The science behind the report:

Achieve better online transaction processing performance on MariaDB with new Amazon EC2 M6i instances 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 Achieve better online transaction processing performance on MariaDB with new Amazon EC2 M6i instances featuring 3rd Generation Intel Xeon Scalable processors.

We concluded our hands-on testing on October 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 October 1, 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: Average rate of transactions per minute (TPM) and new orders per minute (NOPM) achieved by Amazon EC2 M5 series and M6i series instances running MariaDB and the HammerDB TPROC-C online transaction processing benchmark.

8vCPU instance results		
Instance	TPM	NOPM
m5.2xlarge	280,463	92,550
m6i.2xlarge	360,268	118,862
16vCPU instance results		
Instance	TPM	NOPM
m5.4xlarge	523,085	172,607
m6i.4xlarge	674,863	223,313
64vCPU instance results		
Instance	TPM	NOPM
m5.16xlarge	1,572,375	519,054
m6i.16xlarge	1,903,609	628,675

System configuration information

Table 2: Detailed information on the M6i series instances we tested.

System configuration information	m6i.2xlarge	m6i.4xlarge	m6i.16xlarge
Tested by	Principled Technologies	Principled Technologies	Principled Technologies
Test date	10/06/2021	10/06/2021	10/06/2021
CSP / Region	us-east-1b	us-east-1b	us-east-1b
Workload & version	HammerDB v4.2 TPROC-C	HammerDB v4.2 TPROC-C	HammerDB v4.2 TPROC-C
WL specific parameters	250 Warehouses 12 virtual users	500 Warehouses 24 virtual users	2000 Warehouses 96 virtual users
Iterations and result choice	3 runs, median	3 runs, median	3 runs, median
Server platform	m6i.2xlarge	m6i.4xlarge	m6i.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	Red Hat Enterprise Linux 8.4	Red Hat Enterprise Linux 8.4	Red Hat Enterprise Linux 8.4
Date of last OS updates/ patches applied	09/30/2021	09/30/2021	09/30/2021
Processor			
Number of processors	1	1	2
Vendor and model	Intel® Xeon® Platinum 8375C	Intel Xeon Platinum 8375C	Intel Xeon Platinum 8375C
VM core count (per processor)	32	32	32
Core frequency (GHz)	2.90	2.90	2.90
Stepping	6	6	6
Hyper-Threading	Yes	Yes	Yes
Turbo	Yes	Yes	Yes
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	12.5 Gbps	12.5 Gbps	25 Gbps
Storage BW / Instance	10 Gbps	10 Gbps	20 Gbps

System configuration information	m6i.2xlarge	m6i.4xlarge	m6i.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)	80	128	512
Drive information (speed, interface, type)	io1, EBS, 4000 IOPS	io1, EBS, 6000 IOPS	io1, EBS, 20000 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 3: Detailed information on the M5 series instances we tested.

System configuration information	m5.2xlarge	m5.4xlarge	m5.16xlarge
Tested by	Principled Technologies	Principled Technologies	Principled Technologies
Test date	10/06/2021	10/06/2021	10/06/2021
CSP / Region	us-east-1b	us-east-1b	us-east-1b
Workload & version	HammerDB v4.2 TPROC-C	HammerDB v4.2 TPROC-C	HammerDB v4.2 TPROC-C
WL specific parameters	250 Warehouses 12 virtual users	500 Warehouses 24 virtual users	2000 Warehouses 96 virtual users
Iterations and result choice	3 runs, median	3 runs, median	3 runs, median
Server platform	m5.2xlarge	m5.4xlarge	m5.16xlarge
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	Red Hat Enterprise Linux 8.4	Red Hat Enterprise Linux 8.4	Red Hat Enterprise Linux 8.4
Date of last OS updates/ patches applied	09/30/2021	09/30/2021	09/30/2021
Processor			
Number of processors	1	1	2
Vendor and model	Intel Xeon Platinum 8259CL	Intel Xeon Platinum 8259CL	Intel Xeon Platinum 8259CL
VM core count (per processor)	32	32	32
Core frequency (GHz)	2.50	2.50	2.50
Stepping	7	7	7
Hyper-Threading	Yes	Yes	Yes
Turbo	Yes	Yes	Yes
Total number of vCPUs per VM	8	16	64

System configuration information	m5.2xlarge	m5.4xlarge	m5.16xlarge
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	10 Gbps	10 Gbps	20 Gbps
Storage BW / Instance	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)	80	128	512
Drive information (speed, interface, type)	io1, EBS, 4000 IOPS	io1, EBS, 6000 IOPS	io1, EBS, 20000 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

We tested two sets of Amazon EC2 instances: New M6i series instances featuring 3rd Generation Intel Xeon Scalable processors and older M5 series instances featuring 2nd Generation Intel Xeon Scalable processors. We ran a TPC-C-like workload on MariaDB on the AWS instances to show the performance increase in terms of transactions per minute on OLTP databases that customers could expect to see using the newer instance series vs. the older.

Creating the MariaDB instance

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

Creating the MariaDB instance

1. Log into AWS and navigate to the AWS Management Console.
2. Click EC2
3. Click Launch instance. To open the Launch Instance wizard, select Launch instance from the dropdown menu.
4. In the search window, type `Red Hat` and press Enter.
5. On the Quick Start tab, choose your architecture (x86). Click the Select button next to "Red Hat Enterprise Linux 8 (HVM), SSD Volume Type)" by Amazon Web Services. ami-0b0af3577fe5e3532 (64-bit x86) / ami-01fc429821bf1f4b4 (64-bit Arm)
6. On the Choose Instance Type tab, select {m6i,m5}.{2,4,16}xlarge, and 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. We chose us-east-1b.
    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
8. Click Next: Add Storage.
9. 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: {80GB,128GB,512GB}
    g. Volume Type: io1
    h. IOPS: {4000,6000,20000}
    i. Delete on Termination: Checked
    j. Encryption: Not Encrypted
10. Click Next: Add Tags
11. On the Add Tags tab, add any appropriate tags, and click Next: Configure Security Group.
12. On the Configure Security Group tab, set the following:
    a. Create a new security group.
    b. Leave the rules default.
13. Click Review and Launch.
14. On the Review Tab, click Launch.
15. Choose the appropriate option for the key pair, and click Launch Instances.

Creating the HammerDB 4.2 client instance

1. Log into AWS, and navigate to the AWS Management Console.
2. Click EC2
3. Click Launch instance. To open the Launch Instance wizard, select Launch instance from the dropdown menu.
4. In the search window, enter `Red Hat` and press Enter.
5. On the Quick Start tab, choose x86, and click the Select button next to "Red Hat Enterprise Linux 8 (HVM), SSD Volume Type)" by Amazon Web Services. ami-0b0af3577fe5e3532 (64-bit x86) / ami-01fc429821bf1f4b4 (64-bit Arm)
6. On the Choose Instance Type tab, select m5n.2xlarge, and 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
8. Click Next: Add Storage.
9. On the Add Storage tab, set the following:
    a. Size: 10GB
    b. Volume Type: gp2
    c. Delete on Termination: Checked
    d. Encryption: Not Encrypted
10. Click Next: Add Tags
11. On the Add Tags tab, add any appropriate tags, and Click Next: Configure Security Group.
12. On the Configure Security Group tab, set the following:
    a. Select an existing security group
    b. Chose the group created for MariaDB and HammerDB.
13. Click Review and Launch.
14. On the Review Tab, click Launch.
15. Choose the appropriate option for the key pair, and click Launch Instances.

Configuring Red Hat Enterprise Linux 8 and installing MariaDB on the MariaDB instance

1. Using SSH, log into the MariaDB instance.
2. Run the mariadb_host_prepare.sh script:

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

3. Install the appropriate MariaDB repository for x86 architecture on RHEL 8 from https://mariadb.org/download/?tab=repo-config
4. Install MariaDB:

```
sudo dnf 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. Copy the appropriate my.cnf config file from the appendix depending on your MySQL instance and target database size. Here is an example for a 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 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;
```

11. Shut down the instance.

```
sudo poweroff
```

Configuring Red Hat Enterprise Linux 8 and installing HammerDB 4.2 on the mariadb-client instance

1. Using SSH, log into the HammerDB instance.
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 the required packages:

```
sudo dnf install https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm
sudo dnf install -y wget vim tar zip unzip lz4 pigz nmon sysstat numactl ksh
```

5. Install the appropriate MariaDB repository for x86 architecture on RHEL 8 from https://mariadb.org/download/?tab=repo-config
6. Install MariaDB:

```
sudo dnf install MariaDB-server
```

7. Download HammerDB 4.2:

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

8. Extract the HammerDB package:

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

9. Download and extract the nmonchart tool:

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

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

```
sudo poweroff
```

Creating the database schema with HammerDB

1. Using SSH, log into the mariadb-client instance.
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 root
    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
```

Back up the database

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

```
sudo 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 instances under test. As each instance had different hardware and database sizes, please refer to Table 4 to see the number of users to run on each instance.

1. Login to the hammerdb mariadb-client instance via ssh.
2. Execute the run_test.sh script substituting IP_ADDRESS with the AWS private IP of the mariadb 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 mariadb instance, restore the correct DB_SIZE, and run the test automatically. Results will be saved to the "results" folder in your home directroy by default.
4. To parse all results run the parse_results.sh script.

```
./parse_results.sh
```

5. Reboot the mariadb instance and client instance.
6. Repeat these steps 2 more times for a total of 3 runs. Do this for each mariadb instance type and warehouse size combination.

Table 4: Disk and database settings for each instance size.

Instance type	{m5,m6i}.2xlarge	{m5,m6i}.4xlarge	{m5,m6i}.16xlarge
Number of vCPU	8	16	64
Memory (GB)	32	64	256
Data disk (size, IOPS)	80 GB 4,000 IOPS	128 GB 6,000 IOPS	512 GB 20,000 IOPS
Number of warehouses	250	500	2,000
Number of users	12	24	96
Warmup (min)	2	2	2
Runtime (min)	5	5	5

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 12
diset tpcc maria_rampup 2
diset tpcc maroa_duration 5

vuset logtotemp 1
loadscript
vuset vu 12
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 maria
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 24
diset tpcc maria_rampup 2
diset tpcc maria_duration 5

vuset logtotemp 1
loadscript
vuset vu 24
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 maria
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 96
diset tpcc maria_rampup 2
diset tpcc maria_duration 5

vuset logtotemp 1
loadscript
vuset vu 96
vucreate
vurun
wait_to_complete
vwait forever
```

mariadb_host_prepare.sh

```
#!/bin/bash

setenforce 0
sed -i 's/SELINUX=.*/SELINUX=Permissive/' /etc/selinux/config

#### 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 https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm
dnf install -y wget vim tar zip unzip lz4 pigz nmon sysstat numactl ksh

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

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

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

my-250.cnf

```
# 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
```

parse_results.sh

```
#!/bin/bash
RAMPUP=2    # 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
```

run_test.sh

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

RAMPUP=2    # minutes
DURATION=5  # minutes

STEP=2# seconds
IDLE=30 # seconds

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

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

# Check for files

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

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

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

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

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

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

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

scp ${MYCNF} ${TEST_HOST}:tmp-my.cnf
ssh ${TEST_HOST} "sudo systemctl stop ${APP} ; sudo cp -vf tmp-my.cnf /etc/my.cnf"
ssh ${TEST_HOST} "sudo systemctl start ${APP} && \
  sleep 10 && \
  sync && \
  sudo systemctl stop ${APP} && \
  sudo systemctl start ${APP}" || 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
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 maria/" \
    -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

M6i series instances featuring Intel Xeon Platinum 8375C processors

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

M5 series instances featuring Intel Xeon Platinum 8259CL processors

```
itlb_multihit: KVM: Vulnerable
l1tf: Mitigation: PTE Inversion
mds: Vulnerable: Clear CPU buffers attempted, no microcode; SMT Host state unknown
meltdown: Mitigation: PTI
spec_store_bypass: Vulnerable
spectre_v1: Mitigation: usercopy/swapgs barriers and __user pointer sanitization
spectre_v2: Mitigation: Full generic retpoline, STIBP: conditional, RSB filling
srbds: Not affected
tsx_async_abort: Not affected
```

Read the report at http://facts.pt/TK7B9CA ▶

This project was commissioned by Intel.

Principled Technologies®

Facts matter.®