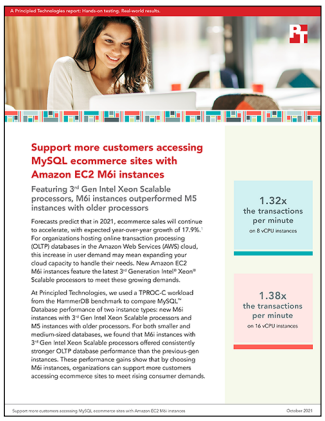

The science behind the report:

Support more customers accessing MySQL ecommerce sites with Amazon EC2 M6i instances

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 Support more customers accessing MySQL ecommerce sites with Amazon EC2 M6i instances.

We concluded our hands-on testing on September 1, 2021. 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 August 25, 2021 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.

MySQL instance	TPM
8 vCPUs	
m5.2xlarge	298,347
m6i.2xlarge	395,501
16 vCPUs	
m5.4xlarge	570,066
m6i.4xlarge	788,027

System configuration information

Table 2: Detailed information on the instances we tested.

System configuration information	m5.2xlarge	m5.4xlarge
Tested by	Principled Technologies	Principled Technologies
Test date	09/01/2021	09/01/2021
CSP and region	us-east-1b	us-east-1b
Workload and version	HammerDB v4.2 TPROC-C	HammerDB v4.2 TPROC-C
WL specific parameters	250 warehouses 16 virtual users	500 warehouses 32 virtual users
Iterations and result choice	3 runs, median	3 runs, median
Server platform	m5.2xlarge	m5.4xlarge
BIOS name and version	Amazon EC2 1.0 10/16/2017	Amazon EC2 1.0 10/16/2017
Operating system name and version/build number	Ubuntu 20.04 LTS	Ubuntu 20.04 LTS
Date of last OS updates/patches applied	08/25/2021	08/25/2021
Processor		
Number of processors	1	1
Vendor and model	Intel® Xeon® Platinum 8175M	Intel Xeon Platinum 8175M
Core count (per processor)	24	24
Core frequency (GHz)	2.50	2.50
Stepping	4	4
Hyper-threading	Yes	Yes
Turbo	Yes	Yes
Number of vCPUs per VM	8	16
Memory module(s)		
Total memory in system (GB)	32	64
NVMe™ memory present?	No	No
Total memory in GB (DDR+NVMe RAM)	32	64
General hardware		
Storage: NW or Direct Att / Instance	NW	NW
Network bandwidth/instance	10 Gbps	10 Gbps
Storage bandwidth/instance	4,750 Mbps	4,750 Mbps
Local storage		
OS		
Number of drives	1	1
Drive size (GB)	20	20
Drive information	gp2, EBS, 100/3000 IOPS	gp2, EBS, 100/3000 IOPS

System configuration information	m5.2xlarge	m5.4xlarge
Data drive		
Number of drives	1	1
Drive size (GB)	64	128
Drive information	io1, EBS, 2000 IOPS	io1, EBS, 4000 IOPS
Network adapter		
Vendor and model	Amazon Elastic Network Adapter	Amazon Elastic Network Adapter
Number and type of ports	1x 10Gb	1x 10Gb

Table 3: Detailed information on the instances we tested.

System configuration information	m6i.2xlarge	m6i.4xlarge
Tested by	Principled Technologies	Principled Technologies
Test date	08/25/2021	08/26/2021
CSP and region	us-east-1b	us-east-1b
Workload and version	HammerDB v4.2 TPROC-C	HammerDB v4.2 TPROC-C
WL specific parameters	250 warehouses 16 virtual users	500 warehouses 32 virtual users
Iterations and result choice	3 runs, median	3 runs, median
Server platform	m6i.2xlarge	m6i.4xlarge
BIOS name and version	Amazon EC2 1.0 10/16/2017	Amazon EC2 1.0 10/16/2017
Operating system name and version/build number	Ubuntu 20.04 LTS	Ubuntu 20.04 LTS
Date of last OS updates/patches applied	08/25/2021	08/25/2021
Processor		
Number of processors	1	1
Vendor and model	Intel Xeon Platinum 8375C	Intel Xeon Platinum 8375C
Core count (per processor)	32	32
Core frequency (GHz)	2.90	2.90
Stepping	6	6
Hyper-threading	Yes	Yes
Turbo	Yes	Yes
Number of vCPUs per VM	8	16
Memory module(s)		
Total memory in system (GB)	32	64
NVMe™ memory present?	No	No
Total memory in GB (DDR+NVMe RAM)	32	64

System configuration information	m6i.2xlarge	m6i.4xlarge
General hardware		
Storage: NW or Direct Att / Instance	NW	NW
Network bandwidth/instance	12.5 Gbps	12.5 Gbps
Storage bandwidth/instance	10 Gbps	10 Gbps
Local storage		
OS		
Number of drives	1	1
Drive size (GB)	20	20
Drive information	gp2, EBS, 100/3000 IOPS	gp2, EBS, 100/3000 IOPS
Data drive		
Number of drives	1	1
Drive size (GB)	64	128
Drive information	io1, EBS, 2000 IOPS	io1, EBS, 4000 IOPS
Network adapter		
Vendor and model	Amazon Elastic Network Adapter	Amazon Elastic Network Adapter
Number and type of ports	1x 10Gb	1x 10Gb

How we tested

Testing overview

We tested two types of Amazon Web Services (AWS) EC2 instances: M6i instances with 3rd Gen Intel Xeon Scalable processors and M5 instances with older Intel Xeon Scalable processors. We ran a TPROC-C workload on MySQL™ on the AWS instances to show the performance increase in terms of transactions per minute on OLTP databases that you could expect to see using the newer instance series vs. the older. Note that we sized the MySQL 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 MySQL instance

1. Log into AWS, and navigate to the AWS Management Console.
2. Click EC2.
3. Click Launch instance, and from the drop-down menu, click Launch instance to open the Launch Instance wizard.
4. In the search window, enter `Ubuntu`, and press Enter.
5. On the Quick Start tab, select the button next to Ubuntu Server 20.04 LTS (HVM), SSD-Volume Type - ami09e67e426f25ce0d7.
6. On the Choose Instance Type tab, select mysql-{m6I,m6g,m5n}.{2,4,16}xlarge. Click Next: Configure Instance Details.
7. On the Configure Instance tab, set the following:
   - Number of instances: 1
   - Purchasing option: Leave unchecked
   - Network: Default VPC
   - Subnet: Choose the region you're working in (we chose us-east-1b)
   - Auto-assign Public IP: Enable
   - Placement Group: Leave unchecked
   - Capacity Reservation: Open
   - Domain join directory: No Directory
   - IAM role: None
   - Shutdown behavior: Stop
8. Click Next: Add Storage.
9. On the Add Storage tab, set the following:
   - Size: 10GB
   - Volume Type: gp2
   - Delete on Termination: Checked
   - Encryption: Not Encrypted
   - Click Add New Volume
   - Size: {64GB,128GB,512GB}
   - Volume Type: io1
   - IOPS: {2000,4000,16000}
   - Delete on Termination: Checked
   - Encryption: Not Encrypted
10. Click Next: Add Tags.
11. On the Add Tags tab, add any appropriate tags, and click Next: Configure Security Group.
12. On the Configure Security Group tab, set the following:
    a. Create a new security group.
    b. Leave the rules default.
    c. Click Review, and Launch.
13. On the Review Tab, click Launch.
14. Choose the appropriate option for the key pair, and click Launch Instances.

Creating the HammerDB 4.2 client instance

1. Log into AWS, and navigate to the AWS Management Console.
2. Click EC2.
3. Click Launch instance, and from the drop-down menu, select Launch instance to open the Launch Instance wizard.
4. In the search window, enter Ubuntu, and press enter.
5. On the Quick Start tab, click the Select button next to Ubuntu Server 20.04 LTS (HVM), SSD-Volume Type - ami09e67e426f25ce0d7.
6. On the Choose Instance Type tab, select m5.2xlarge, and click Next: Configure Instance Details.
7. On the Configure Instance tab, set the following:
   - Number of instances: 1
   - Purchasing option: Leave unchecked
   - Network: Default VPC.
   - Subnet: Choose the region you're working in
   - Auto-assign Public IP: Enable
   - Placement Group: Leave unchecked
   - Capacity Reservation: Open
   - Domain join directory: No Directory
   - IAM role: None
   - Shutdown behavior: Stop
8. Click Next: Add Storage.
9. On the Add Storage tab, set the following:
   - Size: 10GB
   - Volume Type: gp2
   - Delete on Termination: Checked
   - Encryption: Not Encrypted
10. Click Next: Add Tags
11. On the Add Tags tab, add any appropriate tags, and click Next: Configure Security Group.
12. On the Configure Security Group tab, set the following:
    a. Select an existing security group.
    b. Chose the group you created for MySQL and HammerDB.
    c. Click Review, and Launch.
13. On the Review Tab, click Launch.
14. Choose the appropriate option for the key pair, and click Launch Instances.

Configuring Ubuntu 20.04 LTS and installing MySQL on the MySQL instance

1. Log into the MySQL instance via SSH.
2. Run the mysql_host_prepare.sh script:

```
sudo ./mysql_host_prepare.sh
```

3. Download the appropriate MySQL bundle for Ubuntu from https://dev.mysql.com/downloads/mysql/.
4. Extract the MySQL bundle:

```
tar -xf mysql-server_8.0.26-1ubuntu20.04_amd64.deb-bundle.tar
```

5. Install MySQL Community Server 8 and all dependencies:

```
sudo dpkg -i mysql-community-server_8.0.26-1ubuntu20.04_amd64.deb
```

6. Stop the MySQL service:

```
sudo service mysql stop
```

7. Copy the MySQL data directory to your data disk:

```
sudo cp -R -p /var/lib/mysql /mnt/mysqldata/
```

8. Copy the appropriate my.cnf config file from the Scripts section depending on your MySQL instance and target database size. Example for a 250-warehouse database:

```
cp -p /etc/my.cnf{,.bak}
    cp -f my-250.cnf /etc/my.cnf
```

9. Start the MySQL service:

```
sudo service mysqld start
```

10. Log into the MySQL instance as the root user:

```
mysql -u root -p
```

11. Create a new user named mysql with full permissions:

```
CREATE USER 'mysql'@'localhost' IDENTIFIED BY '[password]';
GRANT ALL PRIVILEGES ON *.* TO 'mysql'@'localhost'
->WITH GRANT OPTION;
CREATE USER 'mysql'@'%' IDENTIFIED BY '[password]';
GRANT ALL PRIVILEGES ON *.* TO 'mysql'@'%'
->WITH GRANT OPTION;
```

12. Enable mysql_native_password authentication on the mysql user:

```
ALTER USER 'mysql'@'localhost' IDENTIFIED WITH mysql_native_password BY '[password]';
ALTER USER 'mysql'@'%' IDENTIFIED WITH mysql_native_password BY '[password]';
```

13. Shut down the instance:

```
sudo poweroff
```

Configuring Ubuntu and installing HammerDB 4.2 on the MySQL-client instance

1. Log into the HammerDB instance via SSH.
2. Turn off SSH strict host key checking:

```
echo 'StrictHostKeyChecking no' > .ssh/config
chmod 400 ~/.ssh/config
```

3. Install required packages:

```
sudo apt install -y wget vim tar zip unzip lz4 pigz nmon sysstat numactl ksh
```

4. Download the MySQL x86 RPM bundle from https://dev.mysql.com/downloads/mysql/.
5. Extract the MySQL bundle:

```
tar -xf mysql-server_8.0.26-1ubuntu20.04_amd64.deb-bundle.tar
```

6. Install the MySQL 8 Community Client and its dependencies:

```
sudo dpkg -i mysql-community-client_8.0.26-1ubuntu20.04_amd64.deb
```

7. Download HammerDB 4.2:

```
sudo wget https://github.com/TPC-Council/HammerDB/releases/download/v4.2/HammerDB-4.2-Linux.tar.gz
```

8. Extract the HammerDB package:

```
tar -xf HammerDB-4.2-Linux.tar.gz
```

9. Download and extract the nmonchart tool:

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

10. Copy all scripts and config files from the Scripts section to the HammerDB MySQL-client instance.
11. Shut down the instance:

```
sudo poweroff
```

Creating the database schema with HammerDB

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

```
cd HammerDB-4.2
```

3. Start hammerdbcli:

```
./hammerdbcli
```

4. Set the following variables:

```
dbset db mysql
    diset connection mysql_host <IP_ADDRESS>
    diset tpcc mysql_user root
    diset tpcc mysql_pass <Password>
    diset tpcc mysql_count_ware <DB_SIZE>
    diset tpcc mysql_partition true
    diset tpcc mysql_num_vu 8
    diset tpcc mysql_storage_engine innodb
```

5. Build the schema:

```
buildschema
```

Backing up the database

1. Log into the MySQL instance.
2. Shut down the database:

```
systemctl stop mysqld
```

3. Delete the log files:

```
cd /mnt/mysqldata/
    rm -f data/ib_logfile*
```

4. Back up the database:

```
tar -cf- data/ | pigz -9 -c > mysql_tpcc_<DB_SIZE>warehouses_data.tar.gz
```

5. Repeat all database creation 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 4 to see the number of users to run on each instance.

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

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

   The script will prepare the MySQL instance, restore the correct DB_SIZE, and run the test automatically. By default, results will save to the Results folder in your home directory.

3. To parse all results, run the parse_results.sh script:

```
./parse_results.sh
```

4. Reboot the MySQL 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 MySQL instance type and warehouse size combination.

Table 4: Specifications for testing.

VM type	m5.2xlarge, m6i.2xlarge	m5.4xlarge, m6i.4xlarge
Number of vCPUs	8	16
Memory (GB)	32	64
Data disk (size, IOPS)	64GB, 2,000 IOPS	128GB, 4,000 IOPS
Number of warehouses	250	500
Number of users	16	32
Warm-up time (min)	5	5
Run time (min)	10	10

Scripts

mysql_host_prepare.sh

```
#!/bin/bash

setenforce 0
#sed -i 's/SELINUX=.*/SELINUX=Permissive/' /etc/selinux/config
systemctl disable --now firewalld

#### System tuning ####
tuned-adm profile virtual-guest

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

sysctl -p

#### Install tools ####
#dnf install -y epel-release
apt-get install -y wget vim tar zip unzip lz4 pigz nmon sysstat numactl ksh


#### Prepare storage ####
umount -v /mnt/mysqldata
mkdir -p /mnt/mysqldata

sed -i '/mysqldata/d' /etc/fstab
if [ -e /dev/nvme1n1 ]; then
  mkfs.xfs -f /dev/nvme1n1
  echo '/dev/nvme1n1  /mnt/mysqldata  xfs  defaults,nofail,x-systemd.device-timeout=5  0
    2' >> /etc/fstab
else
  mkfs.xfs -f /dev/xvdb
  echo '/dev/xvdb  /mnt/mysqldata  xfs  defaults,nofail,x-systemd.device-timeout=5  0
    2' >> /etc/fstab
fi

mount -v /mnt/mysqldata
restorecon -Rv /mnt/mysqldata
```

my-250.cnf

```
[mysqld]
datadir=/mnt/mysqldata/mysql
default_authentication_plugin=mysql_native_password
socket=/var/run/mysqld/mysqld.sock
log-error=/var/log/mysql/error.log
pid-file=/var/run/mysqld/mysqld.pid
port=3306
bind_address=0.0.0.0

# general
max_connections=4000
table_open_cache=8000
table_open_cache_instances=16
back_log=1500
default_password_lifetime=0
ssl=0
performance_schema=OFF
max_prepared_stmt_count=128000
skip_log_bin=1
character_set_server=latin1
collation_server=latin1_swedish_ci
transaction_isolation=REPEATABLE-READ

# files
innodb_file_per_table
innodb_log_file_size=1024M
innodb_log_files_in_group=8 #scale
innodb_open_files=4000

# buffers
innodb_buffer_pool_size=24000M #scale
innodb_buffer_pool_instances=16
innodb_log_buffer_size=64M

# tune
innodb_doublewrite=0
innodb_thread_concurrency=0
innodb_flush_log_at_trx_commit=0
innodb_max_dirty_pages_pct=90
innodb_max_dirty_pages_pct_lwm=10
join_buffer_size=32K
sort_buffer_size=32K
innodb_use_native_aio=1
innodb_stats_persistent=1
innodb_spin_wait_delay=6
innodb_max_purge_lag_delay=300000
innodb_max_purge_lag=0
innodb_flush_method=O_DIRECT_NO_FSYNC
innodb_checksum_algorithm=none
innodb_io_capacity=1000
innodb_io_capacity_max=2000
innodb_lru_scan_depth=9000
innodb_change_buffering=none
innodb_read_only=0
innodb_page_cleaners=4
innodb_undo_log_truncate=off
```

```
# perf special
innodb_adaptive_flushing=1
innodb_flush_neighbors=0
innodb_read_io_threads=16
innodb_write_io_threads=16
innodb_purge_threads=4
innodb_adaptive_hash_index=0

# monitoring
innodb_monitor_enable='%'
```

my-500.cnf

```
[mysqld]
datadir=/mnt/mysqldata/mysql
default_authentication_plugin=mysql_native_password
socket=/var/run/mysqld/mysqld.sock
log-error=/var/log/mysql/error.log
pid-file=/var/run/mysqld/mysqld.pid
port=3306
bind_address=0.0.0.0

# general
max_connections=4000
table_open_cache=8000
table_open_cache_instances=16
back_log=1500
default_password_lifetime=0
ssl=0
performance_schema=OFF
max_prepared_stmt_count=128000
skip_log_bin=1
character_set_server=latin1
collation_server=latin1_swedish_ci
transaction_isolation=REPEATABLE-READ

# files
innodb_file_per_table
innodb_log_file_size=1024M
innodb_log_files_in_group=16 #scale
innodb_open_files=4000

# buffers
innodb_buffer_pool_size=48000M #scale
innodb_buffer_pool_instances=16
innodb_log_buffer_size=64M

# tune
innodb_doublewrite=0
innodb_thread_concurrency=0
innodb_flush_log_at_trx_commit=0
innodb_max_dirty_pages_pct=90
innodb_max_dirty_pages_pct_lwm=10
join_buffer_size=32K
sort_buffer_size=32K
innodb_use_native_aio=1
innodb_stats_persistent=1
```

```
innodb_spin_wait_delay=6
innodb_max_purge_lag_delay=300000
innodb_max_purge_lag=0
innodb_flush_method=O_DIRECT_NO_FSYNC
innodb_checksum_algorithm=none
innodb_io_capacity=2000
innodb_io_capacity_max=4000
innodb_lru_scan_depth=9000
innodb_change_buffering=none
innodb_read_only=0
innodb_page_cleaners=4
innodb_undo_log_truncate=off

# perf special
innodb_adaptive_flushing=1
innodb_flush_neighbors=0
innodb_read_io_threads=16
innodb_write_io_threads=16
innodb_purge_threads=4
innodb_adaptive_hash_index=0

# monitoring
innodb_monitor_enable='%'
```

hdb_tpcc_mysql_250wh.tcl

```
#!/bin/tclsh
puts "SETTING CONFIGURATION"
global complete
proc wait_to_complete {} {
global complete
set complete [vucomplete]
if {!$complete} { after 5000 wait_to_complete } else { exit }
}
dbset db mysql
diset connection mysql_host 127.0.0.1
diset connection mysql_port 3306

diset tpcc mysql_user mysql
diset tpcc mysql_pass Password1
diset tpcc mysql_storage_engine innodb
diset tpcc mysql_partition true
diset tpcc mysql_driver timed

diset tpcc mysql_count_ware 250
diset tpcc mysql_num_vu 16
diset tpcc mysql_rampup 2
diset tpcc mysql_duration 5

vuset logtotemp 1
loadscript
vuset vu 16
vucreate
vurun
wait_to_complete
vwait forever
```

hdb_tpcc_mysql_500wh.tcl

```
#!/bin/tclsh
puts "SETTING CONFIGURATION"
global complete
proc wait_to_complete {} {
global complete
set complete [vucomplete]
if {!$complete} { after 5000 wait_to_complete } else { exit }
}
dbset db mysql
diset connection mysql_host 127.0.0.1
diset connection mysql_port 3306

diset tpcc mysql_user mysql
diset tpcc mysql_pass Password1
diset tpcc mysql_storage_engine innodb
diset tpcc mysql_partition true
diset tpcc mysql_driver timed

diset tpcc mysql_count_ware 500
diset tpcc mysql_num_vu 32
diset tpcc mysql_rampup 2
diset tpcc mysql_duration 5

vuset logtotemp 1
loadscript
vuset vu 32
vucreate
vurun
wait_to_complete
vwait forever
```

run_test.sh

```
#!/bin/bash
echo "Usage: $0 TEST_HOST WAREHOUSE_COUNT"
TEST_HOST=${1:-remotehost}
CLIENT_HOST=$(hostname -s)
WAREHOUSE_COUNT=${2}
APP=mysql
MYCNF=my-${WAREHOUSE_COUNT}.cnf
HDB_DIR=HammerDB-4.2/
HDB_SCRIPT=hdb_tpcc_${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 ${APP} stop && \
  sudo rm -rf /mnt/${APP}data && \
  pigz -d -c /mnt/${APP}data/${APP}_tpcc_${WAREHOUSE_COUNT}warehouses_data.tar.gz | sudo tar C /
    mnt/${APP}data -xf- ; sync
  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}/" \
    -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"
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}
```

run_test.sh

```
#!/bin/bash
RAMPUP=5    # minutes
STEP=2  # seconds
SKIP=$(((RAMPUP*60)/STEP+1))
echo "RAMPUP: ${RAMPUP} minutes"
echo "STEP: ${STEP} seconds"
echo "SKIP: ${SKIP} records"

echo -e "Benchmark\tInstance\tCPU type\tTimestamp\tTPM\tNOPM\tServer CPU%\tClient CPU%\tServer apts\
    tClient apts\tServer AZ\tClient AZ"
for result in `find results/* -type d | sort -V`;
do
  echo "$result" | awk -F'[/,_:]' '{printf("%s\t%s\t%s\t%d\t",$2,$3,$4,$5$6)}'
  for hammerdb in $result/*_hammerdb.log; do
    [ -f "$hammerdb" ] || continue
    awk '/NOPM/{printf("%d\t%d\t",$7,$11)}' ${hammerdb}
  done
  for server in $result/server_*.nmon; do
    [ -f "$server" ] || continue
    awk -F',' "/CPU_ALL/{rows+=1;if(rows>${SKIP}) {count+=1;idle+=\$6}}END{printf(\"%.2f\t\",100-
    idle/count)}" $server
  done
  for client in $result/client_*.nmon; do
    [ -f "$client" ] || continue
    awk -F',' "/CPU_ALL/{rows+=1;if(rows>${SKIP}) {count+=1;idle+=\$6}}END{printf(\"%.2f\t\",100-
    idle/count)}" $client
  done
  SERVER_CKSUM=$(sort ${result}/server_apts.txt | sha1sum)
  CLIENT_CKSUM=$(sort ${result}/client_apts.txt | sha1sum)
  echo -en "${SERVER_CKSUM::7}\t${CLIENT_CKSUM::7}\t$(cat ${result}/server_av.txt)\t$(cat ${result}/
    client_av.txt)"
  echo
done
```

Read the report at http://facts.pt/lpGOWpp ▶

This project was commissioned by Intel.