MYSQL ON AWS M6I INSTANCES ENABLED BY 3RD GENERATION INTEL XEON SCALABLE PROCESSORS VS. AWS M5N INSTANCES

At Principled Technologies, we used a TPROC-C workload from the HammerDB benchmark to compare MySQL™ Database performance of two instance types: new M6i instances with 3rd Gen Intel® Xeon® Scalable processors and M5n instances with older processors. HammerDB reports results in both transactions per minute (TPM) and new orders per minute (OPM). We tested three instance sizes to reflect different business needs: 8 vCPU, 16 vCPU, and 64 vCPU.

We purchased two sets of instances from two general-purpose AWS EC2 series:

- Newer M6i instances featuring 3rd Generation Intel Xeon Platinum 8375C processors (Ice Lake)
- Older M5n instances featuring 2nd Generation Intel Xeon Platinum 8259CL processors (Cascade Lake)

We ran each instance in the US East 1 region.

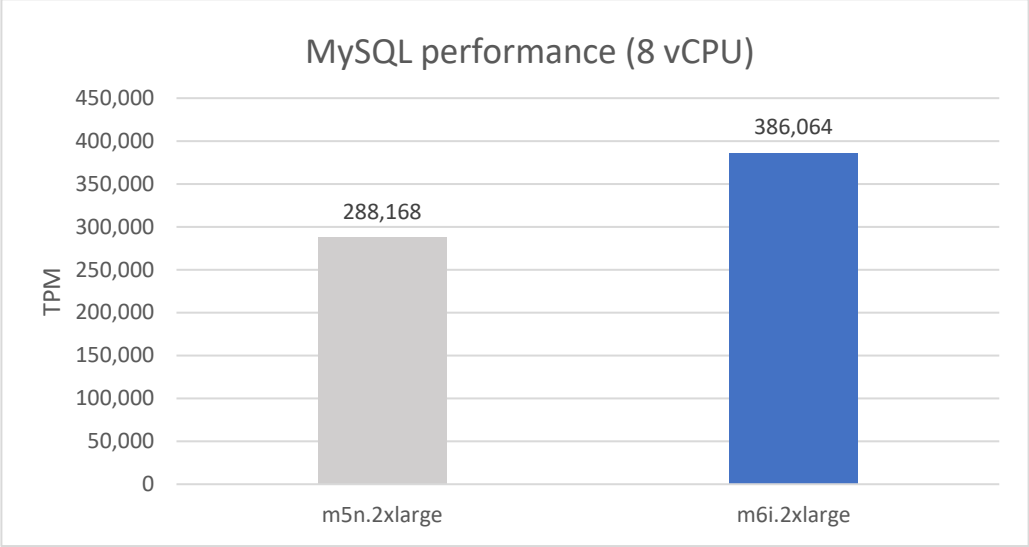
Figure 1 details the results of our HammerDB tests. We found that at each instance size, the M6i instances with 3rd Gen Intel Xeon Scalable processors outperformed the M5n instances with older processors.

8 vCPU results					
Instance	**CPU type**	**TPM**	**NOPM**	**Host CPU%**	**Client CPU%**
m5n.2xlarge	8259CL	288,168	94,903	94.64	4.27
m6i.2xlarge	8375C	386,064	127,322	93.98	6.02
Times win for M6i instance		**1.33**	**1.34**		
16 vCPU results					
Instance	**CPU type**	**TPM**	**NOPM**	**Host CPU%**	**Client CPU%**
m5n.4xlarge	8259CL	591,512	195,269	96.25	9.07
m6i.4xlarge	8375C	746,857	246,659	91.15	10.88
Times win for M6i instance		**1.26**	**1.26**		
64 vCPU results					
Instance	**CPU type**	**TPM**	**NOPM**	**Host CPU%**	**Client CPU%**
m5n.16xlarge	8259CL	1,995,105	658,549	93.28	34.19
m6i.16xlarge	8375C	2,834,552	935,261	93.29	50.52
Times win for M6i instance		**1.42**	**1.42**		

Figure 1: Results summary for the HammerDB tests. Source: Principled Technologies.

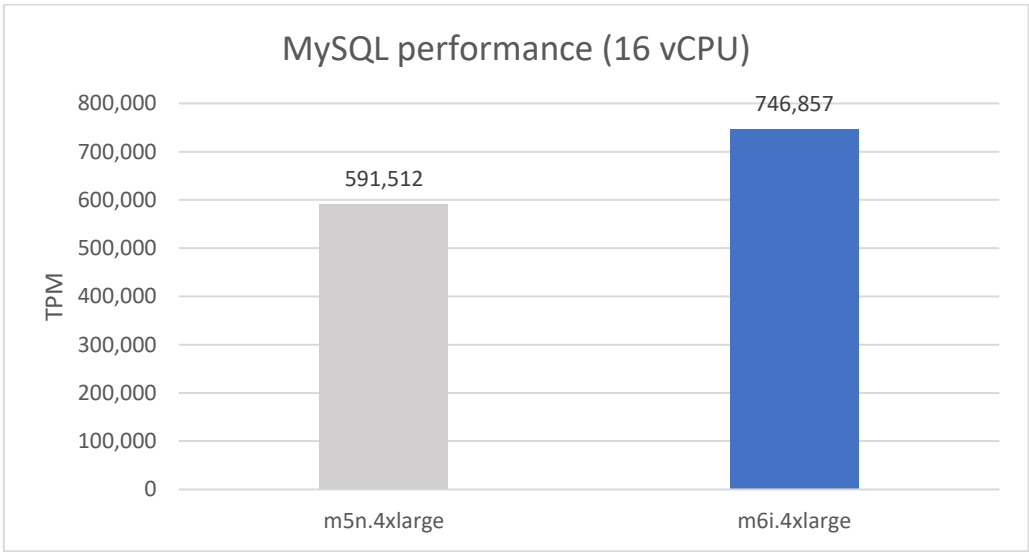
As Figure 2 shows, at 8 vCPUs, the AWS M6i instances featuring 3rd Gen Intel Xeon Scalable processors handled 1.33x as many transactions per minute as the M5n instances with older processors did.

Figure 2: Transactions per minute that the small-size instances (8 vCPUs) with a 22GB MySQL database handled using the HammerDB benchmark. Higher numbers are better. Source: Principled Technologies.



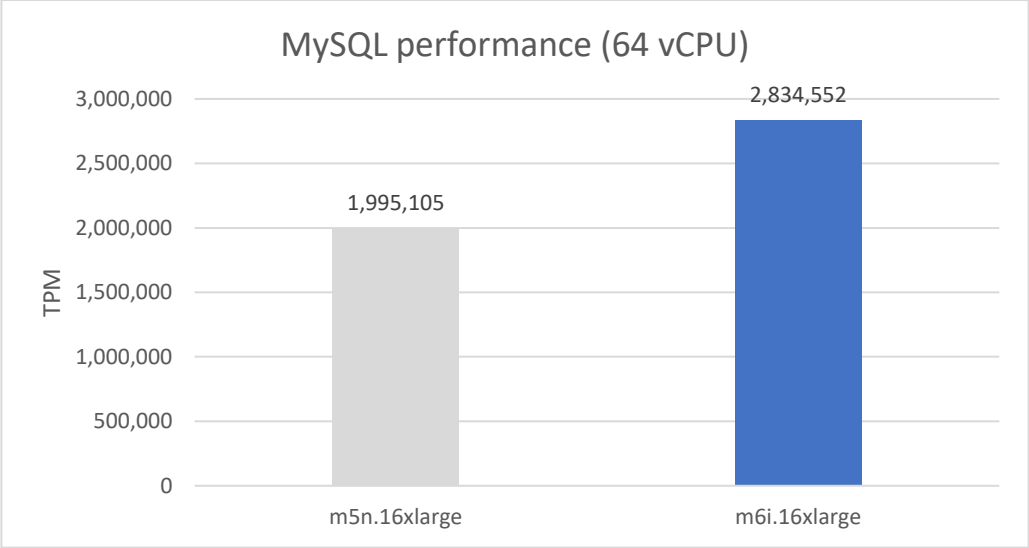
With 16 vCPUs per instance, the Amazon EC2 M6i instances featuring 3rd Gen Intel Xeon Scalable processors achieved 1.26x the transactions per minute of the M5n instances with older processors (see Figure 3).

Figure 3: Transactions per minute that the medium-sized instances (16 vCPUs) with a 45GB MySQL database handled using the HammerDB benchmark. Higher numbers are better. Source: Principled Technologies.



As Figure 4 shows, at 64 vCPUs, the Amazon EC2 M6i instances featuring 3rd Gen Intel Xeon Scalable processors handled 1.42x the transactions per minute of the M5n instances with older processors.

Figure 4: Transactions per minute that the large-size instances (64 vCPUs) with a 180GB MySQL database handled using the HammerDB benchmark. Higher numbers are better. Source: Principled Technologies.



APPENDIX A – SYSTEM CONFIGURATION INFORMATION

Figures 5 and 6 provide detailed configuration information for the test instances.

Server configuration information	m5n.2xlarge (8vCPU)	m5n.4xlarge (16vCPU)	m5n.16xlarge (64vCPU)
Tested by	Principled Technologies	Principled Technologies	Principled Technologies
Test date	09/16/2021	09/16/2021	09/17/2021
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 16 virtual users	500 Warehouses 32 virtual users	2000 Warehouses 128 virtual users
Iterations and result choice	3 runs, median	3 runs, median	3 runs, median
Server platform	m5n.2xlarge	m5n.4xlarge	m5n.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	09/08/2021	09/08/2021	09/08/2021
Processor			
Number of processors	1	1	2
Vendor and model	Intel Xeon Platinum 8259CL	Intel Xeon Platinum 8259CL	Intel Xeon Platinum 8259CL
Core count (per processor)	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
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

Server configuration information	m5n.2xlarge (8vCPU)	m5n.4xlarge (16vCPU)	m5n.16xlarge (64vCPU)
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
Local storage			
OS			
Number of drives	1	1	1
Drive size (GB)	20	20	20
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)	64	128	512
Drive information (speed, interface, type)	io1, EBS, 2000 IOPS	io1, EBS, 4000 IOPS	io1, EBS, 16000 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

Figure 5: Configuration information for the M5n instances.

Server configuration information	m6i.2xlarge (8 vCPU)	m6i.4xlarge (16 vCPU)	m6i.16xlarge (64 vCPU)
Tested by	Principled Technologies	Principled Technologies	Principled Technologies
Test date	09/08/2021	09/08/2021	09/09/2021
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

Server configuration information	m6i.2xlarge (8 vCPU)	m6i.4xlarge (16 vCPU)	m6i.16xlarge (64 vCPU)
WL specific parameters	250 Warehouses 16 virtual users	500 Warehouses 32 virtual users	2000 Warehouses 128 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	09/08/2021	09/08/2021	09/08/2021
Processor			
Number of processors	1	1	2
Vendor and model	Intel Xeon Platinum 8375C	Intel Xeon Platinum 8375C	Intel Xeon Platinum 8375C
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
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 HW			
Storage: NW or Direct Att / Instance	NW	NW	NW

Server configuration information	m6i.2xlarge (8 vCPU)	m6i.4xlarge (16 vCPU)	m6i.16xlarge (64 vCPU)
Network BW / Instance	12.5 Gbps	12.5 Gbps	25 Gbps
Storage BW / Instance	10 Gbps	10 Gbps	20 Gbps
Local storage			
OS			
Number of drives	1	1	1
Drive size (GB)	20	20	20
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)	64	128	512
Drive information (speed, interface, type)	io1, EBS, 2000 IOPS	io1, EBS, 4000 IOPS	io1, EBS, 16000 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

Figure 6: Configuration information for the M6i instances.

APPENDIX B – HOW WE TESTED

We tested AWS instances featuring older processors vs. those with 3rd Gen Intel Xeon Scalable processors. We ran a TPROC-C workload on MySQL on the AWS instances to show the performance increase in terms of transactions per minute on OLTP databases that customers can expect to see using the newer instance series vs. the older.

Creating the MySQL instance

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

1. Log into AWS, and navigate to the AWS Management Console.
2. Click EC2.
3. Click Launch instance, and from the drop-down menu, select Launch instance.
4. In the search window, type `Red Hat` and press Enter.
5. On the Quick Start tab, choose your architecture (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 mysql-{m6i,m5n}.{2,4,16}xlarge. Click Next: Configure Instance Details.
7. On the Configure Instance tab, set the following:
    - Number of instances: 1
    - Purchasing option: Leave unchecked
    - Network: Default VPC.
    - Subnet: Choose the region you're working in. We chose us-east-1b.
    - Auto-assign Public IP: Enable.
    - Placement Group: Leave unchecked.
    - Capacity Reservation: Open
    - Domain join directory: No Directory
    - IAM role: None
    - Shutdown behavior: Stop
8. Click Next: Add Storage.
9. On the Add Storage tab, set the following:
    - Size: 10GB
    - Volume Type: gp2
    - Delete on Termination: Checked
    - Encryption: Not Encrypted
    - Click Add New Volume
    - Size: {64GB,128GB,512GB}
    - Volume Type: io1

- IOPS: {2000,4000,16000}
- Delete on Termination: Checked
- Encryption: Not Encrypted

10. Click Next: Add Tags.
11. On the Add Tags tab, add any appropriate tags, and click Next: Configure Security Group.
12. On the Configure Security Group tab, set the following:
    - Create a new security group.
    - Leave the rules default.
    - Click Review and Launch.
13. On the Review Tab, click Launch.
14. Choose the appropriate option for the key pair, and click Launch Instances.

Creating the HammerDB 4.2 client instance

1. Log into AWS and navigate to the AWS Management Console.
2. Click EC2.
3. Click Launch instance, and from the drop-down menu, select Launch instance.
4. In the search window, enter `CentOS 8` 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 m5.2xlarge. Click Next: Configure Instance Details.
7. On the Configure Instance tab, set the following:
    - Number of instances: 1
    - Purchasing option: Leave unchecked
    - Network: Default VPC.
    - Subnet: Choose the region you're working in.
    - Auto-assign Public IP: Enable.
    - Placement Group: Leave unchecked.
    - Capacity Reservation: Open
    - Domain join directory: No Directory
    - IAM role: None
    - Shutdown behavior: Stop
8. Click Next: Add Storage.
9. On the Add Storage tab, set the following:
    - Size: 10GB
    - Volume Type: gp2
    - Delete on Termination: Checked

- Encryption: Not Encrypted

10. Click Next: Add Tags
11. On the Add Tags tab, add any appropriate tags, and Click Next: Configure Security Group.
12. On the Configure Security Group tab, set the following:
    - Select an existing security group
    - Chose the group created for MySQL and HammerDB.
    - Click Review and Launch.
13. On the Review Tab, click Launch.
14. Choose the appropriate option for the key pair, and click Launch Instances.

Configuring Red Hat Enterprise Linux 8 and installing MySQL on mysql instance

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

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

3. Download the appropriate MySQL bundle for either x86 or Arm architecture on RHEL from https://dev.mysql.com/downloads/mysql/
4. Extract the MySQL bundle:

```
tar -xf mysql-8.0.26-1.el8.x86_64.rpm-bundle.tar
```

5. Install MySQL Community Server 8 and all dependencies.

```
sudo yum --disablerepo=* localinstall mysql-community-server-8.0.26-1.el8.x86_64.rpm
```

6. Stop the MySQL service:

```
sudo service mysqld stop
```

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

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

8. Copy the appropriate my.cnf config file from the appendix depending on your mysql instance and target database size. Example for 250 warehouse database:

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

9. Start the MySQL service:

```
sudo service mysqld start
```

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

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

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

```
CREATE USER 'mysql'@'localhost' IDENTIFIED BY '[password]';

GRANT ALL PRIVILEGES ON *.* TO 'mysql'@'localhost'

->  WITH GRANT OPTION;

CREATE USER 'mysql'@'%' IDENTIFIED BY '[password]';
```

```
GRANT ALL PRIVILEGES ON *.* TO 'mysql'@'%'

->  WITH GRANT OPTION;
```

12. Enable mysql_native_password authentication on the mysql user:

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

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

13. Shut down the instance.

```
sudo poweroff
```

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

1. Log into the hammerdb instance via SSH.
2. Disable SELINUX:

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

3. Turn off SSH strict host key checking:

```
echo 'StrictHostKeyChecking no' > .ssh/config

chmod 400 ~/.ssh/config
```

4. Install required packages:

```
sudo dnf install -y epel-release

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

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

```
tar -xf mysql-8.0.26-1.el8.x86_64.rpm-bundle.tar
```

7. Install the MySQL 8 Community Client and all of its dependencies:

```
sudo yum --disablerepo=* localinstall mysql-community-client-8.0.26-1.el8.x86_64.rpm
```

8. Download HammerDB 4.2:

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

9. Extract the HammerDB package:

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

10. Download and extract nmonchart tool:

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

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

```
sudo poweroff
```

Creating the database schema with HammerDB

1. Log into the mysql-client instance via SSH.

2. Navigate to the HammerDB directory:

```
cd HammerDB-4.2
```

3. Start hammerdbcli:

```
./hammerdbcli
```

4. Set the following variables:

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

5. Build the schema:

```
buildschema
```

Backing up the database

1. Log into the mysql instance.
2. Shut down the database.

```
systemctl stop mysqld
```

3. Delete the log files:

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

4. Back up the database:

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

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

Running the tests

In this section, we list the steps to run the HammerDB TPROC-C test on the instances under test. As each instance had different hardware and database sizes, please refer to Figure 7 to see the number of users to run on each instance.

1. Log into the hammerdb mysql-client instance via SSH.
2. Execute the run_test.sh script substituting IP_ADDRESS with the AWS private IP of the mysql instance and DB_SIZE with the number of warehouses. Additional parameters and config options can be tuned by modifying the script and editing the variables at the start of the file:

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

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

```
./parse_results.sh
```

5. Reboot the mysql instance and client instance.

6. Repeat these steps twice more for a total of three runs. Do this for each instance type and warehouse size combination.

VM type	{m5n,m6i}.2xlarge	{m5n,m6i}.4xlarge	{m5n,m6i}.16xlarge
Number of vCPU	8	16	64
Memory (GB)	32	64	256
Data disk (size, IOPS)	64GB, 2000 IOPS	128GB, 4000 IOPS	512GB, 16000 IOPS
Number of warehouses	250	500	2000
Number of users	16	32	128
Warmup (min)	5	5	5
Runtime (min)	10	10	10

Figure 7: Parameters for HammerDB testing

APPENDIX C: PROCESSOR MITIGATIONS

M6i instances

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: Enhanced IBRS, IBPB: conditional, RSB filling
srbds: Not affected
tsx_async_abort: Not affected

M5n instances

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

APPENDIX D: SCRIPTS

mysql_host_prepare.sh

```
#!/bin/bash

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

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

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

sysctl -p

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


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

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

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

my-250.cnf

```
[mysqld]
datadir=/mnt/mysqldata/mysql
default_authentication_plugin=mysql_native_password
socket=/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='%'

[client]
socket=/mnt/mysqldata/mysql/mysqld.sock
```

my-500.cnf

```
[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='%'

[client]
socket=/mnt/mysqldata/mysql/mysqld.sock
```

my-2000.cnf

```
[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='%'

[client]
socket=/mnt/mysqldata/mysql/mysqld.sock
```

hdb_tpcc_mysql_250wh.tcl

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

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

diset tpcc mysql_count_ware 250
diset tpcc mysql_num_vu 16
```

```
diset tpcc mysql_rampup 2
diset tpcc mysql_duration 5

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

hdb_tpcc_mysql_500wh.tcl

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

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

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

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

hdb_tpcc_mysql_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 mysql
diset connection mysql_host 127.0.0.1
diset connection mysql_port 3306

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

diset tpcc mysql_count_ware 2000
diset tpcc mysql_num_vu 128
diset tpcc mysql_rampup 2
diset tpcc mysql_duration 5

vuset logtotemp 1
loadscript
```

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

mysql_host_prepare.sh

```
#!/bin/bash

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

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

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

sysctl -p

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


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

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

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

run_test.sh

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

RAMPUP=5    # minutes
DURATION=10 # minutes

STEP=2      # seconds
IDLE=30     # seconds

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

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

```
# Check for files

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

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

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


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

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

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

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

scp ${MYCNF} ${TEST_HOST}:tmp-my.cnf
ssh ${TEST_HOST} "sudo systemctl stop ${APP}d ; sudo cp -vf tmp-my.cnf /etc/my.cnf"
ssh ${TEST_HOST} "sudo systemctl start ${APP}d && \
    sleep 10 && \
    sync && \
    sudo systemctl stop ${APP}d && \
  sudo rm -rf /mnt/${APP}data && \
  pigz -d -c /mnt/${APP}data/${APP}_tpcc_${WAREHOUSE_COUNT}warehouses_data.tar.gz | sudo tar C
/mnt/${APP}data -xf- ; sync
    sudo 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
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
```