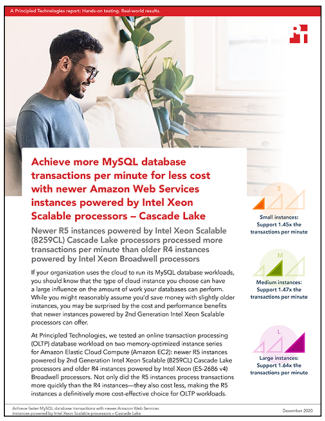

The science behind the report:

Achieve more MySQL database transactions per minute for less cost with newer Amazon Web Services instances featuring Intel Xeon Scalable processors – Cascade Lake

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 Achieve more MySQL database transactions per minute for less cost with newer Amazon Web Services instances featuring Intel Xeon Scalable processors – Cascade Lake.

We concluded our hands-on testing on November 13, 2020. 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 November 19, 2020 or earlier. Unavoidably, these configurations may not represent the latest versions available when this report appears.

Our results

The following tables list our test results in TPM and NOPM. For this report's wins, we used TPM.

Table 1: Comparison of results for the online transaction processing test from the HammerDB benchmarking suite on small (8vCPU) Amazon EC2 instances running MySQL. The r5.2xlarge instance featured Cascade Lake processors, while the r4.2xlarge instance featured Broadwell processors.

2xlarge series (600WH/24 Users)			
	r4	r5	r5 advantage
Transactions per minute	206,360	300,514	1.45x
New orders per minute	68,079	99,412	1.46x

Table 2: Comparison of results for the online transaction processing test from the HammerDB benchmarking suite on medium (16vCPU) Amazon EC2 instances running MySQL. The r5.4xlarge instance featured Cascade Lake processors, while the r4.4xlarge instance featured Broadwell processors.

4xlarge series (1200WH/32 Users)			
	r4	r5	r5 advantage
Transactions per minute	411,092	608,164	1.47x
New orders per minute	135,746	200,760	1.47x

Table 3: Comparison of results for the online transaction processing test from the HammerDB benchmarking suite on large (64vCPU) Amazon EC2 instances running MySQL. The r5.16xlarge instance featured Cascade Lake processors, while the r4.16xlarge instance featured Broadwell processors.

16xlarge series (4800WH/128 Users)			
	r4	r5	r5 advantage
Transactions per minute	1,265,113	2,078,293	1.64x
New orders per minute	417,387	686,059	1.64x

System configuration information

Table 4: Detailed information on the r4 systems we tested.

Server configuration information	r4.2xlarge	r4.4xlarge	r4.16xlarge
Tested by	Principled Technologies	Principled Technologies	Principled Technologies
Test date	11/16/2020	11/16/2020	11/16/2020
CSP / Region	us-east-1f	us-east-1f	us-east-1f
Workload & version	HammerDB v3.3 TPC-C-Like	HammerDB v3.3 TPC-C-Like	HammerDB v3.3 TPC-C-Like
WL specific parameters	600 warehouses 16 virtual users	1,200 warehouses 32 virtual users	4,800 warehouses 128 virtual users
Iterations and result choice	3 runs, median	3 runs, median	3 runs, median
Server platform	r4.2xlarge	r4.4xlarge	r4.16xlarge
BIOS name and version	Xen 4.2.amazon, 8/24/2006	Xen 4.2.amazon, 8/24/2006	Xen 4.2.amazon, 8/24/2006
Operating system name and version/build number	CentOS 8.2 4.18.0-193.19.1.el8_2.x86_64	CentOS 8.2 4.18.0-193.19.1.el8_2.x86_64	CentOS 8.2 4.18.0-193.19.1.el8_2.x86_64
Date of last OS updates/ patches applied	11/02/2020	11/02/2020	11/02/2020
Processor			
Number of processors	1	1	2
Vendor and model	Intel® Xeon® CPU E5-2686 v4	Intel Xeon CPU E5-2686 v4	Intel Xeon CPU E5-2686 v4
Core count (per processor)	4	8	16
Core frequency (GHz)	2.30	2.30	2.30
Stepping	1	1	1
Hyper-Threading	Yes	Yes	Yes
Turbo	Yes	Yes	Yes
Number of vCPU per VM	8	16	64
Memory module(s)			
Total memory in system (GB)	61	122	488
NVMe memory present?	No	No	No
Total memory (DDR+NVMe RAM)	61	122	488
General hardware			
Storage: Network or direct-attached	Network	Network	Network
Network bandwidth per VM instance	Up to 10 Gbps	Up to 10 Gbps	25 Gbps
Storage bandwidth per VM instance	1,700 Mbps	3,500 Mbps	14,000 Mbps

Server configuration information	r4.2xlarge	r4.4xlarge	r4.16xlarge
Local storage (OS)			
Number of drives	1	1	1
Drive size (GB)	10	10	10
Drive information (speed, interface, type)	gp2, EBS, 100/3000 IOPS	gp2, EBS, 100/3000 IOPS	gp2, EBS, 100/3000 IOPS
Data drive			
Number of drives	1	1	1
Drive size (GB)	128	256	1024
Drive information (speed, interface, type)	io1, EBS, 3000 IOPS	io1, EBS, 6000 IOPS	io1, EBS, 24000 IOPS
Network adapter			
Vendor and model	Amazon Elastic Network Adapter	Amazon Elastic Network Adapter	Amazon Elastic Network Adapter
Number and type of ports	1x 10Gb	1x 10Gb	1x 25Gb

Table 5: Detailed information on the r5 systems we tested.

Server configuration information	r5.2xlarge	r5.4xlarge	r5.16xlarge
Tested by	Principled Technologies	Principled Technologies	Principled Technologies
Test date	11/18/2020	11/18/2020	11/18/2020
CSP / Region	us-east-1f	us-east-1f	us-east-1f
Workload & version	HammerDB v3.3 TPC-C-Like	HammerDB v3.3 TPC-C-Like	HammerDB v3.3 TPC-C-Like
WL specific parameters	600 warehouses 16 virtual users	1200 warehouses 32 virtual users	4800 warehouses 128 virtual users
Iterations and result choice	3 runs, median	3 runs, median	3 runs, median
Server platform	r5.2xlarge	r5.4xlarge	r5.16xlarge
BIOS name and version	Amazon EC2 1.0, 10/16/2017	Amazon EC2 1.0, 10/16/2017	Amazon EC2 1.0, 10/16/2017
Operating system name and version/build number	CentOS 8.2 4.18.0-193.19.1.el8_2.x86_64	CentOS 8.2 4.18.0-193.19.1.el8_2.x86_64	CentOS 8.2 4.18.0-193.19.1.el8_2.x86_64
Date of last OS updates/ patches applied	11/02/2020	11/02/2020	11/02/2020
Processor			
Number of processors	1	1	2
Vendor and model	Intel Xeon Platinum 8259CL	Intel Xeon Platinum 8259CL	Intel Xeon Platinum 8259CL
Core count (per processor)	4	8	16
Core frequency (GHz)	2.50	2.50	2.50
Stepping	7	7	7
Hyper-Threading	Yes	Yes	Yes
Turbo	Yes	Yes	Yes
Number of vCPU per VM	8	16	64

Server configuration information	r5.2xlarge	r5.4xlarge	r5.16xlarge
Memory module(s)			
Total memory in system (GB)	64	128	512
NVMe memory present?	No	No	No
Total memory (DDR+NVMe RAM)	64	128	512
General hardware			
Storage: Network or direct-attached	Network	Network	Network
Network bandwidth per VM instance	Up to 10 Gbps	Up to 10 Gbps	20 Gbps
Storage bandwidth per VM instance	Up to 4,750 Mbps	4,750 Mbps	13,600 Mbps
Local storage (OS)			
Number of drives	1	1	1
Drive size (GB)	10	10	10
Drive information (speed, interface, type)	gp2, EBS, 100/3000 IOPS	gp2, EBS, 100/3000 IOPS	gp2, EBS, 100/3000 IOPS
Data drive			
Number of drives	1	1	1
Drive size (GB)	128	256	1024
Drive information (speed, interface, type)	io1, EBS, 3000 IOPS	io1, EBS, 6000 IOPS	io1, EBS, 24000 IOPS
Network adapter			
Vendor and model	Amazon Elastic Network Adapter	Amazon Elastic Network Adapter	Amazon Elastic Network Adapter
Number and type of ports	1x 10Gb	1x 10Gb	1x 20Gb

How we tested

Testing overview

For this project, we tested AWS instances featuring older Intel E5 v4 processors vs. newer 2nd generation Xeon processors. We ran a TPC-C-like workload on MySQL on the AWS instances to show the performance increase in terms of transactions per minute on OLTP databases that customers can expect to see using the newer instance series vs. the older.

Using our methodology to aid your own deployments

While the methodology below describes in great detail how we accomplished our testing, it is not a deployment guide. However, because we include many basic installation steps for operating systems and testing tools, reading our methodology may help with your own installation.

Creating the mysql instance

This section contains the steps we took to create our client instance for remotely running the HammerDB benchmark client software.

Create the mysql instance

1. Log into AWS and navigate to the AWS Management Console.
2. Click on EC2
3. Click Launch instance, then Launch instance in the dropdown to open the Launch Instance wizard.
4. In the search window, enter CentOS 8 and press enter.
5. On the AWS Marketplace tab, click the Select button next to "CentOS 8 (x86_64) - with Updates HVM" by Amazon Web Services CentOS-8-ec2-8.2.2004-20200923-1.x86_64-471d022d-974f-4f9c-8e39-b00d9b583833-ami-03b6a1d995f5a5146.4
6. On the Choose Instance Type tab, select mysql-{r4,r5}.{2,4,16}xlarge , then click "Next: Configure Instance Details".
7. On the Configure Instance tab, set the following:
    a. Number of instances: 1
    b. Purchasing option: Leave unchecked
    c. Network: Default VPC.
    d. Subnet: Choose the region you're working in.
    e. Auto-assign Public IP: Enable.
    f. Placement Group: Leave unchecked.
    g. Capacity Reservation: Open
    h. Domain join directory: No Directory
    i. IAM role: S3READ
    j. Shutdown behavior: Stop
    k. Click Next: Add Storage.
8. On the Add Storage tab, set the following:
    a. Size: 10GB
    b. Volume Type: gp2
    c. Delete on Termination: Checked
    d. Encryption: Not Encrypted
    e. Click Add New Volume
    f. Size: {64GB,128GB,512GB}
    g. Volume Type: io1
    h. IOPS: {2000,4000,16000}
    i. Delete on Termination: Checked
    j. Encryption: Not Encrypted
    k. Click Next: Add Tags
9. On the Configure Security Group tab, set the following:
    a. Select an existing security group
    b. Chose the group created for MySQL and HammerDB.
    c. Click Review and Launch.
10. On the Review Tab, click Launch.
11. Choose the appropriate option for the key pair, then click Launch Instances.

Create the HammerDB 3.3 client instance

1. Log into AWS and navigate to the AWS Management Console.
2. Click on EC2
3. Click Launch instance, then Launch instance in the dropdown to open the Launch Instance wizard.
4. In the search window, enter CentOS 8 and press enter.
5. On the AWS Marketplace tab, click the Select button next to "CentOS 8 (x86_64) - with Updates HVM" by Amazon Web Services. CentOS-8-ec2-8.2.2004-20200923-1.x86_64-471d022d-974f-4f9c-8e39-b00d9b583833-ami-03b6a1d995f5a5146.4
6. On the Choose Instance Type tab, select m5n.2xlarge , then click "Next: Configure Instance Details".
7. On the Configure Instance tab, set the following:
    a. Number of instances: 1
    b. Purchasing option: Leave unchecked
    c. Network: Default VPC.
    d. Subnet: Choose the region you're working in.
    e. Auto-assign Public IP: Enable.
    f. Placement Group: Leave unchecked.
    g. Capacity Reservation: Open
    h. Domain join directory: No Directory
    i. IAM role: None
    j. Shutdown behavior: Stop
    k. Click Next: Add Storage.
8. On the Add Storage tab, set the following:
    a. Size: 10GB
    b. Volume Type: gp2
    c. Delete on Termination: Checked
    d. Encryption: Not Encrypted
    e. Click Next: Add Tags
9. On the Configure Security Group tab, set the following:
    a. Select an existing security group
    b. Chose the group created for MySQL and HammerDB.
    c. Click Review and Launch.
10. On the Review Tab, click Launch.
11. Choose the appropriate option for the key pair, then click Launch Instances.

Configure CentOS 8 and install MySQL on mysql instance

1. Login to the MySQL instance via ssh.
2. Run the "mysql_host_prepare.sh" script: Disable SELINUX.

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

3. Shutdown the instance.

```
sudo poweroff
```

Configure CentOS 8 and install HammerDB 3.3 on mysql-client instance

1. Login to the hammerdb instance via ssh.
2. Disable SELINUX.

```
sudo sed -I 's/SELINUX=enforcing/SELINUX=disabled/' /etc/selinux/config
sudo setenforce 0
```

3. Turn off SSH strict host key checking.

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

4. Install required packages.

```
sudo dnf install -y epel-release
sudo dnf install -y wget vim tar zip unzip lz4 pigz nmon sysstat numactl ksh awscli
```

5. Download and install the MySQL repository.

```
sudo dnf install -y https://dev.mysql.com/get/mysql80-community-release-el8-1.noarch.rpm
```

6. Install the MySQL client.

```
sudo dnf --disablerepo=AppStream install -y mysql-community-client
```

7. Download HammerDB 3.3.

```
sudo wget https://github.com/TPC-Council/HammerDB/releases/download/v3.3/HammerDB-3.3-Linux.tar.gz
```

8. Extract the HammerDB package.

```
tar -xf HammerDB-3.3-Linux.tar.gz
```

9. Download and extract nmonchart tool.

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

10. Copy all scripts and config files in the appendix section to the HammerDB mysql-client instance.
11. Shutdown the instance.

```
sudo poweroff
```

Configure MySQL for database creation and backup

In this section, we list the various MySQL settings that we changed and the steps to do so.

Configure mysql instance and start the database

1. Login to the MySQL instance via ssh.
2. Copy the appropriate my.cnf config file from the appendix depending on your mysql instance and target database size. Example for 600 warehouse database:

```
cp -p /etc/my.cnf{,.bak}
cp -f my-600.cnf /etc/my.cnf
```

3. Run the mysql_host_prepare.sh script:

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

Create the database schema with HammerDB

1. Login to the mysql-client instance via ssh.
2. Navigate to the HammerDB directory.

```
cd HammerDB-3.3
```

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
```

Backup the database

1. Login to the mysql instance.
2. Shutdown the database.

```
systemctl stop mysqld
```

3. Delete the log files:

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

4. Backup 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.

Run the tests

In this section, we list the steps to run the HammerDB TPC-C-like test on the instances under test.

1. Login to 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. Additional parameters and config options can be tuned by modifying the script and editing the variables at the start of the file.

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

3. The script will prepare the mysql instance, restore the correct DB_SIZE, and run the test automatically. Results will be saved to the "results" folder in your home directory by default.
4. To parse all results run the parse_results.sh script.

```
./parse_results.sh
```

5. After destroying the virtual users, login to the postgresql instance and restore the database.
6. Terminate the mysql instance.
7. Repeat these steps 2 more times for a total of 3 runs. Do this for each mysql instance type and warehouse size combination.

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
dnf install -y wget vim tar zip unzip lz4 pigz nmon sysstat numactl awscli


#### Install MySQL ####
dnf install -y https://dev.mysql.com/get/mysql80-community-release-el8-1.noarch.rpm

dnf --disablerepo=AppStream install -y mysql-community-server
systemctl disable --now mysqld

#### 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
```

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
HOST_PREPARE=${APP}_host_prepare.sh
MYCNF=my-${WAREHOUSE_COUNT}.cnf
HDB_DIR=HammerDB-3.3/
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 ${HOST_PREPARE} ]; then
  echo "Missing host prepare script: ${HOST_PREPARE}"
  exit
fi

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 -p ${HOST_PREPARE} ${TEST_HOST}:host_prepare.sh
ssh ${TEST_HOST} "sudo ./host_prepare.sh"

scp ${MYCNF} ${TEST_HOST}:tmp-my.cnf
ssh ${TEST_HOST} "sudo systemctl stop ${APP}d ; sudo cp -vf tmp-my.cnf /etc/my.cnf"
date
echo "Downloading and extracting database: ETA $((WAREHOUSE_COUNT*2/5)) seconds"
#ssh ${TEST_HOST} "curl https://gyasi.s3.amazonaws.com/${APP}_tpcc_${WAREHOUSE_COUNT}warehouses_data.
tar.gz | pigz -d -c | sudo tar -C /mnt/${APP}data -xf- ; sync"
ssh ${TEST_HOST} "time aws s3 cp s3://gyasi/${APP}_tpcc_${WAREHOUSE_COUNT}warehouses_data.tar.gz - |
pigz -d -c | sudo tar -C /mnt/${APP}data -xf- ; sync"
datessh ${TEST_HOST} "sudo systemctl start ${APP}d && \
    sleep 10 && \
    sync && \
    sudo systemctl stop ${APP}d && \
    sudo umount -v /mnt/${APP}data && \
    sudo mount -v /mnt/${APP}data && \
    sudo systemctl start ${APP}d" || 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 ${HOST_PREPARE} ${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
rpm -qa | sort > ${RESULTS_DIR}/client_rpms.txt
curl -s http://169.254.169.254/latest/meta-data/placement/availability-zone > ${RESULTS_DIR}/client_
availability-zone.txt
aws ec2 get-launch-template-data --instance-id $(curl -s http://169.254.169.254/latest/meta-data/
instance-id) --query 'LaunchTemplateData' > ${RESULTS_DIR}/client_launch-template-data.json
# 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} 'curl -s https://download-ib01.fedoraproject.org/pub/fedora/linux/releases/33/
Everything/x86_64/os/Packages/u/util-linux-2.36-3.fc33.x86_64.rpm | rpm2cpio - | cpio -i --to-stdout
./usr/bin/lscpu 2> /dev/null > lscpu ; chmod +x lscpu ; ./lscpu' > ${RESULTS_DIR}/server_lscpu.txtssh
```

```
${TEST_HOST} 'rpm -qa | sort' > ${RESULTS_DIR}/server_rpms.txt
ssh ${TEST_HOST} 'curl -s http://169.254.169.254/latest/meta-data/placement/availability-zone' >
${RESULTS_DIR}/server_availability-zone.txt
aws ec2 get-launch-template-data --instance-id "$(ssh ${TEST_HOST} 'curl -s http://169.254.169.254/
latest/meta-data/instance-id')" --query 'LaunchTemplateData' > ${RESULTS_DIR}/server_launch-template-
data.json
# 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}
```

parse_results.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%\tWarehouses\
tUsers\tVulnerability\tServer RPMs\tClient RPMs\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
  awk '/mysql_count_ware/{printf("%d\t",$4)};/vuset *vu/{printf("%d\t",$3);exit}' ${result}/run_
hdb_*.tcl
  SERVER_VULNERABILITY_CKSUM=$(grep Vulnerability ${result}/server_lscpu.txt | sort | sha1sum)
  SERVER_RPM_CKSUM=$(sort ${result}/server_rpms.txt | sha1sum)
  CLIENT_RPM_CKSUM=$(sort ${result}/client_rpms.txt | sha1sum)
  echo -en "${SERVER_VULNERABILITY_CKSUM::7}\t${SERVER_RPM_CKSUM::7}\t${CLIENT_RPM_CKSUM::7}\t$(cat
${result}/server_availability-zone.txt)\t$(cat ${result}/client_availability-zone.txt)"  echo
done
```

hdb_tpcc_mysql_600wh.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 root
diset tpcc mysql_pass SecureP@ssw0rd1234
diset tpcc mysql_storage_engine innodb
diset tpcc mysql_partition true
diset tpcc mysql_driver timed

diset tpcc mysql_count_ware 600
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_1200wh.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 root
diset tpcc mysql_pass SecureP@ssw0rd1234
diset tpcc mysql_storage_engine innodb
diset tpcc mysql_partition true
diset tpcc mysql_driver timed

diset tpcc mysql_count_ware 1200
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
```

hdb_tpcc_mysql_4800wh.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 root
diset tpcc mysql_pass SecureP@ssw0rd1234
diset tpcc mysql_storage_engine innodb
diset tpcc mysql_partition true
diset tpcc mysql_driver timed

diset tpcc mysql_count_ware 4800
diset tpcc mysql_num_vu 128
diset tpcc mysql_rampup 2
diset tpcc mysql_duration 5

vuset logtotemp 1
```

```
loadscript
vuset vu 128
vucreate
vurun
wait_to_complete
vwait forever
```

my-600.cnf

```
[mysqld]
datadir=/mnt/mysqldata/data
default_authentication_plugin=mysql_native_password
socket=/var/lib/mysql/mysql.sock
log-error=/var/log/mysqld.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=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=1500
innodb_io_capacity_max=3000
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-1200.cnf

```
[mysqld]
datadir=/mnt/mysqldata/data
default_authentication_plugin=mysql_native_password
socket=/var/lib/mysql/mysql.sock
log-error=/var/log/mysqld.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=96000M #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=3000
innodb_io_capacity_max=6000
```

```
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-4800.cnf

```
[mysqld]
datadir=/mnt/mysqldata/data
default_authentication_plugin=mysql_native_password
socket=/var/lib/mysql/mysql.sock
log-error=/var/log/mysqld.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=64 #scale
innodb_open_files=4000

# buffers
innodb_buffer_pool_size=384000M #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=12000
innodb_io_capacity_max=24000
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='%'
```

Determining CPU vulnerability mitigation

The information below shows the Intel processor mitigation settings on the AWS instances.

r4.16xlarge

```
Vulnerability Itlb multihit:     KVM: Vulnerable
Vulnerability L1tf:              Mitigation; PTE Inversion
Vulnerability Mds:               Vulnerable: Clear CPU buffers attempted, no microcode; SMT Host
state unknown
Vulnerability Meltdown:          Mitigation; PTI
Vulnerability Spec store bypass: Vulnerable
Vulnerability Spectre v1:        Mitigation; usercopy/swapgs barriers and __user pointer sanitization
Vulnerability Spectre v2:        Mitigation; Full generic retpoline, STIBP disabled, RSB filling
Vulnerability Srbds:             Not affected
Vulnerability Tsx async abort:   Vulnerable: Clear CPU buffers attempted, no microcode; SMT Host
state unknown
```

r5.16xlarge

```
Vulnerability Itlb multihit:     KVM: Vulnerable
Vulnerability L1tf:              Mitigation; PTE Inversion
Vulnerability Mds:               Vulnerable: Clear CPU buffers attempted, no microcode; SMT Host
state unknown
Vulnerability Meltdown:          Mitigation; PTI
Vulnerability Spec store bypass: Vulnerable
Vulnerability Spectre v1:        Mitigation; usercopy/swapgs barriers and __user pointer sanitization
Vulnerability Spectre v2:        Mitigation; Full generic retpoline, STIBP disabled, RSB filling
Vulnerability Srbds:             Not affected
Vulnerability Tsx async abort:   Not affected
```

Read the report at http://facts.pt/L5S3md0 ▶

This project was commissioned by Intel.