TPCC-UVa: An open-source TPC-C implementation for parallel and distributed systems

Conference Paper · April 2006

DOI: 10.1109/IPDPS.2006.1639646

2 authors:

Diego R. Llanos
University of Valladolid
172 PUBLICATIONS **964** CITATIONS
SEE PROFILE

Belen Palop
Complutense University of Madrid
60 PUBLICATIONS **577** CITATIONS
SEE PROFILE

TPCC-UVa: An Open-Source TPC-C Implementation for Parallel and Distributed Systems

Diego R. Llanos and Belén Palop

Universidad de Valladolid
Departamento de Informática
Valladolid, Spain
{diego,b.palop}@infor.uva.es

Abstract

This paper presents TPCC-UVa, an open-source implementation of the TPC-C benchmark intended to be used in parallel and distributed 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 and architecture and real examples of performance measurements.

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 [2], the NAS Parallel Benchmark [5] and the OLDEN benchmarks [8], 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 [10], 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 [1]. Unfortunately, there is not an official TPC-C benchmark implementation available for research purposes.

In this paper we present TPCC-UVa, an unofficial, open-source implementation of the TPC-C benchmark

D. R. Llanos is partially supported by the European Commission under grant RII3-CT-2003-506079. B. Palop is partially supported by MCYT TIC2003-08933-C02-01.

1-4244-0054-6/06/$20.00

version 5.0. 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. This makes easy to use it for the performance measurement and behavior of real systems or in the context of a simulation environment such as Simics [4]. As an example, TPCC-UVa has been recently used in the performance measurement of different file systems [6].

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), are not comparable 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 and main characteristics. Section 4 shows some experimental results obtained executing TPCC-UVa on a real, multiprocessor system, while Section 5 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 [10]. 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 [10].

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. These modules are:

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.

`http://www.infor.uva.es/~diego/tpcc-uva.html`.

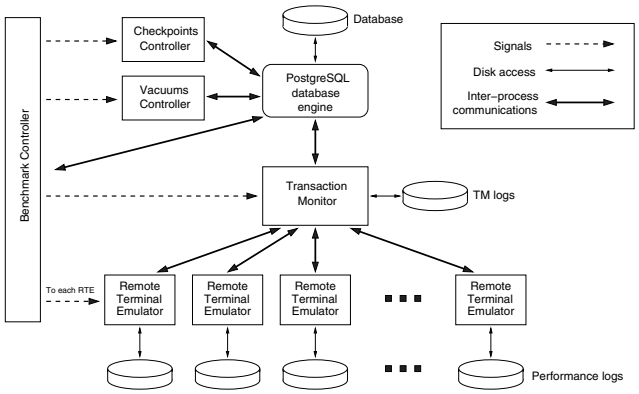


Figure 1. TPCC-UVa architecture.

Figure 1 shows the TPCC-UVa architecture. Interprocess communication is carried out using both shared-memory structures and system signals, allowing to run the benchmark in a shared-memory multiprocessor environment, as we will see in Section 3.6. 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:

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 are also included, such as primary and foreign keys.

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 (see Sect. 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 has been valid and to return the TPCC-UVa Transactions-Per-Minute (tpmC-uva) metric.

As we said in the Introduction, the tpmC-uva metric obtained with TPCC-UVa should not be compared with tpmC values obtained by approved implementations of the benchmark. To highlight this fact, the Transactions-Per-Minute metric returned by TPCC-UVa is called tpmC-uva instead of tpmC.

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 mea-

surement 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 [10] indicates that "if transactions are routed or organized within the SUT, 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 [10]. 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 slows down the database operation. To avoid a performance loss in the execution of long tests (more than two hours), the Vacuum Controller executes periodically the PostgreSQL `vacuum` command [7]. 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 was implemented using the communication procedures provided by Unix System V IPC interface, such as semaphores, shared memory and message queues [9]. 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 Experimental results

As a working example, in this section we present the results of the execution of TPCC-UVa on a multiprocessor system. The system under test is a Compaq ProLiant server, equipped with two Intel Xeon 2.40GHz processors, 1 280 Mb of RAM, three 36GB hard disks in RAID 5 configuration, and running RedHat Linux 9.0 with a 2.4.20-8smp precompiled kernel. Figure 2 shows the results given by TPCC-UVa for a 2-hours

```
Test results accounting performed on 2004-18-10 at 17:58:57 using 9 warehouses.

Start of measurement interval:  20.003233 m
End of measurement interval:  140.004750 m
COMPUTED THROUGHPUT: 107.882 tpmC-uva using 9 warehouses.
29746 Transactions committed.

NEW-ORDER TRANSACTIONS:
12946 Transactions within measurement time (15035 Total).
Percentage:  43.522%
Percentage of "well done" transactions:  90.854%
Response time (min/med/max/90th):  0.006 / 2.140 / 27.930 / 4.760
Percentage of rolled-back transactions:  1.012% .
Average number of items per order:  9.871 .
Percentage of remote items:  1.003% .
Think time (min/avg/max):  0.000 / 12.052 / 120.000

PAYMENT TRANSACTIONS:
12919 Transactions within measurement time (15042 Total).
Percentage:  43.431%
Percentage of "well done" transactions:  90.858%
Response time (min/med/max/90th):  0.011 / 2.061 / 27.387 / 4.760
Percentage of remote transactions:  14.862% .
Percentage of customers selected by C_ID: 39.601% .
Think time (min/avg/max):  0.000 / 12.043 / 120.000

ORDER-STATUS TRANSACTIONS:
1296 Transactions within measurement time (1509 Total).
Percentage:  4.357%
Percentage of "well done" transactions:  91.435%
Response time (min/med/max/90th):  0.016 / 2.070 / 27.293 / 4.640
Percentage of customers chosen by C_ID: 42.284% .
Think time (min/avg/max):  0.000 / 9.998 / 76.000

DELIVERY TRANSACTIONS:
1289 Transactions within measurement time (1502 Total).
Percentage:  4.333%
Percentage of "well done" transactions:  100.000%
Response time (min/med/max/90th):  0.000 / 0.000 / 0.001 / 0.000
Percentage of execution time < 80s :  100.000%
Execution time min/avg/max:  0.241/2.623/27.359
No.  of skipped districts:  0 .
Percentage of skipped districts:  0.000%.
Think time (min/avg/max):  0.000 / 4.991 / 38.000

STOCK-LEVEL TRANSACTIONS:
1296 Transactions within measurement time (1506 Total).
Percentage:  4.357%
Percentage of "well done" transactions:  99.691%
Response time (min/med/max/90th):  0.026 / 2.386 / 26.685 / 5.120
Think time (min/avg/max):  0.000 / 5.014 / 38.000

Longest checkpoints:
Start time Elapsed time (s) Execution time (s)
Mon Oct 18 20:19:56 2004 8459.676000 27.581000
Mon Oct 18 18:49:10 2004 3013.506000 21.514000
Mon Oct 18 19:19:32 2004 4835.039000 14.397000
Mon Oct 18 18:18:57 2004 1200.238000 13.251000

No vacuums executed.

>> TEST PASSED
```

Figure 2. Results summary of a TPCC-UVa benchmark execution on an Intel Xeon multiprocessor system.

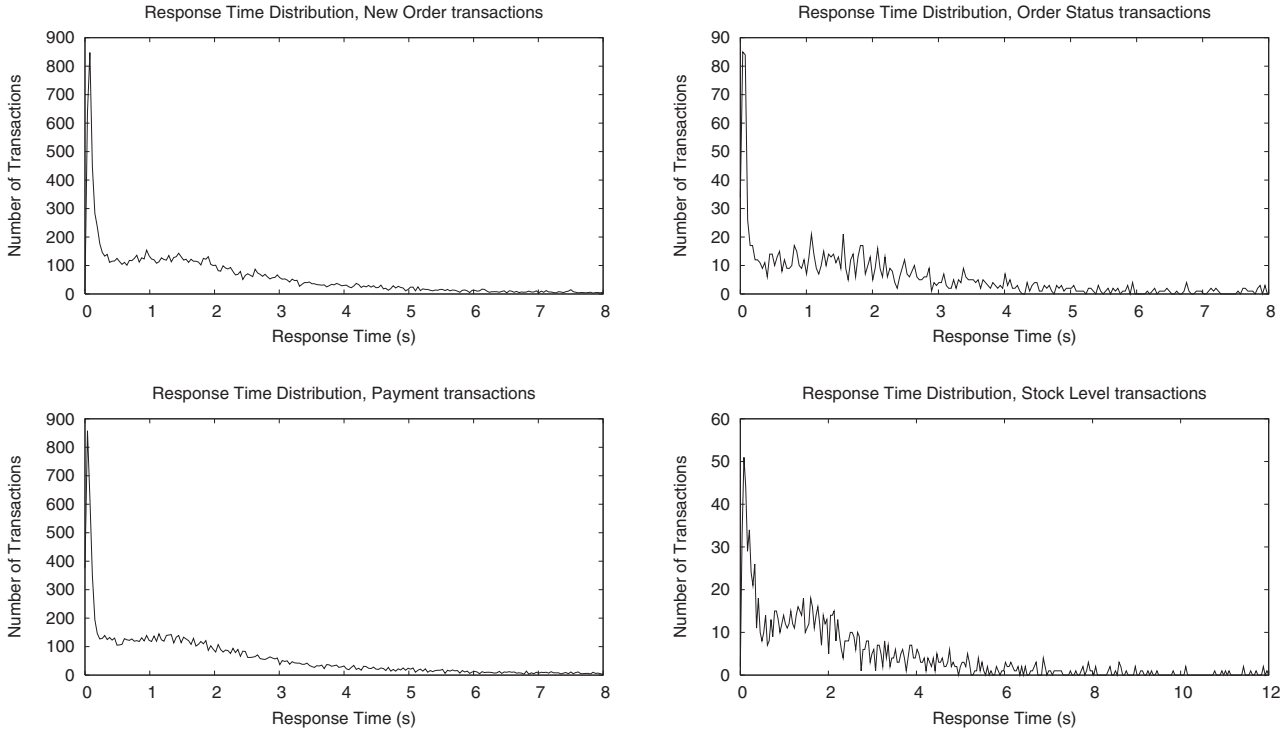


Figure 3. Response time distribution of some transaction types for a 2-hours execution on the system under test.

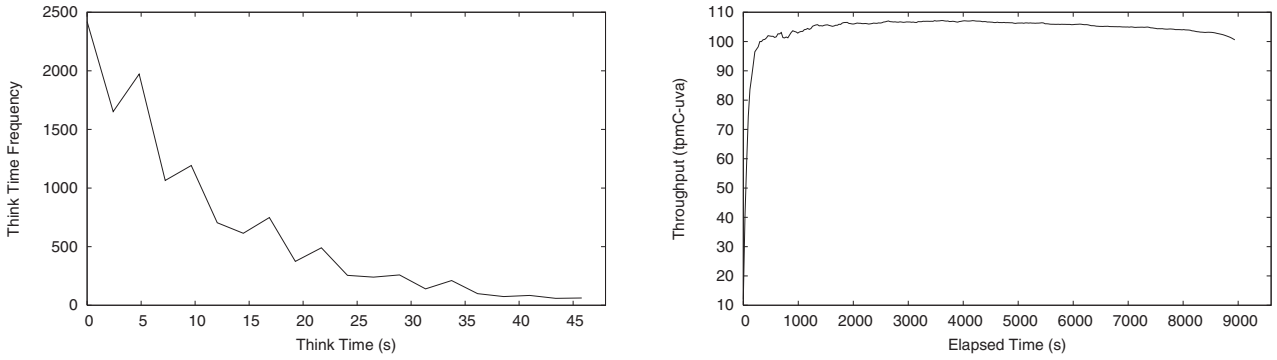
test, using nine warehouses, a ramp-up period of 20 minutes and no vacuum operation. The most important result is the computed throughput, in this case 107.882 tpmC-uva. To be valid, the test should meet some response time requirements: The last line of the results file indicates that these requirements have been met in our experiment.

The TPC-C standard requires some performance plots to be generated after a test run. The execution of TPCC-UVa produces different data files to generate the plots required [3]. Figure 3 shows the response time distribution for some transaction types for the experiment shown in Fig. 2. These plots have been obtained following the requirements described in Clause 5.6.1 of the TPC-C benchmark. Figure 4(a) shows the frequency distribution of think times for the New-Order Transaction (Clause 5.6.3), and Fig. 4(b) shows its corresponding throughput (Clause 5.6.4).

Finally, Fig. 5 shows the effect of the vacuum operations performed by the Vacuum Controlled described in Sec. 3.5. Figure 5(a) shows the performance of the system under test in a 8-hours test with hourly vacuums, while Fig. 5(b) shows the performance obtained in a experiment of similar characteristics but with no vacuums. It can be seen that the performance during the first hour (3 600 seconds) is very similar in both cases. After each vacuum, the performance shown in Fig. 5(a) drops noticeably, but the system can keep a stable number of New-Order transactions per minute during the execution of the experiment. Figure 5(b) shows that the performance of the experiment with no vacuums is very good during the first four hours, but the database pollution gets worse as the experiment proceeds. In fact, this pollution makes the second experiment fail, because the response times obtained at the end does not meet the TPC-C standard requirements.

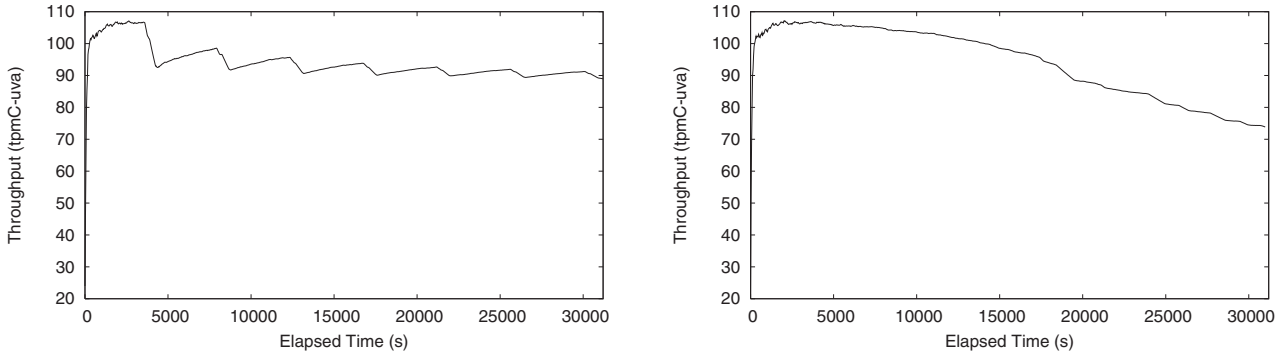
5 Conclusions

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 and real



(a) (b)

Figure 4. (a) Frequency distribution of think times and (b) throughput of the New-Order transaction for a 2-hours execution on the system under test.



(a) (b)

Figure 5. Throughput of the New-Order transaction for a 2-hours execution on the system under test With (a) hourly vacuum operations, and (b) no vacuums.

examples of performance measurements for a parallel system. TPCC-UVa can be freely downloaded from `http://www.infor.uva.es/~diego/tpcc-uva.html`.

Acknowledgments

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.

References

[1] A. Eisenberg and J. Melton. Standards in practice. *SIGMOD Rec.*, 27(3):53–58, 1998.

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

[3] Llanos, Diego R. TPCC-UVA Installation and User Guide. Technical Report Revision 6, Computer Science Department, University of Valladolid, Spain, November 2004.

[4] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hgberg, F. Larson, A. Moestedt, and B. Werner. Simics: A Full System Simulation Platform. *IEEE Computer*, pages 50–58, February 2002.

[5] NAS parallel benchmark. `http://science.nas.nasa.gov/Software/NPB`. Access date: January 2005.

[6] J. Piernas, T. Cortés, and J. M. García. Traditional file systems versus DualFS: a performance comparison

approach. *IEICE Trans. Inf. and Syst.*, E87-D(7), July 2004.

[7] PostgreSQL 7.1 reference manual. PostgreSQL Global Development Group, 2001.

[8] A. Rogers, M. C. Carlisle, J. H. Reppy, and L. J. Hendren. Supporting dynamic data structures on distributed-memory machines. *ACM Trans. Program. Lang. Syst.*, 17(2):233–263, 1995.

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

[10] TPC benchmark C standard specification, revision 5.0. Transaction Processing Performance Council, February 2001.