark run is limited by the number of transactions per user (-tr). If the execution of some statement is excessively long, the benchmark will still be terminated and results reported when the forced time out is reached.
- 50 (MB) memory for –pc and –cl computations

The following options may be used with an optional parameter value:

- If the option –pc is used without a parameter value, the default value is 90
- If the option –cl is used and no parameter value is specified, the default value is 95.

The workload driver assumes that the database is already created. The workload driver can and should be used to populate the database tables (typically with –u and –tr).

DB2 only: If the host/port options are used to run a workload against a DB2 server on a different machine, the following commands should be executed on the DB2 server machine:

```
db2set db2comm=tcpip
db2 update dbm cfg using svcename 36460  -> use the actual port # instead of 36460
db2stop
db2start
```

The available service names (svcename) can be found in the file **/etc/services** on an AIX machine and in **C:\WINNT\system32\drivers\etc\services** on a Windows machine (the location of this file may vary depending on the version of Windows installed).

If the database resides on the same machine as the application, the lines above may not be required, depending on your setup, but could simply be added at the beginning of the @run* scripts mentioned in section 2.1.

4 Transactions and Transaction Templates

Each transaction must be provided in a separate text file. A transaction can consist of one or multiple statements (query, insert, update or delete). Each statement in a transaction must be terminated by the percent sign (%) as the statement separator. When the workload driver executes the transactions, it issues a commit after each one; i.e. after the last statement of each transaction (unless a different commit count is specified in the –cc option). The isolation level currently used is "Read Committed".

DB2 only: XQueries can have the keyword **XQUERY** as the first word of the statement. If they do not, the program prepends this keyword automatically (only if the database is DB2).

The transactions provided to the workload driver are typically transaction *templates*, i.e. they contain either numbered parameter markers denoted by |1, |2, etc. or standard SQL parameter markers denoted by "?". Based on the workload description file, the workload driver replaces these parameter markers with actual literal values.

Here is an example of an XQuery transaction template:

```
declare default element namespace "http://tpox-
benchmark.com/custacc";
for $cust in db2-fn:xmlcolumn("CUSTACC.CADOC")/Customer
      where $cust/@id=|1 and $cust/Nationality="|2"
      return
            <Customer_Profile CUSTOMERID="{$cust/@id}">
                  {$cust/Name}
                  {$cust/DateOfBirth}
                  {$cust/Gender}
                  {$cust/Nationality}
                  {$cust/Addresses}
                  {$cust/EmailAddresses}
            </Customer_Profile>
%
```

This query has two parameter markers for which literal values can be supplied from different input sets (see section 5). The quotes around |2 indicate that the literal value will be used as a string, while |1 must be replaced by a number.

Just like SQL-style comments begin with '**--**', XQuery comments are denoted by '**(: :)**'. The workload driver simply removes the comments before sending the statements to the database.

The workload driver can also execute **select**, **delete, update,** and **insert** SQL statements. The workload driver is not intended to execute DDL statements such as **create** statements for database, tables, indexes, etc., though it is generally possible. It assumes that all required database objects already exist.

5 Workload Description File

A *workload description file* (aka *workload properties file*) tells the workload driver which transaction templates to execute, and how. A workload description file can either specify a directory that contains all transaction templates or list the transaction templates explicitly. The latter approach allows for more flexibility, so we focus on that in the following examples.

5.1 Example 1 (files containing transactions, no parameters, no weights)

Here is an example of workload description file that specifies three transaction templates:

```
numOfTransactions = 3

t1 = newqueries/getCustomerProfile.xqr

t2 = newqueries/listOrders.xqr

t3 = newqueries/listSecurities.xqr
```

numOfTransactions is a required keyword to specify the number of transactions (unless a directory containing all transactions is specified, as shown in section 5.3). The "**=**" sign must be used for all key/value pairs. The path to the files containing the transactions can be specified relative to the WorkloadDriver[1] directory (as the directory *newqueries* above), but can also be absolute.

In this example, the transaction templates are not expected to have parameter markers and each of the three transactions occurs by default equally often in the resulting workload mix. The specification of parameter markers and transaction weights is shown next.

5.2 Example 2 (transaction weights)

In Example 1, all transactions have the same relative weight in the workload; i.e., they will be executed equally often if the benchmark run is long enough. For more detailed workload modeling, here is how **weights** can be assigned to the individual transactions:

```
numOfTransactions = 3

t1 = newqueries/getCustomerProfile.xqr
w1 = 50

t2 = newqueries/listOrders.xqr
w2 = 20

t3 = newqueries/listSecurities.xqr
w3 = 30
```

Whenever a thread ("user") picks the "next" transaction to execute, getCustomerProfile.xqr will have a probability of 0.5, listOrders.xqr a probability of 0.2, and listSecurities.xqr a probability of 0.3. The weights are used to define the **workload mix**. This is particularly useful to control the read/write ratio in a workload.

If `w1` is specified, the program assumes that all transactions will have weights assigned to them. All weights must add up to 100 (i.e., 100% total), otherwise an error is thrown.

The transaction can be specified in any order. There is also no fixed order for the execution of transactions. Once a transaction completes, the submitting user (thread) picks another transaction according to the probabilities expressed by the weights. A transaction "completes" when the entire result (of all of its statements) has been fetched by the thread.

[1] Here and later, WorkloadDriver directory = $HOME/TPoX/WorkloadDriver.

5.3 Example 3 (directory containing transactions)

The examples above showed how to explicitly identify the transaction templates in a workload description file. A simpler alternative is to just specify a directory where all transaction templates are located:

```
directory = newqueries
```

numOfTransactions does not need to be specified in this case. All files in the directory specified will be used (numOfTransactions = # of files).

When this "directory option" is used, the weights and parameter markers can be specified in exactly the same way as discussed above. The number and order of the transaction templates are defined by the alphabetical order in which they appear in the directory.

5.4 Example 4 (parameter markers for queries)

The statements in the transaction templates can have **parameter markers**, which are substituted by actual values during the workload driver execution, based on parameter marker specifications in the workload file. For example:

```
numOfTransactions = 3

t1 = newqueries/listSecurities.xqr
p1|1 = file|input/security_types.txt

t2 = newqueries/getCustomerProfile.xqr
p2|1 = uniform|1000-3000
p2|2 = ids|1002-20001

t3 = newqueries/listOrders.xqr
```

The pipe character ("|") is the divider for different parts of a parameter marker specification. All parts are required. The "p1|1" denotes the first parameter of the first transaction, "p1|2" denotes the second parameter of the first transaction, and so on.

In the example above, the 1st transaction has one parameter marker, the 2nd has two, and the 3rd has none. The value for the parameter marker of the 1st transaction is a string value picked randomly from the file **`input/security_types.txt`**. The value for the 1st parameter marker of the 2nd transaction is an integer, uniformly distributed between 1000 and 3000. If a statement in the 2nd transaction includes a condition such as "*... where $cust/tax* = ***|1***", the |1 is replaced by a random integer between 1000 and 3000. The value of the 2nd parameter marker of the 2nd transaction is a range of ids uniformly distributed between 1002 and 20001.

The difference between a '`uniform`' and an '`ids`' specification is as follows: '`uniform`' allows you to specify an *arbitrary* but *fixed* interval of numbers; an '`ids`' specification should match the range of actual id values in the initial database population (customer IDs in this case) and this range will be dynamically extended by the workload driver as more documents are inserted.

The uniform distribution can be used for both integer and decimal numbers. By adding the "**decimal**" keyword to the end of the parameter marker specification, we create a decimal number uniform distribution instead of an integer number uniform distribution. E. g., consider the following parameter marker specification:

```
p2|1 = uniform|10.05-99.1|decimal
```

This parameter marker will have a uniform distribution between 10.05 and 99.1. All the decimal numbers in this interval have equal probability of being used as the parameter marker value.

Please note that the "**integer**" keyword is also allowed. The specification of **`uniform|1000-3000`** is equivalent to **`uniform|1000-3000|integer`**

In summary, there are currently four ways to specify values for parameter markers:

1. Uniform distribution for numbers, where a fixed range of possible values is specified (as for p2|1 above). This is indicated by the keyword "**uniform**" (with an optional "decimal" keyword at the end when the intention is to use a decimal interval instead of an integer one).
2. Uniform distribution of customers IDs, orders IDs, or account IDs that are used in predicates of queries, deletes or updates. This is indicated by the keyword "**ids**". The respective ID ranges are dynamically extended during the benchmark run as more documents are inserted. More detail is provided in Section 5.6 in the context of deletes.
3. A file containing input values (as for p1|1 above). This is indicated by the keyword "**file**". The path to the file can be relative to the WorkloadDriver directory or absolute. The file must contain a list of values, *each value on a separate line*. The workload driver removes whitespaces on both ends of each line as well as empty lines. Values from the file are drawn randomly (i.e., according to a uniform distribution). Here is an example of what can be contained in such a file:
   ```
   Stock Fund
   Bond Fund
   Mixed Fund
   ```
4. Identical values for two parameter markers. For example, **p2|2 = p2|1** means that p2|2 always uses the same *value* as p2|1, not just the same distribution or input file. See Section 5.7 for more detail.

Further distributions for parameter marker values can be added in a future version of the workload driver, such as non-uniform distributions and distributions for date or decimal values.

In a workload properties file you can first specify all transactions, then all weights and then all parameter markers (or do this in any other order), for example:

```
numOfTransactions = 2

t1 = newqueries/getOrder.xqr
t2 = newqueries/listMoreSecurities.xqr

p1|1 = uniform|1000-3000
p2|1 = uniform|100-200
p2|2 = file|input/security_types.txt

w1=80
w2=20
```

The workload driver also supports SQL and SQL/XML queries with parameter markers denoted by question marks. Example:

```
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")
%
```

If the query above is defined as `t1` and the range of order ids for the parameter marker is 103,283 - 1,103,282 (all orders), `p1|1` should be specified as `ids|103283-1103282`. Each "?" should have a parameter marker specification assigned to it. If there were, say, three "?"-denoted parameter markers in this query, we would have to specify p1|1, p1|2 and p1|3, where p1|1 would refer to the first "?", p1|2 to the second and p1|3 to the third. If a transaction is a multi-statement transaction, then numbering continues across statements.

5.5 Example 5 (parameter markers for document and sub-document level inserts)

There are three tables that are used in the TPoX benchmark: **custacc**, **order** and **security**, each having a single column of type XML. These three tables contain four collections: **custacc**, **order**, **security** and **account**. The first three collections are stored in the table with the same name. The **account** collection is stored in the **custacc** table. Each customer (stored in **custacc**) has one or more accounts. All accounts for one customer are stored in that customer's document in the **custacc** table. A new account is inserted when either a new custacc document is inserted or when a customer opens a new account (an update performing a sub-document level insert).

The workload driver supports XML inserts into the three TPoX tables as follows.

The insert template statements themselves do not contain workload driver-specific parameter markers (|1, |2, etc.), but a single SQL parameter marker (**?**). For example, any of the following could be insert transaction templates:

```
insert into custacc values (?) %

insert into custacc values(xmlvalidate( ? according to xmlschema
id tpox.custacc)) %
```

At run time, the "?" will be replaced by an actual XML document that is to be inserted. The documents will be read from one or multiple directories in the file system. File systems often don't behave nicely if millions of files are stored in a single directory. Therefore, our data generation scripts can distribute documents over many directories ("*batches*"). We suggest using a sufficient number of directories for data generation such that a single directory does not contain more than 100,000 files (some types of file systems may handle larger numbers well). The workload driver is able to feed insert statements with documents from many directories.

For example, if the following insert transaction for the *custacc* collection is the 2nd transaction in a workload description file, then this is how p2|1 can be specified to insert XML documents from one or multiple directories:

```
p2|1 = files|custacc|data/custacc/batch-|1-5|1000|500
```

What we see on the right side of the "=" sign is this:

```
files|<collection name>|<path/directory prefix>|<1st dir#>-<last
dir#>|<docs per dir>|<1st document id>|[<file name prefix>]
```

The "**files**" keyword indicates that this parameter marker specification describes the range and location of input documents. `p2|1` above specifies that the following documents are inserted from the following directories:

Directory	Documents
data/custacc/batch-**1**	custacc**500**.xml – custacc1499.xml
data/custacc/batch-**2**	custacc1500.xml – custacc2499.xml
data/custacc/batch-**3**	custacc2500.xml – custacc3499.xml
data/custacc/batch-**4**	custacc3500.xml – custacc4499.xml
data/custacc/batch-**5**	custacc4500.xml – custacc5499.xml

The directory prefix is relative to the WorkloadDriver directory. In this example, there are five directories (1-5) containing 1000 Custacc documents each (<docs per dir>=1000). The workload driver appends the directory numbers to the directory prefix. The first document is custacc500.xml (<1st doc id> = 500) and the file names are expected to be <*collection name*><*doc id*>*.xml*.

If the XML documents reside in a single directory (which is likely only for the security documents, or for very small benchmark runs), you can also specify just one directory for the source documents. For example:

```
p3|1 = files|security|data/security|1-1|20833|1
```

which is equivalent to

```
p3|1 = files|security|data/security|1|20833|1
```

In both cases, data/security is assumed to be the data directory and no number is appended to the directory prefix. In general, if `<1st dir#>-<last dir#>` is specified as **1** or **1-1**, no number is appended to the directory prefix.

Another example: if you want to insert documents only from the directory `data/custacc/batch-2`, you can do so in any of the following ways:

```
p2|1 = files|custacc|data/custacc/batch-|2-2|1000|500
p2|1 = files|custacc|data/custacc/batch-|2|1000|500
p2|1 = files|custacc|data/custacc/batch-2|1|1000|500
```

The insert statement templates for order and security are similar to the template for custacc. **To insert a specific number of documents into a table, your workload should contain a *single* insert transaction and you should use the –u and –tr options to the workload driver rather than limit the workload run by time. The number of documents inserted equals the number of users times the number of transactions per user. For example: -u 83 –tr 251 would insert all 20833 security document.**

Although TPoX was developed to store XML data using the database XML type, you can also insert XML documents into CLOB columns. This feature was introduced for comparison purposes between inserting XML into an XML column and into a CLOB column.
The only difference from the XML inserts is that the "**files**" keyword is replaced with the "**filesclob**" keyword. Please remember that when "**filesclob**" is specified, your table column must be of type CLOB and not of type XML.

The usage of the workload driver on the workloads other than TPoX is explained in section 5.8.

5.6 Example 6 (parameter markers for deletes)

Delete templates typically use regular SQL parameter markers (**?**) rather than workload driver-specific parameter markers (|1, |2, etc.). Example:

```
DELETE FROM custacc
WHERE XMLEXISTS
('declare namespace ns = "http://tpox-benchmark.com/custacc";
$cadoc/ns:Customer[@id=$id]'
PASSING cadoc AS "cadoc", cast (? as integer) as "id") %
```

At run time, the "?" is replaced by an actual value. For now, only customer and order ids are used to select documents for deletion. More interesting conditions are possible. A TPoX workload must not include deletions from the security table, since the number of securities is fixed (20,833).

If the delete transaction for the custacc table is the 2nd transaction in the workload properties file, then this is how p2|1 can be specified:

```
p2|1 = ids|custacc|1002-101001
```

What we see on the right side of the "=" sign is this:

```
ids|<table>|<start id>-<end id>
```

The `<table>` has to be `custacc` or `order`. It can also be `security` if the transaction isn't a delete. (The use of these 3 predefined table names can be overridden as shown in section 5.8).

The TPoX data generation package produces the following start (lowest) id attribute values:

Document Type	Path	Start Id
Custacc	/Customer/@id	1002
Order	/FIXML/Order/@ID	103282
Security	/Security/@id	10000

For example, if you have inserted 100,000 custacc documents for the initial database population, custacc1.xml – custacc100000.xml, the range of customer ids is `1002-101001` (`<start id>` is 1002 and `<end id>` is 101001). This way each document in the table is considered for delete, and with equal probability.

The "**ids**" keyword indicates that the parameter marker specification provides the initial range for documents that can be deleted in the table `<table name>`. If the workload also contains inserts into the table `<table>`, then the workload driver dynamically extends the `<end id>` to include the newly inserted documents. The `<end id>` keeps getting updated to *<end id> + the number of documents inserted*. This reduces the probability of trying to delete documents that have already been deleted. The "ids" parameter markers can also be used for updates as well as for queries. Although there are no deletions from the *security* table (the number of securities is fixed), an "ids" parameter marker can well be used for querying and updating securities.

Please note that although **ids|account** is supported in this version of TPoX, it should be used with care. The reason for this is that **account** ids are not dense in this release and the random number generator may return non-existent account ids from the specified range. Before the dense account ids are supported in TPoX, we recommend using **ids|custacc** and updating the **first** account of the specified customer. See **U2OpenAccount.xqr** at the end of section 5.7.

The old-style dummy updates (replacing a document by itself) are still supported for backwards-compatibility.

5.7 Example 6 (parameter markers for updates)

Sub-document level updates are now supported in TPoX, in concordance with the W3C XQuery Update Draft published at http://www.w3.org/TR/xqupdate/. Example:

```
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 :=
              $price/PriceToday/Open*$tradecoefficient
       return
       (
       do replace value of
              $price/LastTrade with $newlasttrade,
       do replace value of
              $price/Ask with $newlasttrade*$askcoefficient,
       do replace value of
              $price/Bid with $newlasttrade*$bidcoefficient)
    return  $secdoc'
    PASSING sdoc AS "doc",  CAST(? AS DOUBLE) as "tradecoefficient",
       CAST(? AS DOUBLE) as "askcoefficient", CAST(? AS DOUBLE) as
       "bidcoefficient")
WHERE XMLEXISTS
('declare default element namespace "http://tpox-benchmark.com/security";
$sdoc/Security[Symbol=$symbol]'
PASSING sdoc AS "sdoc", CAST(? AS VARCHAR(10)) AS "symbol") %
```

For a given symbol, the above statement updates the values for the elements LastTrade, Ask and Bid.

Again, standard ('?') parameter markers are used. Delete, update and select statements can have any number of them. In case of the statement above, the specification for four parameter markers is required:

```
t1 = ../DB2/xqupdates_pm/U3SecurityPrice.xqr
p1|1 = uniform | 0.95-1.05 | decimal
p1|2 = uniform | 1.0001-1.01 | decimal
p1|3 = p1|2
p1|4 = file | ../WorkloadDriver/input/security_symbols.txt
```

The specification for p1|3 says that exactly the same value is used for the 3rd parameter marker as for the second. This way the same value will be used for *$askcoefficient* and *$bidcoefficient*.

In the previous example only value updates are being done in the XML document, but you can also use XQuery Update language to remove, replace or insert a new XML node into an existing node in your document. Let's show an example:

```
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
        $acct
          into $c/Customer/Accounts
      else ()
  return $c'
  PASSING cadoc AS "doc", XMLCAST(? AS XML) AS "acct")
WHERE XMLEXISTS
('declare default element namespace "http://tpox-benchmark.com/custacc";
$cadoc/Customer[@id=$id]'
PASSING cadoc AS "cadoc", CAST (? AS DOUBLE) AS "id")
%
```

In this example, we are inserting a new **account** into a **custacc** document. (Remember that a **custacc** document can have multiple accounts.) This transaction is described in the properties file as follows:

```
t2 = ../DB2/xqupdates_pm/U2OpenAccount.xqr
p2|1= files|account|../generatedXML/CUSTOM/account/batch-|1-54|10000|1
p2|2= ids|custacc|1002-3001001
```

Please note the use of the "**files**" keyword. The "**files**" keyword can be used not only to insert new documents into a table but also to perform sub-document updates. The file referenced by this parameter specification is a valid XML file that contains an **account** and will be inserted into the **Accounts** elements of the updated document.

5.8 Example 7 (parameter markers for workloads other than TPoX)

You can freely specify the number and names of the tables used in a workload. This is useful if you want to run some workload other than TPoX.

For example, this is what `acordinsert.properties` may look like:

```
numOfTransactions = 1
numOfTables = 1
tb1  = acord
t1   = inserts/insertAcord.sql
p1|1 = files|acord|data/acord|1|78|1|Acord103Response
```

This tells us that there is only a single table used in the workload. The table is called "**acord**" and there are 78 XML documents for insertion, all located in a single directory. The documents are named Acord103Response1.xml, Acord103Response2.xml, and so on. **`Acord103Response`** is the `<file name prefix>` which allows you to insert documents whose file names prefix is different from the table name.

Now the `acord` table is the one for which the '`ids`' and '`files`' parameter marker specifications can be defined.

numOfTables, **tb1**, **tb2**, etc. are reserved keywords. If `numOfTables` is not specified in the `.properties` file, the program assumes that a TPoX workload is run and that there are three tables (custacc, order, and security).

In the current implementation of the workload driver, the table specified for insert can only contain a single XML column (unless a default value is provided for the other XML column(s)).

6 Program Output

The output can be found in the directory WorkloadDriver**/output/output<timestamp>** (assuming that the program was run from the WorkloadDriver directory). The <timestamp> is the starting time of the program run (e.g., 2005_09_19_1603). You may need to manually remove old output directories that you no longer need.

The following 6 output files can be produced. The output in stats.txt, stats_per_user.txt and output.txt is also displayed on the screen.

1) stats.txt

This file is always produced (unless the program terminates with an exception). Here is an example with verbosity level = 1:

```
STATISTICS OVER THE COMPLETE RUN:

*** SYSTEM WORKLOAD STATISTICS ***

Tr.# Name            Type  Count  %-age Total Time(s) Min Time(s) Max Time(s) Avg Time(s)
1    get_order          Q   5062   10.12  392.37        0.00        0.56        0.08
2    get_security       Q   4991   9.98   393.83        0.00        0.57        0.08
3    customer_profile   Q   5107   10.21  400.38        0.00        0.56        0.08
4    search_securities  Q   4934   9.87   387.38        0.00        0.59        0.08
5    account_summary    Q   4928   9.86   379.52        0.00        0.59        0.08
6    get_security_price Q   4913   9.83   385.20        0.00        0.59        0.08
7    customer_max_order Q   5082   10.16  278.39        0.00        0.53        0.05
8    updcustacc         U   1517   3.03   84.82         0.00        0.31        0.06
9    updsecurity        U   1479   2.96   85.84         0.00        0.28        0.06
10   delcustacc         D   983    1.97   55.73         0.00        0.29        0.06
11   delorder           D   5045   10.09  271.21        0.00        0.27        0.05
12   insNoValidcustacc  I   1018   2.04   97.33         0.03        0.59        0.10
13   insNoValidorder    I   4941   9.88   401.55        0.00        0.62        0.08

*** SYSTEM THROUGHPUT ***
The throughput is 60000 transactions per minute (1000.00 per second).
```

When `–pd <# seconds>` option is used (i.e., stats print delay is specified), then the system workload statistics are displayed as often, as specified by the “pd” argument. At each interval, you see the interval number, its timestamp and what happened during that interval. At the end of the run, the final table showing statistics for the whole run is produced. The transaction type is indicated as **Q**uery, **U**pdate, **D**elete, **I**nsert, or **M**ulti-statement.

Having the stats printed periodically allows you to see how the metrics are changing during the run. For example, this is useful for plotting throughput graphs. Another advantage is that if the program terminates abnormally, you can see the performance metrics prior to the termination.

If the ramp-up period is specified using the `–r <#seconds>` option, the output is displayed in the same way (with the notification when the ramp-up took place), but at the end two final tables are

produced rather than one. The first table shows all the statistics as if there was no ramp-up time (i.e., for the whole run). The second table shows results only for the period after the ramp-up. Comparing these two tables tells you whether the ramp up was really necessary for the system to reach a steady state.

When percentile and/or confidence intervals are computed, one new column is displayed for each.

```
Tr. #  Name                    ...  Avg Time (s)  95% Percentile  95.0% Conf. Interval
1      account_summary_sqlxml  ...  0.03          0.07            [0.03 - 0.24]
2      customer_profile_sqlxml ...  0.02          0.05            [0.01 - 0.17]
```

In this example, the 95th percentile and a confidence interval with confidence level of 95% were computed for each transaction. When using ram-up period, percentile and/or confidence interval are computed for the after ramp-up period and for the overall period, just like all the other reported values.
The supported value for the percentile computation range from 1 to 99, while for confidence intervals the set of values supported for the confidence level are: 70, 75, 80, 85, 90, 95, 97, 98, 99, 99.9, 99.99, 99.999 and 99.9999.

Please note that in the table above several columns were removed in order for each row to fit in one line. All the columns displayed in the previous example (Type, %-age, Total Time(s), Min Time(s) and Max Time(s)) are also printed when using –pc and/or –cl options.

2) stats_per_user.txt

This file is produced only if verbosity level is 1 or greater. It contains the same performance metrics as `stats.txt`, but per user (thread):

```
*** DETAILED STATISTICS FOR ALL USERS OVER COMPLETE RUN ***

*** User # 1 statistics ***
. . .

*** User # 2 statistics ***
. . .
```

A separate table showing how many transactions were complemented by each user is displayed.

The information in this file tells you whether the workload was distributed evenly among users.

If the ramp-up period is specified, **stats_per_user1.txt** and **stats_per_user2.txt** are produced. The first one is for the whole run and the second one is for the period after the ramp-up only.

3) output.txt

This file is always produced, but its contents depend on the verbosity level. Here is an example when the verbosity level is set to 2:

```
The WorkloadDriver program is running...

The following arguments are used (user id/password omitted):
 -u 100 -ht myhost.ibm.com -pt 36460 -w mixed.properties -tr 500 -v 1

100 concurrent users/threads have been initialized and connected to the
database...
```

```
Longest connection time:                3 seconds
Workload execution starting date/time:  Wed Sep 13 15:07:49 EDT 2006
Workload execution finishing date/time: Wed Sep 13 15:08:40 EDT 2006
Workload execution elapsed time:        50 seconds

The system and user statistics have been collected...

The output/output2006_09_13_1507 directory contains the files
output.txt and stats.txt (as well as stats_per_user.txt, if the
verbosity level is 1 or 2, and user1.txt, etc., if the verbosity level
is 2).

The last custacc document inserted was data/custacc/custacc1018.xml
The last order document inserted was data/order/order4941.xml
```

The collected performance metrics are summarized when the workload execution scheduled finishing time is reached (as specified in the -ti or –fto option). The last transaction for each user might still be in flight, but we do not need statistics for it, since it would finish too late to be considered. The connections are closed only after all of these running transactions have completed.

If the program terminates abnormally (i.e., if either an error or an exception is raised), the output.txt file also contains diagnostic information about what went wrong and which actual values were used for the parameter markers. For example:

```
EXCEPTION:
The statement that caused an exception:
insert into security values (?)
Failed during inserting: data/security/security125.xml
Note: the insertions of the documents with the ids in the range
(<current doc id> - <number of users>, <current doc id>) might have not
been committed!
```

For XQueries, the statement that raised the exception is printed (in the output.txt file and on the screen) with parameter markers replaced by the actual values.

When insert transactions are used in the workload, the output.txt file also specifies which document was the last one inserted into each of the three tables. This info is useful in case you want to specify different documents for the next benchmark run (to avoid duplicates).

4) user1.txt – userN.txt

These files are produced, if verbosity level is 2 (to get detailed information for each of N users). userX.txt contains all transactions executed by user X (with parameter markers replaced by actual values) and the output of these transactions. Example of a file user3.txt:

```
*** TRANSACTION # 5 ***

Transaction starting time: Mon Sep 19 16:03:45 EST 2005

Statement:

      XQUERY declare default element namespace
"http://www.fixprotocol.org/FIXML-4-4"; declare namespace
s="http://tpox-benchmark.com/security";  let $stockstemp :=  for $temp
```

```
in db2-fn:xmlcolumn("SECURITY.SDOC")/s:Security where
$temp/@id>="10783" and $temp/@id<10790 return $temp for $temp2 in
$stockstemp let $n := $temp2/s:Symbol return <Stock xmlns="http://tpox-
benchmark.com">   {$n} <Orders> {for $o in db2-
fn:xmlcolumn("ORDER.ODOC")/FIXML[Order/Instrmt/@Sym= $temp2/s:Symbol]
return sum($o/Order/OrdQty/@Qty) </Orders>      </Stock>


Result(s):

1
<Stock xmlns="http://tpox-benchmark.com"><Symbol
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:security="http://tpox-benchmark.com/security" xmlns="http://tpox-
benchmark.com/security">BCIIPRC</Symbol><Orders/></Stock>
<Stock xmlns="http://tpox-benchmark.com"><Symbol
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:security="http://tpox-benchmark.com/security" xmlns="http://tpox-
benchmark.com/security">BNSO</Symbol><Orders/></Stock>

Returned rows = 1

. . .

Transaction ending time: Mon Sep 13 16:04:41 EST 2006


*** TRANSACTION # 1 ***
```

In case of xqueries, the TPoX-specific parameter markers (|1, |2, …) are replaced by the actual values before the statements/results are printed in the user output files. In case of other statements, the '?'-denoted parameter markers are not replaced, but the actual valued used for them are displayed in a tabular format.

If verbosity level 0 or 1 is used, although the query results of the statements are not printed in the files, they are still fetched from the database server into java host variables to ensure that the performance measurements are meaningful. Verbosity level 2 is for debugging purposes (to see that the statements return correct output). Verbosity level 0 or 1 should be used for actual measurement runs because then no extra time is spent on writing data to files.

In the case of query transactions, a summary line is printed at the end of the result set displaying how many rows were returned by the query. This information could be easily processed by using the filtering commands (e.g.: grep and find).

Sometimes (when –ti or –fto option is used), the following lines are the last ones in the userX.txt file, where X is one of the users:

```
THIS TRANSACTION COMPLETED AFTER THE SCHEDULED FINISHING TIME FOR THE
WORKLOAD EXECUTION! THEREFORE, THE INFORMATION ABOUT IT IS NOT USED IN
THE STATISTICS COLLECTED.
Transaction ending time: Mon Sep 13 16:05:24 EST 2006
```

The program waits for the last transaction of each user to complete and only then closes the user files.

5) comment.txt
If the "–c <comment>" option is specified in the WorkloadDriver command line, this output file will contain the <comment> text.

6) other output

There is a WorkloadDriver/**xml** directory created with the output of each run in an xml format. This directory contains files with names such as **tpox2006_09_13_1507.xml**.

Additionally, there is a **jccOutput.txt** file that is located in the WorkloadDriver directory. This file is produced when the verbosity level is equal to 2. It contains debugging information produced by the java database connector. This file needs to be manually removed from time to time for it not to grow too large.