TPCC-UVa: An Open-Source TPC-C Implementation for Global Performance Measurement of Computer Systems

Diego R. Llanos
Departamento de Informática
Universidad de Valladolid, Spain.
diego@infor.uva.es.

Abstract

This paper presents TPCC-UVa, an open-source implementation of the TPC-C benchmark version 5 intended to be used to measure performance of computer systems. TPCC-UVa is written entirely in C language and it uses the PostgreSQL database engine. This implementation includes all the functionalities described by the TPC-C standard specification for the measurement of both uni- and multiprocessor systems performance. The major characteristics of the TPC-C specification are discussed, together with a description of the TPCC-UVa implementation, architecture, and performance metrics obtained. As working examples, TPCC-UVa is used in this paper to measure performance of different file systems under Linux, and to compare the relative performance of multi-core CPU technologies and their single-core counterparts.

Keywords: On-line transaction processing, TPC, performance measurement.

1 Introduction

Workload characterization in order to measure system performance is a major topic in the field of Computer Architecture. Many different benchmarks have been proposed to simulate real working conditions of both existing and proposed systems. Those benchmarks can be classified in terms of their corresponding application domains and their execution characteristics.

The most popular benchmarks are related with numerical processing, such as the SPEC CPU2000 benchmark suite [4], the NAS Parallel Benchmark [7] and the OLDEN benchmarks [10], among others. These benchmarks include many common characteristics of real scientific workloads, and some of them can be executed in both sequential and parallel computing environments. These benchmarks are designed to challenge the CPU and memory subsystem capabilities of the systems under test. However, they do not take into account other aspects of the system architecture, such as process management or I/O subsystem.

Database benchmarks, on the other hand, allow to study not only CPU and memory hierarchy performance, but the global performance of a system. These benchmarks use a synthetic workload against a database engine, measuring the performance of the system in terms of the number of transactions completed in a given period of time. One of the main advantages of this class of benchmarks is that results are very relevant to financial, commercial and corporative fields, where this type of applications is dominant.

The TPC-C benchmark, designed by the Transaction Processing Performance Council [1], simulates the execution of a set of both interactive and deferred transactions. This workload is representative of an OLTP (On-line Transaction Processing) environment, with features such as transaction queries and rollback. These capabilities makes the TPC-C benchmark specification a de-facto standard for measuring server performance. Most vendors publish performance values for their systems, allowing the consumer to accurately compare different architectures.

The Transaction Processing Performance Council only distributes a requirements specification for the TPC-C benchmark. Following this specification, vendors may implement and run a TPC-C benchmark, needing the approval of the TPC consortium to publish its performance results [3]. Unfortunately, there is not an official TPC-C benchmark implementation available for research purposes.

In this paper we describe TPCC-UVa [5], an unofficial, open-source implementation of the TPC-C benchmark version 5. The purpose of TPCC-UVa is to be used as a research benchmark for the scientific community. The TPCC-UVa benchmark is written entirely in C language, and it uses the PostgreSQL database engine. This implementation has been extensively tested on Linux systems, and it is easily portable to other platforms. TPCC-UVa source code is freely distributed from the project website[1]. This makes easy to use it for the performance measurement and behavior of real systems or in the context

[1] http://www.infor.uva.es/~diego/tpcc-uva.html.

of a simulation environment such as Simics [6]. As an example, TPCC-UVa has been recently used in the performance measurement of both existent and experimental file systems [8].

The TPCC-UVa implementation includes all the characteristics described in the TPC-C standard specification, except support for price/performance comparison. The reason is that TPCC-UVa is only intended to be used for measuring performance in research environments. It is important to highlight the fact that TPCC-UVa is not an implementation approved by TPC, and the results of the execution of TPCC-UVa, in particular its performance parameter (tpmC-uva), should not be compared with the performance values obtained by official implementations of TPC-C.

The rest of the article is organized as follows. Section 2 describes the main characteristics of the TPC-C benchmark specification. Section 3 presents the TPCC-UVa implementation, describing its architecture in detail. Section 4 shows the performance reports generated by TPCC-UVa in order to meet TPC-C standard requirements. Section 5 shows the use of TPCC-UVa for measuring different aspects of system performance on real machines. Finally, Section 6 concludes the paper.

2 Overview of the TPC-C standard specification

The TPC-C benchmark specification simulates the execution of a mixture of read-only and update intensive transactions that simulate the activities found in complex OLTP application environments [1]. The TPC-C workload is determined by the activity of a set of terminals that request the execution of different database transactions, simulating the business activity of a wholesale supplier.

Five different transaction types are defined by the standard. The *New Order* transaction consists of entering a complete order through a single database transaction; the *Payment* transaction enters a customer's payment; the *Order Status* transaction queries the status of a customer's last order; the *Delivery* transaction processes a batch of ten new, not-yet-delivered orders; finally, the *Stock Level* transactions determines the number of recently sold items that have a stock level below a specified threshold.

When a terminal send the transaction request it waits to receive the results in all cases, except for the Delivery transaction, that simulates a transaction executed in deferred mode. The structure of the corresponding database is composed by several tables, with different characteristics with respect to their scheme and cardinality. This benchmark includes a scalability criteria that allows to simulate a realistic workload, allowing to change the database size and the number of transaction terminals for a more accurate simulation of the machine capabilities.

After the execution of the benchmark during a given period of time, the number of New Order transactions executed per minute gives the performance metric, called *transactions-per-minute-C* (tpmC). The TPC-C benchmark also includes a performance value that takes into account the cost of the system under test, the *price-per-tpmC*, to allow a comparison in terms of price/performance. Additional details can be found in the TPC-C standard specification [1].

3 TPCC-UVa architecture and implementation

The TPCC-UVa implementation is composed by five different modules that collaborate to perform all the necessary activities to measure the performance of the system under test. Figure 1 shows the TPCC-UVa architecture. The modules are the following.

Benchmark controller This module interacts with the user, populating the database and allowing the launch of different experiments.

Remote Terminal Emulator (RTE) There is one RTE process per active terminal in the benchmark execution. It simulates the activity of a remote terminal, according with TPC-C specifications.

Transaction Monitor This module receives all the requests from the RTEs, executing queries to the underlying database system.

Checkpoints controller This module performs checkpoints periodically in the database system, registering timestamps at the beginning and the end of each checkpoint.

Vacuum Controller This module avoids the degradation produced by the continuous flow of operations to the database.

Interprocess communication is carried out using both shared-memory structures and system signals, allowing to run the benchmark in any Unix-like, shared-memory multiprocessor environment. The following subsections describe each module in more detail.

3.1 Benchmark Controller

The Benchmark Controller (BC) allows the user to access the benchmark functionality. It performs the following functions.

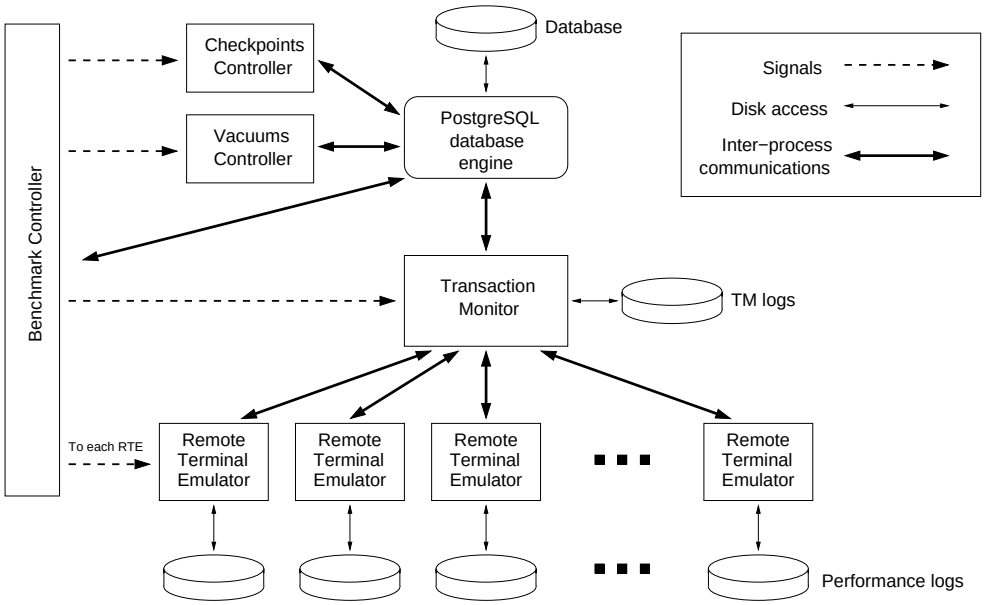

Figure 1: TPCC-UVa architecture.

Database initial population: It creates a new database to run a test. The database is composed by the nine tables defined in the TPC-C specifications, together with their required population and scalability characteristics. Different mechanisms to ensure the reference integrity of the data, such as primary and foreign keys, are also included.

Database consistency check: This option allows the user to check the consistency of the active database, to see if it meets the conditions described by the TPC-C standard to run a test on it.

Restoring an existent database: This option eliminates the modifications performed in the database tables by a previous test run. The purpose of this option is to rebuild a database to run a new test according with the TPC-C requirements without the need of creating a new one from scratch, a time-consuming operation.

Deleting a database: This option allows the user to delete the current database.

Executing a test: This option launches the TPCC-UVa modules that allow to run a measurement test. Such a test is composed by three intervals: the *ramp-up* period, a time when the performance of the system is not stable yet and therefore will not be considered for the performance measurement; the *measurement interval*, where the performance measurement is done; and the *end-of-test period*, when the Benchmark Controller stops all the related processes.

To execute a test, the user should define different execution parameters, such as the number of warehouses to be considered, the ramp-up period, the measurement interval and the configuration of the Vacuum Controller (described in Section 3.5). To run a test, the Benchmark Controller starts the Transaction Monitor, up to ten Remote Terminal Emulators for each one of the selected warehouses, and the Checkpoint and Vacuum Controllers (see Fig. 1). The Benchmark Controller also defines the experiment timings, informing each module about the current interval while executing a test.

Summary results of last test: This option reads and processes the benchmark logs produced by the set of Remote Terminal Emulators and the Transaction Monitor during the execution of the test. The information provided by the logs can be divided in two parts. The first one is the number of New Order transactions executed per minute, together with the response time of the executed transactions. This information will determine the performance of the system under test. The second part is the data needed to ensure that the test has been performed following the TPC-C specifications, such as the terminal response times and the relative percentage of each transaction in the executed transaction set. Both data types should be processed by the Benchmark Controller to ensure that the test is valid and to return the TPCC-UVa Transactions-Per-Minute (tpmC-uva) metric.

The Transactions-Per-Minute metric returned by TPCC-UVa is called tpmC-uva instead of tpmC. The reason is that, as we said in Section 1, the metric obtained

with TPCC-UVa should not be compared with tpmC values obtained by approved implementations of the benchmark.

3.2 Remote Terminal Emulators

The Remote Terminal Emulators (RTE from here on) generate the transaction requests for the system. Each RTE runs as an individual process, generating new transactions according with the requirements of the TPC-C benchmark specification. Once the measurement time is expired, the Benchmark Controller stops each one of the RTE using system signals. The RTE capabilities are the following.

User simulation: Each RTE simulates the behavior of a user connected to it, performing transaction type selection and transaction input data generation. It also simulates two related wait times: "keying time" and "think time".

Terminal simulation: Each RTE generates the output required by each terminal, showing the information introduced by the simulated user and the results obtained once the transaction is executed. Although each RTE can show this information in the standard output, the generated output is usually redirected to `/dev/null` to avoid collapsing the system console.

Transactions management: Each RTE generates a transaction type according with the TPC-C specifications, sending it to the Transactions Monitor. If the transaction is interactive, the results are sent back to the corresponding RTE once the transaction is completed.

Transaction response time measurement: Each RTE measures the response time for each one of the transactions requested. This data is stored locally in a log file, together with additional information that will be needed for the performance measurement of the system under test.

3.3 Transactions Monitor

The Transactions Monitor (TM from here on) receives the transaction requests from all the RTEs, passing them to the database engine and returning the generated results back to the RTEs. The transactions are executed according with their arrival order. The TM also registers the results of the delayed execution of the Delivery transaction and, when needed, data related to errors in the execution of transactions. The TM is activated and deactivated by the Benchmark Controller.

Clause 2.3.5 of the TPC-C standard specification [1] indicates that "if transactions are routed or organized within the SUT [System Under Test], a commercially available transaction processing monitor" is required, with a given set of functionalities. To avoid the use of commercially-available software, our TM does not route or organize transactions, but only queues them for execution in arrival order.

3.4 Checkpoints Controller

The Checkpoints Controller is responsible for ordering checkpoints periodically, registering the timestamps at the beginning and end of each checkpoint, according with Clause 5.5.2.2 of the TPC-C standard specification [1]. The first checkpoint is performed when the Checkpoints Controller is activated, at the beginning of the measurement interval.

3.5 Vacuum Controller

The Vacuum Controller mitigates the negative effects of a continuous flow of transaction executions in the database system. This controller is needed because the chosen database engine (PostgreSQL) keeps residual information that may slow down the database operation. To avoid a performance loss in the execution of long tests (i.e. more than two hours), the Vacuum Controller executes periodically the PostgreSQL `vacuum` command [9]. The user can configure the interval between vacuums and their maximum number.

3.6 TPCC-UVa communication procedures

Communication between the Transaction Monitor and each Remote Terminal Emulator is implemented using the communication procedures provided by Unix System V IPC interface, such as semaphores, shared memory and message queues [11]. The communication between the TM and the RTEs is based on the use of a single queue of pending transaction requests. This queue is used by the RTEs to submit transaction requests to the TM. The incoming order of the requests into the TM determine their execution order. A synchronization semaphore is used to manage reads and writes to this queue. Once a transaction is completed, the results are transmitted from the MT to the RTE that issued the request through a shared-memory data structure. Again, a semaphore is used to manage each data structure.

4 TPCC-UVa reports

The execution of TPCC-UVa on a System-Under-Test (SUT from here on) returns different performance metrics and plots. As an example, in this section we will describe in detail the results and plots obtained by TPCC-UVa version 1.2.3 on a dual core multiprocessor system. The SUT

```
Test results accounting performed on 2006-11-30 at 00:00:59 using 30 warehouses.

Start of measurement interval:  20.004883 m
End of measurement interval:  140.004983 m
COMPUTED THROUGHPUT: 367.791 tpmC-uva using 30 warehouses.
101424 Transactions committed.

NEW-ORDER TRANSACTIONS:
44135 Transactions within measurement time (50583 Total).
Percentage:  43.515%
Percentage of "well done" transactions:  97.100%
Response time (min/med/max/90th):  0.006 / 0.813 / 84.781 / 1.240
Percentage of rolled-back transactions:  0.986% .
Average number of items per order:  9.400 .
Percentage of remote items:  1.036% .
Think time (min/avg/max):  0.000 / 12.029 / 120.000

PAYMENT TRANSACTIONS:
44099 Transactions within measurement time (50712 Total).
Percentage:  43.480%
Percentage of "well done" transactions:  97.746%
Response time (min/med/max/90th):  0.002 / 0.596 / 89.864 / 0.960
Percentage of remote transactions:  14.148% .
Percentage of customers selected by C_ID: 39.087% .
Think time (min/avg/max):  0.000 / 11.971 / 120.000

ORDER-STATUS TRANSACTIONS:
4417 Transactions within measurement time (5081 Total).
Percentage:  4.355%
Percentage of "well done" transactions:  97.804%
Response time (min/med/max/90th):  0.001 / 0.703 / 84.308 / 1.080
Percentage of clients chosen by C_ID: 39.280% .
Think time (min/avg/max):  0.000 / 10.028 / 93.000

DELIVERY TRANSACTIONS:
4387 Transactions within measurement time (5057 Total).
Percentage:  4.325%
Percentage of "well done" transactions:  99.544%
Response time (min/med/max/90th):  0.000 / 0.122 / 72.625 / 0.010
Percentage of execution time < 80s :  100.000%
Execution time min/avg/max:  0.019/0.631/75.965
No.  of skipped districts:  0 .
Percentage of skipped districts:  0.000%.
Think time (min/avg/max):  0.000 / 5.013 / 47.000

STOCK-LEVEL TRANSACTIONS:
4386 Transactions within measurement time (5061 Total).
Percentage:  4.324%
Percentage of "well done" transactions:  99.772%
Response time (min/med/max/90th):  0.007 / 0.714 / 76.328 / 1.120
Think time (min/avg/max):  0.000 / 4.999 / 47.000

Longest checkpoints:
Start time Elapsed time since test start (s) Execution time (s)
Thu Nov 30 01:51:30 2006 6630.787000 11.000000
Thu Nov 30 01:21:20 2006 4820.357000 10.400000
Thu Nov 30 02:21:41 2006 8441.789000 10.200000
Thu Nov 30 00:51:10 2006 3010.331000 10.009000

No vacuums executed.

>> TEST PASSED
```

Figure 2: Results summary of a TPCC-UVa benchmark execution on a Dual Core Opteron multiprocessor system.

is a server equipped with two Dual Core AMD Opteron Processor 265 at 1 800 MHz (seen by Linux as four different processors), 2 GBytes of RAM and a RAID-5 storage system, using a LSI Logic MegaRAID Serial ATA 300-8X disk controller. The system runs Gentoo Linux with a 2.6.17 kernel, and we used PostgreSQL 8.1.4 as the underlying database system.

Figure 2 shows the results given by TPCC-UVa for a 2-hours test, using 30 warehouses, a ramp-up period of 20 minutes and no vacuum operation. The most important result is the computed throughput, in this case 367.791 tpmC-uva. To be valid, the test should meet some response time requirements, stated in Clause 5.5.1.5 of the TPC-C benchmark. The last line of the results file shown in Fig. 2 indicates whether these requirements have been met in this particular experiment.

In addition with the result summary given in Fig. 2, the TPCC-UVa implementation returns the data needed to draw the plots defined by the TPC-C standard specification. According to the standard, four different plot families should be generated. The plot families are described below.

Frequency distribution of Response Times Clause 5.6.1 of the TPC-C standard specification requires to build a graph called Response Time Distribution, that shows the number of transactions of each different transaction type that were completed in a given response time. Figure 3 shows both the plot as described by the TPC-C standard specification, and the response time distributions of the five transaction types given by TPCC-UVa.

Response Times vs. Throughput for the New Order transaction Clause 5.6.2 of the TPC-C standard specification requires to build a graph of response times versus throughput for the New Order transaction. The graph must be plotted at approximately 50%, 80% and 100% of the reported throughput rate (additional data points are optional). Figure 4 shows both the plot as described by the TPC-C standard specification, and the corresponding plot obtained with TPCC-UVa.

Frequency distribution of Think Times Clause 5.6.3 of the TPC-C standard specification requires to build a graph with the frequency distribution of Think Times for the New Order transaction. At least 20 different intervals of equal length must be reported. Figure 5 shows both the plot as described by the TPC-C standard specification, and the corresponding plot obtained with TPCC-UVa.

Throughput of the New Order Transaction Clause 5.6.4 of the TPC-C standard specification requires to build a graph with the throughput of the New Order transaction versus elapsed time, for both the ramp-up period and measurement interval. At least 240 different intervals should be used, with a maximum interval size of 30 seconds. The opening and the closing of the measurement interval must also be reported and shown on the graph. Figure 6 shows both the plot as described by the TPC-C standard specification, and the corresponding plot obtained with TPCC-UVa.

5 System performance measurement

In this section we use TPCC-UVa to measure different aspects of system performance. The following sections discuss some results obtained with TPCC-UVa to compare two important aspects of system performance, such as file system performance and multi-core capabilities. The purpose of these experiments is to show how TPCC-UVa can be used to obtain a reliable measure of different systems under test.

5.1 File systems performance comparison

We have used TPCC-UVa to measure the performance of different file system implementations under Linux. The SUT for this experiment is a server equipped with two Dual Core AMD Opteron Processor 265 at 1 800 MHz with 2 GBytes of RAM and running Gentoo Linux with a 2.6.17 kernel. We have used a Seagate Barracuda 7200.7 Serial ATA disk to store the database. This disk was divided into four primary partitions, each one formatted using a different file system type.

The file system types considered in this experiment were the following: ext2fs, the classical file system for Linux [2]; ext3fs, a version of ext2fs with journaling; ReiserFS version 3.6, a journaling file system for Linux based on balance tree algorithms, developed by Namesys; and JFS, based on IBM's journaled file system technology and now open-source.

We have run several experiments using each file system to store the TPCC-UVa database maintained by PostgreSQL. Each experiment ran for two hours, with ramp-up periods of 20 minutes and no vacuum operations. Figure 7 shows the tpmC-uva obtained using different number of warehouses for each one of the four file systems considered.

As can be seen in Fig. 7, ext2fs gives better results than file systems with journaling, particularly ReiserFS (3.0 percent slower) and JFS (8.9 percent slower). The maximum number of warehouses that the SUT was capable to serve was 27 for ext2fs, ext3fs and ReiserFS, while JFS allowed to run the benchmark with at most 25 warehouses.

The response time requirements defined by TPC-C were not met with more warehouses.

5.2 Dual- and single-core performance comparison

Finally, in this section we compare the performance of a SUT with two dual-core processors with respect to an identical system but with two single-core processors.

The dual-core SUT has two Dual Core Opteron 265 (seen by Linux as four processors), while the single-core SUT has two Single Core Opteron 246 processors. Both systems have 2 GBytes of RAM and a RAID-5 storage system, using a LSI Logic MegaRAID Serial ATA 300-8X disk controller and a ReiserFS file system. Both systems run Gentoo Linux with a 2.6.17 kernel. We have run different 2-hours experiments, with ramp-up period of 20 minutes and with no vacuum operations.

Figure 8 shows the tpmC-uva obtained using different number of warehouses for both SUTs, together with the maximum throughput of the New Order transaction in both cases. The results show that the dual-core architecture allowed a workload of up to 30 warehouses, with a 28.1% performance gain over the single-core architecture, that only dealt with up to 25 warehouses. These results show that, as expected, multi-core architectures are a valid choice for running transaction-oriented database workloads.

6 Conclusions

This paper describes TPCC-UVa, an open-source implementation of the TPC-C benchmark intended for measuring performance of parallel and distributed systems. The implementation simulates the execution of an OLTP environment according with the TPC-C standard specification. The major characteristics of the TPC-C specification has been discussed, together with a description of the TPCC-UVa architecture, performance metrics and plots generated and real examples of performance measurements for parallel systems. TPCC-UVa can be freely downloaded from `http://www.infor.uva.es/~diego/tpcc-uva.html`.

Acknowledgments

The author is partially supported by the European Commission under grant RII3-CT-2003-506079, and by Castilla-Leon Regional Government under grant VA031B06. The author would like to thank Julio A. Hernández and Eduardo Hernández for implementing the first version of TPCC-UVa as part of their BSc. thesis; Belén Palop for her help during the development of the benchmark; Javier Ramos for his expertise and support in Linux environments; and Joaquín Adiego, Pablo de la Fuente and the Grinbd Group for allowing the use of their computing facilities to run some of the experiments.

References

[1] TPC benchmark C standard specification, revision 5. Transaction Processing Performance Council. `http://www.tpc.org`. Access date: December 2006.

[2] CARD, R., TS'O, T., AND TWEEDIE, S. Design and implementation of the second extended filesystem. In *Proc. of the First Dutch International Symposium on Linux* (December 1994). ISBN 90-367-0385-9.

[3] EISENBERG, A., AND MELTON, J. Standards in practice. *SIGMOD Rec. 27*, 3 (1998), 53–58.

[4] HENNING, J. L. SPEC CPU2000: Measuring CPU performance in the new millennium. *Computer 33*, 7 (2000), 28–35.

[5] LLANOS, D. R., AND PALOP, B. TPCC-UVa: An Open-Source Implementation of the TPC-C Benchmark. In *PMEO-PDS 06, Proc. of IPDPS 2006 Workshops* (April 2006), IEEE Press. ISBN 1-4244-0054-6.

[6] MAGNUSSON, P. S., CHRISTENSSON, M., ESKILSON, J., FORSGREN, D., HALLBERG, G., HGBERG, J., LARSON, F., MOESTEDT, A., AND WERNER, B. Simics: A Full System Simulation Platform. *IEEE Computer* (February 2002), 50–58.

[7] NAS parallel benchmark. Access date: Dec. 2006. `http://science.nas.nasa.gov/Software/NPB`.

[8] PIERNAS, J., CORTÉS, T., AND GARCÍA, J. M. Traditional file systems versus DualFS: a performance comparison approach. *IEICE Trans. Inf. and Syst. E87-D*, 7 (July 2004).

[9] PostgreSQL 8.1.4 reference manual. PostgreSQL Global Development Group, 2005.

[10] ROGERS, A., CARLISLE, M. C., REPPY, J. H., AND HENDREN, L. J. Supporting dynamic data structures on distributed-memory machines. *ACM Trans. Program. Lang. Syst. 17*, 2 (1995), 233–263.

[11] STEVENS, W. R. *Advanced programming in the Unix environment*. Addison-Wesley, 1993. ISBN 0-201-56317-7.

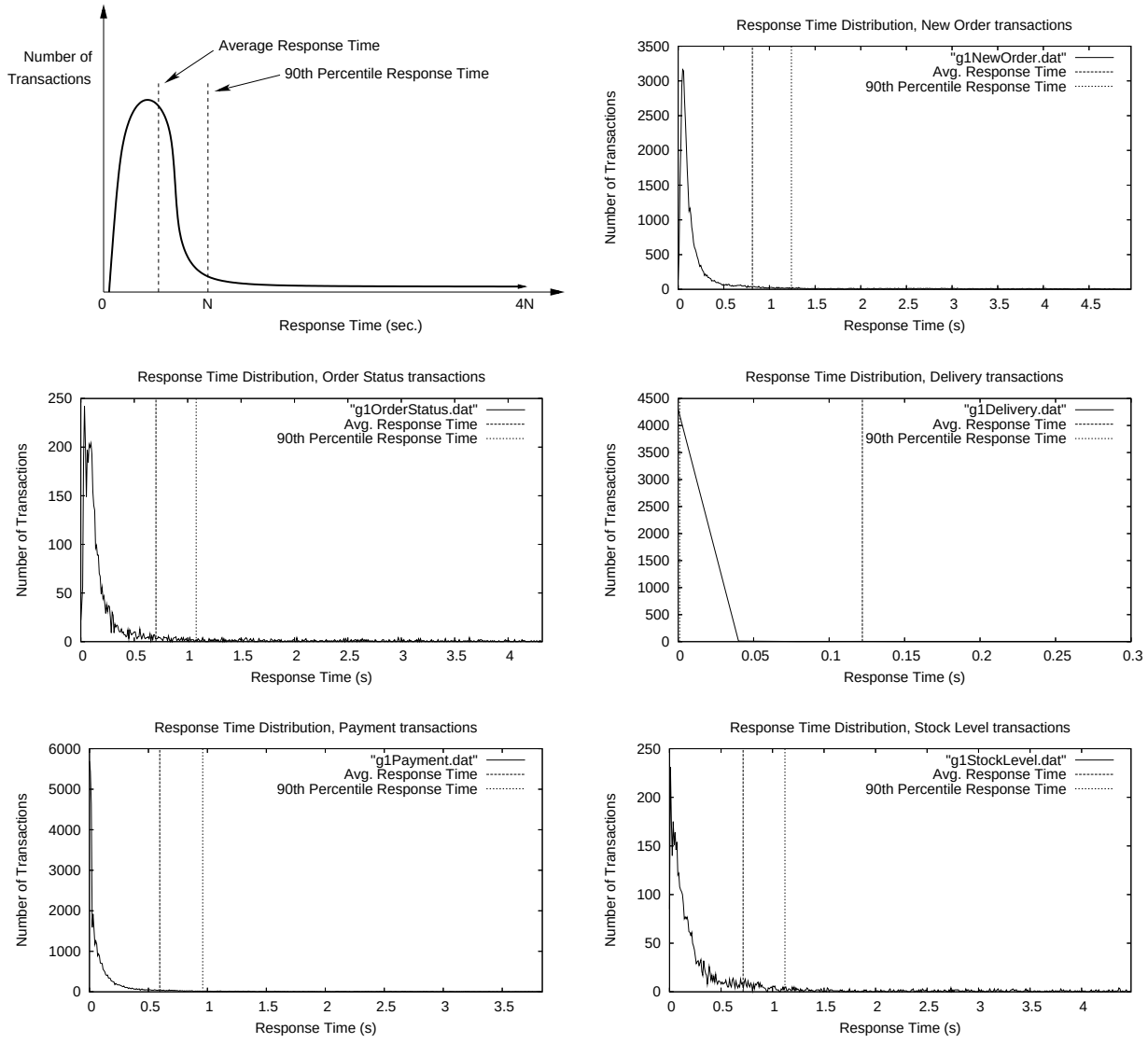

Figure 3: Response time distribution as required by clause 5.6.1 of the TPC-C standard specification (upper left corner) and response time distribution of the five transaction types given by TPCC-UVa for the System Under Test described in Section 4.

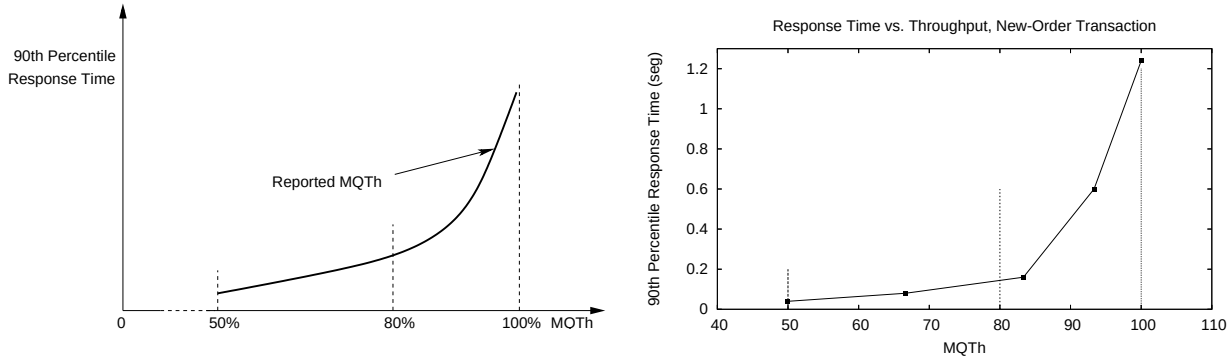

Figure 4: Response times vs. throughput for the New Order transaction as required by clause 5.6.2 of the TPC-C standard specification (left) and the corresponding plot returned by TPCC-UVa for the System Under Test described in Section 4, with configurations of 15, 20, 25, 28 and 30 warehouses (right).

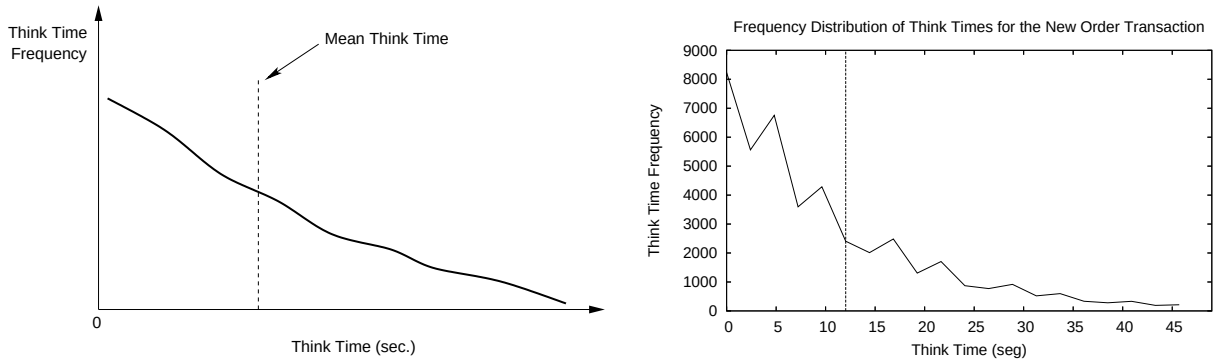

Figure 5: Frequency distribution of Think Times for the New Order transaction as required by clause 5.6.3 of the TPC-C standard specification (left) and the corresponding plot returned by TPCC-UVa for the System Under Test described in Section 4 (right).

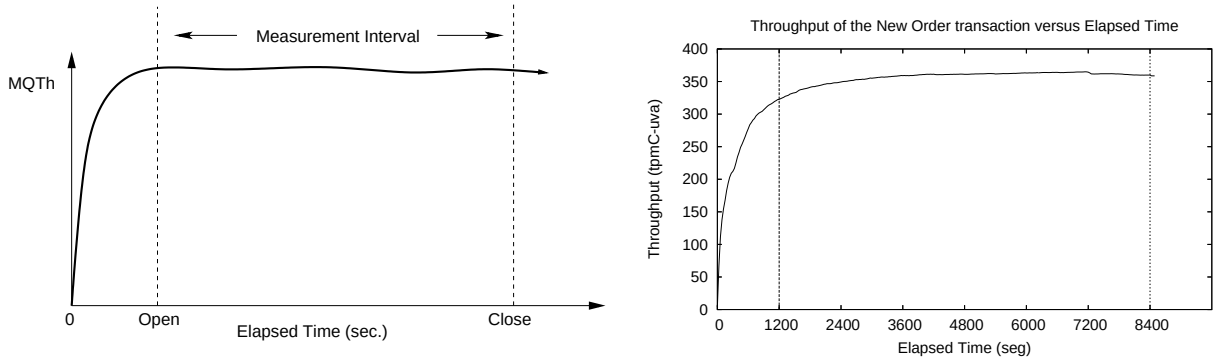

Figure 6: Throughput of the New Order transaction as required by clause 5.6.4 of the TPC-C standard specification (left) and the corresponding plot returned by TPCC-UVa for the System Under Test described in Section 4 (right).

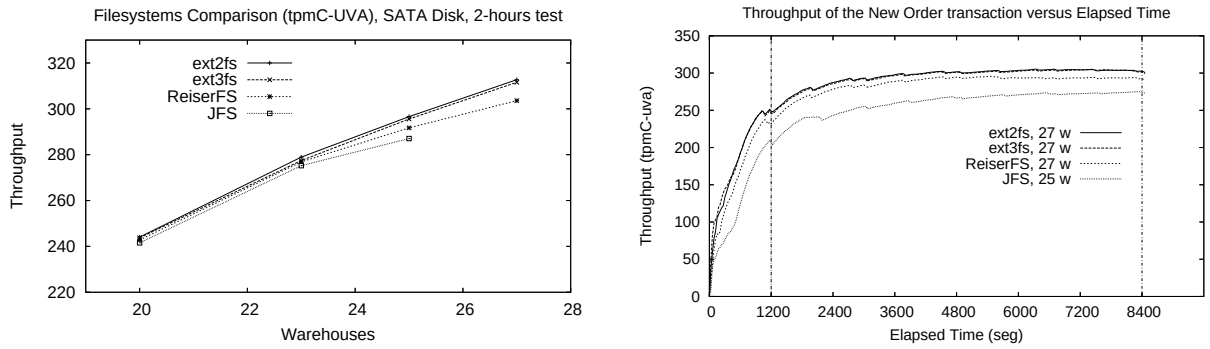

Figure 7: Number of tpmC-uva using different number of warehouses for each file system considered (left) and maximum throughput of the New Order transaction in each case (right).

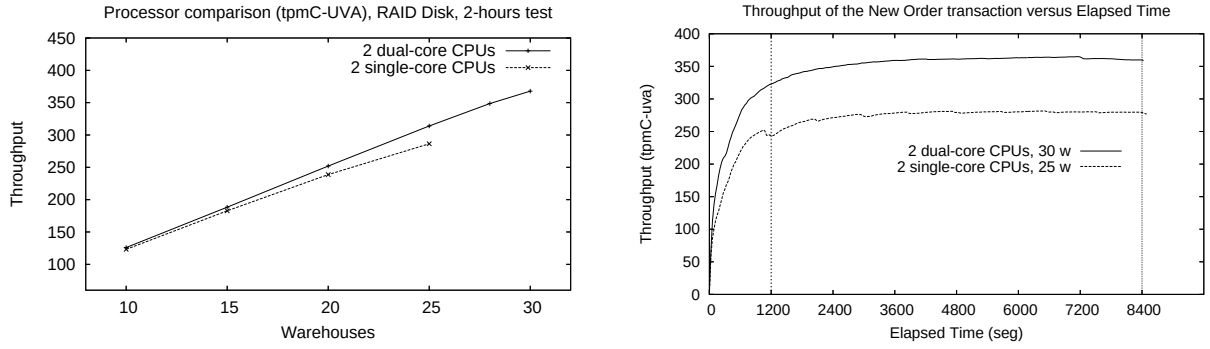

Figure 8: Number of tpmC-uva using different number of warehouses for each SUT considered (left) and maximum throughput of the New Order transaction in each case (right).