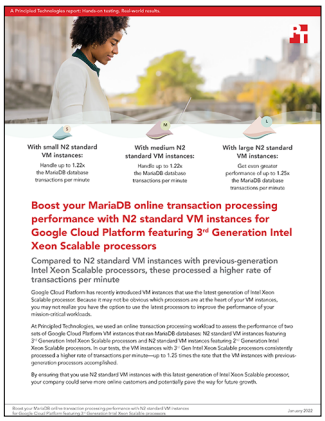


The science behind the report:

# Boost your MariaDB online transaction processing performance with N2 standard VM instances for Google Cloud Platform 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 Boost your MariaDB online transaction processing performance with N2 standard VM instances for Google Cloud Platform featuring 3rd Generation Intel Xeon Scalable processors.

We concluded our hands-on testing on November 15, 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 November 9, 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 online transaction processing tests comparing HammerDB OLTP workload performance on Google Cloud Platform N2 standard VM instances featuring 3rd Generation Intel Xeon Scalable processors to N2 standard VM instances featuring 2nd Generation Intel Xeon Scalable processors. Comparison of transactions per minute (TPM) and new orders per minute (NOPM).

| n2-standard-8 results | | |
|---|---|---|
| Instance | TPM | NOPM |
| n2-standard-8 (2nd Gen Intel) | 276,187 | 91,039 |
| n2-standard-8 (3rd Gen Intel) | 338,312 | 111,843 |
| **n2-standard-16 results** | | |
| Instance | TPM | NOPM |
| n2-standard-16 (2nd Gen Intel) | 546,891 | 180,550 |
| n2-standard-16 (3rd Gen Intel) | 668,416 | 220,834 |
| **n2-standard-64 results** | | |
| Instance | TPM | NOPM |
| n2-standard-64 (2nd Gen Intel) | 815,836 | 269,172 |
| n2-standard-64 (3rd Gen Intel) | 1,020,436 | 336,583 |

# System configuration information

Table 2: Detailed information on the 2nd Generation Intel Xeon Scalable processor-supported N2 standard VM instances we tested.

| System configuration information | n2-standard-8 | n2-standard-16 | n2-standard-64 |
|---|---|---|---|
| Tested by | Principled Technologies | Principled Technologies | Principled Technologies |
| Test date | 11/10/2021 | 11/10/2021 | 11/15/2021 |
| CSP / Region | us-central1-a | us-central1-a | us-central1-a |
| Workload & version | HammerDB v4.2 TPROC-C | HammerDB v4.2 TPROC-C | HammerDB v4.2 TPROC-C |
| WL specific parameters | 250 Warehouses<br>16 virtual users | 500 Warehouses<br>32 virtual users | 2000 Warehouses<br>128 virtual users |
| Iterations and result choice | 3 runs, median | 3 runs, median | 3 runs, median |
| Server platform | n2-standard-8 | n2-standard-16 | n2-standard-64 |
| BIOS name and version | Google 01/01/2011 | Google 01/01/2011 | Google 01/01/2011 |
| Operating system name and version/build number | Ubuntu 20.04 LTS | Ubuntu 20.04 LTS | Ubuntu 20.04 LTS |
| Date of last OS updates/ patches applied | 11/09/2021 | 11/09/2021 | 11/09/2021 |
| Processor | | | |
| Number of processors | 2 | 2 | 2 |
| Vendor and model | Intel® Xeon® Cascade Lake | Intel Xeon Cascade Lake | Intel Xeon Cascade Lake |
| VM core count (per processor) | 4 | 8 | 32 |
| Core frequency (GHz) | 2.80 | 2.80 | 2.80 |
| Stepping | 7 | 7 | 7 |
| 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 HW | | | |
| Storage: NW or Direct Att / Instance | NW | NW | NW |
| Network BW / Instance | 16 Gbps | 32 Gbps | 32 Gbps |
| Storage BW / Instance | 6,400 Mbps | 9,600 Mbps | 9,600 Mbps |

| System configuration information | n2-standard-8 | n2-standard-16 | n2-standard-64 |
|---|---|---|---|
| Local storage | | | |
| OS | | | |
| Number of drives | 1 | 1 | 1 |
| Drive size (GB) | 10 | 10 | 10 |
| Drive information (speed, interface, type) | SSD persistent disk | SSD persistent disk | SSD persistent disk |
| Data drive | | | |
| Number of drives | 1 | 1 | 1 |
| Drive size (GB) | 64 | 128 | 512 |
| Drive information (speed, interface, type) | SSD persistent disk | SSD persistent disk | SSD persistent disk |
| Network adapter | | | |
| Vendor and model | Google VirtIO Ethernet Adapter | Google VirtIO Ethernet Adapter | Google VirtIO Ethernet Adapter |
| Number and type of ports | 1x 100Gb | 1x 100Gb | 1x 100Gb |

Table 3: Detailed information on the 3rd Generation Intel Xeon Scalable processor-supported N2 standard VM instances we tested.

| System configuration information | n2-standard-8 | n2-standard-16 | n2-standard-64 |
|---|---|---|---|
| Tested by | Principled Technologies | Principled Technologies | Principled Technologies |
| Test date | 11/10/2021 | 11/10/2021 | 11/15/2021 |
| CSP / Region | us-central1-a | us-central1-a | us-central1-a |
| Workload & version | HammerDB v4.2 TPROC-C | HammerDB v4.2 TPROC-C | HammerDB v4.2 TPROC-C |
| WL specific parameters | 250 Warehouses<br>16 virtual users | 500 Warehouses<br>32 virtual users | 2000 Warehouses<br>128 virtual users |
| Iterations and result choice | 3 runs, median | 3 runs, median | 3 runs, median |
| Server platform | n2-standard-8 | n2-standard-16 | n2-standard-64 |
| BIOS name and version | Google 01/01/2011 | Google 01/01/2011 | Google 01/01/2011 |
| Operating system name and version/build number | Ubuntu 20.04 LTS | Ubuntu 20.04 LTS | Ubuntu 20.04 LTS |
| Date of last OS updates/ patches applied | 11/09/2021 | 11/09/2021 | 11/09/2021 |
| Processor | | | |
| Number of processors | 2 | 2 | 2 |
| Vendor and model | Intel® Xeon Ice Lake | Intel Xeon Ice Lake | Intel Xeon Ice Lake |
| VM core count (per processor) | 4 | 8 | 32 |
| Core frequency (GHz) | 2.60 | 2.60 | 2.60 |
| Stepping | 6 | 6 | 6 |
| Hyper-Threading | Yes | Yes | Yes |
| Turbo | Yes | Yes | Yes |
| Total number of vCPUs per VM | 8 | 16 | 64 |

| System configuration information | n2-standard-8 | n2-standard-16 | n2-standard-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 HW | | | |
| Storage: NW or Direct Att / Instance | NW | NW | NW |
| Network BW / Instance | 16 Gbps | 32 Gbps | 32 Gbps |
| Storage BW / Instance | 6,400 Mbps | 9,600 Mbps | 9,600 Mbps |
| Local storage | | | |
| OS | | | |
| Number of drives | 1 | 1 | 1 |
| Drive size (GB) | 10 | 10 | 10 |
| Drive information (speed, interface, type) | SSD persistent disk | SSD persistent disk | SSD persistent disk |
| Data drive | | | |
| Number of drives | 1 | 1 | 1 |
| Drive size (GB) | 64 | 128 | 512 |
| Drive information (speed, interface, type) | SSD persistent disk | SSD persistent disk | SSD persistent disk |
| Network adapter | | | |
| Vendor and model | Google VirtIO Ethernet Adapter | Google VirtIO Ethernet Adapter | Google VirtIO Ethernet Adapter |
| Number and type of ports | 1x 100Gb | 1x 100Gb | 1x 100Gb |

# How we tested

## Testing overview

For this project, we compared GCP VM instances featuring 3rd Generation Intel Xeon Scalable processors to the same instances featuring 2nd Generation Intel Xeon Scalable processors. We ran an OLTP workload on MariaDB on the GCP VM instances to show the performance increase in terms of transactions per minute on OLTP databases that customers can expect to see using the newer VM instance series vs. the older.

## Creating the mariadb and client VM instances

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

### Create the mariadb VM instance

1. Log into Google Cloud, and click Go to console.
2. Click Compute engine, and click VM instances.
3. Click Create.
4. In the left window, select New VM instance.
5. Add the following information:
    a. Name: Name your VM instance.
    b. Labels: Use any appropriate labels.
    c. Region: Select your desired region. We used us-central1.
    d. Zone: Select your desired zone.
    e. Machine Configuration:
    f. Machine family: General-purpose
    g. Series: N2
    h. Machine type: n2-standard-{8,16,64}
    i. CPU platform: {Cascade Lake, Ice Lake} or above
    j. Keep Turn on display device unchecked.
    k. Keep Confidential VM Service and Container unchecked.
    l. Boot Disk, click Change:
        i. Operating System: Ubuntu
        ii. Version: Ubuntu 20.04 LTS
        iii. Boot disk type: SSD persistent disk
        iv. Size: 10GB
        v. Click Select
    m. Identity and API access: App Engine default service account.
    n. Firewall: Check Allow HTTP traffic and Allow HTTPs traffic.
    o. Expand Networking, Disks, Security, Management, Sole-Tenancy
    p. Expand Disks
    q. Click Add New Disk:
        i. Name: Name the disk
        ii. Disk source type: Blank disk
        iii. Disk type: SSD persistent disk
        iv. Size: {64,128,512}GB
        v. Labels: add appropriate label
        vi. Mode: Read/write
        vii. Deletion rule: Delete disk
        viii. Click Save.
    r. Click Create.

## Creating the HammerDB 4.2 client VM instance

1. Log into Google Cloud, and click Go to console.
2. Click Compute engine, and click VM instances.
3. Click Create.
4. In the left window, select New VM instance.
5. Add the following information:
    a. Name: Name your VM instance.
    b. Labels: Use any appropriate labels.
    c. Region: Select your desired region. We used us-central1
    d. Zone: Select your desired zone.
    e. Machine Configuration:
    f. Machine family: General-purpose
    g. Series: N2
    h. Machine type: n2-standard-8
    i. CPU platform: Automatic
    j. Keep Turn on display device unchecked.
    k. Keep Confidential VM Service and Container unchecked.
    l. Boot Disk, click Change:
        i. Operating System: Ubuntu
        ii. Version: Ubuntu 20.04 LTS
        iii. Boot disk type: SSD persistent disk
        iv. Size: 10GB
        v. Click Select
    m. Identity and API access: App Engine default service account.
    n. Firewall: Check Allow HTTP traffic and Allow HTTPs traffic.
    o. Click Create.

## Configuring Ubuntu 20.04 LTS and installing MariaDB on the mariadb VM instance

1. Log into the mariadb VM instance via ssh.
2. Run the mariadb_host_prepare.sh script:

```
sudo ./mariadb_host_prepare.sh
```

3. Install the appropriate MariaDB repository for x86 architecture on Ubuntu 20.04 from the following URL: https://mariadb.org/download/?tab=repo-config.
4. Install MariaDB:

```
sudo apt install MariaDB-server
```

5. Stop the MariaDB service:

```
sudo systemctl stop mariadb
```

6. Copy the mariadb data directory to your data disk:

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

7. Using the appendix on page 10 of this document, copy the appropriate my.cnf config file for your MariaDB instance and target database size. Example for 250 warehouse database:

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

8. Start the MariaDB service:

```
sudo systemctl start mariadb
```

9. Log into the MariaDB instance as the root user:

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

10. Create a new user. Name it `mysql` and give it full permissions:
    a. CREATE USER 'mysql'@'localhost' IDENTIFIED BY '[password]';
    b. GRANT ALL PRIVILEGES ON *.* TO 'mysql'@'localhost' → WITH GRANT OPTION;
    c. CREATE USER 'mysql'@'%' IDENTIFIED BY '[password]';
    d. GRANT ALL PRIVILEGES ON *.* TO 'mysql'@'%' → WITH GRANT OPTION;
11. Shut down the instance.

```
sudo poweroff
```

## Configuring Ubuntu and installing HammerDB 4.2 on MariaDB-client VM instance

1. Log into the client VM instance via SSH.
2. Turn off SSH strict host key checking.

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

3. Install the required packages.

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

4. From the following URL, install the appropriate MariaDB repository for x86 architecture on Ubuntu 20.04: https://mariadb.org/download/?tab=repo-config
5. Install MariaDB:

```
sudo apt install MariaDB-server
```

6. Download HammerDB 4.2.

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

7. Extract the HammerDB package.

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

8. Download and extract nmonchart tool.

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

9. Copy all scripts and config files in the page 10 appendix section to the HammerDB mariadb-client VM instance.
10. Shut down the instance.

```
sudo poweroff
```

## Creating the database schema with HammerDB

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

```
cd HammerDB-4.2
```

3. Start hammerdbcli:

```
./hammerdbcli
```

4. Set the following variables:

```
dbset db maria
    diset connection maria_host <IP_ADDRESS>
    diset tpcc maria_user mysql
    diset tpcc maria_pass <Password>
    diset tpcc maria_count_ware <DB_SIZE>
    diset tpcc maria_partition true
    diset tpcc maria_num_vu 8
    diset tpcc maria_storage_engine innodb
```

5. Build the schema:

```
buildschema
```

## Backing up the database

1. Log into the mariadb VM instance.
2. Shut down the database:

```
systemctl stop mariadb
```

3. Delete the log files:

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

4. Back up the database:

```
tar -cf- data/ | pigz -9 -c > mariadb_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 VM instances under test. As each VM instance had different hardware and database sizes, please refer to the table below to see the number of users to run on each VM instance.

1. Log into the hammerdb mariadb-client VM instance via SSH.
2. Execute the run_test.sh script, substituting IP_ADDRESS with the GCP private IP of the mariadb VM instance and DB_SIZE with the number of warehouses. You may 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>
```

3. The script will prepare the mariadb VM instance, restore the correct DB_SIZE, and run the test automatically. By default, the script will save results to the "results" folder in your home directory.
4. To parse all results, run the parse_results.sh script.

```
./parse_results.sh
```

5. Reboot the mariadb VM instance and client VM instance.
6. Repeat these steps twice more for a total of 3 runs. Do this for each mariadb VM instance type and warehouse size combination.

| Instance type | n2-standard-8 | n2-standard-16 | n2-standard-64 |
|---|---|---|---|
| Number of vCPU | 8 | 16 | 64 |
| Memory (GB) | 32 | 64 | 256 |
| Data disk (size, IOPS) | 64 | 128 | 512 |
| Number of warehouses | 250 | 500 | 2,000 |
| Number of users | 16 | 32 | 128 |
| Warmup (min) | 5 | 5 | 5 |
| Runtime (min) | 10 | 10 | 10 |

## Scripts

### hdb_tpcc_mariadb_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 maria
diset connection maria_host 127.0.0.1
diset connection maria_port 3306
diset tpcc maria_user mysql
diset tpcc maria_pass Password1!
diset tpcc maria_storage_engine innodb
diset tpcc maria_partition true
diset tpcc maria_driver timed
diset tpcc maria_count_ware 250
diset tpcc maria_num_vu 16
diset tpcc maria_rampup 2
diset tpcc maria_duration 5
vuset logtotemp 1
loadscript
vuset vu 16
vucreate
vurun
wait_to_complete
vwait forever
```

### hdb_tpcc_mariadb_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 maria
diset connection maria_host 127.0.0.1
diset connection maria_port 3306
diset tpcc maria_user mysql
diset tpcc maria_pass Password1!
diset tpcc maria_storage_engine innodb
diset tpcc maria_partition true
diset tpcc maria_driver timed
diset tpcc maria_count_ware 500
diset tpcc maria_num_vu 32
diset tpcc maria_rampup 2
diset tpcc maria_duration 5
vuset logtotemp 1
```

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

## hdb_tpcc_mariadb_2000wh.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 maria
diset connection maria_host 127.0.0.1
diset connection maria_port 3306
diset tpcc maria_user mysql
diset tpcc maria_pass Password1!
diset tpcc maria_storage_engine innodb
diset tpcc maria_partition true
diset tpcc maria_driver timed
diset tpcc maria_count_ware 2000
diset tpcc maria_num_vu 128
diset tpcc maria_rampup 2
diset tpcc maria_duration 5
vuset logtotemp 1
loadscript
vuset vu 128
vucreate
vurun
wait_to_complete
vwait forever
```

## mariadb_host_prepare.sh

```
#!/bin/bash
setenforce 0
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 ####
apt install -y wget vim tar zip unzip lz4 pigz nmon sysstat numactl psmisc 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/sdb
  echo '/dev/sdb  /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

```
# This group is read both by the client and the server
# use it for options that affect everything
#
[mysqld]
datadir=/mnt/mysqldata/mysql
#default_authentication_plugin=mysql_native_password
#socket=/mnt/mysqldata/mysql/mysqld.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='%'
#
# include *.cnf from the config directory
#
#!includedir /etc/my.cnf.d
```

## my-500.cnf

```
# This group is read both by the client and the server
# use it for options that affect everything
#
[mysqld]
datadir=/mnt/mysqldata/mysql
#default_authentication_plugin=mysql_native_password
#socket=/mnt/mysqldata/mysql/mysqld.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=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='%'
#
# include *.cnf from the config directory
#
#!includedir /etc/my.cnf.d
```

## my-2000.cnf

```
# This group is read both by the client and the server
# use it for options that affect everything
#
[mysqld]
datadir=/mnt/mysqldata/mysql
#default_authentication_plugin=mysql_native_password
#socket=/mnt/mysqldata/mysql/mysqld.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='%'
#
# include *.cnf from the config directory
#
#!includedir /etc/my.cnf.d
```

### run_test.sh

```
#!/bin/bash
echo "Usage: $0 TEST_HOST WAREHOUSE_COUNT"
TEST_HOST=${1:-remotehost}
CLIENT_HOST=$(hostname -s)
WAREHOUSE_COUNT=${2}
APP=maria
MYCNF=my-${WAREHOUSE_COUNT}.cnf
HDB_DIR=HammerDB-4.2/
HDB_SCRIPT=hdb_tpcc_${APP}db_${WAREHOUSE_COUNT}wh.tcl
HDB_RUN=run_${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 ${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="n2-standard-8"
```

```
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 systemctl stop ${APP}d ; sudo cp -vf tmp-my.cnf /etc/my.cnf"
ssh ${TEST_HOST} "sudo systemctl start ${APP} && \
  sleep 10 && \
  sync && \
  sudo systemctl stop ${APP}db && \
  sudo rm -rf /mnt/mysqldata/mysql && \
  pigz -d -c /mnt/mysqldata/${APP}db_${WAREHOUSE_COUNT}warehouses_data.tar.gz | sudo tar -C /mnt/
    mysqldata -xf- ; sync
  sudo systemctl start ${APP}db" || 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}db_${INSTANCE_TYPE}_${INSTANCE_CPU}_${TIMESTAMP}
mkdir -p ${RESULTS_DIR}
RESULTS_FILE=${APP}db_${INSTANCE_TYPE}_${INSTANCE_CPU}_${TIMESTAMP}
# Copy config files to results folder
cp -pvf ${0} ${RESULTS_DIR}/
cp -pvf ${HOST_PREPARE} ${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
# 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
# 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}
```

## Mitigations

### n2 CLX mitigations

```
Vulnerability Itlb multihit:     Not affected
Vulnerability L1tf:              Not affected
Vulnerability Mds:               Mitigation; Clear CPU buffers; SMT Host state unknown
Vulnerability Meltdown:          Not affected
Vulnerability Spec store bypass: Mitigation; Speculative Store Bypass disabled via prctl and seccomp
Vulnerability Spectre v1:        Mitigation; usercopy/swapgs barriers and __user pointer sanitization
Vulnerability Spectre v2:        Mitigation; Enhanced IBRS, IBPB conditional, RSB filling
Vulnerability Srbds:             Not affected
Vulnerability Tsx async abort:   Mitigation; Clear CPU buffers; SMT Host state unknown
```

### n2 ICX mitigations

```
Vulnerability Itlb multihit:     Not affected
Vulnerability L1tf:              Not affected
Vulnerability Mds:               Not affected
Vulnerability Meltdown:          Not affected
Vulnerability Spec store bypass: Mitigation; Speculative Store Bypass disabled via prctl and seccomp
Vulnerability Spectre v1:        Mitigation; usercopy/swapgs barriers and __user pointer sanitization
Vulnerability Spectre v2:        Mitigation; Enhanced IBRS, IBPB conditional, RSB filling
Vulnerability Srbds:             Not affected
Vulnerability Tsx async abort:   Vulnerable: Clear CPU buffers attempted, no microcode; SMT
    Host state unknown
```

**Read the report at https://facts.pt/RqXmDxs ▶**

This project was commissioned by Intel.