Computing and Informatics, Vol. 29, 2010, 719–740

DBT-5: AN OPEN-SOURCE TPC-E IMPLEMENTATION FOR GLOBAL PERFORMANCE MEASUREMENT OF COMPUTER SYSTEMS

Rilson O. Nascimento, Paulo R. M. Maciel

Centro de Informática
Universidade Federal de Pernambuco
Caixa Postal 7851 CDU – 50.732-970
Recife, PE, Brazil
e-mail: {rilson, prmm}@cin.ufpe.br

Manuscript received 2 June 2008; revised 8 December 2008
Communicated by Włodzimierz Funika

Abstract. TPC-E is the new benchmark approved by the TPC council. It is designed to exercise a brokerage firm workload, which is representative of current On-Line Transaction Processing workloads. In this paper we present DBT-5, a fair usage open-source implementation of the TPC-E benchmark. In addition to reporting about the design and implementation of the tool, experimental results from a system running a database engine are also described. The significance of this work is that it provides an environment where recent innovations in the OLTP workload field can be evaluated.

Keywords: Performance evaluation, benchmarks, TPC-E, database workload

Mathematics Subject Classification 2000: 68M20

1 INTRODUCTION

Performance is a significant facet of any transaction processing system. The response time is critical for end users as much as transaction throughput is important for system administrators. In this typical corporative scenario, companies cannot waste money on systems that are unable of sustaining satisfactorily their business with

scalability and performance. Therefore, the companies need instruments that report system performance in a way that rational comparisons can be made among different systems, that finally can lead to support the decision making process.

In order to answer to this need, hardware and software vendors formed an independent consortium called the Transaction Processing Performance Council (TPC), which defines representative transaction processing systems workloads that can be used to make such coherent comparisons [1].

The TPC Benchmark E (TPC-E) [7] is intended to model a complex online transaction processing (OLTP) workload. It is patterned after a brokerage firm workload, with multiple transaction types, balanced mixture of disk input/output and processor usage. Like its predecessor TPC-C, the workload also specifies skewed or non-uniform access to the tables, which better simulates the OLTP activity. The TPC-E aims at replacing the aging TPC-C benchmark, becoming one of the TPC mainstream benchmarks together with TPC-DS.

The DBT-5 [16] performance tool was based on the release 1.0.0 of the TPC-E specification. As of writing this paper, 1.6.0 is the current version of the specification. According to the TPC Policies [8], test results on different versions of the benchmark are considered comparable within the same major revision (the leftmost position of the version number); for instance, test results on **1**.0.0, **1**.5.0, **1**.5.1, and **1**.6.0 are all comparable. We plan to update DBT-5 to adhere to the next major release of TPC-E (2.0.0).

In addition to their commercial purpose, the TPC benchmarks have been extensively used in research. Llanos implemented the TPC-C workload to be used in parallel and distributed systems [2]; Leutenegger and Dias modeled the TPC-C benchmark [6] to study access patterns; Sivasubramaniam created a synthetic workload generator for TPC-H [4]; among other papers. It is important that the scientific community and the industry proceed in their efforts to improve the existing systems by counting on a tool that implements recent innovations in the OLTP workload simulation field. The aim of DBT-5 is to provide an environment where these innovations can be measured, modeled, and characterized, so that new advancements might take place.

While DBT-5 is a compliant TPC-E implementation, as we will demonstrate in later sections, its results or metrics should not be compared to those found in official results published by TPC. DBT-5 is a fair-use TPC-E implementation, thus, its commercial use is prohibited. However, the use for non-commercial purposes is valid and encouraged. The primary metric reported by the DBT-5 workload is the number of trade-result transactions executed per second, which is expressed as Trade-Result Transactions per Second (TRTPS).

The rest of this paper is organized as follows. In the next section we provide a synopsis of the TPC-E workload, intending to let this paper be plausibly self-contained. Section 3 discusses DBT-5's performance metric. Overall design and implementation of the tool is presented in Section 4. After that, we present the validation of our tool against the TPC-E specification, followed by an example of application. Concluding remarks appear in the last section.

2 TPC-E WORKLOAD SYNOPSIS

This section provides an outline of the TPC-E benchmark. For detailed information about the benchmark and the TPC, please see the TPC-E specification [7], and the TPC web site: `http://www.tpc.org`. The TPC-E benchmark exercises a variety of on-line transactions, simulating current OLTP applications. Due to this fact, some TPC-E features, like the database schema and the transactions, are more complex than those of its predecessor, the TPC-C benchmark. The main system component was tested is the central database engine, where the transactions run.

TPC-E models a brokerage firm that interacts with customers and with the market exchange. The customers can trade requests and inquire the brokerage house; the market can send feedback from triggered orders. The central database is the repository from these three entities, Customer, Brokerage House and Market. The database consists of 33 relations, organized in four sets: Market, Customer, Broker and Dimension. The Dimension set contains ancillary data used by the other relations, like tax rates, addresses and zip codes. The database size is defined as a function of the cardinality of the Customer relation, i.e., all other relations must increase according to the size of the Customer table. The maximum throughput attainable also depends upon the Customer's cardinality. Therefore, to get a desired throughput, the Customers table must be sized appropriately.

Transaction type	Weight	Access	Mix %	90 % Response Time Constraint
Trade-Order	Heavy	R/W	10.1	2s
Trade-Result	Heavy	R/W	10	2s
Trade-Lookup	Medium	R/O	8	3s
Trade-Update	Medium	R/W	2	3s
Trade-Status	Light	R/O	19	1s
Customer Position	Mid-Heavy	R/O	13	3s
Broker Volume	Mid-Heavy	R/O	4.9	3s
Security Detail	Medium	R/O	14	3s
Market Feed	Medium	R/W	1	2s
Market Watch	Medium	R/O	18	3s
Data Maintenance	Light	R/W	–	–
Trade-Cleanup	Medium	R/W	–	–

Table 1. TPC-E transactions

The TPC-E benchmark defines twelve transactions; ten of them have a specific mix to be observed during a run. The transactions differ with respect to the type of access (read-only or read-write) and the load it imposes on the system, as depicted in Table 1. The weight attribute provides a means for characterizing the load that each transaction imposes on the system in terms of I/O and CPU. During a workload

run, at least 90 % of each transaction type must have a response time less than or equal to the constraint specified in Table 1.

To publish an official TPC-E result the test sponsors must include the primary metrics that the specification requires: the throughput rating, expressed in tpsE; the price/tpsE ratio, which takes into account the cost of the pre-configured system evaluated; and the availability date, which states when all products used in the test will be commercially available. tpsE represents the number of trade-result transactions executed per second during a run. Unlike TPC-C, the TPC-E workload defines neither display layouts nor thinking/keying times. The absence of these characteristics leads to simplifications in the user emulation.

EGen is a TPC provided software package that accompanies TPC-E and it is designed to facilitate its implementation. The main components of EGen are EGenLoader, EGenDriver and EGenTxnHarness. EGenLoader is used to generate data for the database; it has two built-in loaders, one that generates output flat files, and another that loads a Microsoft SQL Server database; EGenLoader can be extended to support direct loading of other databases. We extended EGenLoader to support PostgreSQL, as we will show in the next section. EGenDriver facilitates the implementation of a driver; it has the following components: EGenDriverCE (Customer Emulator), EGenDriverMEE (Market Exchange Emulator) and EGenDriverDM (Data Maintenance). EGenTxnHarness is a TPC provided C++ class that defines the transaction logic, such class is not allowed to be changed by the sponsors. This logic is used together with the transactions defined on the database server.

With the goal of accurately representing today's real-world OLTP systems, the TPC-E database is populated with data distributions based on the 2000 U.S. and Canada census and actual listings on the New York Stock Exchange [9] and Nasdaq [10]. It is also important to mention that TPC-E was designed to ensure a level playing field between participants, by defining a common software package that all players should use to implement the benchmark. This is an important step to ensure that a fair testing process is in place, avoiding the misleading and biased *benchmarking games* [3].

Besides evaluating the performance of the system under test, our tool might be also used to analyze some aspects of the benchmark itself. The TPC-E benchmark assumes skewed access to the rows, i.e., within a table some rows are referenced more frequently than others. For example, in the Customer table, it is expected some customers are more exercised than others in order to model the behavior of a real OLTP application.

In order to verify that, we collected one million occurrences of the Customer ID field generated by the workload during a test run. These numbers were created by the non-uniform random number generator, as specified by the TPC-E EGen software package [7]. In Figure 1 we plot the frequency distribution of Customer ID. The non-uniformity of access with respect to the Customer ID is visible whereas some IDs are more accessed than others.

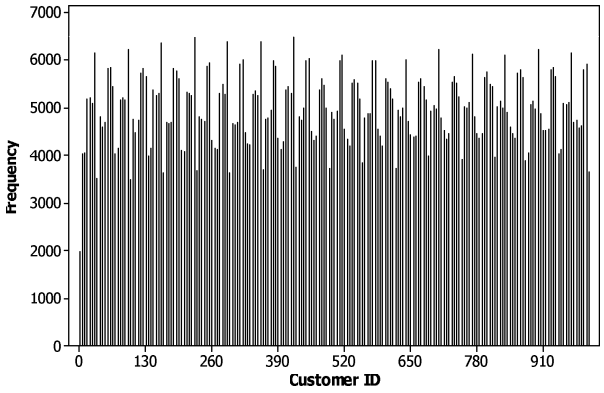


Fig. 1. Frequency distribution of Customer ID

3 PERFORMANCE METRICS

The primary metric reported by the workload is the number of trade-result transactions executed per second, which is expressed as Trade-Result Transactions per Second (TRTPS). This metric is calculated by the following expression:

$$\lambda = \frac{n_{TR}}{t}$$

where n_{TR} denotes the number of completed trade-result transactions executed during the test run, and t is the time length in seconds of the test run.

Given the required mix, complexity and types of transactions, this metric closely simulates a complete business activity. For this reason, TRTPS is considered a *business throughput*. The importance of reporting performance with a single number [5] is that it is easy to understand and easy to be used for apple-to-apple comparisons between different systems, regardless of hardware, operating system or transaction processing system used. In addition, it might be used for comparing alternative configurations on the same system.

4 DESIGN AND IMPLEMENTATION

The workload was written mostly in C++ and PL/pgSQL, which is a loadable procedural language for the PostgreSQL database system. It was used to create the functions that defined the TPC-E transactions. C++ was used to code the servers, the emulators, the transaction tester and the extended database loader. Figure 2 depicts the modules of DBT-5. The other components were coded in scripting languages. Currently, the software only supports Linux as runtime environment.

DBT-5 can be freely downloaded from the internet (a subversion client is needed): `https://svn.sourceforge.net/svnroot/osdldbt/trunk/dbt5`.

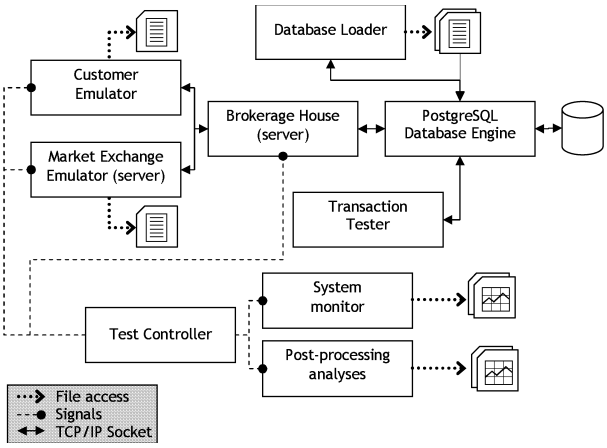


Fig. 2. DBT-5 architecture

Five main modules collaborate to provide the activities that run the workload. The modules are:

Test Controller – This module controls the other components and launches a performance run. This is the main interface available for the user to interact with the workload. It is an optional piece since the other modules can be deployed individually. However, its utilization simplifies and automates the test.

Customer Emulator – A key piece of the driver system. It is responsible for emulating customers' activity: it requests trades, provides input and dispatch transactions. It also collects and logs the transactions' response times.

Market Exchange Emulator – It is part of the driver system. It is responsible for emulating the stock market: it receives trade requests from the Brokerage House, executes the trades, sends transaction data, receives replies, and measures and logs transactions' response times.

Brokerage House – This component represents the multi-threaded server hosted in the brokerage firm. It receives the transaction requests from the drivers, communicates with the database engine and sends replies back to the drivers.

Database loader – The database loader is part of TPC-E EGen. Our loader is an extension of the base loader and can generate and load data directly into a PostgreSQL database.

The following subsections describe the modules in more detail as well as the communication process between modules.

4.1 Test Controller

The Test Controller is a bash script responsible for launching a DBT-5 test. Its main parameters are: number of customers, duration of the test, number of users and seed. The other seven options have defaults defined. This module performs the test in four stages:

1. Brokerage House server activation;
2. Market Exchange Server activation;
3. Customer Emulator activation (driver); and
4. Test results processing.

One of the limitations of the Test Controller is that it cannot operate in a multi-machine environment; it can only start the other modules when they are all on the same machine.

4.2 Customer Emulator

The Customer Emulator (CE) is a two-fold component: the core part is provided by EGen (EGenDriverCE), the rest is implemented by the sponsor. In a TPC-E compliant driver, EGenDriverCE must be used when implementing the CE. The sponsor implements the platform-specific features, like the driver-database communication process and response time logging, while EGenDriverCE is responsible for the core functions of the emulator, which involves deciding which transaction to perform next (following the transaction mix), and generating data for the transaction. Both the CE and the Market Exchange Emulator log the transaction response times in a file to be evaluated at a later time by the Post-processing analyzer.

4.3 Market Exchange Emulator

The Market Exchange Emulator (MEE) follows the same design and implementation of the Customer Emulator. It is divided into two parts, one part provided by EGen (EGenDriverMEE) and the other part implemented by the test sponsor.

It is important to notice that unlike the CE, the MEE is not only an emulator, but also a server. Since it represents the Market, it receives and performs trade requests from the Brokerage House. The MEE is an executable file, implemented as a multi-threaded server that listens on a user-defined port.

4.4 Brokerage House

The Brokerage House (BH) is composed of the following parts: Driver-System Under Test(SUT) connector, EGenTxnHarness, Frame Implementation and the database interface. EGenTxnHarness is provided by the TPC and calls the sponsor's implementation of the transaction frames. The Frame Implementation provides the

transaction functionality that is off the database; the database interface is used in the communication with the database server. We used libpqxx as the client API for PostgreSQL. The BH is implemented as a multi-threaded server that listens on a user-defined port.

4.5 Database Loader

The Database Loader produces and loads data into the test database. It generates the correct number of rows for each table based on defined rules in the TPC-E. EGenLoader is part of EGen; it was designed to generate all data necessary to a test while still allowing sponsors the freedom to customize how the data gets loaded into the database. As a result, it can be extended to populate different database management systems.

EGen provides a C++ template class, named CBaseLoader, which can be used to extend the base loader, while the generator element remains unchanged. We implemented a CPGSQLLoader class derived from the base loader to support direct loading of a PostgreSQL database. The flat file loader that accompanies EGen can still be used to populate any database via text files.

4.6 System Monitor and Post-processing Analyzer

The System Monitor is responsible for collecting performance data from the system resources, including disks, memory and processors during the test. The gathered data is stored in files in raw format, which are later processed to generate plots and reports. These plots provide useful information to analyze system performance; it helps the tester understand the system behavior during the workload, find bottlenecks and tune the system. This module is written in scripting language and uses gnuplot to generate the graphs; other Linux tools are used to collect performance data from the system, namely vmstat, iostat and OProfile.

OProfile is used for system-wide profiling. It profiles the Linux kernel as well as all user applications running on the system to generate: a symbol summary including libraries, call-graph output, and annotated mixed source and assembly. The data is useful in determining what the system is doing in userspace or kernel spaces and the code path exercised.

The Post-processing Analyzer assimilates the log files generated by the emulators during the test. It performs statistical computations from the transactions response times, generating graphs that describe the transactions behavior. In addition, a throughput graph is produced that portrays the main DBT-5 metric all over the test phase. This module is written in perl and uses gnuplot to generate the graphs.

4.7 Communication

TCP/IP sockets are used to enable the communication between the server (Brokerage House) and the emulators (Customer and Market), and between the server and the database engine. The binaries created for these components do not need to be hosted on the same machine because they can communicate over the network. Therefore, it is possible to create different scenarios for performance testing. A current limitation of the Test Controller, however, is that it can only control the components when they are present on the same machine. However, it is possible to launch and run the test without the controller when you have a more complex, distributed testing environment. At the programming level, the TPC-E specification defines interfaces that must be used to derive C++ classes; these classes govern the communication between emulators and servers. The TPC-provided interface classes are: CCESUTInterface, CMEESUTInterface and CDMSUTInterface. We derived CCESUT, CMEESUT and CDMSUT classes from these interfaces, which are used by the Customer Emulator, the Market Emulator and the Data-Maintenance transaction, respectively.

In comparison with TPCC-UVa [2], which is a software tool that implements the TPC-C benchmark, our architecture provides an important facet. TPCC-Uva uses shared-memory for inter-module communication. It enables efficient communication but it also encloses all modules on a single host. DBT-5's client/server approach enables the creation of a networked testing environment, via TCP/IP sockets, where the modules can reside on different machines. This networked testbed includes the following benefits: alleviation of the interference of the workload's instrumentation in the SUT, thus providing more reliable results; improved characterization of a typical database application environment, due to the presence of the network itself.

```
                              Response Time (s)
 Transaction                %    Average :    90th %        Total        Rollbacks      %
-----------------       -----  ---------------------  -----------  ---------------  -----
Trade Order             10.10       0.072 :     0.057         7490               74   1.00
Trade Result            10.08       0.151 :     0.113         7416                0   0.00
Trade Lookup             8.10       0.655 :     0.696         6006                0   0.00
Trade Update             2.15       0.405 :     0.417         1594                0   0.00
Trade Status            18.94       0.137 :     0.132        14045                0   0.00
Customer Position       12.84       0.492 :     0.479         9522                0   0.00
Broker Volume            4.95       0.026 :     0.016         3670                0   0.00
Security Detail         13.84       0.117 :     0.084        10263                0   0.00
Market Feed              1.00       0.269 :     0.262          741                0   0.00
Market Watch            17.94       0.116 :     0.154        13304                0   0.00
Data Maintenance          ---       0.143 :     0.226           59                0   0.00

2.06 trade-result transactions per second (TRTPS)
40.0 minute duration
0 total unknown errors
5 second(s) ramping up
```

Fig. 3. DBT-5's main report

5 VALIDATION

As per the TPC-E specification, an official TPC-E result must be reviewed by a TPC-certified, independent auditor. The TPC organization certifies an auditor by reviewing his/her qualification through a process that assesses the ability to verify compliance of a benchmark result against the specification. More details about the auditor certification process can be found in a TPC document called TPC Policies, that can be freely downloaded from the TPC site [8].

Although our workload does not allow publishing official TPC-E results, we can take advantage of the audit requirements present in the specification to validate our work. Our intent is not to certify a particular result; the objective is to gather representative evidence that allows us to say, to the best extent possible, that our workload is anchored in the specification.

Our validation can be divided into five parts: database validation, transactions validation, driver validation, and execution rules validation. Part of the validation is based on the results and part on the workload itself. A formal validation has not been required by the TPC due to the fact that our intent is to create a non-commercial implementation of the TPC-E benchmark.

5.1 Requirement verification – Database

- The database is populated using data generated by EGenLoader. EGenLoader was extended to provide a direct loading of a PostgreSQL database, which is allowed by the specification.
- The tables' cardinality meets the requirements of the specification. We created and populated a 5 000-customer database and verified that the cardinality of each table meets the initial size required by the specification.
- All primary keys, foreign keys and all check constraints are maintained by the database.
- The 9 tables in the Customer set have all of the specified attributes, as described in the TPC-E specification, clause 2.2.4 [7].
- The 11 tables in the Market set have all of the specified attributes, as described in the TPC-E specification, clause 2.2.6 [7].
- The 4 tables in the Dimension set have all of the specified attributes, as described in the TPC-E specification, clause 2.2.7 [7].
- The data types used to implement the attributes meet the requirements stated in the specification.
- Each attribute is logically discrete and independently accessible.

- The implementation of LOB was carried out by inserting a `text`[1] attribute into the NewsItem table.

5.2 Requirement Verification – Transactions

- The implementation of each transaction is compliant with its respective input parameters, output parameters, database footprint, and frame implementation requirements.
- All frames are implemented without circumventing any specified database references to static or infrequently changing data elements.
- All frames do not exchange data outside of the specified input and output parameters used to communicate with EGenTxnHarness.
- The implementation of each frame is functionally equivalent to the pseudo-code provided for that frame. We can prove this by executing each transaction individually and comparing the output of our implementation with the pseudo-code output given the same input data. The input data was generated by the following parts of EGenDriver: EGenDriverCE, which provides user input data generation; EGenDriverMEE, which provides market input data generation; and EGenDriverDM, which provides input data for the Data-Maintenance transaction. A 5 000-customer, 5-ITD (Initial Trade Day) database was created and loaded for this verification. All output values presented were found to be correct when compared to the specification pseudo-code. The result of the verification test is presented here together with the seed integer used to generate the input data. For the sake of space we present only the results for Broker-Volume, Trade-Order, Trade-Result and Trade-Status.

  Broker-Volume: Seed = 1 974 393 655

Input		Output	
Field	Value	Field	Value
broker list	Terry R. Vowell,	list_len	0
[1..10]	Arthur O. Devan, Maudie U. Shear,	broker_name[0]	
	Imelda T. Jadlowiec, Donna I. Frie,	volume[0]	0
	Alphonso T. Hilgert,	status	0
	Christopher N. Simley, Ida T. Chu,		
	Melissa V. Attard, Eugene T. Belzer		
sector name	Transportation		

[1] To fully adhere to the specification, with respect to the use of a binary data type for the NI_ITEM field of the NewsItem table, we plan to use `bytea` (PostgreSQL binary data type) instead of `text` for the next release of DBT-5.

Trade-Order: Seed = 160 710 521

Frame 1:

Input		Output	
Field	Value	Field	Value
acct id	3341	acct name	John Hance Emergency Expenses
		broker name	Arthur O. Devan
		cust f name	John
		cust id	335
		cust l name	Hance
		cust tier	2
		tax id	276FV7250IH330
		tax status	1
		status	0

Frame 2: This frame is conditionally executed when the transaction executor's first name, last name, and tax id do not match the customer first name, customer last name, and customer tax id returned in Frame 1. For the seed used, executor's data match customer's data. Thus, this frame was not executed.

Frame 3:

Input		Output	
Field	Value	Field	Value
acct id	3341	co name	Atalanta Sosnoff
cust id	335		Capital Corporate
cust tier	2	requested price	23.02
is lifo	0	symbol	ATL
issue		buy value	0
st pending id	PNDG	charge amount	4.5
st submitted id	SBMT	comm rate	0.32
tax status	1	cust assets	0
trade qty	400	market price	23.02
trade type id	TMS	s name	COMMON of Atalanta
type is margin	0		Sosnoff Capital Corporate
co name		sell value	0
requested price	23.44	status id	SBMT
symbol	ATL	tax amount	0
		type is market	1
		type is sell	1
		status	0

Frame 4:

Input		Output	
Field	Value	Field	Value
acct id	3341	trade id	1 454 535
charge amount	4.5	status	0
comm amount	29.47		
exec name	John Hance		
is cash	1		
is lifo	0		
requested price	23.02		
status id	SBMT		
symbol	ATL		
trade qty	400		
trade type id	TMS		
type is market	1		

Frame 6[2]: This Frame has no input parameters. It was executed successfully (status output field returned zero).

Input		Output	
Field	Value	Field	Value
		status	0

Trade-Result: Seed = 160 710 521
Frame 1:

Input		Output	
Field	Value	Field	Value
trade id	1454535	acct id	3341
		acct id	3341
		charge	4.5
		hs qty	0
		is lifo	0
		symbol	ATL
		trade is cash	1
		trade qty	400
		type id	TMS
		type is market	1
		type is sell	1
		type name	Market-Sell
		status	0

[2] Frame 5 and Frame 6 are mutually exclusive. Frame 5 exercises the rollback functionality; by design, 1 % of Trade-Orders rollback. Frame 6 executes a `commit transaction` command.

Frame 2:

Input		Output	
Field	Value	Field	Value
acct id	3341	broker id	10
hs qty	0	buy value	0
is lifo	0	cust id	335
symbol	ATL	sell value	0
trade id	1454535	tax status	1
trade price	23.09	trade dts	2007-12-1 12:18:23
trade qty	400	status	0
type is sell	1		

Frame 3: This frame is only executed if the customer is liquidating existing holdings, and the liquidation has resulted in a gain, and the customer's tax status is either 1 or 2. For this seed, the customer is not under these conditions, thus the frame was not executed.

Frame 4:

Input		Output	
Field	Value	Field	Value
cust id	335	comm rate	0.32
symbol	ATL	s_name	COMMON of Atalanta Sosnoff
trade qty	400		Capital Corporate
type id	TMS	status	0

Frame 5:

Input		Output	
Field	Value	Field	Value
broker id	10	status	0
comm amount	29.56		
st completed id	CMPT		
trade dts	2007-12-1 12:18:23		
trade id	1454535		
trade price	23.09		

Frame 6:

Input		Output	
Field	Value	Field	Value
acct id	3341	acct bal	3.08497e+06
due date	2007-12-3 12:18:23	status	0
s_name	COMMON of Atalanta Sosnoff Capital Corporate		
se amount	9 201.94		
trade dts	2007-12-1 12:18:23		
trade id	1454535		
trade is cash	1		
trade qty	400		
type name	Market-Sell		

Trade-Status: Seed = 352 098 087
Frame 1:

Input		Output	
Field	Value	Field	Value
acct id	9812	cust l name	Mangram
		cust f name	Derrick
		broker name	Arthur O. Devan
		charge[0]	5
		exec name[0]	Derrick Mangram
		ex name[0]	Pacific Exchange
		s name[0]	COMMON of Badger Paper Mills, Inc.
		status name[0]	Completed
		symbol[0]	BPMI
		trade dts[0]	2005/1/7 15:47:57
		trade id[0]	282086
		trade qty[0]	200
		type name[0]	Limit-Buy
		status	0

5.3 Requirement Verification – Driver

- Our implementation uses EGenTxnHarness (code provided by TPC).
- The Customer Emulator was implemented using EGenDriverCE (code provided by TPC).
- The Market-Exchange Emulator was implemented using EGenDriverMEE (code provided by TPC).
- No routing was used within the frame implementation.
- The Data-Maintenance transaction was implemented using EGenDriverDM (code provided by TPC).

5.4 Requirement Verification – Execution Rules

It is required that the mix of transactions over a test run meets the requirements stated in the specification. We can demonstrate that via a real test by showing the percentages presented at the end of the test.

We used an Intel Xeon CPU 3.00GHz, 3 GB of RAM, running Gentoo 2006.1 on kernel 2.6.19-r5, and PostgreSQL 8.2.1. After running the workload for forty minutes, on a database loaded with 1000 customers, scale factor 500, the Test Controller gathered all outputs in a numbered directory which is automatically assigned for each performance run. The main report is presented in Figure 3. It shows the average response time and the 90th percentile for each transaction, together with the total number of transactions executed and the number of rolled back transactions. 1% of the Trade Orders transactions rollback by design. The transaction mix is also showed in the % column. By reviewing Table 1, and comparing the equivalent columns in Figure 3, we can conclude our test is within the required range for each transaction with respect to % (Transaction Mix) and 90th percentile.

6 EXPERIMENTAL RESULTS

In this section, we present the results of two experiments performed with DBT-5. The first study empirically evaluates the performance of some Linux File Systems regarding all Linux 2.6 I/O schedulers. The second one traces the workload's I/O activity and replays it in a disk simulator.

6.1 File Systems Performance

We used a Pentium D 3.40GHz, running Gentoo 2007 i686, kernel 2.6.23 on a single-disk database setup; the test implemented is CPU intensive rather than I/O intensive. A brief introduction to the file systems and I/O schedulers is provided before presenting the performance results. Similar studies were conducted using data warehousing workloads [12] and mail, web and file server workloads [11].

Linux File Systems. The first File System (FS) implemented on Linux was the Minix FS. It had a 64 MB file system and filenames were limited to 14 characters. Ext FS replaced Minix in 1992. It had 2 GB but suffered from low performance problems and poor implementation of inode. As a result, Ext2 was created and became the standard FS for Linux for many years due to its improved overall performance and size limit of 4 GB. However, Ext2 did not support online file system growth and journaling. As a solution to these problems Ext3 was created. The journaling support provided fast reboots and a greater level of reliability. Its compatibility with Ext2 led to increasing adoption among Ext2 users. ReiserFS is also a journaled FS, it was designed to be a general-purpose FS capable of handling small files efficiently.

I/O Schedulers. I/O or disk scheduling is the technique of sorting I/O operations before submitting them to the I/O sub-system. It is the interlocutor between the block layer and the low-level device drivers. From an I/O scheduler perspective, the block layer is the producer of I/O requests and the device drivers are labeled as the actual consumers [11]. The I/O scheduler is not a mandatory component of the operating system. Instead, performance is the I/O scheduler's sole purpose [13]. The current Linux 2.6 kernel provides four I/O schedulers, being possible to determine which one to use during boot time by using the `elevator` kernel flag. The Deadline I/O scheduler has this name because it assigns to each read request a specific deadline. Its logic ensures that the read requests will be serviced within the allotted time. While the write requests do not have associated deadlines, the scheduler's aim is also to ensure that the queued write requests do not starve. The Anticipatory (as) I/O scheduler primary criterion is to reduce seek operations. It introduces a controlled delay component into the dispatching equation [11]. The basic idea is, by the principle of locality, to delay a read request allowing that the producer thread lines up more read requests. Thus, this micro-delay, in the order of a few milliseconds, would avoid additional seek operations by waiting for contiguous requests. The noop scheduler works by placing all requests into a simple, unordered FIFO queue and implements only request merging. It assumes performance of the I/O has been or will be optimized at the block device or with an intelligent HBA or externally attached controller [15]. The Complete Fair Queueing (CFQ) I/O scheduler's main objective is to ensure fair allocation of I/O bandwidth among the I/O requests since kernel 2.6.18 is the default Linux I/O scheduler.

Figure 4 shows the performance comparison between the three file systems and the four I/O schedulers. Three test runs were performed for each pair of the Cartesian product of the 3-element set of File Systems ($\{\text{Ext2}, \text{Ext3}, \text{ReiserFS}\}$) and the 4-element set of I/O Schedulers ($\{\text{Anticipatory}, \text{noop}, \text{Deadline}, \text{CFQ}\}$). The lines in the top of the chart represent the average throughput of the three runs per I/O scheduler. The dashed line represents EXT3, the continuous line EXT2, and the pointed line ReiserFS. The highest performance was achieved by the tuple EXT3, deadline. ReiserFS performed the worst among file systems tested. ReiserFS was designed to have good performance when handling small files, it cannot efficiently treat large files, which is the case in these tests. The test database has 2.5GB in size, 83 % of the data is contained in 10 files of more than 200MB in size on average. EXT3 surpassed EXT2 by 8 % and ReiserFS by 18 % among the four I/O schedulers. A test performed by Oracle, using an OLTP workload, presented similar results between EXT2 and EXT3 [14]. Among I/O schedulers the results were close; Deadline is only 2 % above CFQ, 3 % above AS and 4 % above no-op.

Figure 5 shows the response time in seconds per transaction and I/O scheduler for EXT2 and EXT3 File Systems; for a better visualization we divided the results between light and heavy transactions. EXT3 performed better than EXT2 also in terms of RT. The average RT among the three runs, all transactions considered,

with respect to the I/O schedulers, as shown in Table 2, has drawn a different result of the throughput, possibly because the throughput result take into account just Trade-Result. The result for EXT3 agrees with that presented in [11] where a test with a single Disk – single CPU was performed against different workloads.

I/O Scheduler	RT(all)	RT(EXT3)
CFQ	0.573	0.522
Deadline	0.580	0.537
no-op	0.612	0.563
AS	0.618	0.566

Table 2. Average response time(s) – I/O schedulers

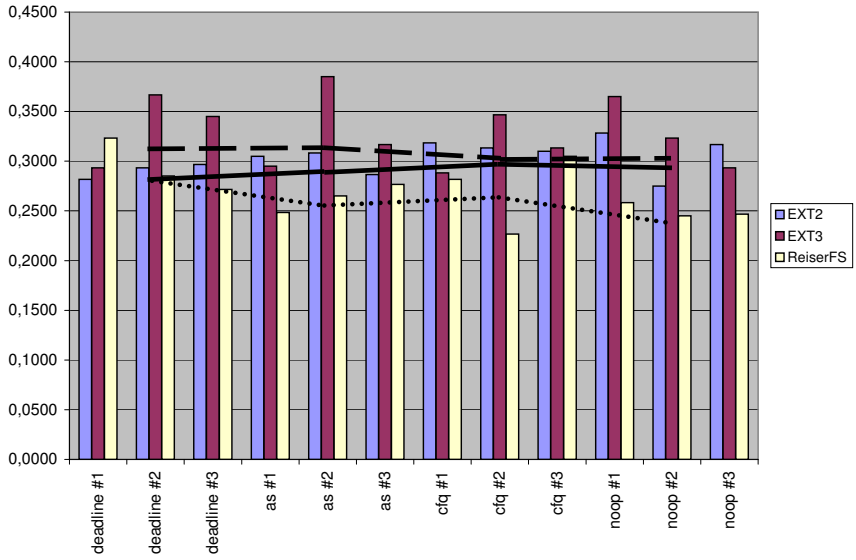
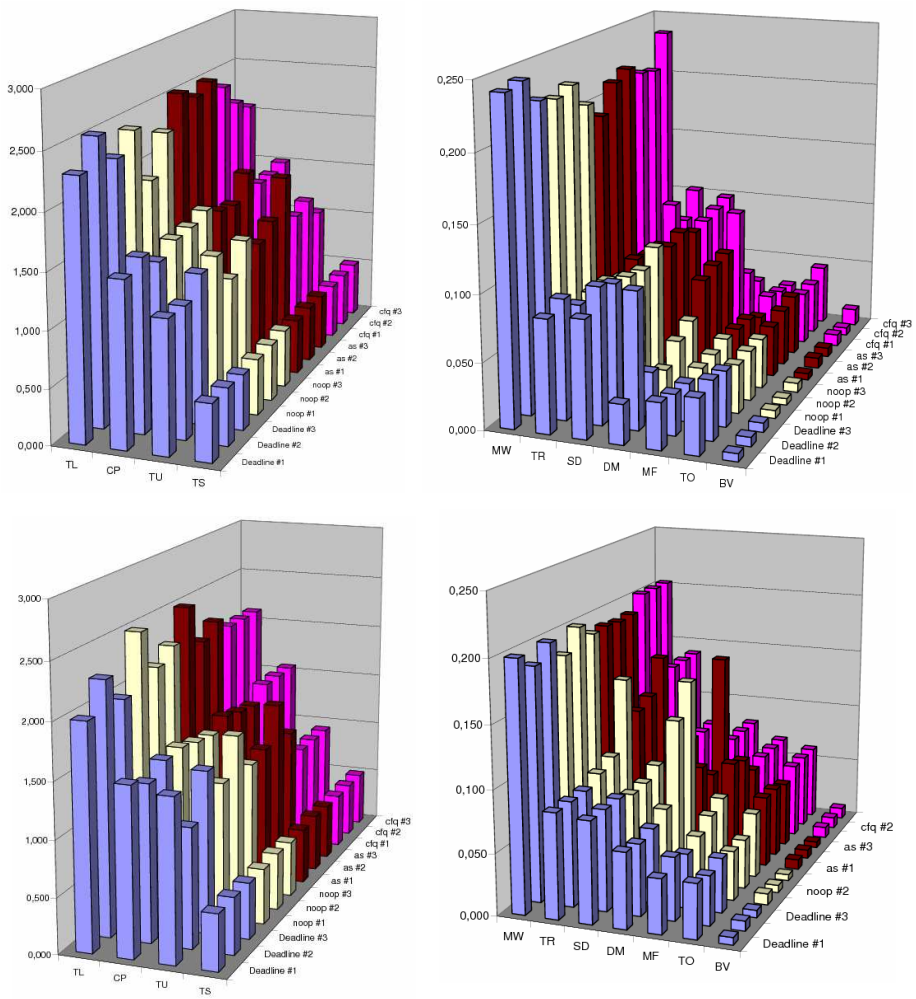


Fig. 4. Throughput. Trade-result transactions per second

6.2 Tracing

In this experiment we used three main software components: a database workload, a tracer and a disk simulator. The blktrace [18] was chosen to be the low-level I/O tracer due to its integration with the Linux kernel 2.6 since version 2.6.17. Before blktrace we tested lltrace and the lttng [19] package; the former only works in the 2.4 kernel and it is not maintained, while the latter did not work in our environment even though we tried to troubleshoot it together with the package creator. The



a) b)

c) d)

Fig. 5. Response time – EXT2: a) light transactions, b) heavy transactions; EXT3: c) light transactions, d) heavy transactions

blktrace tool runs at the block-level, it captures an ample spectre of I/O operations and commands of all processes running on the system for a determined device. We had to change blktrace's output format in order to match DiskSim's input format. DiskSim [17] is a powerful highly-configurable disk system simulator that can be used in place of a real I/O subsystem, given its accuracy and effciency.

The three components were used in a producer-consumer paradigm: the workload feeds the tracer, that in its turn feeds the disk simulator. The test consisted of running our workload in a real system while capturing I/O activity with the help of a backgrounded blktrace process. During the execution of the workload, blktrace captured more than 40 000 read/write operations. In fact there is a fourth component which is the database system, a PostgreSQL database; the target of the workload was the database process, thus the tracer was configured to capture its I/O activity. Once the tracing was finished we configured DiskSim to accept a trace file as input and place the simulator to digest the trace and to generate the output metrics. The scatter plot in Figure 6 presents the I/O requisitions response time as observed by DiskSim during its run.

This experiment demonstrates how an interested researcher can use our workload to study the I/O behavior of current online transactional processing systems. In this work we traced and replayed only the original activity; however, this work can be continued in order to synthesize the workload's I/O signature and eventually to compare the response time plot of both original and synthesized traces.

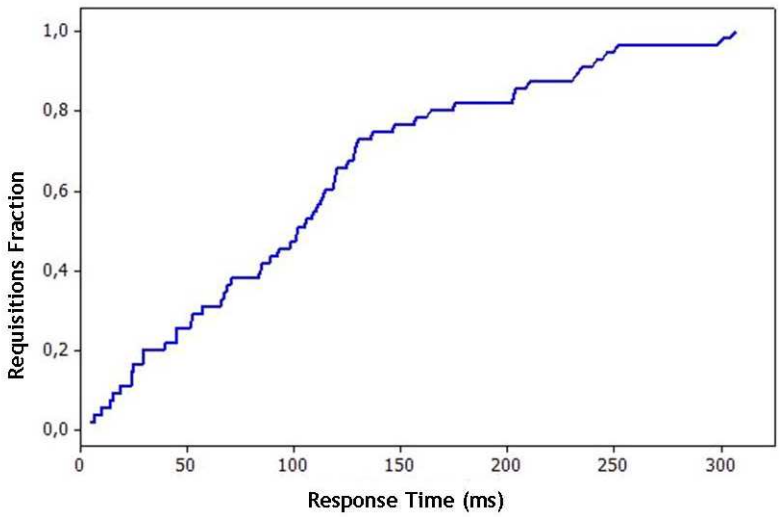


Fig. 6. I/O requisitions response time

7 CONCLUSIONS

In this paper we described DBT-5, a fair usage open-source implementation of the TPC-E benchmark. The workload exercises a modern OLTP system, which simulates an environment for performance evaluation of computer systems. An outline of the TPC-E specification was also presented, together with a description of the DBT-5 architecture and implementation and an example of application. To the best of our knowledge this is the first FOSS (free open-source software) implementation of the TPC-E specification.

The significance of this work is that it provides an environment where recent innovations in the OLTP workload field can be studied. The open source database community can use it to engineer and benchmark their databases. Since the workload characterizes modern and representative database work, it can be useful to improve engines performance. Currently the workload supports only PostgreSQL but it was designed to allow a relatively easy port to other databases.

As future work, we want to simulate the I/O of the TPC-E at the disk drive level by tracing and characterizing the arrival and access patterns during execution of our workload. We plan to update DBT-5 for every major release of the TPC-E specification and port DBT-5 to other Database Management Systems. Finally, we will update the Test Controller module so it can work in a distributed test environment.

REFERENCES

[1] Bernstein, A. P.—Newcomer, E.: Principles of Transaction Processing for the Systems Professional. Morgan Kaufmann, 1997.

[2] Llanos, D. R.—Palop, B.: TPCC-UVa: An Open-Source TPC-C Implementation for Parallel and Distributed Systems. IEEE 6th International Workshop on Performance Modeling, Evaluation, and Optimization of Parallel and Distributed Systems, 2006.

[3] Raj, J.: The Art of Computer Systems Performance Analysis Techniques For Experimental Design Measurements Simulation And Modeling. Wiley Computer Publishing, John Wiley & Sons, Inc, 1991.

[4] Zhang, J.—Sivasubramaniam, A.—Franke, H.—Gautam, N.—Zhang, Y.—Nagar, S.: Synthesizing Representative I/OWorkloads for TPC-H. Proceedings of the 10th International Symposium on High Performance Computer Architecture, 2004.

[5] John, K.—Eeckhout, L.: Performance Evaluation and Benchmarking. CRC Taylor & Francis Group, 2006.

[6] Leutenegger, S.Ṫ.—Dias, D.: A Modeling Study of the TPC-C Benchmark. ACM SIGMOD, 2003.

[7] Transaction Processing Performance Council: TPC BenchmarkTME, 1.3.0, 2007.

[8] Transaction Processing Performance Council. Available on: `http://www.tpc.org`.

[9] New York Stock Exchange web site. Available on: http://www.nyse.com/.

[10] The NASDAQ Stock Market web site. Available on: http://www.nasdaq.com/.

[11] Pratt, S. L.—Heger, D. A.: Workload Dependent Performance Evaluation of the Linux 2.6 I/O Schedulers. Proceedings of the Linux Symposium, 2004.

[12] Wong, P. W. Y.—Hendrickson, R.—Rizvi, H.—Pratt, S.: Performance Evaluation of Linux File Systems for Data Warehousing Workloads. Proceedings of the First International Conference on Scalable Information Systems, 2006.

[13] Robert, L.: The Linux Journal, Kernel Korner – I/O Schedulers. Available on: http://www.linuxjournal.com/article/6931, 2008.

[14] Oracle White Paper: Linux Filesystem Performance Comparison for OLTP with Ext2, Ext3, Raw, and OCFS on Direct-Attached Disks using Oracle 9i Release 2, 2004.

[15] Wikipedia, Noop scheduler. Available on: http://en.wikipedia.com/noop_scheduler.

[16] Do Nascimento, R. O.—Maciel, P. R. M.—Wong, M.: DBT-5: A Fair Usage Open-Source TPC-E Implementation for Performance Evaluation of Computer Systems. Proceedings of the 27th SBC-WPerformance, 2007.

[17] Ganger, G. R.—Worthington, B. L.—Patt, Y. N.: The Disksim Simulation Environment. Technical report CSE-TR-358-98, Dept. of Electrical Engineering and Computer Science, Univ. of Michigan, 1998.

[18] Jens Axboe. Block IO Tracing. Available on: http://www.kernel.org/git/?p=linux/kernel/git/axboe/blktrace.git.

[19] Desnoyers, M.: The LTTng Usertrace Package. Available on: http://ltt.polymtl.ca/svn/ltt-usertrace/README, 2006.

Rilson Nascimento works as Performance Engineer at Ingres. He is also a Masters candidate in the Federal Unversity of Pernambuco. He graduated in computer science in 1998 from the Catholic University of Pernambuco. His research interests include performance evaluation of database systems and database benchmarks.

Paulo Maciel graduated in electronic engineering in 1987 and received his M. Sc. and Ph. D. in electric engineering and computer science, respectively from Univesidade Federal de Pernambuco. He was faculty member of the Electric Engineering Department of Universidade de Pernambuco from 1989 to 2003. Since 2001 he has been a member of the Informatics Center of Universidade Federal de Pernambuco, where he is currently Associate Professor. His research interests include Petri nets, performance and reliability evaluation, and embedded system design.