DBT-5: A Fair Usage Open-Source TPC-E Implementation for Performance Evaluation of Computer Systems

Rilson O. do Nascimento[1], **Mark Wong**[2], **Paulo R. M. Maciel**[1]

[1]Centro de Informática – Universidade Federal de Pernambuco (UFPE)
Caixa Postal 7851 CDU – 50.732-970 – Recife – PE – Brazil

[2]8026 SW 46th Avenue – 97219 – Portland – OR – USA

{rilson, prmm}@cin.ufpe.br, markwkm@gmail.com

Abstract. *The TPC-E is the new benchmark recently approved by the TPC council. It is designed to exercise an On-Line Transaction Processing workload of a brokerage firm and be representative of current database server work. In this paper we present DBT-5, a fair usage open-source implementation of the TPC-E benchmark. In addition to reporting on the design and implementation details of the tool, experimental results on a system running the PostgreSQL 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.*

Resumo. *O TPC-E é o novo benchmark aprovado recentemente pelo conselho TPC. Foi projetado para exercitar uma carga de trabalho de Processamento de Transação On-Line de uma corretora de ações, sendo centrada no trabalho executado pelo servidor de banco de dados. Neste artigo nós apresentamos a ferramenta DBT-5, uma implementação de código aberto, para uso não comercial, do benchmark TPC-E. Além de descrevermos a arquitetura e a implementação do sistema, mostramos resultados experimentais em um sistema rodando o banco de dados PostgreSQL. A importância deste trabalho se dá pela provisão de um ambiente onde as inovações no campo das cargas de trabalho transacionais podem ser avaliadas.*

1. Introduction

Performance is a significant facet of transaction processing systems. 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 an instrument that reports the performance of the systems, in a way that rational comparisons can be made among different systems.

To answer to this need, 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) [6] 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 specifies also skewed or non-uniform access to the tables, which better simulates the OLTP activity. The TPC-E aims to replace the aging TPC-C benchmark, becoming one of the TPC mainstream benchmarks together with TPC-DS. The DBT-5 performance tool was based on a public draft release of the TPC-E specification.

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 [5] to study access patterns; Sivasubramaniam created a synthetic workload generator for TPC-H [3]; among other papers. It is important that the scientific community and the industry proceed on their efforts to excel the existing storage 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.

It is worth notice that the results reported by the DBT-5 workload do not constitute a TPC-E result, and are incomparable with any TPC-E benchmark. 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). However, TRTPS do not and should not be compared to tpsE measurements in any way, since DBT-5 workload does not constitute a compliant TPC-E benchmark. 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 plausibly self-contained. The Section 3 discusses about DBT-5's performance metric. The overall design and implementation of the tool is presented in Section 4, 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 on the benchmark and the TPC, please see the TPC-E specification [6], 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 of its predecessor, the TPC-C benchmark. The main system component being 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 contain generic 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 on the Customer's cardinality. Therefore, to get a desired throughput, the Customers table must be sized appropriately.

Table 1: TPC-E transactions

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

The TPC-E benchmark defines twelve transactions; ten of them have a specific mix to be respected during the run. The transactions differ on the type of access (read-only or read-write) and on the load it imposes on the system, as depicted in Table 1.

To publish an official TPC-E result the sponsors must include the primary metrics that the specifications 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 run will be commercially available. The tpsE represents the number of trade-result transactions executed per second during a run. Unlike TPC-C, the TPC-E workload neither defines display layouts, nor thinking/keying times. The absence of these characteristics leads to simplifications in 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 the EGen are EGenLoader, EGenDriver and EGenTxnHarness. The EGenLoader is used to generate data for the database, it comes with two loaders, one that generates output flat files, and other that loads a Microsoft SQL Server database; it can be extended to support direct loading of other databases. We extended the EGenLoader to support PostgreSQL, as we will show in the next section. The EGenDriver facilitates the implementation of a Driver, and it has the following components: EGenDriverCE (Customer Emulator), EGenDriverMEE (Market Exchange Emulator) and EGenDriverDM (Data Maintenance). The EGenTxnHarness is a TPC provided C++ class defining the transaction logic, which is not allowed to be changed by the sponsors. This logic is used together with the transactions defined on the database server.

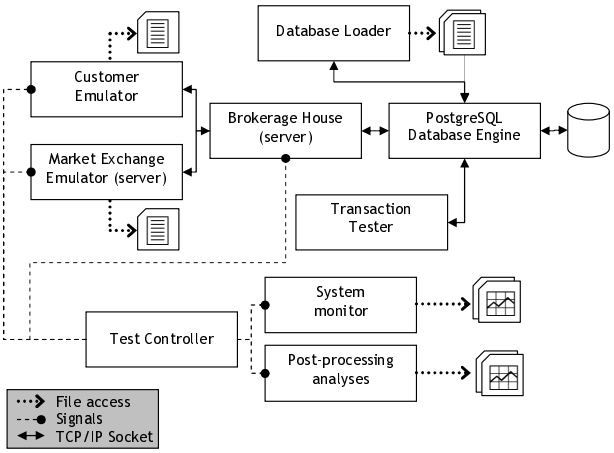


Figure 1: DBT-5 Architecture

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 and the wide range of complexity and types among the transactions, this metric more closely simulates a complete business activity. For this reason, TRTPS is considered a *business throughput*. The importance of reporting performance with a single number [4] 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 being used. In addition, it might be used for comparing configuration alternatives 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 1 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.

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

- **Test Controller** This module controls the other components to launch 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 launched individually. However, its utilization simplifies and automates the test.

- **Customer Emulator** A key piece of the driver system. It is responsible for emulating customers: requesting trades, providing input, sending transaction data, receiving replies, measuring and logging response times.
- **Market Exchange Emulator** It is part of the driver system. It is responsible for emulating the stock market: receiving trade requests from the Brokerage House, performing the trades, sending transaction data, receiving replies, measuring and logging response times.
- **Brokerage House** This component represents the multi-threaded server 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 oriented to generate and direct loading data into the PostgreSQL database.

The following subsections describe the modules in more detail and the communication between them.

4.1. Test Controller

The Test Controller is a bash script responsible for launching a DBT-5 test. Its main options 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) Starting the Brokerage House server; 2) Starting the Market Exchange Server; 3) Starting the Customer Emulator (driver); and 4) Processing the test results. 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, the EGenDriverCE must be used when implementing the CE. The sponsor implements the platform-specific features, i.e., sending the transaction requests and input data to the database, receiving transaction replies and output data from the database, measuring and logging. Both the CE and MEE log the transaction response times in a file to be evaluated at a later time by the Post-processing analyzer. The 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.

4.3. Market Exchange Emulator

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

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 by the following parts: Driver-SUT (System Under Test) connector, EGenTxnHarness, Frame Implementation and the database interface. The EGenTxnHarness is provided by the TPC and calls the sponsor's implementation of the transaction frames. The Frame Implementation provides the transaction functionality; the database interface is used in the communication with the database server. In the project, 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. The 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 (TPC, 2006). 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 the flat 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 processed to generate plots and reports. The plots are very useful to analyze the system performance. For instance, it helps the tester to understand the system behavior during the workload, to find bottlenecks and to 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: 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 works on the log files generated by the emulators during the test. It performs statistical computation based on the transactions response time, generating graphs that describe the transactions behavior, like the Cumulative Distribution Function plots. In addition, a throughput graph is produced that portrays the main DBT-5 metric, TRTPS, all over the test phase. This module is written in perl and uses gnuplot to generate the graphs. The next section presents an example of application of DBT-5, where the main plots and reports are shown.

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

```
                                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
```

Figure 2: DBT-5's main report

database engine. The binaries created for each of these components do not need to be in the same machine, since they can communicate via the network. Therefore, it is possible to create different scenarios for performance testing. A current limitation of the Test Controller is that it cannot control the components when they are not on the same machine. However, it is possible to launch and run the test without the controller when you have a more complex testbed. At the programming level, the TPC-E specification defines interfaces that must be used to derive classes that govern the communication between each emulator and the server. 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 feature. TPCC-Uva uses shared-memory for inter-module communication. It enables efficient communication but encloses all modules inside the same host. DBT-5's client/server approach using TCP/IP sockets enables the creation of a networked testbed where the modules can reside on different machines. This networked environment includes the following benefits: alleviation of the interference of the workload's instrumentation on the system under test, providing more reliable results; improved characterization of a typical database application environment, due to the presence of the network by itself.

5. Experimental Results

We present DBT-5 on a system with 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 gathers all outputs in a numbered directory which is automatically assigned for each performance run. The main report is presented in Figure 2. 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 Trade Orders rollback by design.

The plots can be divided in two groups: System Resources and Transactions. The Figure 3 presents some of the plots for both groups. The System Resources group, which

reports CPU, input/output, memory and paging activity, provides a server-wide view of performance. These plots are useful for finding bottlenecks and for performance tuning. The plots in the Transactions group show statistical information about each transaction. For each graph there is a correspondent data file in flat format that can be used to easily generate other types of graphs.

6. Conclusion

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. An example of application was given and the main graphs were shown.

The significance of this work is that it provides an environment where recent innovations in the OLTP workload field can be studied. For example, using DBT-5 one can model TPC-E with the intent of studying buffer miss rate. Traces can be generated from DBT-5 execution and further analyzed to characterize some aspect of the workload. Presently, we are working towards creating an I/O synthesizer for TPC-E based on traces generated from DBT-5. To our knowledge this is the first FOSS (free open-source software) implementation of the TPC-E specification.

Acknowledgment

The authors would like to thank Google for funding the development of this software tool through the Google Summer of Code 2006 program.

References

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

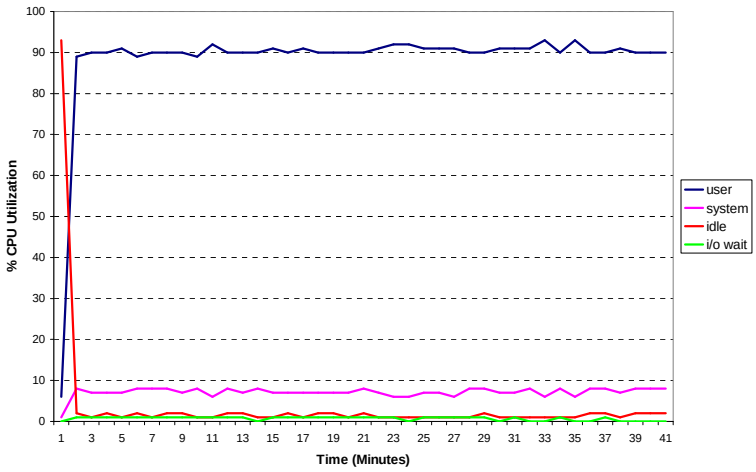
[2] D. R. Llanos and B. Palop. *TPCC-UVa: An Open-Source TPC-C Implementation for Parallel and Distributed Systems*. IEEE 6th International Workshop on Performance Modeling, Evaluation, and Optimization of Parallel and Distributed Systems, 2006.

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

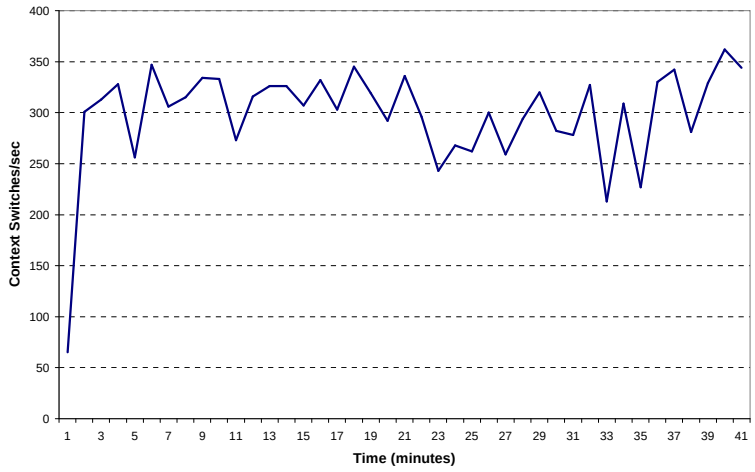
[4] K. John, L., and L. Eeckhout. *Performance Evaluation and Benchmarking*. CRC Taylor & Francis Group, 2006.

[5] S. T. Leutenegger and D. Dias. *A Modeling Study of the TPC-C Benchmark*. ACM SIGMOD, 2003.

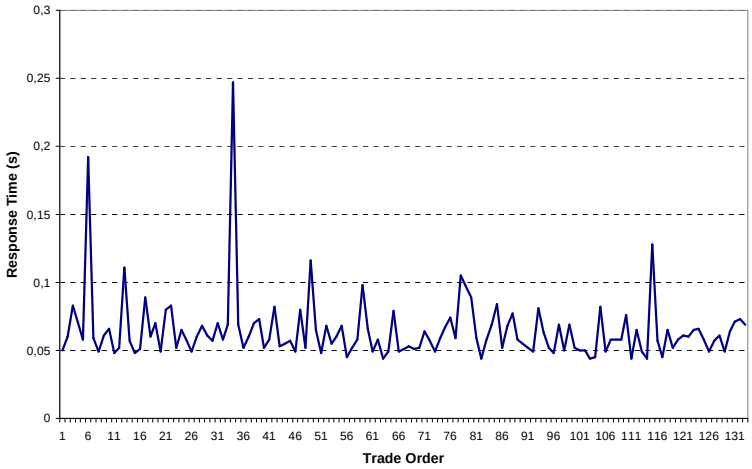
[6] Transaction Processing Performance Council, TPC Benchmark^{TM}E, Draft Revision 0.30 for Public Release, August, 19, 2006.



(a) %CPU Utilization



(b) Context Switches per second



(c) Trade-Order Response Times

Figure 3: :Response Time and System Resources Graphs