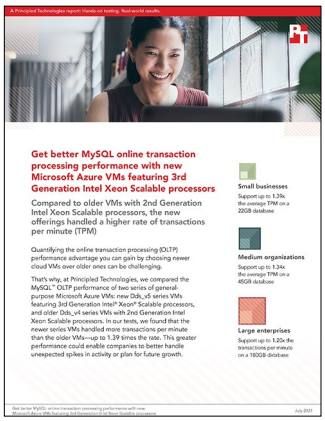

The science behind the report:

Get better MySQL online transaction processing performance with new Microsoft Azure VMs featuring 3rd Generation 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 Get better MySQL online transaction processing performance with new Microsoft Azure VMs featuring 3rd Generation Intel Xeon Scalable processors.

We concluded our hands-on testing on May 27, 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 May 19, 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

Microsoft Azure database	Average transactions per minute processed
8 vCPU results	
D8ds_v5	413,934
D8ds_v4	296,524
16 vCPU results	
D16ds_v5	790,341
D16ds_v4	586,538
64 vCPU results	
D64ds_v5	2,338,305
D64ds_v4	1,937,500

System configuration information

Table 2: Details for the Microsoft Azure Dds_v5 series VMs we tested.

System configuration information	D8ds_v5 VM	D16ds_v5 VM	D64ds_v5 VM
Tested by	Principled Technologies	Principled Technologies	Principled Technologies
Test date	05/26/2021	05/27/2021	05/26/2021
CSP / Region	East US 2	East US 2	East US 2
Workload & version	HammerDB v4.1 TPROC-C	HammerDB v4.1 TPROC-C	HammerDB v4.1 TPROC-C
WL specific parameters	250 warehouses 24 users	500 warehouses 48 users	2,000 warehouses 196 users
Iterations and result choice	3 runs, median	3 runs, median	3 runs, median
Server platform	D8ds_v5	D16ds_v5	D64ds_v5
BIOS name and version	Microsoft Corporation Hyper-V UEFI Release v4.1, 10/27/2020	Microsoft Corporation Hyper-V UEFI Release v4.1, 10/27/2020	Microsoft Corporation Hyper-V UEFI Release v4.1, 10/27/2020
Operating system name and version/build number	Ubuntu Server 20.04 LTS	Ubuntu Server 20.04 LTS	Ubuntu Server 20.04 LTS
Date of last OS updates/ patches applied	05/19/2021	05/19/2021	05/19/2021
Processor			
Number of processors	1	1	2
Vendor and model	Intel® Xeon® Platinum 8370C	Intel Xeon Platinum 8370C	Intel Xeon Platinum 8370C
VM core count (per processor)	4	8	16
Core frequency (GHz)	2.80	2.80	2.80
Stepping	6	6	6
Hyper-Threading	Yes	Yes	Yes
Turbo	Yes	Yes	Yes
Total number of vCPUs per VM	8	16	64
Memory module(s)			
Total memory in system (GB)	32	64	256
NVMe memory present?	No	No	No
Total memory (DDR+NVMe RAM)	32	64	256
General hardware			
Storage: NW or Direct Att / Instance	NW Att	NW Att	NW Att
Network BW / Instance	N/A	N/A	N/A
Storage BW / Instance	N/A	N/A	N/A

System configuration information	D8ds_v5 VM	D16ds_v5 VM	D64ds_v5 VM
Local storage			
OS			
Number of drives	1	1	1
Drive size (GB)	30	30	30
Drive information (speed, interface, type)	Standard SSD	Standard SSD	Standard SSD
Temporary drive			
Number of drives	1	1	1
Drive size (GB)	300	600	2,400
Data drive			
Number of drives	1	1	1
Drive size (GB)	64	128	512
Drive information (speed, interface, type)	Ultra, 2,000 IOPS	Ultra, 4,000 IOPS	Ultra, 16,000 IOPS
Network adapter			
Vendor and model	Microsoft Hyper-V Network Adapter	Microsoft Hyper-V Network Adapter	Microsoft Hyper-V Network Adapter
Number and type of ports	1x 100Gb	1x 100Gb	1x 100Gb

Table 3: Details for the Microsoft Azure Dds_v4 series VMs we tested.

System configuration information	D8ds_v4 VM	D16ds_v4 VM	D64ds_v4 VM
Tested by	Principled Technologies	Principled Technologies	Principled Technologies
Test date	05/25/2021	05/27/2021	05/26/2021
CSP / Region	East US 2	East US 2	East US 2
Workload & version	HammerDB v4.1 TPROC-C	HammerDB v4.1 TPROC-C	HammerDB v4.1 TPROC-C
WL specific parameters	250 Warehouses 24 users	500 Warehouses 48 users	2,000 Warehouses 196 users
Iterations and result choice	3 runs, median	3 runs, median	3 runs, median
Server platform	D8ds_v4	D16ds_v4	D64ds_v4
BIOS name and version	Microsoft Corporation Hyper-V UEFI Release v4.1, 10/27/2020	Microsoft Corporation Hyper-V UEFI Release v4.1, 10/27/2020	Microsoft Corporation Hyper-V UEFI Release v4.1, 10/27/2020
Operating system name and version/build number	Ubuntu Server 20.04 LTS	Ubuntu Server 20.04 LTS	Ubuntu Server 20.04 LTS
Date of last OS updates/ patches applied	05/19/2021	05/19/2021	05/19/2021

System configuration information	D8ds_v4 VM	D16ds_v4 VM	D64ds_v4 VM
Processor			
Number of processors	1	1	2
Vendor and model	Intel Xeon Platinum 8272CL	Intel Xeon Platinum 8272CL	Intel Xeon Platinum 8272CL
Core count (per processor)	24	24	24
Core frequency (GHz)	2.60	2.60	2.60
Stepping	7	7	7
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)	32	64	256
NVMe memory present?	No	No	No
Total memory (DDR+NVMe RAM)	32	64	256
General hardware			
Storage: NW or Direct Att /Instance	NW Att	NW Att	NW Att
Network BW / Instance (Mbps)	4,000	8,000	30,000
Storage BW / Instance (MB/s)	192	384	1,200
Local storage			
OS			
Number of drives	1	1	1
Drive size (GB)	30	30	30
Drive information (speed, interface, type)	Standard SSD	Standard SSD	Standard SSD
Temporary drive			
Number of drives	1	1	1
Drive size (GB)	300	600	2400
Data drive			
Number of drives	1	1	1
Drive size (GB)	64	128	512
Drive information (speed, interface, type)	Ultra, 2000 IOPS	Ultra, 4000 IOPS	Ultra, 16000 IOPS
Network adapter			
Vendor and model	Microsoft Hyper-V Network Adapter	Microsoft Hyper-V Network Adapter	Microsoft Hyper-V Network Adapter
Number and type of ports	1x 50Gb	1x 50Gb	1x 50Gb

How we tested

Testing overview

For this project, we tested Azure VMs featuring 2nd Generation Intel Xeon Scalable processors vs. 3rd Generation Intel Xeon Scalable processor versions. We ran a TPROC-C workload on MySQL on the Azure VMs to show the performance increase in terms of transactions per minute on OLTP databases that customers can expect to see using the newer VM series vs. the older.

Creating the mysql VM and mysql-client VMs

This section contains the steps we took to create a VM under test and a client VM for remotely running the HammerDB benchmark client software.

Creating the HammerDB 4.1 client VM

1. Log into the Azure Portal, and navigate to the Virtual Machines service.
2. To open the Add VM wizard, click Add.
3. On the Basics tab, set the following:
    a. Choose your Subscription.
    b. Choose your Resource group.
    c. Name the Virtual Machine.
    d. Choose your Region. We used East US 2.
    e. Leave the Availability options set to No infrastructure redundancy required.
    f. Click See all images.
    g. In the Search field, enter Rogue Wave.
    h. From the CentOS-based drop-down menu, select CentOS-based 8.3 - Gen2.
    i. Leave Azure Spot instance set to No.
    j. Select the VM size you wish to use. We used Standard D8ds_v4.
    k. Leave the Authentication type set to SSH public key.
    l. Either choose a new Username or leave the default.
    m. For the SSH public key source, choose Generate new key pair.
    n. Enter a name for the Key pair name.
    o. Leave Public inbound ports set to Allow selected ports.
    p. For Select inbound ports, choose SSH (22).
4. On the Disks tab, set the following:
    a. For the OS disk type, choose Standard SSD.
    b. Leave the default Encryption type.
5. On the Networking tab, set the following:
    a. Choose your Virtual network.
    b. To create a new Public IP, choose Create new.
    c. Leave the rest of the settings at defaults.
6. On the Management tab, set the following:
    a. Choose your Diagnostics storage account.
    b. Leave the rest set to defaults.
7. On the Advanced tab, leave all defaults.
8. On the Tags tab, add any tags you wish to use.
9. On the Review + create tab, review your settings, and click Create.

Creating the mysql VM under test

1. Log into the Azure Portal, and navigate to the Virtual Machines service.
2. To open the Add VM wizard, click Add.
3. On the Basics tab, set the following:
    a. Choose your Subscription.
    b. Choose your Resource group.
    c. Name the Virtual Machine.
    d. Choose your Region. We used US East 2.
    e. From the Availability options drop-down menu, choose Availability Zone.
    f. For Availability Zone, select Zone 2. This is to ensure access to Ultra Disks.
    g. Click See all images.
    h. In the Search field, enter Rogue Wave.
    i. Select "CentOS-based 8.3 - Gen2" from the CentOS-based dropdown menu.
    j. Leave Azure Spot instance set to No.
    k. Select the VM size you wish to use. We used Standard D{8,16,64}ds_v{4,5}.
    l. Leave the Authentication type set to SSH public key.
    m. Either choose a new Username or leave the default.
    n. Choose Generate new key pair for the SSH public key source, or use your previously created key.
    o. Leave Public inbound ports set to Allow selected ports.
    p. For Select inbound ports, choose SSH (22).
4. On the Disks tab, set the following:
    a. For the OS disk type, choose Standard SSD.
    b. Leave the default Encryption type.
    c. Check the Enable Ultra Disk compatibility box.
    d. Click Create and attach a new disk.
    e. Leave Name and Source type default, and click Change size.
    f. Under Disk SKU, select Ultra Disk (locally-redundant storage).
    g. Choose your size {64GB, 128GB, 512GB}.
    h. Choose IOPS {2000, 4000, 16000}, and set Throughput to 500 MB/s.
    i. Click OK, and click Next: Networking.
5. On the Networking tab, set the following:
    a. Choose your Virtual network.
    b. Choose Create new to create a new Public IP.
    c. Leave the rest of the settings at defaults.
6. On the Management tab, set the following:
    a. Choose your Diagnostics storage account.
    b. Leave the rest set to defaults.
7. On the Advanced tab, leave all defaults.
8. On the Tags tab, add any tags you wish to use.
9. On the Review + create tab, review your settings, and click Create.

Configuring CentOS 8 and installing MySQL on the mysql VM

1. Log into the MySQL VM via ssh.
2. Disable the older default MySQL module:

```
sudo dnf module disable mysql
```

3. Run mysql_host_prepare.sh:

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

4. After installing MySQL, find your temporary root password:

```
sudo grep 'temporary password' /var/log/mysqld.log
```

5. Log into mysql as root:

```
Mysql -u root -p
```

6. Enter the temporary password from the previous step.
7. Change the root user password:

```
ALTER USER 'root'@'localhost' IDENTIFIED BY '[password]';
```

8. 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;
```

9. Shut down the VM.

```
sudo poweroff
```

Creating a baseline image of the mysql VM

Creating a snapshot of your baseline VM

1. In your Azure portal, navigate to the Snapshots service.
2. To open the Snapshot wizard, click Add.
3. On the Basics tab, set the following:
    a. Choose your Subscription.
    b. Choose your Resource group.
    c. Enter a name for your snapshot.
    d. Choose your Region.
    e. For the Snapshot type, select Full - make a complete read-only copy of the selected disk.
    f. Choose the OS disk for your baseline VM.
    g. Choose Standard HDD for the Storage type.
4. On the Encryption tab, leave all defaults.
5. On the Tags tab, add any tags you wish to use.
6. On the Review + create tab, review your settings, and click Create.

Creating your image with the baseline snapshot

To create an image, you must first have a Shared Image Gallery. The steps below will walk you through the creation of the gallery as well as the image creation steps. Once you have created your gallery, you will not need to do so again to add new images.

1. In your Azure portal, navigate to the Shared image galleries service.
2. To open the Add gallery wizard, click Add.
3. On the Basics tab, set the following:
    a. Choose your Subscription.
    b. Choose your Resource.
    c. Name your gallery.
    d. Choose your Region.
    e. Enter a Description if you like.
4. On the Tags tab, add any tags you wish to use.
5. On the Review + create tab, review your settings, and click Create.
6. Click your new image gallery, and click Add new image definition to open the wizard.
7. On the Basics tab, set the following:
    a. Set the Operating System to Windows.
    b. Set the VM generation to Gen 2.
    c. Set the Operation system state to Specialized.
    d. Enter your desired values for the Publisher, Offer, and SKU entries.

8. Skip the Version tab.
9. Skip the Publishing options tab.
10. On the Tags tab, add any tags you wish to use.
11. On the Review + create tab, review your settings, and click Create.
12. Click on the image definition you've created, and click Add version to open the wizard.
13. On the Basics tab, set the following:
    a. Enter a version number, such as 1.0.0.
    b. Choose the OS disk snapshot of the baseline VM you created.
    c. Leave the rest as defaults.
14. On the Encryption tab, leave defaults.
15. On the Tags tab, add any tags you wish to use.
16. On the Review + create tab, review your settings, and click Create.

Configuring CentOS 8 and installing HammerDB 4.1 on the client VM

1. Log into the hammerdb VM 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/configInstall required packages.
sudo dnf install -y epel-release
sudo dnf install -y wget vim tar zip unzip lz4 pigz nmon sysstat numactl kshDisable the older
    default MySQL module:
sudo dnf module disable mysql
```

4. Download and install the MySQL repository:

```
sudo dnf install -y https://dev.mysql.com/get/mysql80-community-release-el8-1.noarch.rpmInstall
    the MySQL client.
sudo dnf --disablerepo=AppStream install -y mysql-community-clientDownload HammerDB 3.3.
sudo wget https://github.com/TPC-Council/HammerDB/releases/download/v4.1/HammerDB-4.1-Linux.tar.gz
```

5. Extract the HammerDB package:

```
tar -xf HammerDB-4.1-Linux.tar.gzDownload and extract nmonchart tool.
wget https://sourceforge.net/projects/nmon/files/nmonchart40.tar
    tar -xf nmonchart40.tar ./nmonchartCopy all scripts and config files in the appendix section to the
    HammerDB mysql-client VM.
```

6. Shut down the VM:

```
sudo poweroff
```

Configuring MySQL for database creation and backup

In this section, we list the various MySQL settings that we changed and the steps to do so. To see the MySQL configuration files used for each VM type, see the Scripts section. To see the exact settings for each VM, please refer to Table 4.

Configuring the MySQL VM and starting the database

1. Log into the mysql VM via SSH.
2. Copy the appropriate my.cnf config file from the appendix depending on your mysql VM and target database size. Example for 250 warehouse database:

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

3. Run the mysql_host_prepare.sh script:

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

Creating the database schema with HammerDB

1. Log into the mysql-client VM via ssh.
2. Navigate to the HammerDB directory:

```
cd HammerDB-4.1Start hammerdbcli.
./hammerdbcli
```

3. 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
```

4. Build the schema:

```
buildschema
```

Backing up the database

1. Log into the mysql VM.
2. Shut down the database:

```
systemctl stop mysqldDelete the log files:
cd /mnt/mysqldata/
    rm -f data/ib_logfile*
```

3. Back up the database:

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

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

Generating and exchanging SSH keys

1. On the VM under test, run the following command to create a new SSH key pair:

```
ssh-keygen
```

2. Press enter four times to save the key pair to the default location with no password.
3. Once the key pair has been generated, run the following command to obtain your new public SSH key:

```
cat .ssh/id_rsa.pub
```

4. Copy the output key to your clipboard.
5. Log into the mysql-client VM.
6. Open the authorized_key file with the following command, and copy the public SSH key from the VM under test into the file:

```
sudo vim .ssh/authorized_keys
```

7. Repeat steps 1 through 6 for the mysql-client VM, copying the public SSH key to the authorized_keys file on the VM under test.

Running the tests

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

1. Log into the HammerDB mysql-client VM via SSH.
2. Execute the run_test.sh script substituting IP_ADDRESS with the Azure private IP of the mysql VM 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>The script will prepare the mysql VM, restore the correct DB_
    SIZE, and run the test automatically. Results will be saved to the "results" folder in your home
    directory by default.
```

3. To parse all results, run the parse_results.sh script: ./parse_results.sh
4. After destroying the virtual users, log into the mysql VM and restore the database.
5. Reboot the mysql VM.
6. Repeat these steps two more times for a total of three runs. Do this for each mysql VM type and warehouse size combination.

Table 4: Specifications for each of the VMs we tested. Source: Principled Technologies.

	D8ds_{v4,v5}	D16ds_{v4,v5}	D64ds_{v4,v5}
Number of vCPU	8	16	64
Memory (GB)	32	64	256
Data disk (size, performance)	64GB, 2,000 IOPS	128GB, 4,000 IOPS	512GB, 16,000 IOPS
Number of warehouses	250	500	2000
Number of users	24	48	196
Warmup (min)	5	5	10
Runtime (min)	10	10	20

Determining CPU vulnerability mitigation

These are the specific CPU mitigation settings that Azure reported at the time of testing.

3rd Generation Intel Xeon Scalable processor mitigation information

```
Vulnerability Itlb multihit: Not affected
Vulnerability L1tf: Not affected
Vulnerability Mds: Not affected
Vulnerability Meltdown: Not affected
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
```

2nd Generation Intel Xeon Scalable processor mitigation information

```
Vulnerability Itlb multihit:     KVM: Mitigation: Split huge pages
Vulnerability L1tf:              Not affected
Vulnerability Mds:               Not affected
Vulnerability Meltdown:          Not affected
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
```

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

#### 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})

Process MySQL database transactions in Amazon Web Services faster with newer May
2021 (Revised) | 10
```

```
instances powered by 2nd Generation Intel Xeon Scalable processors – Cascade Lake

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"

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} "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_

av.txt

# Copy server info to results folder

Process MySQL database transactions in Amazon Web Services faster with newer May
2021 (Revised) | 11

instances powered by 2nd Generation Intel Xeon Scalable processors – Cascade Lake

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} '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_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}
```

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%\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

SERVER_CKSUM=$(sort ${result}/server_rpms.txt | sha1sum)

CLIENT_CKSUM=$(sort ${result}/client_rpms.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


hdb_tpcc_mysql_200wh.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 200

diset tpcc mysql_num_vu 12

diset tpcc mysql_rampup 2

diset tpcc mysql_duration 5

vuset logtotemp 1

loadscript

vuset vu 12

vucreate

vurun

wait_to_complete

vwait forever
```

hdb_tpcc_mysql_400wh.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 400
diset tpcc mysql_num_vu 24
diset tpcc mysql_rampup 2
diset tpcc mysql_duration 5
vuset logtotemp 1
loadscript
vuset vu 24
vucreate
vurun
wait_to_complete
vwait forever
hdb_tpcc_mysql_1600wh.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 1600

diset tpcc mysql_num_vu 96

diset tpcc mysql_rampup 2

diset tpcc mysql_duration 5

vuset logtotemp 1

loadscript

vuset vu 96

vucreate

vurun

wait_to_complete

vwait forever

my-200.cnf

[mysqld]

datadir=/mnt/mysqldata/data

default_authentication_plugin=mysql_native_password

Process MySQL database transactions in Amazon Web Services faster with newer May
2021 (Revised) | 14

instances powered by 2nd Generation Intel Xeon Scalable processors – Cascade Lake

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=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-400.cnf

```
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='%'
```

my-1600.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=192000M #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=8000
innodb_io_capacity_max=16000
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='%'
```

Read the report at http://facts.pt/9FUbQFr ▶

This project was commissioned by Intel.

Facts matter.®