tpcc-uva: an open-source implementation of the TPC-C benchmark

Installation and User Guide

Version 1.2, rev. 1

Diego R. Llanos
diego@infor.uva.es
University of Valladolid, Spain

October 10, 2006

Contents

Chapter 1

Introduction

The TPC-C benchmark is a well-know benchmark used to measure the performance of high-end systems. TPC-C simulates the execution of a set of distributed, on-line transactions (OLTP), for a period between two and eight hours. TPC-C incorporates five types of transactions with different complexity, for on-line and deferred execution on a database system. The execution of the benchmark produces a performance parameter, called "TPC-C transactions per minute" (tpmC). This parameter allows to compare the speed of different systems[1].

Although the TPC-C specifications are public, there was not a public implementation: each company develops and uses its own. We have developed an open-source version of the TPC-C benchmark, that can be used and distributed under the terms of the General Public License (GPL). This program, called tpcc-uva, is written entirely in C language and can be run in any Linux system. It uses the PostgreSQL database system and a simple transaction monitor. This implementation can be used to measure the speed of different computers or to analyze the behavior of individual components, either hardware or software, and its impact on the overall performance of the system. The development was started by Eduardo Hernández-Perdiguero and Julio A. Hernández-Gonzalo as part of their Bachelor's Thesis in the Escuela Universitaria Politécnica, University of Valladolid, with the advise of Diego R. Llanos.

To keep tpcc-uva as an open-source program, we use our own version of a very simple Transaction Monitor (TM), instead of using any commercial TM. This is the main deviation of tpcc-uva with respect to the TPC-C standard, which says that the TM used should be commercially available. Because of that, the results obtained with tpcc-uva, particularly the main performance parameter "tpcc-uva Transactions per minute (tpmC-uva)" can not be compared with values of tpmC obtained with other implementations.

1.1 Installation highlights

Version 1.2 of tpcc-uva has more flexible installation requirements than its predecessors. It is not even necessary to have root permissions on the system. All the user should do is to install the PostgreSQL database server and to compile tpcc-uva in order to successfully run the benchmark.

In order to allow to reproduce the experiments, we strongly suggest to indicate, together with the performance metric given by tpcc-uva, the operating system and the kernel and compiler used.

1.2 Copyright notice

tpcc-uva is distributed under the terms of the General Public License (GPL).

[1] The official comparison is available at `http://www.tpc.org/tpcc/results/tpcc_perf_results.asp`.

1.3 We need your opinion

We are happy to allow the use of this benchmark for research purposes. To improve it we need your feedback: if you use it, please let us know, writing to `diego@infor.uva.es`.

1.4 Acknowledgements

We would like to acknowledge the contribution of the following persons: Eduardo Hernández-Perdiguero and Julio Alberto Hernández-Gonzalo for developing the first version of `tpcc-uva` [1] under our advice; Belén Palop, for her useful work in the improvement of the benchmark [2]; and Carmen Pilar Martínez-Sánchez for helping us to debug the code and the documentation.

Chapter 2

Installation of tpcc-uva

2.1 Introduction

In this section we will describe how to correctly build the environment needed by the benchmark. To keep the compatibility with most Linux distributions, and to be fair with the characteristics of the system under test, the benchmark is distributed together with the source code of the database engine.

The time needed by the entire installation process depends on the characteristics of the system, but for version 1.2.0 can be estimated in less than an hour.

2.2 Initial requirements

The only requirements is to have a GNU/Linux system with 1Gb of recommended free disk space, and a running GCC compiler. The installation is much simpler in `tpcc-uva` 1.2.0 than for its predecessors: it is just a matter of installing the database engine, to compile the benchmark source code, to start the database and then to start the benchmark.

2.3 Installing PostgreSQL

We recommend the use of version 8.1.4 of the PostgreSQL engine. Other versions may work as well, but we have not try them. Do not use versions older than version 8.0.

1. Create a `tpcc-uva` directory, and unpack there the PostgreSQL tarball. Only the tarball labeled `base` is needed:

   ```
   diego@linux:> mkdir $HOME/tpcc-uva
   diego@linux:> mv postgresql-base-8.1.4.tar.bz2 tpcc-uva
   diego@linux:> cd $HOME/tpcc-uva
   diego@linux:> bunzip2 postgresql-base-8.1.4.tar.bz2
   diego@linux:> tar xvfz postgresql-base-8.1.4.tar
   ```

2. Configure the PostgreSQL database source code. We suggest to create a `tpcc-uva/pgsql` directory to install the binaries, libraries and include files of PostgreSQL. It is not really necessary to create a `postgres` user: you can run the database engine and the benchmark as the same user.

   Here we will summarize the steps needed to compile and install PostgreSQL. The detailed instructions can be found in the `INSTALL` file that comes with the source tarball of PostgreSQL.

```
diego@linux:> mkdir $HOME/tpcc-uva/pgsql
diego@linux:> cd $HOME/tpcc-uva/postgresql-8.1.4
diego@linux:> ./configure --prefix=$HOME/tpcc-uva/pgsql
```

`configure` should not fail. If it fails, the most common reason is that a particular library is missing in the system, such as `readline` or `zlib`. Contact your system administrator in order to install them, and run `configure` again.

3. Compile and install PostgreSQL. According to PostgreSQL documentation, the `gmake` binary should be used to compile it: other `make` implementations may not work. This is not a problem in Linux systems, but for other systems it should be taken into account.

```
diego@linux:> gmake
diego@linux:> gmake install
diego@linux:> mkdir $HOME/tpcc-uva/pgsql/data
```

4. Modify the `PATH` and `LD_LIBRARY_PATH` environment variables of your account. To do so, edit `$HOME/.bash_profile` and add the following lines:

```
PATH=$HOME/tpcc-uva/bin:$HOME/tpcc-uva/pgsql/bin:$PATH
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/tpcc-uva/pgsql/lib
export PATH LD_LIBRARY_PATH
```

5. Run the modified `$HOME/.bash_profile`. Be careful with the exact format of the following command:

```
diego@linux:> . $HOME/.bash_profile
```

6. Ensure that the paths are correct. Issue the following command:

```
diego@linux> which initdb
```

The answer should be `$HOME/tpcc-uva/bin/initdb`, indicating that the executable file is the one we have just compiled.

7. Run PostgreSQL.

```
diego@linux:> initdb -D $HOME/tpcc-uva/pgsql/data
diego@linux:> postmaster -D $HOME/tpcc-uva/pgsql/data >log.out 2>log.err &
```

From here on, PostgreSQL is running in the system. To use the `tpcc-uva` benchmark, this command should be executed again each time the machine reboots.

8. Change the PosgreSQL parameters. Perform the following changes in the `$HOME/tpcc-uva/pgsql/data/postgresql.conf` file:

   (a) Replace

   ```
   #checkpoint_segments = 3 # in logfile segments (16MB each), min 1
   ```

   in line 115 of `postgresql.conf` with the following:

   ```
   checkpoint_segments = 10 # in logfile segments (16MB each), min 1
   ```

This increments the number of WAL (Write Ahead Logging) files that should become full to force PostgreSQL to perform an automatic checkpoint.

(b) Replace

`#checkpoint_timeout = 300 # in seconds, range 30-3600`

in line 116 of `postgresql.conf` with the following:

`checkpoint_timeout = 3600 # in seconds, range 30-3600`

This moves to the maximun the time between automatic checkpoints.

9. Force PostgreSQL to re-read the configuration file:

```
diego@linux:> killall -HUP postmaster
```

Now PosgreSQL is ready for the test.

2.4 Installing Gnuplot

Gnuplot is needed to build the plots that summarize the results obtained with the benchmark. If you do not have Gnuplot already installed in the system, you will need it. Again, it is enough to compile it inside our user space.

1. Unpack the Gnuplot tarball under `$HOME/tpcc-uva`:

```
diego@linux:> cd $HOME/tpcc-uva
diego@linux:> gunzip gnuplot-3.7.1.tar.gz
diego@linux:> tar xvf gnuplot-3.7.1.tar
```

2. Go inside the Gnuplot source code directory and configure it:

```
diego@linux:> cd $HOME/tpcc-uva/gnuplot-3.7.1
diego@linux:> ./configure --prefix=$HOME/tpcc-uva/bin --without-x
```

To use only the libraries installed in the previous sections, we will not build the X interface of Gnuplot.

3. Compile and install the package:

```
diego@linux:> make
diego@linux:> make install
```

2.5 Installing tpcc-uva

1. Unpack the tpcc-uva tarball under `$HOME/tpcc-uva`:

```
diego@linux:> cd $HOME/tpcc-uva
diego@linux:> gunzip tpccuva-1.2.0.tar.gz
diego@linux:> tar xvf tpccuva-1.2.0.tar
```

2. Create a `tpcc-uva/var/tpcc` directory to store the files generated by *tpcc-uva* during the execution of the benchmark.

```
diego@linux:> mkdir $HOME/tpcc-uva/var
diego@linux:> mkdir $HOME/tpcc-uva/var/tpcc
```

3. **Important:** Adjust the environment variables present in `tpccuva-1.2.0/Makefile`.

4. Compile and install the package:

   ```
   diego@linux:> cd $HOME/tpcc-uva/tpccuva-1.2.0
   diego@linux:> make
   diego@linux:> make install
   ```

   Congratulations! The benchmark is now installed. Chapter 3 explains how to run the benchmark, and Chapter 4 how to obtain the performance plots according to TPCC specifications.

2.6 How to remove everything

To remove all the software can not be easier. Just kill the `postmaster` server and remove the directory tree.

```
diego@linux:> killall -9 postmaster
diego@linux:> rm -rf $HOME/tpcc-uva/
```

Chapter 3

Running the benchmark

This chapter explains how to run the benchmark. Each run is called a *test*. The next chapter is devoted to explain the basics of the analysis of the performance data after the execution of a test.

To run the benchmark, first create a directory to store the results. From there issue the following command:

```
diego@linux:> $HOME/tpcc-uva/bin/bench
```

This will launch the benchmark controller. A menu with several options will appear. The basic steps are the following:

1. Create a database.
2. Run the test and store the results.
3. Choose to run another test on the same database or to delete it and create a new one.

Therefore, if no database is present, only options 1 (create a database) and 8 (quit) are shown in the menu. The following sections describe each option in detail.

3.1 Option 1: Create a New Test Database

This option allows the user to crate a new test database for running the benchmark. This database will be populated according to the TPC-C standard requisites. If this option is not available, this means that the database has been already created. The database will be created in the `$HOME/tpcc-uva/pgsql/data` directory.

After choosing this option, the program asks for the number of warehouses the database will contain. This number should be between 1 and 100. A greater number of warehouses means a larger workload.

Note: Take into account that a large enough workload will make the benchmark test fail. Typical values are 1 warehouse for desktop PCs, or more for more powerful systems. To find the optimum value, run a 2-hours test with a given number of warehouses and, if it succeeds, try to increment it and run another one. Choose the maximum number your machine can process, and keep it during the remaining measurements.

Note: Each additional warehouse means 137Mb of additional disk space.

The database population process can be stopped at any time just by pressing `<Caryl>C`. The program will ask the user for confirmation. The number of rows that the database will have is described in Table 3.1.

`Item` table	100,000 rows.
`Warehouse` table	1 row for each warehouse.
`District` table	10 rows for each warehouse.
`Customer` table	30,000 rows for each warehouse.
`History` table	30,000 rows for each warehouse.
`Orderr` table	30,000 rows for each warehouse.
`Order-line` table	A mean value of 300,000 rows for each warehouse.
`New-Order` table	9,000 rows for each warehouse.
`Stock` table	100,000 rows for each warehouse.

Table 3.1: Number of rows of the database

3.2 Option 2: Restore Existing Database

This option undo the changes on the database due to the execution of a test. The option only appears in the menu when a database has already been created.

Although is it possible to run a test with a restored database, the results of the test will be worse than using a brand new database. On the other hand, to restore a database is much less time-consuming than to create a new one. We recommend to use restored database only for preliminary tests.

The benchmark will not try to recover corrupted databases. If in doubt, remove the existing database and build a new one.

3.3 Option 3: Run The Test

This option runs the performance test. This option is only shown when a database has already been created or restored.

The program will ask for the parameters needed to run the test. These parameters are the following:

Number of warehouses: The number should be lower than or equal to the number of warehouses stored in the database.

Number of terminals per warehouse: The official TPC-C specifications set this value to 10, although for preliminary tests we may choose to use a lower value.

Ramp-up period: The terminal processes are launched during the so-called "ramp-up period". After this period the performance is expected to be stable, in order to start the measurements (as specified in Clause 5.5 of the TPC-C benchmark). The ramp-up period should be set in minutes: A typical value is around 20 minutes.

Measurement period: This is the time when the performance will be calculated. The TPC-C standard specifies that this period should be last for 2 hours (120 minutes) to 8 hours (480 minutes). The chosen period should be set in minutes.

After setting the values described above, the program will ask for a confirmation to continue the process. If the values are not correct the program will ask for them again.

Once the values are correct the system will ask the user if he/she wants to perform **vacuums** during the test. These vacuums are needed in order to eliminate residual information from the database that degrade performance. The maximum number of vacuums and the interval between vacuums can be set by the user. For eight-hours test it is useful to perform vacuums each 60 minutes, with a maxi-mun of six vacuums[1]. The effect of these vacuums can be clearly observed in the performance plots obtained at the end of the test.

[1] The maxi-mun number of vacuums is set to avoid performing a vacuum just before finishing the test.

After a new confirmation message, the test will begin by checking the database consistency, and later executing the test itself. If the check fails for a given table, the program ask the user for confirmation before proceeding. Although it is possible to proceed, we recommend to restore or rebuild the database in case of error. The only exception is the **New-Order** table, because the transactions that works with this table permit it to have fewer rows than the initial population. In any other case we recommend not to continue with the test.

After the database check, the Transactions Monitor (TM) and the Remote Terminal Emulators (RTE) will be launched, together with the Vacuum Controller, and the test will begin.

3.4 Option 4: Check Database Consistency

This option launches a check on the database, to ensure that the database follows the conditions of clauses 3.3.2.1, 3.3.2.2, 3.3.2.3 and 3.3.2.4 of the TPC-C standard [3]. Take into account that, although the TPC-C standard defines 12 consistency conditions, only the four first conditions should be explicitly demonstrated. This option is shown only if a database already exists.

3.5 Option 5: Delete Database

This option allow the user to delete an existing database.

3.6 Option 6: Perform Data Analysis

This option makes the program analyze the result data of the test. All the information is showed in the screen, including information checkpoint files and vacuums. This information, together with the files needed to build the performance graphics described in Clauses 5.6.1, 5.6.2, 5.6.3, 5.6.4 of the TPC-C standard [3], can be stored in the current directory on user request.

After showing all the information, a final message is written in the screen telling the user if the test has been passed or not. *The resulting tpmC-uva value is valid only if the test has been passed.*

3.7 Option 7: Check Database State

This option checks the number of rows of the database, showing the information to the user. This information is useful to see if the database has already been used to run a previous test. If so, the user may choose to delete it and create a new one, to restore it or to keep on using it. Take into account that the TPC-C standard specifies that valid test should only be run with a database with the number of rows showed in Table 3.1. See Section 3.1 for further information.

3.8 Option 8: Quit

This option closes the program. Any created database will remain for a next run of the program.

Chapter 4

Performance analysis

This chapter explains how to obtain the performance plots of a test run. As an example, we will use the results obtained running `tpcc-uva` on a standard Linux box. The characteristics of this System Under Test (SUT) are the following:

- Intel Pentium IV processor at 1.6GHz
- 256 Mb RAM, with 512 Mb swap space.
- 40 Gb hard disk, ext2 filesystem.
- Red Hat Linux 9, Linux kernel 2.4.20-8.

4.1 General results

The maximum number of warehouses the SUT can handle while keeping the response time requirements is three, with 10 terminals each. After determining this number of warehouses (running several experiments) we ran an eight-hours experiment, with 20 minutes of ramp-up period and with up to 6 vacuums, performed every 60 minutes. The general output file we obtained is the following:

```
Test results accounting performed on 2004-10-06 at 16:06:35 using 3 warehouses.

Start of measurement interval: 20.000317 m

End of measurement interval: 500.020467 m

COMPUTED THROUGHPUT: 35.226 tpmC-uva using 3 warehouses.

 38896 Transactions committed.

NEW-ORDER TRANSACTIONS:
  16909 Transactions within measurement time (17637 Total).
  Percentage: 43.472%
  Percentage of "well done" transactions: 97.155%
  Response time (min/med/max/90th): 0.025 / 1.960 / 253.407 / 1.680
  Percentage of rolled-back transactions: 1.005% .
  Average number of items per order: 9.908 .
  Percentage of remote items: 1.008% .
  Think time (min/avg/max): 0.000 / 12.021 / 115.000

PAYMENT TRANSACTIONS:
  16914 Transactions within measurement time (17653 Total).
```

```
  Percentage: 43.485%
  Percentage of "well done" transactions: 97.446%
  Response time (min/med/max/90th): 0.005 / 1.427 / 253.271 / 1.600
  Percentage of remote transactions: 15.153% .
  Percentage of customers selected by C_ID: 40.280% .
  Think time (min/avg/max): 0.000 / 11.996 / 115.000

ORDER-STATUS TRANSACTIONS:
  1691 Transactions within measurement time (1767 Total).
  Percentage: 4.347%
  Percentage of "well done" transactions: 97.694%
  Response time (min/med/max/90th): 0.035 / 1.031 / 152.543 / 1.680
  Percentage of customer selected by C_ID: 41.632% .
  Think time (min/avg/max): 0.000 / 9.706 / 90.000

DELIVERY TRANSACTIONS:
  1689 Transactions within measurement time (1764 Total).
  Percentage: 4.342%
  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 : 99.526%
  Execution time min/avg/max: 0.286/2.761/233.914
  No. of skipped districts: 0 .
  Percentage of skipped districts: 0.000%.
  Think time (min/avg/max): 0.000 / 4.834 / 45.000

STOCK-LEVEL TRANSACTIONS:
  1693 Transactions within measurement time (1766 Total).
  Percentage: 4.353%
  Percentage of "well done" transactions: 98.878%
  Response time (min/med/max/90th): 0.049 / 2.579 / 174.609 / 3.200
  Think time (min/avg/max): 0.000 / 4.803 / 45.000

Longest checkpoints:
Start time Elapsed time since test start (s) Execution time (s)
Thu Jun 10 21:27:49 2004 19274.158000 124.806000
Thu Jun 10 16:26:41 2004 1206.341000 10.490000
Thu Jun 10 23:30:14 2004 26619.928000 9.553000
Thu Jun 10 16:56:52 2004 3017.618000 8.884000

Longest vacuums:
Start time Elapsed time since test start (s) Execution time (s)
Thu Jun 10 20:19:04 2004 15149.690000 359.056000
Thu Jun 10 21:25:03 2004 19108.908000 328.324000
Thu Jun 10 22:30:32 2004 23037.508000 297.155000
Thu Jun 10 19:14:38 2004 11283.958000 265.560000

>> TEST PASSED
```

As we can see, the computed throughput has been 35.226 tpmC-uva using 3 warehouses. It is important to note again that this value is valid if and only if the test has been passed, that means that the different response times follows the standard requirements.

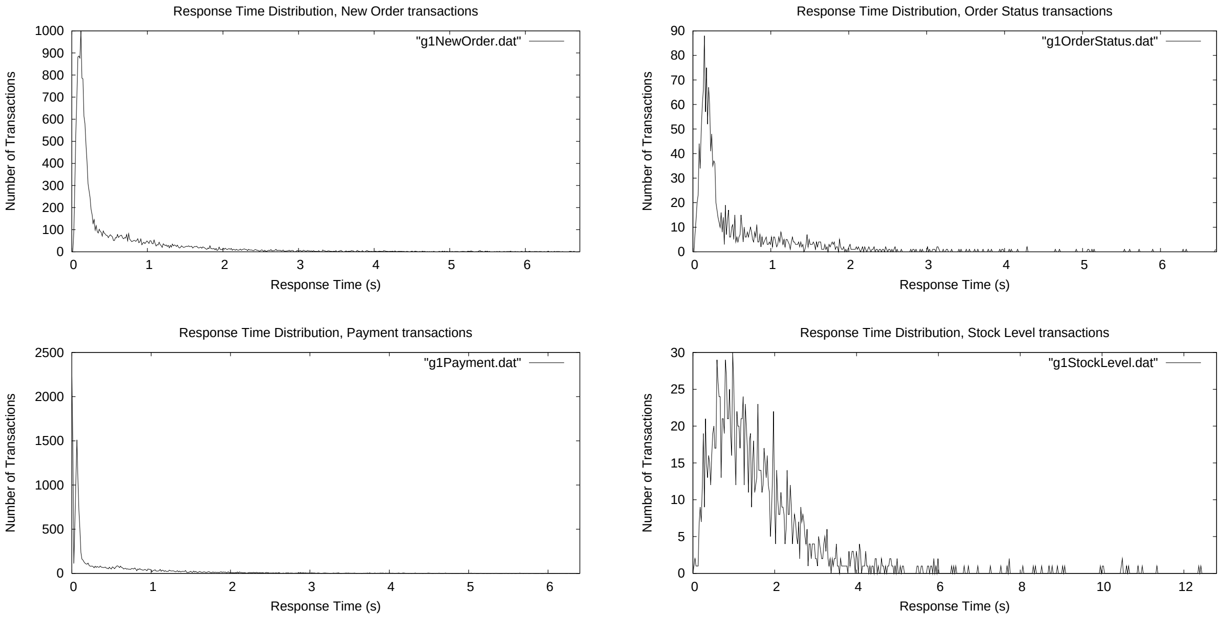

Figure 4.1: Example of plots generated according to Clause 5.6.1 of the TPCC benchmark

4.2 Frequency Distribution of Response Times (Clause 5.6.1)

The purpose of this report is explained in Clause 5.6.1 of the TPC-C Benchmark. The data needed to build each one of the required plots are in the following files: `g1Delivery.dat`, `g1NewOrder.dat`, `g1OrderStatus.dat`, `g1Payment.dat` and `g1StockLevel.dat`.

As an example, the Frequency Distribution of Response Times for our SUT in the experiment shown above can be seen in Figure 4.1. The response time for the Delivery transaction is usually to small to be accurately represented.

The plots shown in Fig. 4.1 were generated using Gnuplot and the script shown below. To generate his/her new plots, the user should replace the "<4x90thPERCENTILE>" label that appear in the script file below with four times the 90th percentile time for each transaction (the values can be found in the main, text-only output file). Save the script as `561.gnp` in the same directory as the output files and execute `gnuplot 561.gnp`.

```
# 561.eps file generation
# (Plot according to Clause 5.6.1 TPC-C Standard)

set terminal postscript 22
set size  29.7/21 , 1.
set pointsize 1

set output "561-NewOrder.eps"
set title "Response Time Distribution, New Order transactions"
set xlabel "Response Time (s)"
set ylabel "Number of Transactions"
plot [0: <4x90thPERCENTILE> ] "g1NewOrder.dat" with lines

set output "561-Delivery.eps"
set title "Response Time Distribution, Delivery transactions"
set xlabel "Response Time (s)"
```

```
set ylabel "Number of Transactions"
plot [0: <4x90thPERCENTILE> ] "g1Delivery.dat" with lines

set output "561-OrderStatus.eps"
set title "Response Time Distribution, Order Status transactions"
set xlabel "Response Time (s)"
set ylabel "Number of Transactions"
plot [0: <4x90thPERCENTILE> ] "g1OrderStatus.dat" with lines

set output "561-Payment.eps"
set title "Response Time Distribution, Payment transactions"
set xlabel "Response Time (s)"
set ylabel "Number of Transactions"
plot [0: <4x90thPERCENTILE> ] "g1Payment.dat" with lines

set output "561-StockLevel.eps"
set title "Response Time Distribution, Stock Level transactions"
set xlabel "Response Time (s)"
set ylabel "Number of Transactions"
plot [0: <4x90thPERCENTILE> ] "g1StockLevel.dat" with lines
```

4.3 Response Times vs. Throughput for the New Order Transaction (Clause 5.6.2)

The purpose of this report is explained in Clause 5.6.2 of the TPC-C Benchmark. The three points needed to build it should be obtained manually by the user. To do so, the user should first complete a valid test. Then the user should run at least other two tests, during at least 20 minutes of measure time, with 50% and 80% of the number of active terminals were used in the first experiment. Other workloads may be added to the plot as well. The 90th percentile of the response time for the New Order transaction for the three experiments should be plotted.

For example, in our SUT we obtained a value of 1.680 for the 90th percentile of response time for New Order transactions, with a workload of three warehouses (see the main output file at the beginning of this chapter). To obtain the corresponding values for the 50% and 80% of the workload, we ran two experiments, using the same number of warehouses but 5 and 8 terminals per warehouse instead of 10. The experiment have a ramp-up time period of 20 minutes, and another 20 minutes of measurement (the minimum specified by the standard).

After obtaining the three 90th percentile response times, the user should create a file with the workload percentages and the times, like the following one. Save it as `g2.dat`:

```
50    0.640
80    0.960
100   1.680
```

Finally, the user may use one script like the following one to produce the desired plot. The `<MAX-VALUE>` should be replaced with a value slightly greater than the maximum 90th percentile to be plotted (for example, two in our case).

```
# 562.eps file generation
# (Plot according to Clause 5.6.2 TPC-C Standard)

# set terminal postscript landscape 22
set terminal postscript 22
set size  29.7/21 , 1.
set pointsize 1
```

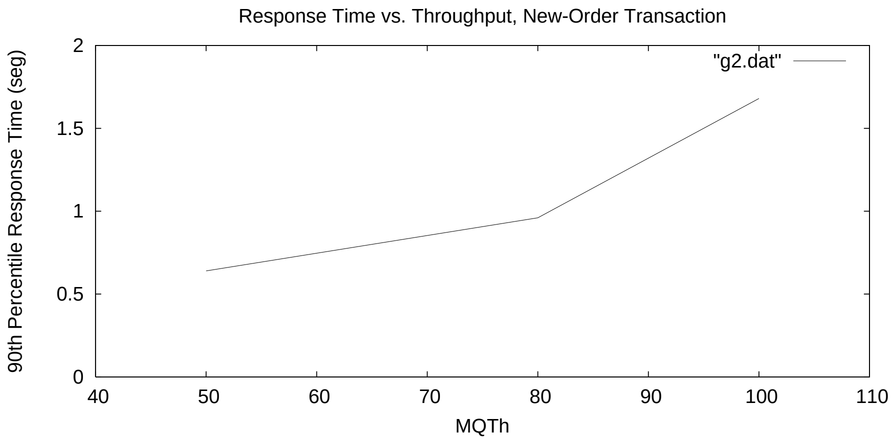

Figure 4.2: Example of plot generated according to Clause 5.6.2 of the TPCC benchmark.

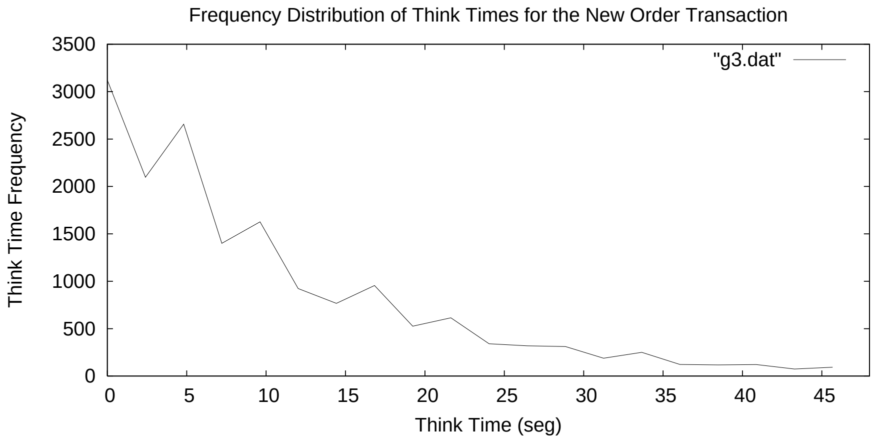

Figure 4.3: Example of plot generated according to Clause 5.6.3 of the TPCC benchmark.

```
set output "562.eps"
set title "Response Time vs. Throughput, New-Order Transaction"
set xlabel "MQTh"
set ylabel "90th Percentile Response Time (seg)"
plot [40:110][0: <MAX-VALUE> ] "g2.dat" with lines
```

Figure 4.2 shows the result for our SUT.

4.4 Frequency Distribution of Think Times (Clause 5.6.3)

The purpose of this report is explained in Clause 5.6.3 of the TPC-C Benchmark. This is the Frequency Distribution of Think Times for the New Order Transaction. The resulting plot obtained in our SUT is shown Figure 4.3.

The data needed to build the required plot is in the `g3.dat` file. The plot can be generated using Gnuplot and the following script. To use it, the user should replace the “`4xMEAN-THINK-TIME`” label that appear in the script file with four times the mean think time for the New-Order transaction (shown in

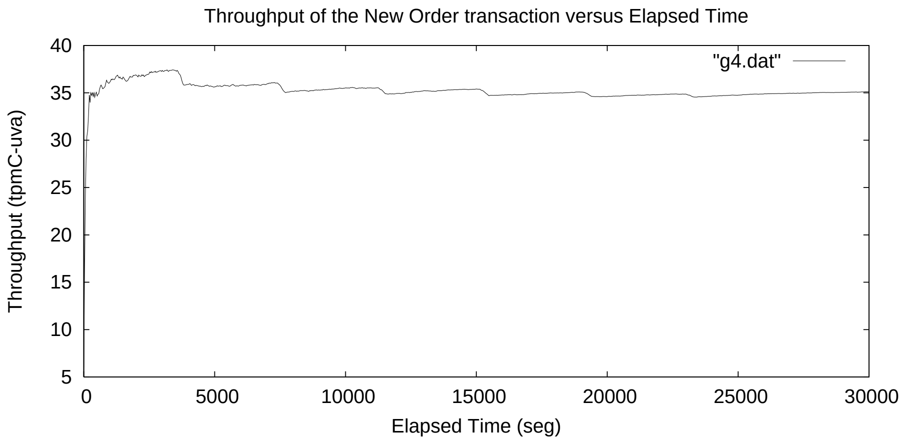

Figure 4.4: Example of plot generated according to Clause 5.6.4 of the TPCC benchmark.

the main text-only output file). Save the script as `563.gnp` in the same directory as the output files and execute `gnuplot 563.gnp`.

```
# 563.eps file generation
# (Plot according to Clause 5.6.3 TPC-C Standard)

set terminal postscript 22
set size  29.7/21 , 1.
set pointsize 1

set output "563.eps"
set title "Frequency Distribution of Think Times for the New Order Transaction"
set xlabel "Think Time (seg)"
set ylabel "Think Time Frequency"
plot [0: 4xMEAN-THINK-TIME] "g3.dat" with lines
```

4.5 Throughput of the New Order Transaction (Clause 5.6.4)

The purpose of this report is explained in Clause 5.6.4 of the TPC-C Benchmark. This is the most relevant plot to understand what happened during the measurement interval of the test. As an example, Figure 4.4 shows the evolution of the number of New Order transactions during the ramp-up period and the measurement interval. It is easy to see the effect of the vacuums in the evolution of the performance.

The data needed to build the required plot is in the `g4.dat` file. The plot can be generated using Gnuplot and the following script. To use it, the user should replace the `ELAPSED-TIME` value that appear in the script file with total elapsed time in seconds. This value is equivalent to the length of the measurement interval plus two times the ramp-up period, measured in seconds. For example, for a 8-hours test with 20 minutes of ramp-up period, the elapsed time will be 31200 seconds. Save the following script as `564.gnp` in the same directory as the output files and execute `gnuplot 564.gnp`.

```
# 564.eps file generation
# (Plot according to Clause 5.6.4 TPC-C Standard)

# set terminal postscript landscape 22
set terminal postscript 22
set size  29.7/21 , 1.
set pointsize 1
```

```
set output "564.eps"
set title "Throughput of the New Order transaction versus Elapsed Time"
set xlabel "Elapsed Time (seg)"
set ylabel "Throughput (tpmC-uva)"
plot [0: ELAPSED-TIME] "g4.dat" with lines
```

Bibliography

[1] Julio A. Hernández, Eduardo Hernández, and Diego R. Llanos. TPCC-UVA: Implementación del benchmark TPC-C. In *Proc. XIII Jornadas de Paralelismo*, Lérida, Spain, September 2002. ISBN 84-8409-159-7.

[2] Diego R. Llanos and Belén Palop. Tpcc-uva: An open-source implementation of the tpc-c benchmark. In *Proceedings of IPDPS 2006 Workshops (PMEO-PDS 06, 5th Workshop on Performance Modeling, Evaluation, and Optimization of Parallel and Distributed Systems.* IEEE Press, April 2006.

[3] Transaction Processing Performance Council. TPC Benchmark C Standard Specification. Technical Report Revision 5.0, February 2001.