
The science behind the report:

# Open up new possibilities with higher transactional database performance from Amazon EC2 R7i instances with 4th Gen Intel Xeon Scalable processors

This document describes what we tested, how we tested, and what we found. To learn how these facts translate into real-world benefits, read the report Open up new possibilities with higher transactional database performance from Amazon EC2 R7i instances with 4th Gen Intel Xeon Scalable processors.

We concluded our hands-on testing on January 26, 2024. During testing, we determined the appropriate hardware and software configurations and applied updates as they became available. The results in this report reflect configurations that we finalized on January 25, 2024 or earlier. Unavoidably, these configurations may not represent the latest versions available when this report appears.

## Our results

To learn more about how we have calculated the wins in this report, go to http://facts.pt/calculating-and-highlighting-wins. Unless we state otherwise, we have followed the rules and principles we outline in that document.

Table 1: Results of our testing.

| Instance | CPU | Storage | Database Warehouses | NOPM | % increase |
|---|---|---|---|---|---|
| 4vCPU | | | | | |
| r7i.xlarge | 8488C | 40GB gp3 SSD | 150 | 173,063 | 13.56% |
| r6i.xlarge | 8375C | 40GB gp3 SSD | 150 | 152,385 | - |
| 16vCPU | | | | | |
| r7i.4xlarge | 8488C | 150GB io1 SSD | 600 | 468,374 | 13.85% |
| r6i.4xlarge | 8375C | 150GB io1 SSD | 600 | 411,370 | - |
| 64vCPU | | | | | |
| r7i.16xlarge | 8488C | 500GB io1 SSD | 2400 | 1,471,049 | 8.57% |
| r6i.16xlarge | 8375C | 500GB io1 SSD | 2400 | 1,354,852 | - |

This project was commissioned by Intel.

# System configuration information

Table 2: Detailed information on the systems we tested.

| System configuration information | r6i.xlarge | r6i.4xlarge | r6i.16xlarge |
|---|---|---|---|
| Tested by | Principled Technologies | Principled Technologies | Principled Technologies |
| Date testing ended | 01/26/2024 | 01/24/2024 | 01/23/2024 |
| Cloud service provider (CSP) | AWS | AWS | AWS |
| Region | us-east-1d | us-east-1d | us-east-1d |
| Workload and version | HammerDB v4.9 TPROC-C | HammerDB v4.9 TPROC-C | HammerDB v4.9 TPROC-C |
| Workload-specific parameters | 150 warehouses, 32 virtual users, pg_storedprocs=true, 5 minutes warmup, 10 minutes run time | 600 warehouses, 144 virtual users, pg_storedprocs=true, 5 minutes warmup, 10 minutes run time | 2400 warehouses, 112 virtual users, pg_storedprocs=true, 5 minutes warmup, 10 minutes run time |
| Iterations and result choice | 3 runs, median | 3 runs, median | 3 runs, median |
| Server platform | r6i.xlarge | r6i.4xlarge | r6i.16xlarge |
| SQL version | PostgreSQL 16 | PostgreSQL 16 | PostgreSQL 16 |
| Operating system name and version/build number | Ubuntu 22.04 LTS | Ubuntu 22.04 LTS | Ubuntu 22.04 LTS |
| Date of last OS updates/ patches applied | 01/23/2024 | 01/23/2024 | 01/23/2024 |
| Processor | | | |
| Number of vCPU | 4 | 16 | 64 |
| Vendor and model | Intel® Xeon® Platinum 8375C | Intel Xeon Platinum 8375C | Intel Xeon Platinum 8375C |
| Core count (per processor) | 32 | 32 | 32 |
| Core frequency (GHz) | 2.90 | 2.90 | 2.90 |
| Stepping | 6 | 6 | 6 |
| Hyper-threading | Yes | Yes | Yes |
| Turbo | Yes | Yes | Yes |
| Memory module(s) | | | |
| Total memory in system (GB) | 32 | 128 | 512 |
| Local storage | | | |
| OS | | | |
| Number of drives | 1 | 1 | 1 |
| Drive size (GB) | 20 | 20 | 20 |
| Drive information (speed, interface, type) | gp3, EBS, 125/3000 IOPS | gp3, EBS, 125/3000 IOPS | gp3, EBS, 125/3000 IOPS |
| Data drive | | | |
| Number of drives | 1 | 1 | 1 |
| Drive size (GB) | 40 | 150 | 500 |
| Drive information (speed, interface, type) | gp3, EBS, 125/16000 IOPS | io1, EBS, 7500 IOPS | io1, EBS, 25000 IOPS |

Table 3: Detailed information on the systems we tested.

| System configuration information | r7i.xlarge | r7i.4xlarge | r7i.16xlarge |
|---|---|---|---|
| Tested by | Principled Technologies | Principled Technologies | Principled Technologies |
| Date testing ended | 01/26/2024 | 01/24/2024 | 01/23/2024 |
| Cloud service provider (CSP) | AWS | AWS | AWS |
| Region | us-east-1d | us-east-1d | us-east-1d |
| Workload and version | HammerDB v4.9 TPROC-C | HammerDB v4.9 TPROC-C | HammerDB v4.9 TPROC-C |
| Workload-specific parameters | 150 warehouses, 32 virtual users, pg_storedprocs=true, 5 minutes warmup, 10 minutes run time | 600 warehouses, 144 virtual users, pg_storedprocs=true, 5 minutes warmup, 10 minutes run time | 2400 warehouses, 128 virtual users, pg_storedprocs=true, 5 minutes warmup, 10 minutes run time |
| Iterations and result choice | 3 runs, median | 3 runs, median | 3 runs, median |
| Server platform | r7i.xlarge | r7i.4xlarge | r7i.16xlarge |
| SQL version | PostgreSQL 16 | PostgreSQL 16 | PostgreSQL 16 |
| Operating system name and version/build number | Ubuntu 22.04 LTS | Ubuntu 22.04 LTS | Ubuntu 22.04 LTS |
| Date of last OS updates/ patches applied | 01/23/2024 | 01/23/2024 | 01/23/2024 |
| Processor | | | |
| Number of vCPU | 4 | 16 | 64 |
| Vendor and model | Intel Xeon Platinum 8488C | Intel Xeon Platinum 8488C | Intel Xeon Platinum 8488C |
| Core count (per processor) | 48 | 48 | 48 |
| Core frequency (GHz) | 2.40 | 2.40 | 2.40 |
| Stepping | 8 | 8 | 8 |
| Hyper-threading | Yes | Yes | Yes |
| Turbo | Yes | Yes | Yes |
| Memory module(s) | | | |
| Total memory in system (GB) | 32 | 128 | 512 |
| Local storage | | | |
| OS | | | |
| Number of drives | 1 | 1 | 1 |
| Drive size (GB) | 20 | 20 | 20 |
| Drive information (speed, interface, type) | gp3, EBS, 125/3000 IOPS | gp3, EBS, 125/3000 IOPS | gp3, EBS, 125/3000 IOPS |
| Data drive | | | |
| Number of drives | 1 | 1 | 1 |
| Drive size (GB) | 40 | 150 | 500 |
| Drive information (speed, interface, type) | gp3, EBS, 125/16000 IOPS | io1, EBS, 7500 IOPS | io1, EBS, 25000 IOPS |

# How we tested

## Testing overview

We tested two types of Amazon Web Services (AWS) EC2 instances: R7i instances with 4th Gen Intel Xeon Scalable processors and R6i instances with 3rd Gen Intel Xeon Scalable processors. We ran a TPROC-C workload on PostgreSQL on the AWS instances to show the performance increase in terms of new orders per minute on OLTP databases that you could expect to see using the newer instance series vs. the older. Note that we sized the PostgresSQL databases to fit in RAM in order to avoid a disk bottleneck and show the performance differences in the CPUs. Your results may vary.

## Creating the HammerDB client instance

1. Log into AWS and navigate to the AWS Management Console.
2. Click EC2.
3. Click Launch instance to open the Launch an instance wizard.
4. Enter a name and any desired tags.
5. Under Quick Start, click the Ubuntu button.
6. Under Amazon Machine Image (AMI), select Ubuntu Server 22.04 LTS (HVM), SSD Volume Type.
7. Keep the Architecture set to x86.
8. Under Instance type, select m6i.2xlarge.
9. Under key pair (login), click Create key pair:
   - Enter a key pair name.
   - Leave RSA and .pem selected and click Create key pair.
10. Next to Network settings, click Edit, and set the following:
    - Leave the default VPC selected.
    - Choose your subnet. We selected us-east-1d.
    - Leave Auto-assign public IP enabled.
    - Under Firewall (security groups), click Create security group.
    - Name the group and give it a description.
    - Ensure that there is an inbound rule for SSH over TCP on port 22.
11. Under Configure storage, set the root volume to 20GiB on gp3 storage.
12. Click Launch instance.

## Editing the Security Group inbound rules

1. Log into AWS and navigate to the AWS Management Console.
2. Click EC2.
3. In the left pane, select Security Groups.
4. Select your newly created Security Group.
5. Under the Inbound rules tab, click Edit inbound rules.
6. Click Add rule, and set the following:
   - Type: All traffic
   - Source: In the search bar, enter your security group name, and select it
7. Click Save rules.

## Creating the PostgreSQL instance

1. Log into AWS, and navigate to the AWS Management Console.
2. Click EC2.
3. Click Launch instance to open the Launch an instance wizard.
4. Enter a name and any desired tags.
5. Under Quick Start, click the Ubuntu button.
6. Under Amazon Machine Image (AMI), select Ubuntu Server 22.04 LTS (HVM), SSD Volume Type.
7. Keep the Architecture set to x86.
8. Under Instance type, select {r7i,r6i}.{xlarge,4xlarge,16xlarge}.
9. Under key pair (login), select the key pair you previously created.
10. Next to Network settings, click Edit, and set the following:
    - Leave the default VPC selected.
    - Choose your subnet. We selected us-east-1d.
    - Leave Auto-assign public IP enabled.
    - Under Firewall (security groups), select the security group you previously created.
11. Next to Configure storage, click Advanced.
12. For Volume 1 (AMI Root), set the following:
    - Size: 20GiB
    - Volume type: gp3
    - IOPS: default
    - Delete on Termination: Checked
    - Encryption: Not encrypted
    - Throughput: default
13. Click Add new volume and set the following:
    - Size: {40GiB,150Gib,500GiB}
    - Volume type: {gp3,io1,io1}
    - IOPS: {16000,7500,25000}
    - Delete on Termination: Checked
    - Encryption: Not encrypted
    - Throughput: {1000,N/A,N/A}
14. Click Launch instance.

## Configuring Ubuntu 22.04 LTS and installing PostgreSQL on the PostgreSQL instance

1. Log into the PostgreSQL instance via SSH.
2. Change GRUB huge page size, and reboot the instance:

```
sudo sed -i 's/\(GRUB_CMDLINE_LINUX="\)/\1default_hugepagesz=1G hugepagesz=1G /' /etc/default/grub
sudo update-grub
sudo reboot
```

3. Run the appropriate postgresql_host_prepare.sh script depending on your instance size. Example for 4-vCPU instance:

```
sudo ./postgresql_host_prepare_4vcpu.sh
```

4. Add the repository entry for PostgreSQL to the package list:

```
sudo sh -c 'echo "deb https://apt.postgresql.org/pub/repos/apt $(lsb_release -cs)-pgdg main" > /etc/
apt/sources.list.d/pgdg.list'
```

5. Import the repository signing key:

```
wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add -
```

6. Update the package lists, and install PostgreSQL:

```
sudo apt-get update
sudo apt-get -y install postgresql-16
```

7. Log into PostgreSQL, and update the postgres user password:

```
sudo -u postgres psql postgres
postgres=# \password postgres
```

8. Edit pg_hba.conf:

```
sudo vim /etc/postgresql/16/main/pg_hba.conf
```

9. Add the following to the top of the table at the bottom:

```
host all all 0.0.0.0/0 scram-sha-256
local all postgres md5
local all all md5
host all all 127.0.0.1/32 md5
host all all ::1/128 md5
```

10. Stop the PostgreSQL service:

```
sudo systemctl stop postgresql
```

11. Copy the PostgreSQL data directory to your data disk:

```
sudo rsync -av /var/lib/postgresql /pgmnt/data
```

12. Copy the appropriate postgresql.conf file from the Scripts section of this document depending on your PostgreSQL instance size. Example for 4-vCPU instance:

```
sudo cp -p /etc/postgresql/16/main/postgresql.conf{,.bak}
sudo cp -f 4vcpu.conf /etc/postgresql/16/main/postgresql.conf
```

13. Enable and start the PostgreSQL service:

```
sudo systemctl enable postgresql
sudo systemctl start postgresql
```

## Configuring Ubuntu 22.04 LTS and installing HammerDB 4.9 on the HammerDB client instance

1. Log into the HammerDB client instance via SSH.
2. Install required packages:

```
sudo apt-get install -y wget vim tar zip unzip pigz nmon sysstate
```

3. Download and extract HammerDB 4.9:

```
sudo wget https://github.com/TPC-Council/HammerDB/releases/download/v4.9/HammerDB-4.9-Linux.tar.gz
tar -xf HammerDB-4.9-Linux.tar.gz
```

4. Download and extract the nmonchart tool:

```
wget https://sourceforge.net/projects/nmon/files/nmonchart40.tar
tar -xf nmoncgart40.tar ./nmonchart
```

5. Copy all scripts and config files from the Scripts section of this document to the HammerDB client instance.

## Creating the database schema with HammerDB

1. Log into the HammerDB client instance via SSH.
2. Navigate to the HammerDB directory:

```
cd HammerDB-4.9
```

3. Start hammerdbcli:

```
./hammerdbcli
```

4. Set the following variables:

```
dbset db pg
diset connection pg_host <IP_ADDRESS>
diset tpcc pg_user postgres
diset tpcc pg_pass <PASSWORD>
diset tpcc pg_count_ware <DB_SIZE>
diset tpcc pg_num_vu 8
diset tpcc pg_storedprocs true
```

5. Build the schema:

```
Buildschema
```

## Backing up the database

1. Log into the PostgreSQL instance via SSH.
2. Stop the PostgreSQL service:

```
sudo systemctl stop postgresql
```

3. Back up the database:

```
cd /pgmnt
sudo tar -cf- data/ | pigz -9 -c > /backups/pg<VCPU_COUNT>vCPU_backup.tar.gz
```

4. Repeat all database creation and backup steps for all warehouse sizes.

## Running the tests

In this section, we list the steps to run the HammerDB TPROC-C test on the instances under test. As each instance had different hardware and database sizes, please refer to Table 1 to see the number of users to run on each instance.

1. Log into the HammerDB client instance via SSH.
2. Execute the run_test.sh script, substituting IP_ADDRESS with the AWS private IP of the PostgreSQL instance and DB_SIZE with the number of warehouses. You can tune additional parameters and configuration options by modifying the script and editing the variables at the start of the file.

```
./run_test.sh <IP_ADDRESS> <DB_SIZE>
```

3. By default, HammerDB will save results to a folder within the Results folder in your home directory on the client instance. Copy the corresponding folder to your preferred location from which to parse the results.
4. Reboot the PostgreSQL instance and client instance.
5. Repeat these steps two more times for a total of three runs, and report the median result. Do this for each PostgreSQL instance type and warehouse size combination.

Table 4: Specifications for testing.

| VM type | {r6i,r7i}.xlarge | {r6i,r7i}.4xlarge | {r6i,r7i}.16xlarge |
|---|---|---|---|
| Number of vCPUs | 4 | 16 | 64 |
| Memory (GB) | 32 | 128 | 512 |
| Data disk (EBS type, size) | gp3, 40 GB | io1, 150 GB | io1, 500 GB |
| Number of warehouses | 150 | 600 | 2,400 |
| Number of users | 32 | 144 | 112 for r6i.16xlarge<br>128 for r7i.16xlarge |
| Warmup (min) | 5 | 5 | 5 |
| Runtime (min) | 10 | 10 | 10 |

## Scripts

### postgres_host_prepare_4vcpu.sh

```
#!/bin/bash

### Install tools ###
sudo apt-get update -y
sudo apt-get upgrade -y
sudo apt-get install wget tar zip unzip vim sysstat pigz nmon tuned

### System tuning ###
tuned-adm profile throughput-performance

sudo sed -i -e '/vm.swappiness/d' -e '/fs.aio-max-nr/d' -e '/vm.nr_hugepages/d' /etc/sysctl.conf
sudo cat <<EOF >>/etc/sysctl.conf
vm.swappiness = 1
fs.aio-max-nr = 1048576
vm.nr_hugepages = 24
EOF

sudo sed -i -e '/^EOF$/i postgres soft memlock 500000000\npostgres hard memlock 500000000' /etc/
security/limits.conf

sysctl -p

### Prepare storage ###
sudo mkdir /pgmnt
sudo mkdir /backups

sudo mkfs.xfs -f /dev/nvme1n1
sudo mkfs.xfs -f /dev/nvme2n1
sudo mount /dev/nvme1n1 /pgmnt
sudo mount /dev/nvme2n1 /backups

sudo sh -c 'echo "/dev/nvme1n1 /pgmnt xfs defaults 0 0" >> /etc/fstab'
sudo sh -c 'echo "/dev/nvme2n1 /backups xfs defaults 0 0" >> /etc/fstab'

sudo mkdir /pgmnt/data
```

### postgres_host_prepare_16vcpu.sh

```
#!/bin/bash

### Install tools ###
sudo apt-get update -y
sudo apt-get upgrade -y
sudo apt-get install wget tar zip unzip vim sysstat pigz nmon tuned

### System tuning ###
tuned-adm profile throughput-performance

sudo sed -i -e '/vm.swappiness/d' -e '/fs.aio-max-nr/d' -e '/vm.nr_hugepages/d' /etc/sysctl.conf
sudo cat <<EOF >>/etc/sysctl.conf
vm.swappiness = 1
fs.aio-max-nr = 1048576
vm.nr_hugepages = 90
EOF

sudo sed -i -e '/^EOF$/i postgres soft memlock 500000000\npostgres hard memlock 500000000' /etc/
security/limits.conf

sysctl -p

### Prepare storage ###
sudo mkdir /pgmnt
sudo mkdir /backups

sudo mkfs.xfs -f /dev/nvme1n1
```

```
sudo mkfs.xfs -f /dev/nvme2n1
sudo mount /dev/nvme1n1 /pgmnt
sudo mount /dev/nvme2n1 /backups

sudo sh -c 'echo "/dev/nvme1n1 /pgmnt xfs defaults 0 0" >> /etc/fstab'
sudo sh -c 'echo "/dev/nvme2n1 /backups xfs defaults 0 0" >> /etc/fstab'

sudo mkdir /pgmnt/data
```

## postgres_host_prepare_64vcpu.sh

```
#!/bin/bash

### Install tools ###
sudo apt-get update -y
sudo apt-get upgrade -y
sudo apt-get install wget tar zip unzip vim sysstat pigz nmon tuned

### System tuning ###
tuned-adm profile throughput-performance

sudo sed -i -e '/vm.swappiness/d' -e '/fs.aio-max-nr/d' -e '/vm.nr_hugepages/d' /etc/sysctl.conf
sudo cat <<EOF >>/etc/sysctl.conf
vm.swappiness = 1
fs.aio-max-nr = 1048576
vm.nr_hugepages = 400
EOF

sudo sed -i -e '/^EOF$/i postgres soft memlock 500000000\npostgres hard memlock 500000000' /etc/
security/limits.conf

sysctl -p

### Prepare storage ###
sudo mkdir /pgmnt
sudo mkdir /backups

sudo mkfs.xfs -f /dev/nvme1n1
sudo mkfs.xfs -f /dev/nvme2n1
sudo mount /dev/nvme1n1 /pgmnt
sudo mount /dev/nvme2n1 /backups

sudo sh -c 'echo "/dev/nvme1n1 /pgmnt xfs defaults 0 0" >> /etc/fstab'
sudo sh -c 'echo "/dev/nvme2n1 /backups xfs defaults 0 0" >> /etc/fstab'

sudo mkdir /pgmnt/data
```

## 4vcpu.conf

```
listen_addresses = '*'
port = 5432
data_directory = '/pgmnt/data/postgresql/16/main/'
max_connections = 256
shared_buffers = 20480MB
huge_pages = on
temp_buffers = 1024MB
work_mem = 1024MB
maintenance_work_mem = 512MB
autovacuum_work_mem = -1
max_stack_depth = 5MB
dynamic_shared_memory_type = posix
max_files_per_process = 4000
effective_io_concurrency = 32
wal_level = minimal
synchronous_commit = off
wal_buffers = 512MB
checkpoint_timeout = 1h
max_wal_size = 1GB
```

```
min_wal_size = 80MB
checkpoint_completion_target = 1
checkpoint_warning = 0
max_wal_senders = 0
log_destination = 'stderr'
logging_collector = on
log_directory = 'log'
log_filename = 'postgresql-%a.log'
log_truncate_on_rotation = on
log_rotation_age = 1d
log_rotation_size = 0
log_min_messages = error
log_min_error_statement = error
log_line_prefix = '%m [%p] '
log_timezone = 'America/New_York'
autovacuum = off
datestyle = 'iso, mdy'
timezone = 'America/New_York'
lc_messages = 'en_US.UTF-8'
lc_monetary = 'en_US.UTF-8'
lc_numeric = 'en_US.UTF-8'
lc_time = 'en_US.UTF-8'
default_text_search_config = 'pg_catalog.english'
max_locks_per_transaction = 64
max_pred_locks_per_transaction = 64
```

### 16vcpu.conf

```
listen_addresses = '*'
port = 5432
data_directory = '/pgmnt/data/postgresql/16/main/'
max_connections = 256
shared_buffers = 81920MB
huge_pages = on
temp_buffers = 1024MB
work_mem = 2048MB
maintenance_work_mem = 512MB
autovacuum_work_mem = -1
max_stack_depth = 5MB
dynamic_shared_memory_type = posix
max_files_per_process = 4000
effective_io_concurrency = 32
wal_level = minimal
synchronous_commit = off
wal_buffers = 512MB
checkpoint_timeout = 1h
max_wal_size = 1GB
min_wal_size = 80MB
checkpoint_completion_target = 1
checkpoint_warning = 0
max_wal_senders = 0
log_destination = 'stderr'
logging_collector = on
log_directory = 'log'
log_filename = 'postgresql-%a.log'
log_truncate_on_rotation = on
log_rotation_age = 1d
log_rotation_size = 0
log_min_messages = error
log_min_error_statement = error
log_line_prefix = '%m [%p] '
log_timezone = 'America/New_York'
autovacuum = off
datestyle = 'iso, mdy'
timezone = 'America/New_York'
lc_messages = 'en_US.UTF-8'
lc_monetary = 'en_US.UTF-8'
lc_numeric = 'en_US.UTF-8'
lc_time = 'en_US.UTF-8'
default_text_search_config = 'pg_catalog.english'
max_locks_per_transaction = 64
max_pred_locks_per_transaction = 64
```

## 64vcpu.conf

```
listen_addresses = '*'
port = 5432
data_directory = '/pgmnt/data/postgresql/16/main/'
max_connections = 256
shared_buffers = 384000MB
huge_pages = on
temp_buffers = 1024MB
work_mem = 4096MB
maintenance_work_mem = 1024MB
autovacuum_work_mem = -1
max_stack_depth = 5MB
dynamic_shared_memory_type = posix
max_files_per_process = 4000
effective_io_concurrency = 32
wal_level = minimal
synchronous_commit = off
wal_buffers = 1024MB
checkpoint_timeout = 1h
max_wal_size = 1GB
min_wal_size = 80MB
checkpoint_completion_target = 1
checkpoint_warning = 0
max_wal_senders = 0
log_destination = 'stderr'
logging_collector = on
log_directory = 'log'
log_filename = 'postgresql-%a.log'
log_truncate_on_rotation = on
log_rotation_age = 1d
log_rotation_size = 0
log_min_messages = error
log_min_error_statement = error
log_line_prefix = '%m [%p] '
log_timezone = 'America/New_York'
autovacuum = off
datestyle = 'iso, mdy'
timezone = 'America/New_York'
lc_messages = 'en_US.UTF-8'
lc_monetary = 'en_US.UTF-8'
lc_numeric = 'en_US.UTF-8'
lc_time = 'en_US.UTF-8'
default_text_search_config = 'pg_catalog.english'
max_locks_per_transaction = 64
max_pred_locks_per_transaction = 64
```

## hdb_tpcc_pg_150wh.tcl

```
dbset db pg
diset tpcc pg_driver timed
diset tpcc pg_user postgres
diset tpcc pg_pass Password1
diset tpcc pg_rampup 5
diset tpcc pg_duration 10
diset tpcc pg_count_ware 150
diset tpcc pg_total_iterations 10000000

loadscript
vuset vu 32
vuset logtotemp 1
vucreate
vurun
vudestroy
```

### hdb_tpcc_pg_600wh.tcl

```
dbset db pg
diset tpcc pg_driver timed
diset tpcc pg_user postgres
diset tpcc pg_pass Password1
diset tpcc pg_rampup 5
diset tpcc pg_duration 10
diset tpcc pg_count_ware 600
diset tpcc pg_total_iterations 10000000

loadscript
vuset vu 144
vuset logtotemp 1
vucreate
vurun
vudestroy
```

### hdb_tpcc_pg_2400wh.tcl

```
dbset db pg
diset tpcc pg_driver timed
diset tpcc pg_user postgres
diset tpcc pg_pass Password1
diset tpcc pg_rampup 5
diset tpcc pg_duration 10
diset tpcc pg_count_ware 2400
diset tpcc pg_total_iterations 10000000

loadscript
vuset vu 128
vuset logtotemp 1
vucreate
vurun
vudestroy
```

### run_test.sh

```
#!/bin/bash
echo "Usage: $0 TEST_HOST WAREHOUSE_COUNT"
TEST_HOST=${1:-remotehost}
CLIENT_HOST=$(hostname -s)
WAREHOUSE_COUNT=${2}
APP=postgresql
#MYCNF=my-${WAREHOUSE_COUNT}.cnf
HDB_DIR=HammerDB-4.9/
HDB_SCRIPT=hdb_tpcc_${APP}_${WAREHOUSE_COUNT}wh.tcl
HDB_RUN=run_${HDB_SCRIPT}
RUNNING_FILE=benchmark_running.txt

RAMPUP=5# minutes
DURATION=10   # minutes

STEP=2  # seconds
IDLE=30 # seconds

WARMUP=$((RAMPUP*60))
RUNTIME=$((DURATION*60))
SAMPLES_TOTAL=$(((WARMUP+RUNTIME)/STEP+5))

TIMESTAMP=$(date '+%Y%m%d_%H%M%S')

# Check for files

#if [ ! -e ${MYCNF} ]; then
#  echo "Missing my.cnf config: ${MYCNF}"
#  exit
```

```
#fi

if [ ! -e ${HDB_DIR}/hammerdbcli ]; then
  echo "Missing hammerdbcli missing: ${HDB_DIR}/hammerdbcli"
  exit
fi

if [ ! -e ${HDB_SCRIPT} ]; then
  echo "Missing HammerDB script: ${HDB_SCRIPT}"
  exit
fi


# Test SSH host access
sed -i "/${TEST_HOST}/d" ~/.ssh/known_hosts
ssh ${TEST_HOST} 'hostname -f' || exit

# Get AWS info
REMOTE_HOSTNAME="$(ssh ${TEST_HOST} 'hostname -s')"
INSTANCE_TYPE="$(ssh ${TEST_HOST} 'curl -s http://169.254.169.254/latest/meta-data/instance-type |
sed -e "s/ //g"')"
echo "INSTANCE_TYPE: ${INSTANCE_TYPE}"
INSTANCE_CPU="$(ssh ${TEST_HOST} 'awk "/model name/{print \$7\$8;exit}" /proc/cpuinfo | sed -e "s/ //g"
-e "s/CPU//"')"
echo "INSTANCE_CPU: ${INSTANCE_CPU}"
sleep 1

# Check if benchmark is already running
if [ -e ${RUNNING_FILE} ]; then
  echo "Benchmark already running: $(cat ${RUNNING_FILE})"
  RUNNING_HOST=$(awk '{print $1}' ${RUNNING_FILE})
  if [[ "${RUNNING_HOST}" == "${TEST_HOST}" ]]; then
    echo "Test already running on the same remote host. Exiting..."
      exit
  fi
  sleep 3
  echo "If this is incorrect manually remove the benchmark running file: ${RUNNING_FILE}"
  sleep 3
  echo "Benchmark will pause after restoring database until current benchmark finishes."
  sleep 3
fi

# Prepare Test Host
echo -e "\nPreparing test host.\n"

#scp ${MYCNF} ${TEST_HOST}:tmp-my.cnf
#ssh ${TEST_HOST} "sudo service ${APP} stop ; sudo cp -vf tmp-my.cnf /etc/mysql/mysql.conf.d/mysqld.cnf"
ssh ${TEST_HOST} "sudo service ${APP} start && \
      sleep 10 && \
      sync && \
      sudo service ${APP} stop && \
      sudo service ${APP} start" || exit

# Check if benchmark is already running and if so wait till it finishes
if [ -e ${RUNNING_FILE} ]; then
  echo "Benchmark running: $(cat ${RUNNING_FILE})"
  echo "Please wait for it to finish or manually remove the benchmark running file: ${RUNNING_FILE}"
  date
  echo -n "Waiting"
  while [ -e ${RUNNING_FILE} ];
  do
    echo -n "."
    sleep ${STEP}
  done
  echo "Done!"
  date
fi

echo "${TEST_HOST} ${WAREHOUSE_COUNT} ${INSTANCE_TYPE} ${INSTANCE_CPU} ${TIMESTAMP}" > ${RUNNING_FILE}

# Make results folder
echo -e "\nCreating results folder and saving config files.\n"
```

```
RESULTS_DIR=results/${APP}_${INSTANCE_TYPE}_${INSTANCE_CPU}_${TIMESTAMP}
mkdir -p ${RESULTS_DIR}
RESULTS_FILE=${APP}_${INSTANCE_TYPE}_${INSTANCE_CPU}_${TIMESTAMP}

# Copy config files to results folder
cp -pvf ${0} ${RESULTS_DIR}/
#cp -pvf ${MYCNF} ${RESULTS_DIR}/
cp -pvf ${HDB_SCRIPT} ${RESULTS_DIR}/

# Copy client info to results folder
sudo dmidecode > ${RESULTS_DIR}/client_dmidecode.txt
dmesg > ${RESULTS_DIR}/client_dmesg.txt
lscpu > ${RESULTS_DIR}/client_lscpu.txt
apt list --installed > ${RESULTS_DIR}/client_apts.txt
#curl -s http://169.254.169.254/latest/meta-data/placement/availability-zone > ${RESULTS_
DIR}/client_av.txt

# Copy server info to results folder
ssh ${TEST_HOST} 'sudo dmidecode' > ${RESULTS_DIR}/server_dmidecode.txt
ssh ${TEST_HOST} 'dmesg' > ${RESULTS_DIR}/server_dmesg.txt
ssh ${TEST_HOST} 'lscpu' > ${RESULTS_DIR}/server_lscpu.txt
ssh ${TEST_HOST} 'apt list --installed' > ${RESULTS_DIR}/server_apts.txt
#ssh ${TEST_HOST} 'curl -s http://169.254.169.254/latest/meta-data/placement/availability-zone' >
${RESULTS_DIR}/server_av.txt

# Save memory and disk info
cat /proc/meminfo > ${RESULTS_DIR}/client_meminfo.txt
ssh ${TEST_HOST} 'cat /proc/meminfo' > ${RESULTS_DIR}/server_meminfo.txt
ssh ${TEST_HOST} 'df -T --sync' > ${RESULTS_DIR}/server_df.txt

# Prepare HammerDB run script
# sed -e "s/dbset db .*/dbset db ${APP}/" \
sed -e "s/dbset db .*/dbset db pg/" \
    -e "s/_host.*/_host ${TEST_HOST}/" \
    -e "s/_count_ware.*/_count_ware ${WAREHOUSE_COUNT}/" \
    -e "s/_rampup.*/_rampup ${RAMPUP}/" \
    -e "s/_duration.*/_duration ${DURATION}/" \
      ${HDB_SCRIPT} > ${HDB_DIR}/${HDB_RUN}
cp -pvf ${HDB_DIR}/${HDB_RUN} ${RESULTS_DIR}/

# Prepare nmon on client and server
sudo killall -q -w nmon ; sudo sync ; sudo rm -f /tmp/client.nmon
ssh ${TEST_HOST} "sudo killall -q -w nmon ; sudo sync ; sudo rm -f /tmp/server.nmon"

# Idle wait for DB to settle
echo -e "\nIdle benchmark for ${IDLE} seconds."
sleep ${IDLE}

# Start nmon on client and server and wait 1 step
sudo nmon -F /tmp/client.nmon -s${STEP} -c$((SAMPLES_TOTAL)) -J -t
ssh ${TEST_HOST} "sudo nmon -F /tmp/server.nmon -s${STEP} -c$((SAMPLES_TOTAL)) -J -t -g auto -D"
sleep ${STEP}

# Run benchmark
echo -e "\nRunning benchmark for $((RAMPUP+DURATION)) minutes!"
rm -f /tmp/hammerdb.log
pushd ${HDB_DIR}
./hammerdbcli auto ${HDB_RUN}
pushd

# Stop nmon and copy to results folder on client and server
ssh ${TEST_HOST} "sudo killall -w nmon"
sudo killall -w nmon
cp -vf /tmp/client.nmon ${RESULTS_DIR}/client_${RESULTS_FILE}.nmon
scp ${TEST_HOST}:/tmp/server.nmon ${RESULTS_DIR}/server_${RESULTS_FILE}.nmon

# Save results
cp -vf /tmp/hammerdb.log ${RESULTS_DIR}/${RESULTS_FILE}_hammerdb.log

# Parse nmon files using nmonchart
for nmonfile in `find ${RESULTS_DIR}/*.nmon`;
do
```

```
  ./nmonchart $nmonfile
done

# Update memory and disk info
cat /proc/meminfo >> ${RESULTS_DIR}/client_meminfo.txt
ssh ${TEST_HOST} 'cat /proc/meminfo' >> ${RESULTS_DIR}/server_meminfo.txt
ssh ${TEST_HOST} 'df -T --sync' >> ${RESULTS_DIR}/server_df.txt

# Shutdown server
#ssh ${TEST_HOST} 'sudo poweroff'

# Remove benchmark running file
rm -f ${RUNNING_FILE}
```

Read the report at https://facts.pt/1pNm4GN ▶

This project was commissioned by Intel.

Principled Technologies®

Facts matter.®