

TPCx-V User's Guide

Version 1.0.1, 2/25/2016

[1.0 Overview](#)

[2.0 Setting Up the SUT Environment](#)

[2.0.1 Using the pre-configured OVF template](#)

[2.0.2 Create the User postgres](#)

[2.1 Create the Virtual Machines](#)

[2.2 Install the Operating System](#)

[2.3 Configure the Operating System](#)

[2.4 Install PostgreSQL](#)

[2.4.1 Clone Virtual Machines](#)

[2.5 Configure a Tier B Virtual Machine](#)

[2.5.1 Create Tablespace Directories](#)

[2.5.2 Setup PostgreSQL](#)

[2.5.3 Create the Flat Files](#)

[2.5.4 Create the TPCx-V Database](#)

[2.5.5 Back up the TPCx-V Database](#) [2.5.6 Restoring the TPCx-V Database](#)

[2.6 Configure a Tier A Virtual Machine](#)

[2.6.1 Setup the ODBC Environment](#)

[3.0 Setting Up the Driver Environment](#)

[3.1 Building the Benchmark Driver](#)

[3.2 Copying Benchmark Driver Files](#)

[3.3 Editing the Configuration File](#)

[4.0 Using the supplied bash scripts to run a test](#)

[5.0 Manually starting a Test Run](#)

[5.1 Starting the Component Processes](#)

[5.1.1 Starting the VConnector Process](#)

[5.1.2 Starting the VMee Process](#) [5.1.3](#)

[Starting the VCe Process](#)

[5.1.4 Starting the VDM Process](#)

[5.2 Starting the Prime Driver Process](#)

[6.0 Post-Run Reporting](#)

[6.1 Mix Log File Sorting](#)

[6.2 Mix Log File Merging](#)

[6.3 Transaction Mix and Audit Check Log File Creation](#)

[6.4 Executive Summary Creation](#)

[6.5 Advanced Reporter Features](#)

[7.0 TPCx-V Auditing](#)

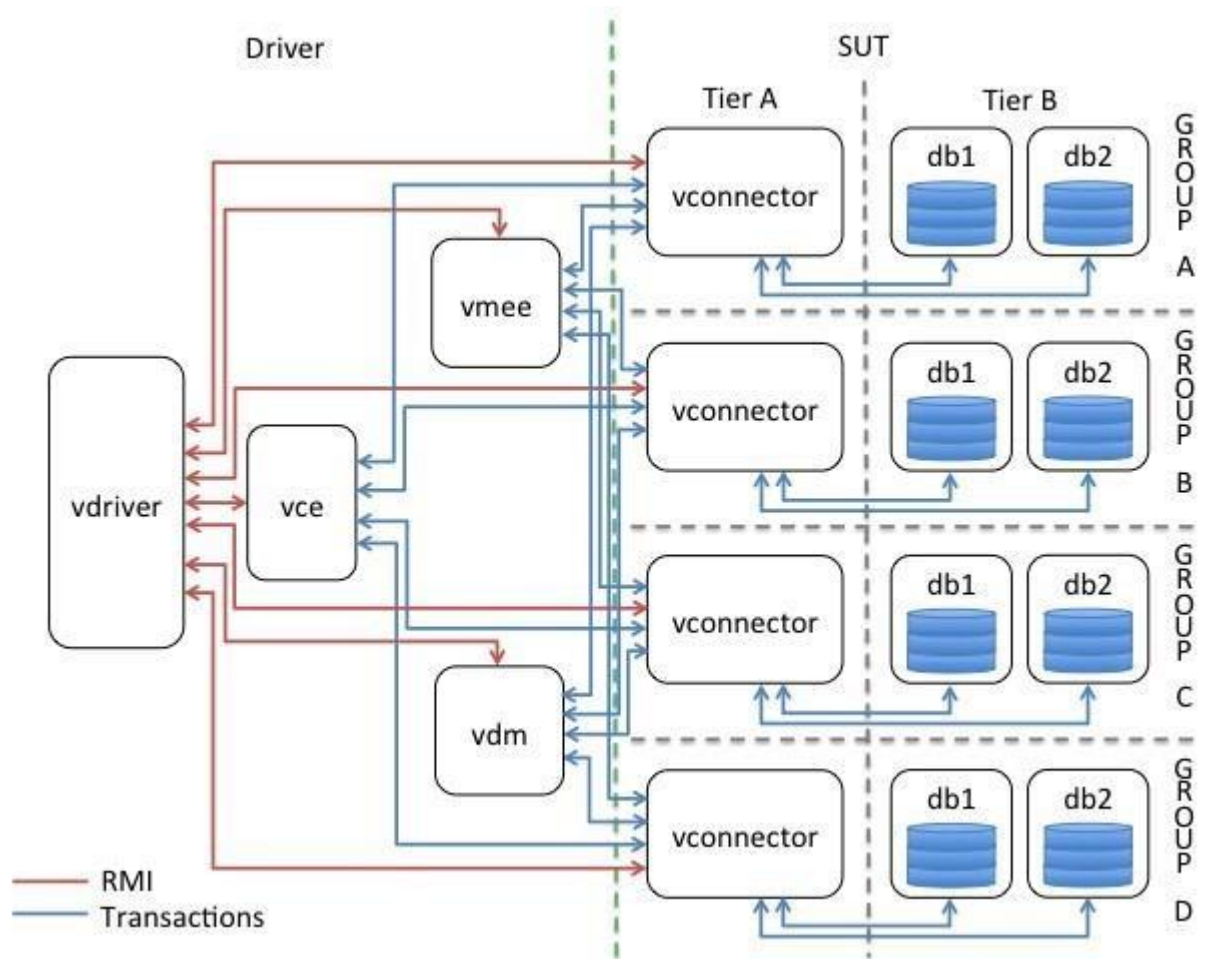
1.0 Overview

This document will help you set up one TPCx-V Tile using Virtual Machines running the following software configuration.

Operating System: RHEL 7 or CentOS 7

Database: PostgreSQL 9.3

The following figure represents all of the components of the benchmark driver and the corresponding network connections between them. This figure may be a useful visual reference as you work through these instructions. Note that this figure represents a System Under Test (SUT) with a single-tile test configuration using only a single driver. For each additional tile, an additional twelve SUT VMs, potentially additional vmee and vce processes, and corresponding connections to those SUT VMs and the vdriver prime driver would be required.

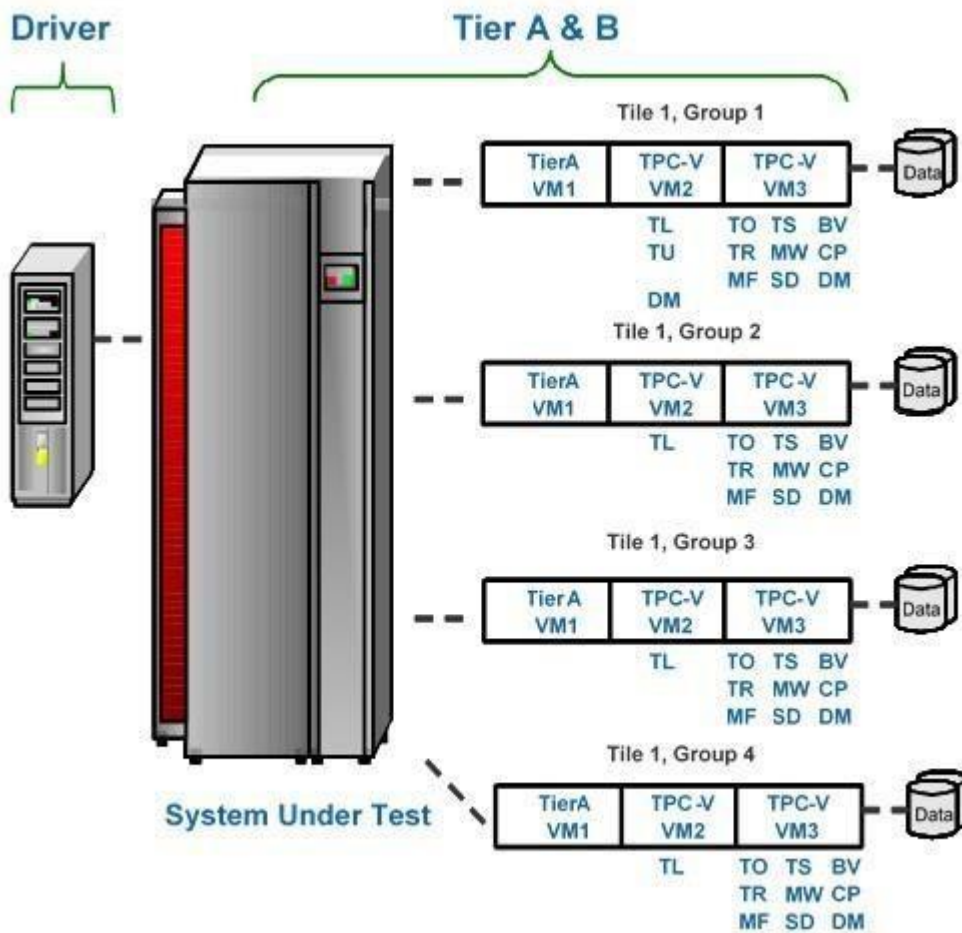


2.0 Setting up the SUT Environment

A general note: the TPCx-V benchmark started out life under the name "TPC-V", without the letter "x". The name of the database, names of some files in the benchmark structure, etc. still bear names such as tpcv, TPC-V, etc.

As you can see from the diagram in Section 1.0, a single-tile SUT configuration will consist of 12 different virtual machines. The instructions in this section will describe how to set up a single virtual machine. You can either run through the instructions 12 times to set up the various virtual machines or you can set up one Tier A virtual machine and one Tier B virtual machines using the instructions, clone those images as needed, and then make the necessary modifications to the cloned images to customize them for their purpose (unique hostname, unique ip address, secondary disk devices, etc).

The figure below show a sample 1-Tile SUT configuration, showing the 3 VMs per Tile, as well as the division of transaction types between the two Tier B VMs.



2.0.1 Using the OVF template

A preconfigured OVF template is available to download from the TPC website. This template has all the necessary software installed and configured. The root password of this VM is "tpc-v" (note: no letter "x" in the password). If you have downloaded and deployed this template, you can skip to section [2.4.1 Clone Virtual Machines](#)

2.0.2 Run as root or postgres?

Once the VMs are configured, most of the commands are run as user postgres. When we install the postgres software, we get a user postgres in /usr/passwords. But it does not have a password, and cannot be directly logged into. You can give it a password, or log into the VM as root, and then do a `su - postgres`

Unless otherwise noted, the commands in sections [2.1](#) to [2.5.2](#) are meant to be run as the user root.

2.1 Create the Virtual Machines

Create a virtual machine with the following minimum characteristics

- 2 virtual cpus
- 4GB memory
- 6GB primary disk device
- 1 network interface

2.1.1 Disk Space for Database Files

The 6GB of space in the root file system should be enough to install the operating system and all the software packages. Ordinarily, you should install all the software and finish configuring the first VM, then clone it into Tier A and Tier B VMs. At that point, you will need the extra space for the databases in the Tier B VMs.

The Tier B virtual machines will need additional disk storage for the database, backups, and flat files. The amount of storage needed will depend on the Group that will be running in the VM. The VM3 Tier B VMs will require slightly more space than VM2 Tier B VMs. Use the chart below to determine the extra disk space needed. The primary disk device of

Tier B VMs can be sized to include enough space for the database, backup, and flat files (the flat files may be created on a single VM and nfs-mounted on all other VMs), or other storage devices can be added to provide the needed size.

	VM2 Database Size (GB)	VM2 Backup Size (GB)	VM3 Database Size (GB)	VM3 Backup Size (GB)	Flat Files (GB)
--	------------------------	----------------------	------------------------	----------------------	-----------------

Group 1	37	3.8	38.5	3.8	60
Group 2	70.5	6.9	73.5	6.9	60
Group 3	105	10	110	10	60
Group 4	138	14	144	14	60

2.2 Install the Operating System

Install the 64-bit, **server** version of RHEL 7 or CentOS 7 in the virtual machine on the primary disk device of the VM. In the "INSTALLATION SUMMARY" window, click on "SOFTWARE SELECTION", and under "Base Environment", choose "Server with GUI". Under "Add-Ons for Selected Environment", choose the following:

1. Guest Agents/File and Storage Server + Network File System Client
2. Large Systems Performance
3. Performance Tools
4. Compatibility Libraries
5. Development Tools

Since you will install Java and PostgreSQL 9.3 in later steps, it is a good idea to **NOT** install the "PostgreSQL Database Server" or "Java Platform" Add-On packages during the Operating System installation to avoid having two possibly conflicting versions of the same software present on the VM.

2.3 Configure the Operating System

1. Increase the limits in /etc/security/limits.conf

```
* soft nproc 578528
* hard nproc 578528
* soft memlock 33554432 * hard memlock 33554432 postgres soft nproc
578528 postgres hard nproc 578528 postgres soft memlock 536870912
postgres hard memlock 536870912
```

2. Add these to /etc/sysctl.conf

```
kernel.sem = 500 64000 400 512
fs.file-max = 500000
vm.min_free_kbytes = 65536
kernel.shmall = 134217728
kernel.shmmax = 549755813888
kernel.shmmni = 4096 fs.file-max =
6815744 fs.aio-max-nr=1048576
kernel.core_uses_pid = 1
```

```
kernel.wake_balance = 0
kernel.perf_event_paranoid = -1
vm.swappiness = 0
```

3. Disable the firewall:

```
systemctl stop firewalld
systemctl disable firewalld
systemctl stop iptables
systemctl disable iptables
```

4. We need the Java 7 JDK (the kit hasn't been tested with Java 8). You can use the OpenJDK package that comes stock with RHEL7.

Alternatively, you can install Java from [Oracle Java](#), for example, 7u72 from [Oracle Java 7u72](#)

If you want to ensure that you are using the newly installed bits rather than OpenJDK, it is recommended that you change the system defaults (at least /usr/bin/java) to point to the newly-installed Oracle jdk with the following commands:

```
update-alternatives --auto java
update-alternatives --install /usr/bin/java java
/usr/java/default/bin/java 180051 \
--slave /usr/bin/keytool keytool /usr/java/default/bin/keytool \
--slave /usr/bin/orbd orbd /usr/java/default/bin/orbd \
--slave /usr/bin/pack200 pack200 /usr/java/default/bin/pack200 \
--slave /usr/bin/rmid rmid /usr/java/default/bin/rmid \
--slave /usr/bin/rmiregistry rmiregistry
/usr/java/default/bin/rmiregistry \
--slave /usr/bin/servertool servertool
/usr/java/default/bin/servertool \
--slave /usr/bin/tnameserv tnameserv /usr/java/default/bin/tnameserv \
--slave /usr/bin/unpack200 unpack200 /usr/java/default/bin/unpack200
\
--slave /usr/lib/jvm/jre jre /usr/java/default/jre \
--slave /usr/share/man/man1/java.1 java.1
/usr/java/default/man/man1/java.1 \
--slave /usr/java/default/man/man1/keytool.1 keytool.1
/usr/java/default/man/man1/keytool.1 \
--slave /usr/share/man/man1/orbd.1 orbd.1
/usr/java/default/man/man1/orbd.1 \
--slave /usr/share/man/man1/pack200.1 pack200.1
/usr/java/default/man/man1/pack200.1 \
--slave /usr/share/man/man1/rmid.1 rmid.1
/usr/java/default/man/man1/rmid.1 \
--slave /usr/share/man/man1/rmiregistry.1 rmiregistry.1
/usr/java/default/man/man1/rmiregistry.1 \
```

```

--slave /usr/share/man/man1/servertool.1 servertool.1
/usr/java/default/man/man1/servertool.1 \
--slave /usr/share/man/man1/tnameserv.1 tnameserv.1
/usr/java/default/man/man1/tnameserv.1 \
--slave /usr/share/man/man1/unpack200.1 unpack200.1
/usr/java/default/man/man1/unpack200.1

```

2.4 Install PostgreSQL

Install and configure PostgreSQL 9.3 in the virtual machine. The binaries and libraries will be installed in /usr/pgsql-9.3.

1. Install ODBC by downloading unixODBC-2.3.1-10.el7.x86_64.rpm and unixODBCdevel-2.3.1-10.el7.x86_64.rpm from rpmfind.net: `yum install unixODBC-2.3.1-10.el7.x86_64.rpm unixODBC-devel-2.3.1-10.el7.x86_64.rpm`
2. Download PostgreSQL
 1. Go to yum.postgresql.org/repopackages.php
 2. Click on the link for version 9.3 for "Red Hat Enterprise Linux 7 – x86_64" or "CentOS 7 - x86_64" depending on which OS you are using.
 3. Download the `pgdg-redhat93-9.3-1.noarch.rpm` or `pgdg-centos93-9.3-1.noarch.rpm` file depending on which OS you are using and put it on the VM in the /root directory
 4. Depending on OS, run either: `rpm -ivh pgdg-redhat93-9.3-1.noarch.rpm`
 or
`rpm -ivh pgdg-centos93-9.3-1.noarch.rpm`
 5. Get `postgresql93-odbc-09.03.0300-1PGDG.rhel7.x86_64.rpm` from postgresql.org
 6. Run: `rpm -ivh postgresql93-odbc-09.03.0300-1PGDG.rhel7.x86_64.rpm`
3. Run: `yum install postgresql93 postgresql93-libs postgresql93-devel postgresql93-server postgresql93-odbc`
4. You should now have a user *postgres* on the VM; if not, create it. Perform the following steps as user *postgres*
 1. Put the TPCx-V kit tar files `xVAudit.tgz`, `VGen.tgz`, `VDb.tgz`, and `VDriver.tgz` into the /opt directory
 2. Untar the kit files to their target locations
`tar xzvf /opt/VGen.tgz -directory=/opt`

```
tar xzvf /opt/VDB.tgz -directory=/opt
tar xzvf /opt/VDriver.tgz -
directory=/opt tar xzvf /opt/xVAudit.tgz
-directory=/opt chown -R postgres:postgres /opt/VDb
/opt/VDriver /opt/VGen /opt/xVAudit
```

3. Add the following lines to the end of `~postgres/.bash_profile` to set these values for future logins.

Note the setting for PGDATA to `/dbstore/tpcv-data`, which is used throughout this Guide. If you want to store the postgres data and configuration files in a different location, change PGDATA accordingly.

```
export PGHOME=/usr/pgsql-9.3 export
PATH=$PATH:$PGHOME/bin export
PGDATA=/dbstore/tpcv-data export
LD_LIBRARY_PATH=${PGHOME}/lib:/opt/VGen/lib:/usr/local/lib
export JAVA_INC=/usr/java/latest/include
```

4. We use the `pgstatspack` package to collect performance data. **As user postgres**, download the file `pgstatspack_version_2.3.1.tar.gz` from pgfoundry.org/, and simply untar it in postgres home directory

2.4.1 Clone virtual machines

At this point, your VM has all the required software. You can now clone it 12 (or 24, etc.) times for all the VMs in a TPCx-V configuration, and customize the Tier A and Tier B

VMs by following the instructions below

On the VMware vSphere platforms, you need to install the `deployPkg` plug-in as described in KB article 2075048. You can then clone VMs by simply providing the new VM's network name and IP address.

If you clone the Tier B VMs from a single source, remember to properly resize the virtual disk that holds the `/dbstore` file system, and run `resize2fs(8)` after powering up the VM.

You can use the `VDriver/scripts/cloneVMs.ps1` PowerCLI script to clone the VMs in a VMware vSphere environment. You will need to invest time in installing VMware PowerCLI on a Windows PC (or VM!), and customizing the script for your environment. That time will be well-spent as it will automate the cloning process, and will save a lot of time in the long run.

The run-time bash scripts depend on the prime driver being able to ssh into all 13 VMs, as both root and postgres, using the file `./ssh/authorized_keys`. It's a good idea to set this up on the first VM before making the clones `ssh-keygen #` You can simply hit the Return Key in response to all the

```
questions
ssh-copy-id root@nameofmyvm # Use the Network Name of your VM here
```



```
ssh-copy-id postgres@nameofmyvm
```

Test this by logging in as user postgres and running a simple command like "ssh

nameofmyvm date". If it asks for a password, run the following command as *root*:

```
restorecon -R -v /root/.ssh
```

2.5 Configure a Tier B Virtual Machine

Use the instructions in this section to finish configuring a VM for use in Tier B. Note that Tier B VMs require additional virtual disk space per table in section 2.1.1

2.5.1 Create Tablespace Directories

If a separate file system will be used to store the postgres data files, create it with:

```
mkfs.ext4 -F -v -t ext4 -T largefile4 -m 0 -j -O extent -O dir_index and  
mount it as /dbstore with mount arguments:  
nofail,noatime,nodiratime,nobarrier
```

Run the following commands on the Tier B VM:

```
mkdir -p /dbstore/tpcv-data mkdir  
/dbstore/tpcv-index mkdir /dbstore/tpcv-  
temp chown -R postgres:postgres /dbstore
```

2.5.2 Setup PostgreSQL

1. Since we have customized the location of PostgreSQL data files to */dbstore/tpcv-data*, we need to let the operating system know where to find these files

1. Create a */etc/systemd/system/postgresql-9.3.service* file with the following contents:

```
.include /lib/systemd/system/postgresql-9.3.service  
Environment=PGDATA=/dbstore/tpcv-data
```

2. `mkdir /etc/systemd/system/postgresql-9.3.service.d`

3. Create a */etc/systemd/system/postgresql-9.3.service.d/restart.conf* file with the following contents:

```
[Service]  
Environment=PGDATA=/dbstore/tpcv-data
```

2. Run the following commands on the Tier B VM to set up PostgreSQL:

```
systemctl enable postgresql-9.3.service
/usr/pgsql9.3/bin/postgresql93-setup initdb systemctl start
postgresql-9.3.service
```

3. Edit the \$PGDATA/pg_hba.conf file to make the connection security levels trust rather than peer and ident.
4. Update the postgresql.conf file (\$PGDATA/postgresql.conf) to have PostgreSQL listen on the local ip address and not just localhost. Update (may need to uncomment) the listen_addresses parameter and add the ip address of the server.

The following postgresql.conf changes are useful when loading the database

```
shared_buffers = 1024MB wal_sync_method =
open_datasync wal_writer_delay = 10ms
checkpoint_segments = 30
checkpoint_completion_target = 0.9
```

5. Restart the PostgreSQL service so the changes will take affect

```
systemctl restart postgresql-9.3.service
```

2.5.3 Create the Flat Files

Run this step as user postgres.

All Tier B VMs use the same flat files for populating the databases, even in a multi-Tile configuration. Each Group 1/2/3/4 loads some or all the flat files. So there is no need to generate identical flat files on all 8 database VMs and waste disk space. The 8 database VMs can nfs-mount /vgenstore from one VM that has generated the flat files. To create the file system for the flat files, run the following command as user root:

```
mkfs.ext4 -F -v -t ext4 -T largefile4 -m 0 -j -O extent -O dir_index
```

Mount the new file system as /vgenstore chown

```
postgres:postgres /vgenstore
```

Note: once the file system is created and mounted, all the rest of the following commands are executed as user postgres.

Build the VGen binaries by executing the following commands:

```
cd /opt/VGen/prj
cd GNUmake make cd
../..
```

The space for the flat files (60GB for the minimum-sized database) can be anywhere as long as all Tier B VMs can access it, and as long as the "-o" argument to VGenLoader and

the parameter flatfile_dir in VDb/pgsql/scripts/linux/env.sh are the same path. Here we assume the flat files are in /vgenstore/flat_out

Run the following commands (which are in the script /opt/VGen/scripts/create_TPCxV_flat_files.sh) to create one set of flat files that can be used for all TPCx-V Tiles and Groups.

```
#!/bin/bash cd /opt/VGen mkdir -p
/vgenstore/flat_out/Fixed mkdir -p
/vgenstore/flat_out/Scaling1 mkdir -p
/vgenstore/flat_out/Scaling2 mkdir -p
/vgenstore/flat_out/Scaling3 mkdir -p
/vgenstore/flat_out/Scaling4 bin/VGenLoader xf -c
20000 -t 20000 -w 125 -o
/vgenstore/flat_out/Fixed & bin/VGenLoader -xd -b
1 -c 5000 -t 20000 -w 125 -o
/vgenstore/flat_out/Scaling1 & bin/VGenLoader -xd -b
5001 -c 5000 -t 20000 -w 125 -o
/vgenstore/flat_out/Scaling2 & bin/VGenLoader -xd -b
10001 -c 5000 -t 20000 -w 125 -o
/vgenstore/flat_out/Scaling3 & bin/VGenLoader -xd -b
15001 -c 5000 -t 20000 -w 125 -o
/vgenstore/flat_out/Scaling4 & wait
```

Flat files created with the above script creates flat files for all four Groups with the minimum customer counts of

Group A: 5000
Group B: 10000
Group C: 15000 Group
D: 20000 and uses

60GB of disk space.

The database and backup sizes of the various groups of this size:

Group A: 35GB DB, 3.8GB backup (22 minutes to load, 17 minutes to create the backup)
(16 minute restore from/to flash)
Group B: 67GB DB, 6.9GB backup (48 minutes to load, 29 minutes to create the backup)
Group C: 100GB DB, 10GB backup (70 minutes to load, 40 minutes to create the backup)
Group D: 131GB DB, 14GB backup (98 minutes to load, 53 minutes to create the backup)

2.5.4 Create the TPCx-V database

1. Make sure the parameter `flatfile_dir` in `/opt/VDb/pgsql/scripts/linux/env.sh` points to where the flat files were loaded. Change the parameter `scaling_tables_dirs` in `/opt/VDb/pgsql/scripts/Linux/env.sh` to `"/Scaling[1-1]"`, `"/Scaling[1-2]"`, and `"/Scaling[1-3]"` for G1, G2, and G3 VMs, respectively.
2. On each Tier B VM, run the following to populate and set up the TPCx-V database:

```
cd /opt/VDb/pgsql/scripts/linux
./setup.sh
```

This phase will take about one hour when loading a default set of flat files. You will see some errors when we create the objects role and language, which already exist, or when we try to drop objects (tables and indexes) that were gone when we dropped the database. These errors are benign.

2.5.5 Back up the TPCx-V database

On each Tier B VM, run the following commands: `rm -rf /dbstore/backup; mkdir /dbstore/backup; pg_dump -j 4 -Fd tpcv -f /dbstore/backup -Z 9`

2.5.6 Restoring the TPCx-V database

On each Tier B VM, run the following commands: `pg_restore -j 4 -c --disabletriggers -d tpcv /dbstore/backup; cd /opt/VDb/pgsql/scripts/linux; ./analyze.sh`

2.6 Configure a Tier A Virtual Machine

Use the instructions in this section to finish configuring a VM for use in Tier A

2.6.1 Setup the ODBC environment

Run this step as user *root*.

This section only needs to be done on the Tier A virtual machines.

1. Populate the `odbc.ini` file from the kit

```
cat /opt/VDb/pgsql/prj/osfiles/odbc.ini >> /etc/odbc.ini
```

2. Edit `/etc/odbcinst.ini` and set

```
Driver64=/usr/pgsql-9.3/lib/psqlodbcw.so
```

3. Edit `/etc/odbc.ini` and set `ServerName` to the name of the two Tier B VMs that serve this Tier A. Note that you need to create two *Data Sources*, one that points to VM2, and one that points to VM3. You can name these Data Sources anything that you want as long as they match `VCONN_DSN_LABELS` in `vcfg.properties`; `PSQL2` and `PSQL3` are good choices.
4. Test out the TPCx-V and ODBC connections. First let's make sure the Tier B VM is reachable on the network, and the database is up and running. Do this by setting `Pghost` to the network name of a Tier B VM, then running a simple database query from the Tier A, such as the following command with its expected output:

```
-bash-4.2$ export Pghost=THE_NETWORK_NAME_OF_YOUR_TIERB bash-4.2$
psql tpcv -c "select count(*) from sector" count
-----
12
(1 row)
```

Now, check that we can reach the same database via ODBC. Do this by using the `tradesatus` test application in `/opt/VDb/pgsql/dml/test_programs`, which connects to the database on the datasource name `PSQL1` (if you have used `PSQL2` and `PSQL3` in `/etc/odbc.ini`, change replace the `PSQL1` string with `PSQL2` or `PSQL3`, and re-make the application).

```
cd /opt/VDb/pgsql/dml/test_programs make
tradesatus
./tradesatus
```

3.0 Setting Up the Driver Environment

Run this step as user `postgres`.

These instructions assume you have already used Section 2 of this document to set up four Tier A VMs, each connected to two Tier B database VMs, for a total of 12 VMs.

Optionally, you can create a 13th clone to be the prime driver. Although this is not required, if done, it will simplify the process since the prime driver will immediately have all the required software installed.

These instructions are intended to assist the user in setting up the TPCx-V prime driver, all subordinate drivers, market exchange emulators, and database connectors used to generate the database load. They also assume you have read the TPCx-V Benchmark Kit Design Document, Clause 10 of the benchmark Specification, and are familiar with concepts and terminology introduced there.

3.1 Building the Benchmark Driver

The following build instructions assume you are building the driver on Linux using GNU C and the Oracle Java Development Kit. Other environments may use different build scripts and build steps, though the fundamental build process should be very similar.

1. Copy the benchmark driver kit file(s) first to your prime driver in the desired directory.
 2. If a Java Development Kit has not already been installed, install one on the prime driver in the desired directory. (Adding a soft link called "java" in the /usr directory that points to the Java development kit's base directory will reduce the editing requirements in the Makefile.)
 3. From the base directory of the benchmark driver kit on the prime driver, go to the VDriver/prj/GNUMake directory.
 4. Modify the included Makefile as appropriate for your environment. Most fields should require no modification. However, be certain that JAVA_HOME points to the correct base directory for your Java development kit (JDK). Likewise, confirm that the C++ compiler and flags are correct for your environment.
 5. Run "make" from the GNUMake directory to build the benchmark driver.
- When the benchmark driver has built successfully, you should find a file named "vdriver.zip" in the /opt/VDriver/dist directory.

3.2 Copying Benchmark Driver Files

The vdriver.zip file contains all of the necessary benchmark driver files. You can copy the zip file to the desired directory on each Tier A SUT VM, as well as to each system that will be used as a CE driver and Market Exchange Emulator, and extract all files to each system. For example, copy vdriver.zip from the /opt/VDriver/dist directory of the prime driver to the /opt/VDriver/dist directory of each of the 12 VMs, then on each VM run the following commands:

```
cd /opt/VDriver/jar    unzip -o ../dist/vdriver.zip -x  
vcfg.properties
```

However, the following instructions detail how to copy only the necessary files for each component.

1. Unzip vdriver.zip on the prime driver (or any other system from which you can copy the extracted files to all of the target systems).
2. Copy vconnector.jar and the "lib" directory to the desired directory on all Tier A VMs.
3. Copy vmee.jar and the "lib" directory to the desired directory on all systems that will be used as market exchange emulators.
4. Copy vdm.jar and the "lib" directory to the desired directory on the system that will be used to drive the data maintenance transactions.

5. Copy vce.jar and the "lib" directory to the desired directory on all systems that will be used as customer emulators.
6. If the system on which vdriver.zip was extracted is not the system that will be used as the prime driver or is not in the desired location on the prime driver, then copy vdriver.jar, reporter.jar, the "lib" directory, vcfg.properties, and testbed.properties to the desired directory on the prime driver.

Finally, confirm that you have a Java runtime environment (JRE) or development kit (JDK) installed on all systems that will be running these benchmark driver components, and that the installed version is the same major revision level or newer than the one used to build the benchmark driver.

3.3 Editing the Configuration File

The configuration file (vcfg.properties) is where the benchmark run configuration is defined. This section deals only with the most commonly modified parameters. See Appendix A for details of configuration parameters not explained here.

VM_TILES

Set this to the number of Tiles being used for the test. (These setup instructions assume a single-Tile configuration.)

NUM_DRIVER_HOSTS Set this value to the number of CE driver processes used in a test.

NUM_CE_DRIVERS

NUM_CE_DRIVERS[x]

Modify NUM_CE_DRIVERS in order to change the total number of CE driver *threads* used to generate load in a test. Note that these threads are evenly distributed between all CE driver processes. In order to override this default distribution of threads, use the indexed version to specify how many driver threads are run in each driver process.

NUM_MKT_EXCHANGES

Set this value to the number of *Tiles* of Tier A VMs. There is a single MEE process for each set of four Tier A VMs.

VDRIVER_RMI_HOST

VDRIVER_RMI_PORT

Set these values to the network interface hostname and port on which the prime driver will listen for RMI commands.

VCE_RMI_HOST[x]

VCE_RMI_PORT[x]

Set these values to the network interface hostname and port on which each vce client process will listen for RMI commands. Note that any vce processes not defined in this way will default to the value assigned to the non-indexed form of these parameters.

VCONN_RMI_HOST[tile][group]

VCONN_RMI_PORT[tile][group]

Set these values to the network interface hostname and port on which each vconnector process will listen for RMI commands. Note that any vconnector processes not defined in this way will default to the value assigned to the non-indexed form of these parameters.

VCONN_TXN_HOST[tile][group]

VCONN_TXN_PORT[tile][group]

Set these values to the network interface hostname and port on which each vconnector process will accept transactions from the vce or vmee processes. Note that any vconnector processes not defined in this way will default to the value assigned to the non-indexed form of these parameters.

VCONN_MEE_INDEX[tile][group]

Set this to the index of the vmee process used for that set of Tier A VMs. Note that this value must be identical for all vconnector processes in the same Tile. Effectively, this value corresponds to the Tile index for a group of four Tier A VMs.

VCONN_DSN_LABELS[tile][group]

Set this to the label strings for the databases used, as defined in your Tier A VM's odbc.ini file, with each label delimited by a comma. There must be as many label names as Tier B databases in use.

VCONN_NUM_DBS[tile][group]

Set this to the number of databases being used by the Tier A VM of the same Tile and Group indexes.

MEE_RMI_HOST[tile]

MEE_RMI_PORT[tile]

Set these values to the network interface hostname and port on which each vmee process will listen for RMI commands. Note that any vmee processes not defined in this way will default to the value assigned to the non-indexed form of these parameters.

MEE_TXN_HOST[tile][group]

MEE_TXN_PORT[tile][group]

Set these values to the network interface hostname and port on which each vmee process will accept transactions from VGen. Four of these pairs must be defined; one for each Tier A VM in each Group. Any vmee processes used but not defined in this way will default to the value assigned to the non-indexed form of these parameters.

VDM_RMI_HOST

VDM_RMI_PORT

Set these values to the network interface hostname and port on which the vdm process will listen for RMI commands.

RUN_ITERATION_SEC

Set this value to the number of seconds that a measurement interval to run. (Modifying this value from the default will result in a non-compliant run.)

WARMUP_SEC

RAMPUP_SEC

RAMPDN_SEC

Set these values to the number of seconds of ramp-up and ramp-down at the beginning and end of an entire test run, and the number of seconds of warm-up at the beginning of the *first* iteration of the test run.

NUM_RUN_PHASES

Set this to the number of run phases you wish to run in a single measurement interval. There must be a GROUP_PCT_DIST_PHASE defined for as many run phases as specified. (Modifying this value from the default will result in a non-compliant run.)

NUM_RUN_ITERATIONS

Set this to the number of times to run a full set of run phases.

4.0 Using the supplied bash scripts to run a test

Run this step as user postgres.

There are a number of shell scripts in the directory `/opt/VDriver/scripts/rhel6` that facilitate the running of a test and post processing the results.

`runme.sh` is the main script. It starts the component processes on all 13 VMs, runs the test, post processes the logs to produce results all the way to a TPCx-V Executive Summary, and can optionally collect stats, run the consistency tests, and run the Trade-Cleanup transaction before and after the run. The results of each run are saved in the directory `/opt/VDriver/results/RUN_ID` with a unique `RUN_ID` for each run.

`iostat_summary.sh` averages the `iostat` output of each VM over the Measurement Interval, and prints a line for each database (virtual) disk of each Tier B VM.

`killrun.sh` logs into all 13 VMs, and kills the benchmark processes. Note that `killrun.sh` gets the network names of the VMs for a given run from the file `/opt/VDriver/results/RUN_ID/allhosts`, which is created by `runme.sh`. If for some reason this file is not created (e.g., you hit CTRL-C before `runme.sh` had a chance to create `allhosts`; or because of syntax issues in `vcfg.properties`), `killrun.sh` won't work, and you will need to manually log

into each VM, and kill the benchmark processes. `summary.sh` gives a 1-line summary of the performance for a given run number.

`trade-cleanup.sh` runs the Trade-Cleanup transaction for the last completed run. Alternatively, the "-t" option to `runme.sh` will automatically run Trade-Cleanup after every run.

`collect_config.sh`, `tools.sh`, `finish_tools.sh`, `killlocal.sh`, and `run_consistency.sh` are auxiliary bash scripts that get invoked by other scripts.

5.0 Manually starting a Test Run

If instead of using the supplied bash scripts, you prefer to manually launch individual component processes, follow the instructions below.

5.1 Starting the Component Processes

As the diagram, above, illustrates, the `vconnector`, `vmee`, and `vce` processes all receive RMI calls from the prime driver, and as such, the hostname of the network interface on which the user intends to communicate with the prime driver must be specified when invoking these processes. Likewise, unless the user intends to listen on the default RMI network port for these RMI calls, the network port must also be specified when invoking these processes.

5.1.1 Starting the VConnector Process

Go to the directory in which the `vconnector.jar` file is installed on each Tier A VM and start each `vconnector` process using the following invocation string format:

```
java -jar [-Djava.library.path=<path_to_c_libs>] vconnector.jar  
-rh <hostname> [-rp <port_num>]
```

An example invocation string:

```
java -jar -Djava.library.path=lib vconnector.jar -rh tpcv_vm1 rp  
30000
```

5.1.2 Starting the VMee Process

Go to the directory in which the `vmee.jar` file is installed on each system being used as a market exchange emulator and start each `vmee` process using the following invocation string format:

```
java -jar [-Djava.library.path=<path_to_c_libs>] vmee.jar -rh  
<hostname> [-rp <port_num>]
```

An example invocation string:

```
java -jar -Djava.library.path=lib vmee.jar -rh tpcv_meel -rp  
32000
```

Be certain to specify different RMI ports for each vmee process on the same system and using the same network interface.

5.1.3 Starting the VCe Process

Go to the directory in which the vce jar file is installed on each system being used as a customer emulator and start each vce process using the following invocation string format:
java -jar [-Djava.library.path=<path_to_c_libs>] vce.jar -rh
<hostname> [-rp <port_num>]

An example invocation string:

```
java -jar -Djava.library.path=lib vce.jar -rh tpcv_cel -rp  
31000
```

Be certain to specify different RMI ports for each vce process on the same system and using the same network interface.

5.1.4 Starting the VDM Process

Go to the directory in which the vdm jar file is installed on the system being used as a data maintenance transaction driver and start the vdm process using the following invocation string format:

```
java -jar [-Djava.library.path=<path_to_c_libs>] vdm.jar -rh  
<hostname> [-rp <port_num>]
```

An example invocation string:

```
java -jar -Djava.library.path=lib vdm.jar -rh tpcv_dm1 -rp 34000
```

Once all of these driver components are started, they are listening on their RMI ports and waiting for the prime driver to send them the needed configuration information and to start the test run.

5.2 Starting the Prime Driver Process

After confirming the values in the configuration file are complete and correct for the desired run characteristics, go to the directory in which the vdriver jar file is installed on the prime driver and start the vdriver process using the following invocation string format:

```
java -jar [-Djava.library.path=<path_to_c_libs>] vdriver.jar  
-i <flat_file_location>
```

An example invocation string:

```
java -jar -Djava.library.path=lib vdriver.jar -i  
/opt/tpcv/VGen/flat_in
```

This will start the test run, and all vce, vmee, and vconnector processes, as well as the vdriver process should terminate at the end of the run.

6.0 Post-Run Reporting

The `runme.sh` script will automatically invoke the `reporter.jar` application multiple times to sort the log files, merge them into a single log file, produce an ASCII file with the required performance and compliance data, and finally generate an Executive Summary. (Run this step as user `postgres`.)

Alternatively, you can follow the instructions below to invoke `reporter.jar` to post-process the logs.

During the measurement interval, each `vce` and `vmee` process records the response time for each request type on a *per-phase* basis in a transaction mix log. At the end of the run, the log files from all of these `vce` and `vmee` processes must be pre-sorted and then merged into a single mix log file using the reporter. (Note that if you set `SORT_MIX_LOGS = "1"` in `vcfg.properties`, the `vce` and `vmee` processes will sort their log files by timestamp at the end of the run. Otherwise, it must be done by the reporter.)

The following instructions assume you have collected all of the `vce` and `vmee` log files and placed them in the same directory as the `reporter.jar` file. They also assume that there is a "results" subdirectory that includes the timestamped subdirectories for each run, and in which subdirectory exists a `runtime.properties` file that includes the specific configuration information for the run that the reporter is intended to process.

There are four flags that you may need to use when invoking the reporter:

- `-s`: this flag tells the reporter that the file or files that it will process need to be *sorted* by timestamp
- `-m`: this flag tells the reporter to *merge* all sorted mix files into a single, time-sorted file
- `-r`: this flag tells the reporter to write a final transaction rate *report* based on the log entries from the single, merged, time-sorted file
- `-es`: this flag tells the reporter to write an HTML-formatted *Executive Summary* report based on the log entries from the single, merged, time-sorted file as well as the information contained in the `testbed.properties` file

If you pass the reporter none of these flags, it expects that you have also passed it the name of a single, sorted, merged log file that only requires writing a final transaction rate report. Otherwise, you should pass it the combination of these four flags that reflects the tasks you would like the reporter to perform.

The reporter process is invoked using the following format:

```
java -jar [-Djava.library.path=<path_to_c_libs>] reporter.jar
-c <path_to_runtime.properties> [-t
<path_to_testbed.properties>] [-h] [-s ] [-m] [-r] [-es] [-o
```

```
<merged_log_name/path>] [-i  
<desired_sample_interval_seconds>] [-si  
<measurement_starting_iteration>] [-sp  
<measurement_starting_phase>] [-mp  
<total_measurement_phases>] <mix_log_file_name(s)>
```

6.1 Mix Log File Sorting

In order to process the mix log files from the CE and MEE processes, these files have to be sorted (by timestamp), and then merged. The following is an example reporter invocation string that might be used for sorting a single mix log file:

```
java -jar -Djava.library.path=lib reporter.jar -c  
results/20151118-142511/runtime.properties -s CE_Mix-0.log
```

This would create a new log file named CE_Mix-0-sorted.log that is ready to be merged with any other pre-sorted logs from the same run.

6.2 Mix Log File Merging

Once all of your CE and MEE mix logs are sorted, they are ready to be merged into a single, combined mix log file. The following is an example reporter invocation string that might be used for merging multiple mix log files:

```
java -jar -Djava.library.path=lib reporter.jar -c  
results/20151118-142511/runtime.properties -m -o  
mixlogmerged.log CE_Mix-0-sorted.log CE_Mix-1-sorted.log  
MEE_Mix-0sorted.log MEE_Mix-1-sorted.log
```

This would merge the four (pre-sorted) mix logs into a single sorted and merged log file named mixlog-merged.log. (A single, combined mix log file is required for creating the transaction mix validation and audit check log files as well as an HTML-formatted Executive Summary document.)

6.3 Transaction Mix and Audit Check Log File Creation

Once you have a single sorted and merged log file to use as input, you can then create the transaction mix validation and audit check log files by using a reporter invocation string similar to the following:

```
java -jar -Djava.library.path=lib reporter.jar -c  
results/20151118-142511/runtime.properties -i 30  
mixlogmerged.log
```

Note that the "-i 30" was not actually required because 30 seconds is the default time sampling interval. Additionally, the "-r" flag was not used in this invocation. That flag is only needed when including the these log files' creation with the previous merging step. But if you provide the reporter with a single, pre-sorted and pre-merged mix log file, the creation of this performance log will occur with or without this flag's inclusion in the invocation string. Note also that the reporter will try to write these log files to the same directory as the directory in which the driver wrote the runtime.properties file.

6.4 Executive Summary Creation

The HTML-formatted Executive Summary is created using a similar format to that of the performance summary log, above. In fact, the performance summary log is created as a part of the Executive Summary creation, making it unnecessary to create the former prior to creating the Executive Summary pages. However, the Executive Summary requires additional details about the hardware and software configuration used for the test, as well as component pricing information. So the user must also fill out the testbed.properties file, and optionally, the sample_pricing.xls file as an easier alternative method for providing pricing data.

If you choose to use the provided sample_pricing.xls file to fill in the test system's pricing data, after filling in the information required, you will need to save the file in csv format, because the reporter expects a csv-formatted file if a pricing file is passed in separately. You can alternatively choose to provide the pricing information in the testbed.properties file. You will find the pricing section commented out in this file. (This is considerably more difficult, and therefore prone to error, and is therefore commented out by default.)

Pass these files' names and locations to the reporter using the -t and -p flags, respectively, when invoking it to create an Executive Summary. The reporter invocation string would look similar to the following in format:

```
java -jar -Djava.library.path=lib reporter.jar -c
results/20151118-142511/runtime.properties -t
testbed.properties -p pricing.csv mixlog-merged.log
```

If unsure how to fill out the testbed.properties file, the user can first invoke the reporter using the sample testbed.properties file provided with the kit, and the resulting Executive Summary Report should help clarify where and how these properties are being used in the report.

6.5 Advanced Reporter Features

Up to here, all of the reporter instructions have implicitly been assuming that the test that was run was a single iteration of ten phases. However, neither the benchmark kit nor the reporter is so limited. If the user runs multiple iterations of the 10-phase test, any ten consecutive phases may be selected as the measurement interval to use for the report (or even greater or fewer than ten phases for a non-compliant test).

To use this reporter functionality the user will include one or more of the following flags when invoking the reporter:

- `-si <starting_iteration>`: This flag is used to specify the iteration that is the beginning of the measurement interval chosen. (Default: 0)
- `-sp <starting_phase>`: This flag is used to specify the phase number that is to be the first phase of the measurement interval chosen. (Default: 0)
- `-mp <total_phases>`: This flag is used to specify the total number of phases that are to be included in the measurement interval. (Default: 10; note that any number other than 10 will make this measurement interval non-compliant)

As an example, assume that a test is run with two iterations of ten phases, and it seems as though the user may have used too short of a warmup time, because prior to the first phase of the first iteration the log data suggests that the system has still not reached a steady state condition. So the user wants to create a report starting at the *second* phase of the first iteration and continue through the first phase of the second iteration. The reporter invocation would look similar to:

```
java -jar -Djava.library.path=lib reporter.jar -c
results/20151118-142511/runtime.properties -t
testbed.properties -si 0 -sp 1 mixlog-merged.log
```

Note that the `-mp` flag was not included because the default value is being used (and must be used for a compliant run). Similarly, because the default starting interval is also 0, the `-si` flag could have been left out, as well.

7.0 TPCx-V Auditing

One of the components of the TPCx-V benchmark kit is the self-audit module. A test sponsor can execute the various tests implemented in the xVAudit application, and provide the outputs to the auditor. The file `/opt/xVAudit/doc/README` lists these tests along with short descriptions of them.

The xVAudit application executes the audit tests on all the databases in the testbed, so it needs to know about all these databases, and know how to access them. That information is collected during the run by the `runme.sh` script, and is deposited in the file `tierb.properties` in the results directory of the run. The `tierb.properties` file should be passed to the xVAudit application with the `-c` option.

The xVAudit application needs a jdbc connection jar, such as `postgresql-9.4.1201.jdbc41.jar` (or newer), which you can download from jdbc.postgresql.org.