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

Installation and User Guide

Version 1.1.1, rev. 7

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

May 17, 2006

Contents

1 Introduction **5**
1.1 Installation highlights 5
1.2 We need your opinion 6
1.3 Acknowledgements 6

2 Installation of tpcc-uva **7**
2.1 Introduction 7
2.2 Initial requirements 7
2.3 Preliminary steps 8
2.4 Compiling Binutils 9
2.5 Compiling GCC 10
2.6 Installing Linux headers 11
2.7 Installing Glibc 11
2.8 "Locking in" Glibc 13
2.9 Testing the new toolchain 13
2.10 Installing PostgreSQL 14
2.11 Installing Gnuplot 16
2.12 Installing tpcc-uva 16
2.13 How to remove everything 17

3 Running the benchmark **19**
3.1 Option 1: Create a New Test Database 19
3.2 Option 2: Restore Existing Database 20
3.3 Option 3: Run The Test 20
3.4 Option 4: Check Database Consistency 21
3.5 Option 5: Delete Database 21
3.6 Option 6: Perform Data Analysis 21
3.7 Option 7: Check Database State 21
3.8 Option 8: Quit 21

4 Performance analysis **23**
4.1 General results 23
4.2 Frequency Distribution of Response Times (Clause 5.6.1) 25
4.3 Response Times vs. Throughput for the New Order Transaction (Clause 5.6.2) 26
4.4 Frequency Distribution of Think Times (Clause 5.6.3) 27
4.5 Throughput of the New Order Transaction (Clause 5.6.4) 28

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

Although previous versions of tpcc-uva required Red Hat Linux 7.2 or 7.3, because the version of PostgreSQL used would not compile otherwise, the new version can be used in any GNU/Linux system in any architecture. To allow this platform-Independence, we do not distributed pre-compiled or RPM versions of the benchmark anymore, just the source code.

For this reason, and to avoid distortions in the measurement and compiler problems due to the use of the particular version of the toolchain present in the target system, tpcc-uva should be compiled with the exact versions of the toolchain distributed with it. Consequently, the steps involved with the installation of tpcc-uva includes the compilation of the exact versions of Binutils, Gcc and Glibc distributed with our package, together with the PostgreSQL database engine and tpcc-uva itself. The process is not difficult at all: detailed instructions are given in the following chapters.

The entire process can be done without modifying the versions of the toolchain actually installed in the system. Furthermore, as long as the software is installed under a specific directory, the entire software installed can be easily removed later through a couple of `rm` commands.

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

The building process is based on the excellent book *Linux From Scratch v5.0*, available online at `http://www.linuxfromscratch.org/`

1.2 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`.

1.3 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` under our advice; 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 toolchain and the database engine.

The time needed by the entire installation process depends on the characteristics of the system, but can be estimated in one to two hours.

Note: Although the benchmark could be compiled by other versions of GCC and/or use other dynamic libraries, the only reference value of Transactions-Per-Minute (*tpmC-uva*) for a given system is the one obtained with the toolchain described in this chapter.

2.2 Initial requirements

The only requirements is to have a GNU/Linux system with 1Gb of recommended free space, and a running GCC compiler. We only need this compiler to compile our own version of the Binutils and GCC: after this we will only use our new compiler.

It may seem that it would be easier to use just the default compiler of the system under test. However, there are several reasons to build a brand new toolchain:

- To avoid problems with non-stable versions of GCC installed on the host system. For example, GCC 3.2.2 (default compiler in RedHat Linux 9 distribution) refuses to compile the PostgreSQL package distributed with the TPCC benchmark.
- To avoid dependence problems with different versions of Glibc (the dynamic libraries of the system).
- To make the results of the benchmark reproducible, independently of the Linux distribution and its default compiler.
- To generate executable code optimized for the host architecture. This goal is not always possible with the compilers provided by some Linux distributions.

Note: The installation process should be performed step by step. Do not try to save time compiling different packages at the same time in different terminals of the same machine, since each package need to have all their predecessors correctly compiled and installed.

2.3 Preliminary steps

1. Log in as root.

2. Create a working directory called `opt` in a partition with enough free space. Here we will create it in the root directory:

   ```
   # cd /
   # mkdir opt
   ```

3. Define a new environmental variable called `$OPT` to point to the new directory created above. With `bash`, this can be done with the following command:

   ```
   # export OPT=/opt
   ```

4. Go to the `$OPT` directory and create there two more directories: one called `tools` and another called `source`:

   ```
   # cd $OPT
   # mkdir tools
   # mkdir sources
   ```

5. Create a `/tools` symlink from the root directory to the `tools` directory created in the preceding step:

   ```
   # ln -s $OPT/tools/ /tools
   ```

6. Copy into the `$OPT/sources` directory the tpcc-uva software environment tarball, and untar it:

   ```
   # cp /tmp/tpccuva-1.1.0-tarball.tar $OPT/sources/
   # cd $OPT/sources
   # tar xvf tpccuva-1.1.0-tarball.tar
   ```

   This will untar the following files:

   - The source code for the basic binary utilities (Binutils) needed to compile the GCC compiler.
   - The source code of the GCC compiler itself.
   - The source code of Linux kernel. We will not compile the kernel: we just need its include files.
   - The source code of the dynamic libraries (Glibc), together with the specific source code for Linux threads (Glibc-linuxthreads).
   - The source code of the PostgreSQL database engine.
   - The source code of the tpcc-uva benchmark.
   - The source code of Gnuplot, to produce the output plots.

2.4 Compiling Binutils

1. Uncompress the Binutils package:

```
# gunzip binutils-2.14.tar.gz
# tar xvf binutils-2.14.tar
```

2. The Binutils documentation recommends building Binutils outside of the source directory in a dedicated build directory. Create it and move into:

```
# mkdir binutils-build
# cd binutils-build
```

3. Run the `configure` script with the following parameters:

```
# ../binutils-2.14/configure --prefix=/tools --disable-nls
```

   This will configure Binutils according with your system architecture. With the `--prefix=/tools` option, Binutils will be installed inside your `/tools` directory, avoiding any conflict with your existing software. The `--disable-nls` option will disable internationalization, an option not needed by our benchmark.

4. Compile and install the package:

```
# make configure-host
# make LDFLAGS="-all-static"
# make install
```

   The package will be installed under the `/tools` directory.

5. Now prepare the newly compiled linker for the "locking in" of Glibc later on:

```
# make -C ld clean
# make -C ld LDFLAGS="-all-static" LIB_PATH=/tools/lib
```

   This will remove the `ld` executable created in the previous step and recompile it indicating the linker's new default library search path.

6. Add the `/tools/bin` and `/tools/sbin` directories as the first directories of our path. **Note:** This step should be executed whenever we open a new terminal to continue with the installation process.

```
# export PATH=/tools/bin:/tools/sbin:$PATH
```

7. We are not done yet with the Binutils package: do not remove the `binutils-build` nor the `binutils-2.14` directories. We will need them later, in Section 2.8.

2.5 Compiling GCC

1. From the `sources` directory, unpack the GCC tarball:

```
# cd $OPT/sources
# gunzip gcc-2.95.3.tar.gz
# tar xvf gcc-2.95.3.tar
```

2. The GCC documentation recommends building GCC outside of the source directory in a dedicated build directory. Create it and move into:

```
# mkdir gcc-build
# cd gcc-build
```

3. Prepare GCC for compilation:

```
# ../gcc-2.95.3/configure --prefix=/tools --with-local-prefix=/tools \
   --disable-nls --enable-shared --enable-languages=c,c++
```

The `-enable-shared` allows the compilation of `libgcc_s.so.1` and `libgcc_eh.a`, ensuring that the `configure` script of Glibc (the next package we will compile) produces the proper results. Note that the `gcc` binaries will still be linked statically, as this is controlled by the `-static` value of `BOOT_LDFLAGS` in the following step.

4. Compile the package:

```
# make BOOT_LDFLAGS="-static" bootstrap
```

The `bootstrap` parameter forces to compile Gcc several times, using the programs generated in a first round to compile itself a second and a third time. If the results of the second and third compiles are the same, this most probably means that it was compiled correctly.

5. Install the package:

```
# make install
```

6. Create the `/tools/bin/cc` symlink, pointing to `/tools/bin/gcc` (note the particular syntax of the following command):

```
# ln -sf gcc /tools/bin/cc
```

7. We are done with the GCC package. Remove all the GCC source and build directories, together with the original tarball, from the `$OPT/sources` directory:

```
# cd $OPT/sources
# rm -rf gcc*
```

2.6 Installing Linux headers

1. From the `sources` directory, unpack the Linux tarball:

   ```
   # cd $OPT/sources
   # gunzip linux-2.4.22.tar.gz
   # tar xvf linux-2.4.22.tar
   ```

2. Go to the new directory and prepare for the installation:

   ```
   # cd linux-2.4.22
   # make mrproper
   ```

3. Create the `include/linux/version.h` file:

   ```
   # make include/linux/version.h
   ```

4. Create the platform-specific `include/asm` symlink:

   ```
   # make symlinks
   ```

5. Install the platform-specific header files:

   ```
   # mkdir /tools/include/asm
   # cp include/asm/* /tools/include/asm/
   # cp -R include/asm-generic /tools/include/
   ```

6. Install the cross-platform kernel header files:

   ```
   # cp -R include/linux /tools/include/
   ```

7. To avoid compilation failures, create an empty `autoconf.h` file:

   ```
   # touch /tools/include/linux/autoconf.h
   ```

8. We are done with the Linux kernel package. Remove all the Linux kernel files from the `$OPT/sources` directory:

   ```
   # cd $OPT/sources
   # rm -rf linux*
   ```

2.7 Installing Glibc

1. From the `sources` directory, unpack the Glibc tarball:

   ```
   # cd $OPT/sources
   # gunzip glibc-2.2.5.tar.gz
   # tar xvf glibc-2.2.5.tar
   ```

2. Move the Glibc-linuxthreads file to the new directory, and uncompress it there:

   ```
   # cp glibc-linuxthreads-2.2.5.tar.gz glibc-2.2.5
   # cd glibc-2.2.5
   # gunzip glibc-linuxthreads-2.2.5.tar.gz
   # tar xvf glibc-linuxthreads-2.2.5.tar
   ```

3. This package is known to behave badly when you have changed its default optimization flags, including the `-march` and `-mcpu` options. Therefore, if you have defined any environment variables that override default optimizations, such as `CFLAGS` and `CXXFLAGS`, we recommend unsetting them when building Glibc.

4. Create an empty `/tools/etc/ld.so.conf` file:

```
# mkdir /tools/etc
# touch /tools/etc/ld.so.conf
```

5. The Glibc documentation recommends building Glibc outside of the source directory in a dedicated build directory. Create it and move into:

```
# cd $OPT/sources/
# mkdir glibc-build
# cd glibc-build
```

6. Prepare Glibc for compilation:

```
# ../glibc-2.2.5/configure --prefix=/tools --disable-profile \
  --enable-add-ons --with-headers=/tools/include --without-gd
```

The `-disable-profile` parameter disables the building of the libraries with profiling information. Omit this option if you plan to do profiling, but take into account that it might affect the performance of the benchmark.

7. Compile the package:

```
# make
```

After running the command, you can check any failed tests in the `check.err` file.

8. Install the package:

```
# make install
```

9. Install the Glibc locales. Not all of them are mandatory, but it is easier to install them all than to choose some of them one by one.

```
# make localedata/install-locales
```

10. We are done with the Glibc package. Remove all the Glibc source and build directories, together with the original tarball, from the `$OPT/sources` directory:

```
# cd $OPT/sources
# rm -rf glibc*
```

2.8 "Locking in" Glibc

1. Return to the `binutils-build` directory and install the adjusted linker:

```
# cd $OPT/sources/binutils-build/
# make -C ld install
```

2. Now we are done with the Binutils package. Remove all the Binutils files from the `$OPT/sources` directory:

```
# cd $OPT/sources
# rm -rf binutils*
```

3. The next thing to do is to amend our GCC specs file to point to the new dynamic linker. We will use a `sed` command to do this:

```
# SPECFILE=/tools/lib/gcc-lib/*/*/specs && \
    sed -e 's@ /lib/ld-linux.so.2@ /tools/lib/ld-linux.so.2@g' \
    $SPECFILE > tempspecfile && mv -f tempspecfile $SPECFILE && \
    unset SPECFILE
```

4. Finally, ensure that no include files from the host system have found their way into GCC's private include dir because of GCC "fixincludes" process. Run the following command:

```
# rm -f /tools/lib/gcc-lib/*/*/include/{pthread.h,bits/sigthread.h}
```

2.9 Testing the new toolchain

It is imperative at this point to ensure that the basic functions (compiling and linking) of the new toolchain are working as expected. Go to the `$OPT/sources` directory and run the following commands *exactly as they appear below*:

```
# echo 'main(){}' > dummy.c
# gcc dummy.c
# readelf -l a.out | grep ': /tools'
```

If everything is working correctly, there should be no errors, and the output of the last command will be:

```
[Requesting program interpreter:  /tools/lib/ld-linux.so.2]
```

If you do not receive the output as shown above, or received no output at all, then something is seriously wrong. The only exception to this is if you are working on a platform where the name of the dynamic linker is different than `ld-linux.so.2`. In this case, the output shown above may be slightly different. In any other case, retrace your steps to find out where the problem is and correct it.

Once you are satisfied that all is well, clean up the test files:

```
# rm -f dummy.c a.out
```

2.10 Installing PostgreSQL

1. From the `sources` directory, unpack the PostgreSQL tarball:

```
# cd $OPT/sources
# gunzip postgresql-7.1.3.tar.gz
# tar xvf postgresql-7.1.3.tar
```

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

```
# cd postgresql-7.1.3
# ./configure --prefix=/tools
```

3. Compile and install it. According with 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 in other systems it may be taken into account. Issue the following commands:

```
# gmake
# gmake install
```

4. Add a postgres user to the host system, choosing a password for him:

```
# adduser postgres
# passwd postgres
```

5. Create the directory where the database will be stored:

```
# mkdir /tools/pgsql
# mkdir /tools/pgsql/data
```

6. Make the postgres user the owner of the new directory:

```
# chown postgres /tools/pgsql/data
# chgrp postgres /tools/pgsql/data
```

7. Enter as the postgres user:

```
# su - postgres
```

8. As the postgres user, change the `$PATH` variable to have `/tools/bin` in first place:

```
$ export PATH=/tools/bin:$PATH
```

Note: It is a good idea to perform this modification also in the `.bashrc` file (or any other login script file you are using), to avoid confusions with any other version of PostgreSQL installed in the system

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

```
$ which initdb
```

The answer should be `/tools/bin/initdb`, indicating that the executable file is the one we have just compiled.

10. Set up the database structure:

```
$ initdb -D /tools/pgsql/data
```

11. Start the PostgreSQL database. As user postgres, execute:

    ```
    $ postmaster -D /tools/pgsql/data -i &
    ```

    Note: 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.

12. Change the PosgreSQL parameters. Perform the following changes in the `/tools/pgsql/data/postgresql.conf` file:

    (a) Replace

    ```
    #fsync = true
    ```

    in line 52 of `postgresql.conf` with the following:

    ```
    fsync = off
    ```

    This disconnects the “fsync” mode.

    (b) Replace

    ```
    #wal_files = 0 # range 0-64
    ```

    in line 109 of `postgresql.conf` with the following:

    ```
    wal_files = 10 # range 0-64
    ```

    This increments the number of WAL (Write Ahead Logging) files.

    (c) 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.

    (d) 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.

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

    ```
    $ killall -HUP postmaster
    ```

    Now PosgreSQL is ready for the test.

14. We are done with the PostgreSQL package. Remove all the PostgreSQL source directory, together with the original tarball, from the `$OPT/sources` directory:

    ```
    # cd $OPT/sources
    # rm -rf postgresql*
    ```

2.11 Installing Gnuplot

1. Exit the postgres user shell:

```
$ exit
```

2. As root, go to the sources directory and unpack the Gnuplot tarball:

```
# cd $OPT/sources
# gunzip gnuplot-3.7.1.tar.gz
# tar xvf gnuplot-3.7.1.tar
```

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

```
# cd gnuplot-3.7.1
# ./configure --prefix=/tools --without-x
```

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

4. Compile and install the package:

```
# make
# make install
```

5. We are done with the Gnuplot package. Remove all the Gnuplot source directory, together with the original tarball, from the $OPT/sources directory:

```
# cd $OPT/sources
# rm -rf gnuplot*
```

2.12 Installing tpcc-uva

1. Go to the sources directory and unpack the Gnuplot tarball:

```
# cd $OPT/sources
# gunzip tpccuva-1.1.0.tar.gz
# tar xvf tpccuva-1.1.0.tar
```

2. Compile and install the package:

```
# make
# 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 with TPCC specifications.

2.13 How to remove everything

These are the instructions to remove all the software installed in this chapter.

1. Kill the `postmaster` server:

   ```
   # killall -9 postmaster
   ```

2. After moving elsewhere any executions of the tpcc-uva benchmark you wish to keep, remove the `$OPT` directory:

   ```
   # cd /
   # rm -rf $OPT
   ```

3. Remove the `/tools` symlink:

   ```
   # rm -f /tools
   ```

4. Delete the postgres user:

   ```
   # userdel postgres
   ```

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, as root, run the following command:

```
# /tools/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 with 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 `/tools/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 [1]. 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 [1], 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.

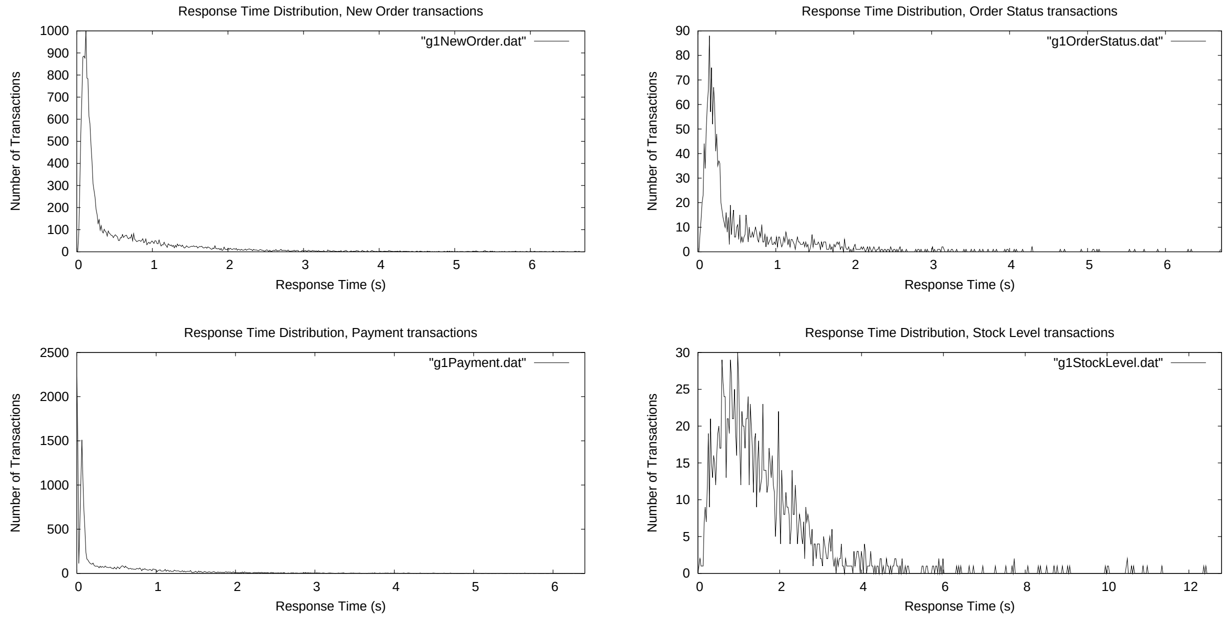

Figure 4.1: Example of plots generated according with Clause 5.6.1 of the TPCC benchmark

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 with 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 with 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
```

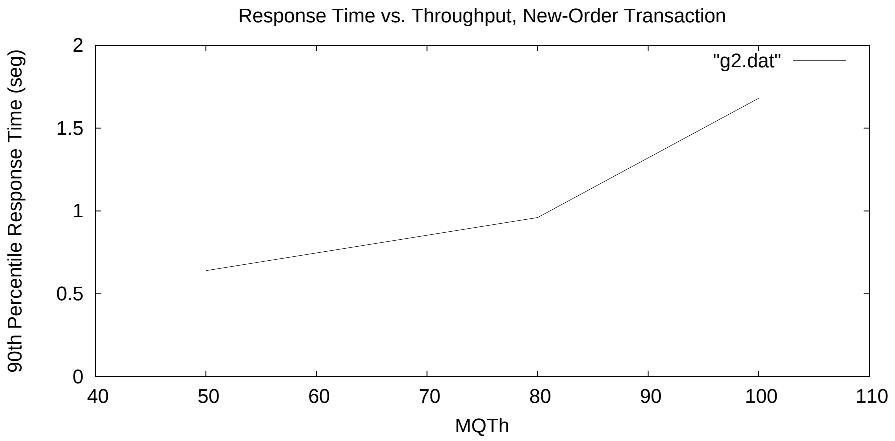

Figure 4.2: Example of plot generated according with Clause 5.6.2 of the TPCC benchmark.

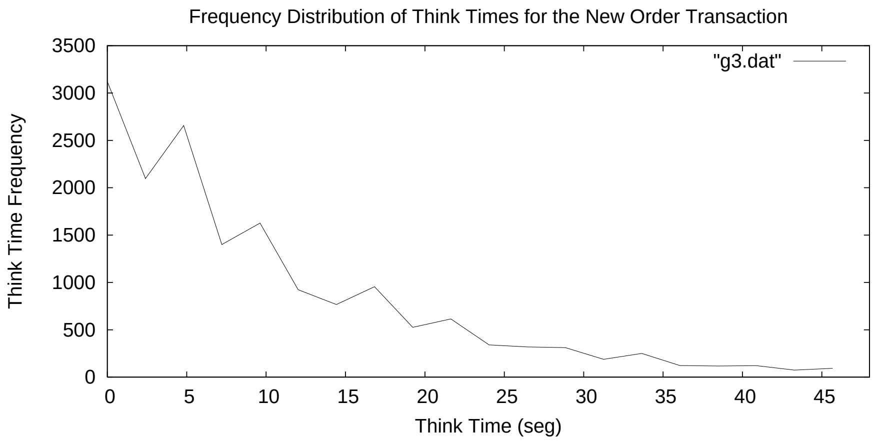

Figure 4.3: Example of plot generated according with Clause 5.6.3 of the TPCC benchmark.

```
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

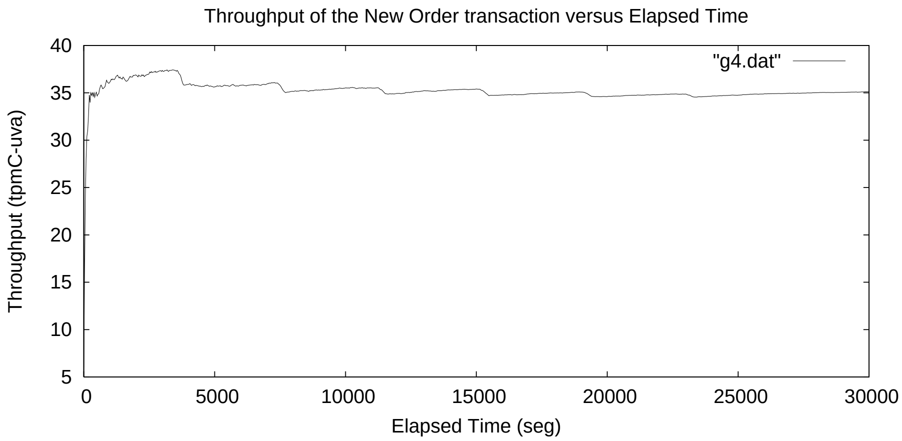

Figure 4.4: Example of plot generated according with Clause 5.6.4 of the TPCC benchmark.

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