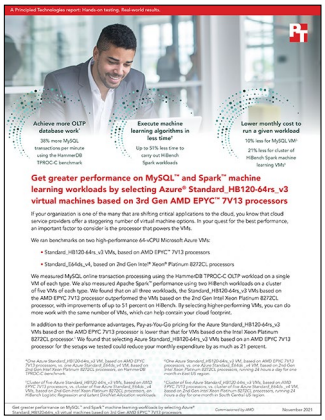

The science behind the report:

Get greater performance on MySQL™ and Spark™ machine learning workloads by selecting Azure® Standard_HB120-64rs_v3 virtual machines based on 3rd Gen AMD EPYC™ 7V13 processors

This document describes what we tested, how we tested, and what we found. To learn how these facts translate into real-world benefits, read the report Get greater performance on MySQL™ and Spark™ machine learning workloads by selecting Azure® Standard_HB120-64rs_v3 virtual machines based on 3rd Gen AMD EPYC™ 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 September 8, 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: MySQL HammerDB TPROC-C test results.

Azure region	Azure VM	Transactions per minute (TPM)	Advantage	CPU busy %	Advantage
East US	Standard_HB120-64rs_v3 AMD EPYC 7V13 Processor	2,486,790	38.53%	85.38	1.98%
	Standard_E64ds_v4 Intel Xeon Platinum 8272CL	1,795,114	-	83.73	-

Table 2: HiBench Logistic Regression (LR) test results.

Azure region	Azure VM	Throughput per node (bytes/sec)	Advantage	CPU busy %	Advantage	Test duration (Seconds)	Advantage
South Central US	Standard_HB120-64rs_v3 AMD EPYC 7V13 Processor	11,825,560	59.01%	43.11	16.29%	1,184	37.11%
	Standard_E64ds_v4 Intel Xeon Platinum 8272CL	7,437,164	-	51.50	-	1,882	-

Table 3: HiBench Latent Dirichlet Allocation (LDA) test results.

Azure region	Azure VM	Throughput per node (bytes/sec)	Advantage	CPU busy %	Advantage	Test duration (Seconds)	Advantage
South Central US	Standard_HB120-64rs_v3 AMD EPYC 7V13 Processor	2,241,737	107.47%	48.02	21.52%	495	51.97%
	Standard_E64ds_v4 Intel Xeon Platinum 8272CL	1,080,528	-	61.18	-	1031	-

Estimated costs

Table 4: Monthly Pay-as-You-Go pricing for MySQL test environment as of October 19, 2021.

Azure region	Azure VM	Configuration details	Estimated monthly cost	Advantage
East US	Standard_HB120-64rs_v3 AMD EPYC 7V13 Processor	1 MySQL Server VM Standard_ HB120-64rs_v3 (64 vCPUs, 448 GB RAM) x 730 hours, Linux; 1 managed disk – P80; internet egress, 5 GB outbound data transfer from US region routed via public internet	$6,232	10.56%
	Standard_E64ds_v4 Intel Xeon Platinum 8272CL	1 MySQL Server VM Standard_ E64ds_v4 (64 vCPUs, 504 GB RAM) x 730 hours, Linux; 1 managed disk – P80; internet egress, 5 GB outbound data transfer from US region routed via public internet	$6,968	-

Table 5: Monthly Pay-as-You-Go pricing for HiBench test environment as of October 19, 2021.

Azure region	Azure VM	Configuration details	Estimated monthly cost	Advantage
South Central US	Standard_HB120-64rs_v3 AMD EPYC 7V13 Processor	1 manager VM; 4 worker node VMs 5 Standard_ HB120-64rs_v3 (64 vCPUs, 448 GB RAM) x 730 hours, Linux; 4 managed disk – P80; internet egress, 5 GB outbound data transfer from US region routed via public internet	$27,561	21.19%
	Standard_E64ds_v4 Intel Xeon Platinum 8272CL	5 Standard_ E64ds_v4 (64 vCPUs, 504 GB RAM) x 730 Hours, Linux; 4 managed disk – P80; internet egress, 5 GB outbound data transfer from US region routed via public internet	$34,970	-

Licensing Program - Microsoft Online Services Agreement
Support - $0.00

All prices shown are in US Dollar ($). This is a summary estimate, not a quote. For up to date pricing information please visit https://azure.microsoft.com/pricing/calculator/.

System configuration information

Table 6: Detailed information on the system we tested.

System configuration information	Standard_HB120-64rs_v3	Standard_E64ds_v4
Tested by	Principled Technologies	Principled Technologies
Test date	9/21/21	10/5/21
Iterations and result choice	3 runs, median	3 runs, median
Workload 1		
Workload and version	TPROC-C HammerDB 4.1	TPROC-C HammerDB 4.1
Workload-specific parameters	4,250 warehouses; 240 virtual users; 10-minute ramp-up time; 20-minute test run time	4,250 warehouses; 240 virtual users; 10-minute ramp-up time; 20-minute test run time
CSP/region	Microsoft Azure East US	Microsoft Azure East US
Workload 2		
Workload and version	HiBench 7.1; Java™ 1.8.0; Apache Spark 3.1.2; Hadoop 3.3.0	HiBench 7.1; Java™ 1.8.0; Apache Spark 3.1.2; Hadoop 3.3.0
Workload-specific parameters	Logistic Regression (LR); Latent Dirichlet Allocation (LDA); both bigdata & 80% Reserved Memory	Logistic Regression (LR); Latent Dirichlet Allocation (LDA); both bigdata & 80% Reserved Memory
CSP/region	Microsoft Azure South Central US	Microsoft Azure South Central US
Number of instances	1	1
Platform details		
VM series and size	Standard_ HB120-64rs_v3	Standard_ E64ds_v4
BIOS name and version	SMBIOS 2.3	SMBIOS 2.3
Operating system name and version/build number	CentOS Linux® release 8.4.2105 4.18.0-305.7.1.el8_4.x86_64	CentOS Linux release 8.4.2105 4.18.0-305.7.1.el8_4.x86_64
Date of last OS updates/patches applied	9/8/21	9/8/21
Number of instances	5	5
Processor		
Vendor and model	AMD EPYC™ 7V13 Processor	Intel® Xeon® Platinum 8272CL
Core count	64	32
Core frequency (GHz)	3.1	2.6
Stepping	0	7
vCPU count	64	64
Memory module(s)		
Total memory in system (GB)	448	504
NVMe™ module present?	No	No
Total memory in system (GB) (DDR+NVMe RAM)	448	504

System configuration information	Standard_HB120-64rs_v3	Standard_E64ds_v4
General hardware		
Storage type (network- or direct-attached)	Network-attached	Network-attached
Network bandwidth per VM (Mb/s)	30,000	30,000
Storage bandwidth per VM (MB/s)	925	925
Local storage (OS)		
Number of drives	1	1
Drive size (GB)	30	30
Drive information	Premium SSD LRS	Premium SSD LRS
Local storage (data drive)		
Number of drives	1	1
Drive size (TB)	8	8
IOPS	20,000	20,000
Drive bandwidth (MB/s)	900	900
Drive information	Premium SSD LRS	Premium SSD LRS
Network		
Network bandwidth	50 Gb/s Ethernet (40 Gb/s usable)	50 Gb/s Ethernet (40 Gb/s usable)
NIC hardware	Azure Second Gen SmartNIC	Azure Second Gen SmartNIC

How we tested

CentOS Linux 8.4 with MySQL 8 deployment methodology

Creating and configuring the Azure VM instances

This section contains the steps we took to create our two instances for running HammerDB benchmark and MySQL database software.

Installing CentOS Linux 8.4 on Azure VMs

1. In Azure Services Home screen, click the Virtual Machines icon, choose Create, and choose Virtual Machine.
2. Under Basic, make the following selections:
    a. Select the appropriate Resource Group.
    b. For Virtual Machine Name, name the VM.
    c. For Region, choose the appropriate region.
    d. For Availability options, Select No infrastructure redundancy required
    e. For Image, click the drop-down menu, and select CentOS-Based 8.4 - Gen 1.
    f. For Size, click the drop-down menu, and select Standard_HB120-64rs_v3.
    g. For Authentication type, select SSH Public key.
    h. For Username, leave the default of azureuser.
    i. For SSH Public key source, select Generate new key pair.
    j. For Key pair name, select MySQL-key.
    k. For Public inbound ports, click the radio button beside Allow selected ports.
    l. For Select inbound ports, select SSH (22).
    m. Click Next: Disks >.
3. Under Disks, make the following selections:
    a. For OS Disk type: Premium SSD (locally redundant).
    b. For SSE Encryption type, select default.
    c. Click Next: Networking >.
4. Under Networking, make the following selections:
    a. For Virtual Network, select the appropriate Virtual Network.
    b. For Subnet: select default Subnet.
    c. For Public IP, select (new) name.
    d. For NIC security group, select Advanced.
    e. For Configure network security group, select the appropriate NSG.
    f. Set Accelerated networking to on.
    g. Set load balancing to off.
    h. Click Next: Management >.
    i. Click through Next: Advanced >, then Next: Tags >, accepting all defaults.
    j. Click Next: Review + create >.
5. Ensure all values are correct, and click Create.

Configuring security and host access

To allow for easy access from one host to another and to provide access to the test hosts from an SSH utility such as Putty, you must configure certain files associated with security.

1. Using PuTTy, log onto the VM using azureuser and private key pair.
2. Issue the sudo su command switch user to root:

```
sudo su
```

3. Change the root password:

```
passwd root
```

4. Change the /etc/ssh/sshd_config file to allow password authentication, and reset the ssh daemon:

```
vi /etc/ssh/sshd_config
```

- uncomment PasswordAuthentication yes
- Comment out PasswordAuthentication no
- uncomment `PermitRootLogin yes`

5. Reset the ssh daemon:

```
systemctl restart sshd
```

6. On both the SUT and the HammerDB driver client, create the mysql_host_prepare.sh shell script:

```
cd /root ; mkdir scripts ; cd scripts
vi 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
```

7. Change permissions and make executable:

```
chmod 755 mysql_host_prepare.sh
```

8. Run host prep script:

```
./mysql_host_prepare.sh
```

9. Configure the systems to log into each other without prompting for a password:

```
mkdir -p /root/.ssh
chmod 700 /root/.ssh
cd /root/.ssh
ssh-keygen -t rsa -q -f id_rsa -N ''
cp id_rsa.pub authorized_keys
echo "StrictHostKeyChecking=no" > config
```

10. Copy keys from server to client:

```
scp /root/.ssh/* <IP address of HDB driver client>:/root/.ssh/
```

11. Test access from server to client and client to server by using the ssh command on both systems:

```
ssh <IP address of other system>
```

12. Each system should be able to access the other without prompting for a password.

Creating storage for DB

The MySQL server SUT requires a separate disk to hold the database.

1. On the SUT instance only, add a second disk:
2. In Azure Services Home screen, click the Virtual Machines icon, and click the server name.
3. In the center of the window, click Disks.
4. Click Create and attach a new disk.
5. Specify the Disk Name and size, and click Save.

Creating file system on Disk

1. From the PuTTy console, partition the Disk, create an XFS filesystem and mount it to the instance:

```
umount -v /mnt/mysqldata
mkdir -p /mnt/mysqldata
mkfs.xfs -f /dev/sdc
echo '/dev/sdc  /mnt/mysqldata  xfs  defaults,nofail,x-systemd.device-timeout=5  0 2' >> /etc/fstab
mount -v /mnt/mysqldata
restorecon -Rv /mnt/mysqldata
df -h
```

2. Ensure the Disk is mounted correctly before proceeding.
3. Define code repositories to be used in software package installations.
4. Create epel repo file:

```
yum install -y https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm
```

Installing MySQL Server

1. Install various tools needed by MySQL:

```
dnf makecache
dnf install -y wget vim tar zip unzip lz4 pigz nmon sysstat numactl libXft psmisc
```

2. Install MySQL:

```
yum  -y module disable mysql
dnf install -y https://dev.mysql.com/get/mysql80-community-release-el8-1.noarch.rpm
dnf install -y mysql-community-server
systemctl enable mysqld
yum update -y
reboot
```

3. After the systems have finished rebooting, verify that MySQL is running:

```
systemctl status mysqld
```

Configuring MySQL

1. Configure and grant access to the HDB system. Execute the following commands on each system so that they can access each other both at the Operating System level and via the MySQL console:

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

2. The temporary MySQL password from the initial installation will appear at the end of the line. Copy the password to the clipboard (you will use it in the next step):

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

3. Enter the temporary password. For this test, we used the password "Password1!"

```
ALTER USER 'root'@'localhost' IDENTIFIED BY 'Password1!';
SELECT CONCAT("'", user, "'@'", host, "'") FROM mysql.user;
```

4. On the MySQL server only, configure MySQL to allow the HammerDB driver system to access the MySQL database:

```
CREATE USER 'root'@'<IP address of the other system>' IDENTIFIED BY 'Password1!';
GRANT ALL ON *.* TO 'root'@'<IP address of the other system>';
FLUSH PRIVILEGES;
ALTER USER 'root'@'<IP address of the other system>' IDENTIFIED WITH mysql_native_password
  BY 'Password1!';
FLUSH PRIVILEGES;
Quit
```

5. Test MySQL access by entering the following command on each host. Note that there is no space between the "-u" & "-p" switches:

```
mysql -uroot -pPassword1! -h <IP address of the other host>
```

Installing HammerDB

1. Copy HammerDB-4.1-Linux-x64-installer.run to /tmp.
2. Switch to tmp and install HDB:

```
cd /tmp;chmod 755 *.run;./HammerDB-4.1-Linux-x64-installer.run
```

3. Follow the instructions, but do not display the ReadMe or run HammerDB.
4. Run the HammerDB CLI to verify that it will operate on MySQLi:

```
cd /opt/HammerDB-4.1 ; ./hammerdbcli
librarycheck
quit
```

5. Copy 4250_schemabuild.tcl to the hosts then to the HammerDB directory and also to root's script folder:

```
cp /tmp/4250_schemabuild.tcl /opt/HammerDB-4.1/;chmod 755 /opt/HammerDB-4.1/*.tcl
```

Running the tests

1. From the HDB driver client, run the test:

```
cd ~/scripts
date;./run_test.sh <IP address of the MySQL Server> 4250;date
```

Scripts

mysql_host_prepare.sh

```
#!/bin/bash

setenforce 0

sed -i 's/SELINUX=.*/SELINUX=Disabled/' /etc/selinux/config

systemctl disable --now firewalld

#### System tuning ####

tuned-adm profile virtual-guest

cat <<EOF >/etc/sysctl.conf

vm.swappiness = 1

fs.aio-max-nr = 1048576

EOF

sysctl -p
```

Run_test.sh

```
#!/bin/bash

echo "Usage: $0 TEST_HOST WAREHOUSE_COUNT"

TEST_HOST=${1:-remotehost}

CLIENT_HOST=$(hostname -s)

WAREHOUSE_COUNT=${2}

APP=mysql

HOST_PREPARE=${APP}_host_prepare.sh

SCHEMA_BUILD=${WAREHOUSE_COUNT}_schemabuild.tcl

MYCNF=my-${WAREHOUSE_COUNT}.cnf

HDB_DIR=/opt/HammerDB-4.1

HDB_SCRIPT=hdb_tpcc_${APP}_${WAREHOUSE_COUNT}wh.tcl

HDB_RUN=run_${HDB_SCRIPT}

RUNNING_FILE=benchmark_running.txt

RAMPUP=10 # minutes

DURATION=20 # minutes

STEP=2  # seconds

IDLE=30 # seconds

WARMUP=$((RAMPUP*60))

RUNTIME=$((DURATION*60))

SAMPLES_TOTAL=$(((WARMUP+RUNTIME)/STEP+5))
```

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

# Check for files

if [ ! -e ${HOST_PREPARE} ]; then

  echo "Missing host prepare script: ${HOST_PREPARE}"

  exit

fi

if [ ! -e ${MYCNF} ]; then

  echo "Missing my.cnf config: ${MYCNF}"

  exit

fi

if [ ! -e ${HDB_DIR}/hammerdbcli ]; then

  echo "Missing hammerdbcli missing: ${HDB_DIR}/hammerdbcli"

  exit

fi

if [ ! -e ${HDB_SCRIPT} ]; then

  echo "Missing HammerDB script: ${HDB_SCRIPT}"

  exit

fi

# Test SSH host access

sed -i "/${TEST_HOST}/d" ~/.ssh/known_hosts

ssh ${TEST_HOST} 'hostname -f' || exit

# Get host info

REMOTE_HOSTNAME="$(ssh ${TEST_HOST} 'hostname -s')"

INSTANCE_TYPE=Azure_HB120-64rs_v3

echo "INSTANCE_TYPE: ${INSTANCE_TYPE}"

INSTANCE_CPU="$(ssh ${TEST_HOST} 'awk "/model name/{print \$4\$5\$6;exit}" /proc/cpuinfo | sed -e "s/
//g" -e "s/CPU//"')"

echo "INSTANCE_CPU: ${INSTANCE_CPU}"

sleep 1

# Check if benchmark is already running

if [ -e ${RUNNING_FILE} ]; then

  echo "Benchmark already running: $(cat ${RUNNING_FILE})"

  RUNNING_HOST=$(awk '{print $1}' ${RUNNING_FILE})

  if [[ "${RUNNING_HOST}" == "${TEST_HOST}" ]]; then

    echo "Test already running on the same remote host. Exiting..."

      exit

  fi
```

```
  sleep 3
  echo "If this is incorrect manually remove the benchmark running file: ${RUNNING_FILE}"
  sleep 3
  echo "Benchmark will pause after restoring database until current benchmark finishes."
  sleep 3
fi
# Prepare Test Host
echo -e "\nPreparing test host.\n"
scp -p ${HOST_PREPARE} ${TEST_HOST}:host_prepare.sh
ssh ${TEST_HOST} "sudo ./host_prepare.sh"
scp ${MYCNF} ${TEST_HOST}:tmp-my.cnf
ssh ${TEST_HOST} "sudo systemctl stop ${APP}d ; sudo cp -vf tmp-my.cnf /etc/my.cnf"
ssh ${TEST_HOST} "sudo systemctl start ${APP}d"
sleep 10
ssh ${TEST_HOST} "sudo mysqladmin -uroot -pPassword1! -f drop tpcc"
# 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
echo "Local" > ${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} 'Local' > ${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
# Build Schema
echo -e "\nBuilding Schema.\n"
rm -f /tmp/hammerdb.log
pushd ${HDB_DIR}
./hammerdbcli auto ${SCHEMA_BUILD} > /tmp/hammerdb_${SCHEMA_BUILD}.log
pushd
cp -pvf /tmp/hammerdb.log ${RESULTS_DIR}/
cp -pvf /tmp/hammerdb_${SCHEMA_BUILD}.log ${RESULTS_DIR}/
cp -pvf ${HDB_DIR}/${SCHEMA_BUILD} ${RESULTS_DIR}/
# 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
# Remove benchmark running file
rm -f ${RUNNING_FILE}
```

hdb_tpcc_mysql_4250wh.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 10.0.0.4

diset connection mysql_port 3306

diset tpcc mysql_user root

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 4250

diset tpcc mysql_num_vu 240

diset tpcc mysql_rampup 10

diset tpcc mysql_duration 20

vuset logtotemp 1

loadscript

vuset vu 240

vucreate

vurun

wait_to_complete

vwait forever
```

my-4250.cnf

```
[mysqld]

datadir=/mnt/mysqldata/data

default_authentication_plugin=mysql_native_password

socket=/var/lib/mysql/mysql.sock

log-error=/var/log/mysqld.log

pid-file=/var/run/mysqld/mysqld.pid

port=3306

bind_address=0.0.0.0

# general

max_connections=4000

table_open_cache=8000

table_open_cache_instances=16

back_log=1500

default_password_lifetime=0

ssl=0

performance_schema=OFF

max_prepared_stmt_count=128000

skip_log_bin=1

character_set_server=latin1

collation_server=latin1_swedish_ci

transaction_isolation=REPEATABLE-READ

# files

innodb_file_per_table

innodb_log_file_size=4G #changed

innodb_log_files_in_group=8 #changed

innodb_open_files=4000

# buffers

innodb_buffer_pool_size=358G #Milan for 448GB RAM

#innodb_buffer_pool_size=403G #Cascade Lake for 504GBs RAM

innodb_buffer_pool_instances=16

innodb_log_buffer_size=64M

# tune

#innodb_numa_interleave=OFF

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='%'
```

4250_schemabuild.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 10.0.0.4

diset connection mysql_port 3306

diset tpcc mysql_user root

diset tpcc mysql_pass Password1!

diset tpcc mysql_storage_engine innodb

diset tpcc mysql_partition true

diset tpcc mysql_count_ware 4250

diset tpcc mysql_num_vu 240

print dict

buildschema

waittocomplete
```

CentOS Linux 8.4 with a HiBench deployment

Creating and configuring the Azure VM instances

This section contains the steps we took to create our two instances for running HammerDB benchmark and MySQL database software.

Installing CentOS Linux 8.4 on Azure VMs

1. In Azure Services Home screen, click the Virtual Machines icon, choose Create, and choose Virtual Machine.
2. Under Basic, make the following selections:
    a. Select the appropriate Resource Group.
    b. For Virtual Machine Name, name the VM.
    c. For Region, choose the appropriate region.
    d. For Availability options, Select No infrastructure redundancy required
    e. For Image, click the drop-down menu, and select CentOS-Based 8.4 - Gen 1.
    f. For Size, click the drop-down menu, and select Standard_HB120-64rs_v3.
    g. For Authentication type, select SSH Public key.
    h. For Username, leave the default of azureuser.
    i. For SSH Public key source, select Generate new key pair.
    j. For Key pair name, create a new key named HiBench-key.
    k. For Public inbound ports, click the radio button beside Allow selected ports.
    l. For Select inbound ports, select SSH (22).
    m. Click Next: Disks.
3. Under Disks, make the following selections:
    a. For OS Disk type: Premium SSD (locally redundant).
    b. For SSE Encryption type, select default.
    c. Click Next: Networking.
4. Under Networking, make the following selections:
    a. For Virtual Network, select the appropriate Virtual Network.
    b. For Subnet: select default Subnet.
    c. For Public IP, select (new) name.
    d. For NIC security group, select Advanced.
    e. For Configure network security group, select the appropriate NSG.
    f. Set Accelerated networking to on.
    g. Set load balancing to off.
    h. Click Next: Management.
    i. Click through Next: Advanced, then Next: Tags, accepting all defaults.
    j. Click Next: Review + create.
5. Ensure all values are correct, and click Create.

Configuring security and host access

To allow for easy access from one host to another and to provide access to the test hosts from an SSH utility such as Putty, you must configure certain files associated with security.

1. Using PuTTy, log onto the VM using azureuser and private key pair.
2. Issue the sudo su command switch user to root:

```
sudo su
```

3. Change the root password:

```
passwd root
```

4. Change the /etc/ssh/sshd_config file to allow password authentication, and reset the ssh daemon:

```
vi /etc/ssh/sshd_config
```

   - uncomment PasswordAuthentication yes
   - Comment out PasswordAuthentication no
   - uncomment `PermitRootLogin yes`

5. Reset the ssh daemon:

```
systemctl restart sshd
```

6. Disable the firewall:

```
systemctl stop firewalld
systemctl disable firewalld
```

7. Edit the SELinux config file by using the vi text editor:

```
vi /etc/selinux/config
SELINUX=permissive
```

Setting up host access

1. Using PuTTy, log into the instance using azureuser as the user using the private key pair that we created earlier.
2. Issue the sudo su command to switch the user to root:

```
sudo su
```

3. Change the root password:

```
passwd root
```

4. Create the epel repo:

```
yum install -y https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm
```

5. Rebuild the DNF cache:

```
dnf makecache
```

6. Install ancillary software:

```
yum -y install nmon python3 vim tar wget java-1.8.0-openjdk maven git blas64 lapack64 python2 bc curl
```

7. Set Python2 as the primary version:

```
alternatives --set python /usr/bin/python2
```

8. Update the host:

```
yum update -y
```

9. Edit the hosts file and add the IP addresses of all nodes in the cluster:

```
vi /etc/hosts
<IP address of manager node>      manager-01     manager-01
<IP address of worker node 1>     worker-01    worker-01
<IP address of worker node 2>     worker-02    worker-02
<IP address of worker node 3>     worker-03    worker-03
<IP address of worker node 4>     worker-04    worker-04
```

10. Configure the systems to log into each other without prompting for a password:

```
mkdir -p /root/.ssh
chmod 700 /root/.ssh
cd /root/.ssh
ssh-keygen -t rsa -q -f id_rsa -N ''
cp id_rsa.pub authorized_keys
echo "StrictHostKeyChecking=no" > config
```

11. Copy keys from server to worker nodes:

```
scp /root/.ssh/* <IP address of worker node 1>:/root/.ssh/
scp /root/.ssh/* <IP address of worker node 2>:/root/.ssh/
scp /root/.ssh/* <IP address of worker node 3>:/root/.ssh/
scp /root/.ssh/* <IP address of worker node 4>:/root/.ssh/
```

12. Test access from server to client and client to server by using the ssh command on both systems:

```
ssh <IP address of other systems>
```

    Each system should be able to access the other without prompting for a password.

13. Go back to /root directory:

```
cd
```

14. Download Hadoop and Spark:

```
wget http://www.gtlib.gatech.edu/pub/apache/spark/spark-3.1.2/spark-3.1.2-bin-hadoop3.2.tgz
wget http://www.gtlib.gatech.edu/pub/apache/hadoop/common/hadoop-3.3.0/hadoop-3.3.0.tar.gz
```

15. Modify your bash profile by adding the following lines:

```
JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.302.b08-0.el8_4.x86_64/jre
PATH=$PATH:$HOME/bin:/opt/yarn/hadoop-3.3.0/bin
```

16. Reboot the system.
17. Add in the Hadoop users:

```
groupadd hadoop
useradd -g hadoop yarn
useradd -g hadoop hdfs
useradd -g hadoop mapred
```

18. Create default Hadoop directories and set their permissions:

```
mkdir -p /var/data/hadoop/hdfs/nn
mkdir -p /var/data/hadoop/hdfs/snn
mkdir -p /var/data/hadoop/hdfs/dn
chown hdfs:hadoop /var/data/hadoop/hdfs/ -R
mkdir -p /var/log/hadoop/yarn
chown yarn:hadoop /var/log/hadoop/yarn/ -R
mkdir -p /opt/yarn
```

19. Extract the Hadoop and Spark compressed files:

```
cd /opt/yarn
tar xvzf /root/hadoop-3.3.0.tar.gz
tar -xvzf ~/spark-3.1.2-bin-hadoop3.2.tgz
```

20. Move into the Hadoop directory and make a yarn directory:

```
cd hadoop-3.3.0/
mkdir logs
chmod g+w logs
chown yarn:hadoop . -R
```

21. Navigate into the Hadoop configuration directory:

```
cd etc/hadoop/
```

22. Modify the Hadoop configuration files with the following settings:

core-site.xml

```
<configuration>
       <property>
               <name>fs.default.name</name>
               <value>hdfs://[ MANAGER IP ADDRESS]:9000</value>
       </property>
       <property>
               <name>hadoop.http.staticuser.user</name>
               <value>hdfs</value>
       </property>
</configuration>
```

hdfs-site.xml

```
<configuration>
 <property>
   <name>dfs.replication</name>
   <value>3</value>
 </property>
 <property>
   <name>dfs.namenode.name.dir</name>
   <value>file:/var/data/hadoop/hdfs/nn</value>
 </property>
 <property>
   <name>fs.checkpoint.dir</name>
   <value>file:/var/data/hadoop/hdfs/snn</value>
 </property>
 <property>
   <name>fs.checkpoint.edits.dir</name>
   <value>file:/var/data/hadoop/hdfs/snn</value>
 </property>
 <property>
   <name>dfs.datanode.data.dir</name>
   <value>file:/var/data/hadoop/hdfs/dn</value>
 </property>
</configuration>
```

mapred-site.xml

```
<configuration>
 <property>
   <name>mapreduce.framework.name</name>
   <value>yarn</value>
 </property>
    <property>
            <name>yarn.app.mapreduce.am.env</name>
            <value>HADOOP_MAPRED_HOME=$HADOOP_HOME</value>
    </property>
    <property>
            <name>mapreduce.map.env</name>
            <value>HADOOP_MAPRED_HOME=$HADOOP_HOME</value>
    </property>
    <property>
            <name>mapreduce.reduce.env</name>
            <value>HADOOP_MAPRED_HOME=$HADOOP_HOME</value>
    </property>
</configuration>
```

yarn-site.xml

```
<configuration>
        <property>
                <name>yarn.resourcemanager.hostname</name>
                <value>[MANAGER IP ADDRESS]</value>
        </property>
 <property>
   <name>yarn.nodemanager.aux-services</name>
   <value>mapreduce_shuffle</value>
 </property>
 <property>
   <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
   <value>org.apache.hadoop.mapred.ShuffleHandler</value>
 </property>
</configuration>
```

hadoop-env.sh

23. Uncomment the JAVA_HOME line and replace with the following information:

```
JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.302.b08-0.el8_4.x86_64/jre
```

24. Power off the instance:

```
Shutdown now -P
```

Creating storage for DB

The HiBench Worker Nodes require a separate disk to hold the database.

1. On each of the worker nodes, add a second disk:
2. In the Azure Services Home screen, click the Virtual Machines icon, and click the VM name.
3. In the center of the window, click Disks.
4. Click Create and attach a new disk.
5. Specify the disk name and size, and click Save.

Configuring and starting the cluster

1. Copy /opt/yarn/spark-3.1.2-bin-hadoop3.2/conf/spark-env.sh to each instance and make executable.
2. Append the following line to /opt/yarn/spark-3.0.1-bin-hadoop3.2/conf/spark-env.sh on each instance:

```
export SPARK_LOCAL_DIRS=/var/data/hadoop/hdfs/tmp
```

3. Run /root/setup-testbed script.
4. Perform the following steps on the manager node to install and configure HiBench:

   • Create the directories you will use for HiBench:

```
hdfs dfs -mkdir -p /user/root
hdfs dfs -mkdir /HiBench
hdfs dfs -chown -R root:hadoop /HiBench
hdfs dfs -chown root /user/root
```

   • Navigate to your home directory and download HiBench:

```
cd ~
git clone https://github.com/intel-hadoop/HiBench.git
```

   • Install HiBench for Spark 3.0:

```
cd HiBench/
mvn -Dspark=3.0 -Dscala=2.12 clean package | tee hibench_build.log
cd conf/
```

- Modify the HiBench configuration files with the following information:

hadoop.conf

```
# Hadoop home
hibench.hadoop.home      /opt/yarn/hadoop-3.3.0

# The path of hadoop executable
hibench.hadoop.executable      ${hibench.hadoop.home}/bin/hadoop

# Hadoop configraution directory
hibench.hadoop.configure.dir  ${hibench.hadoop.home}/etc/hadoop

# The root HDFS path to store HiBench data
hibench.hdfs.master       hdfs://[MANAGER IP ADDRESS]:9000

# Hadoop release provider. Supported value: apache, cdh5, hdp
hibench.hadoop.release    apache
```

spark.conf

```
# Spark home
hibench.spark.home      /opt/yarn/spark-3.1.2-bin-hadoop3.2/

# Spark master
#   standalone mode: spark://xxx:7077
#   YARN mode: yarn-client
hibench.spark.master spark://[MANAGER IP ADDRESS]:7077
```

5. Reboot the systems.

Running the tests

1. On the manager node, navigate to /root and execute the following scripts in order:
   - setup-testbed.sh
   - run-test.sh

The run-test script will copy the results copied into the /root/results folder.

Cleaning up the testbed for next run

1. On the manager node, navigate to /root, and execute the reset-testbed.sh script.

Worker scripts

mount-drives.sh

```
DATADRIVE=`lsblk | grep 8T | grep -v efi | awk '{print $1}'`
mkfs.xfs -f /dev/$DATADRIVE
rm -rf /var/data/hadoop/*
mount /dev/$DATADRIVE /var/data/hadoop
mkdir -p /var/data/hadoop/hdfs/nn
mkdir -p /var/data/hadoop/hdfs/snn
mkdir -p /var/data/hadoop/hdfs/dn
chown hdfs:hadoop /var/data/hadoop/hdfs/ -R
mkdir -p /var/data/hadoop/hdfs/tmp
echo "Current drive mappings:"
df -h | grep hadoop
```

start-spark.sh

```
/opt/yarn/hadoop-3.3.0/bin/hdfs --daemon start datanode
/opt/yarn/spark-3.1.2-bin-hadoop3.2/sbin/start-worker.sh spark://10.0.0.5:7077
```

stop-spark.sh

```
/opt/yarn/hadoop-3.3.0/bin/hdfs --daemon stop datanode
/opt/yarn/spark-3.1.2-bin-hadoop3.2/sbin/stop-worker.sh
```

dismount-format-drives.sh

```
DATADRIVE=`lsblk | grep 8T | grep -v efi | awk '{print $1}'`
umount /dev/$DATADRIVE
mkfs.xfs -f /dev/$DATADRIVE
```

Manager scripts

setup-testbed.sh

```
#!/bin/sh
echo " "
echo "Mounting and formatting drives on Workers"
echo " "
ssh -t worker-01 '/root/mount-drives.sh'
ssh -t worker-02 '/root/mount-drives.sh'
ssh -t worker-03 '/root/mount-drives.sh'
ssh -t worker-04 '/root/mount-drives.sh'
sleep 2
echo " "
echo "Clearing caches on Manager"
echo " "
sync; echo 3 > /proc/sys/vm/drop_caches
sleep 2
echo " "
echo "Clearing caches on Workers."
echo " "
ssh -t worker-01 'sync; echo 3 > /proc/sys/vm/drop_caches'
ssh -t worker-02 'sync; echo 3 > /proc/sys/vm/drop_caches'
ssh -t worker-03 'sync; echo 3 > /proc/sys/vm/drop_caches'
ssh -t worker-04 'sync; echo 3 > /proc/sys/vm/drop_caches'
sleep 2
echo " "
echo "Systems ready. Starting Spark on Manager then Workers"
echo " "
echo " "
rm -rf /var/data/hadoop/hdfs/*
/root/start-spark.sh
```

run-test.sh

```
#!/bin/sh
#VARIABLES
PROC=Azure-HB120-64rs_v3
NMON=3600  #Time to let nmon run.
WORKLOAD=kmeans
HBSIZE=bigdata
START=`date +%m%d%y%H%M`
LOGDIR="HiBench_"$PROC"_"$WORKLOAD"_"$HBSIZE"_"$START
echo "Making results folder."
mkdir /root/results/$LOGDIR
sleep 2
echo "Copying config files to the results folder."
cp -r /root/HiBench/conf/*.conf /root/results/$LOGDIR
cp /opt/yarn/hadoop-3.3.0/etc/hadoop/core-site.xml /root/results/$LOGDIR
cp /opt/yarn/hadoop-3.3.0/etc/hadoop/yarn-site.xml /root/results/$LOGDIR
cp /opt/yarn/hadoop-3.3.0/etc/hadoop/mapred-site.xml /root/results/$LOGDIR
cp /opt/yarn/hadoop-3.3.0/etc/hadoop/hdfs-site.xml /root/results/$LOGDIR
cp /opt/yarn/hadoop-3.3.0/etc/hadoop/hadoop-env.sh /root/results/$LOGDIR
cp /root/HiBench/report/kmeans/spark/conf/kmeans.conf /root/results/$LOGDIR
cp -r /root/*.sh /root/results/$LOGDIR
echo "Running the Kmeans Prepare script."
/root/HiBench/bin/workloads/ml/kmeans/prepare/prepare.sh
echo "Pausing 2 minutes before running test"
sleep 120
echo "Starting nmon on Manager and Workers to run for "$NMON" seconds."
nmon -F /tmp/manager-01.nmon -s1 -c$NMON -J -t
ssh -t worker-01 nmon -F /tmp/worker-01.nmon -s1 -c$NMON -J -t
ssh -t worker-02 nmon -F /tmp/worker-02.nmon -s1 -c$NMON -J -t
ssh -t worker-03 nmon -F /tmp/worker-03.nmon -s1 -c$NMON -J -t
ssh -t worker-04 nmon -F /tmp/worker-04.nmon -s1 -c$NMON -J -t
sleep 2
echo "Starting the Kmeans test."
time /root/HiBench/bin/workloads/ml/kmeans/spark/run.sh
echo "Sleeping for 60 seconds to let things settle."
sleep 60
echo "Copying out files to the results folder."
cp /opt/yarn/spark-3.1.2-bin-hadoop3.2/logs/*.out /root/results/$LOGDIR
scp worker-01:/opt/yarn/spark-3.1.2-bin-hadoop3.2/logs/*.out /root/results/$LOGDIR
scp worker-02:/opt/yarn/spark-3.1.2-bin-hadoop3.2/logs/*.out /root/results/$LOGDIR
scp worker-03:/opt/yarn/spark-3.1.2-bin-hadoop3.2/logs/*.out /root/results/$LOGDIR
scp worker-04:/opt/yarn/spark-3.1.2-bin-hadoop3.2/logs/*.out /root/results/$LOGDIR
echo "Copying nmon data from Workers to Manager in the /tmp folder."
scp worker-01:/tmp/worker-01.nmon /tmp/
scp worker-02:/tmp/worker-02.nmon /tmp/
scp worker-03:/tmp/worker-03.nmon /tmp/
scp worker-04:/tmp/worker-04.nmon /tmp/
mv /tmp/*.nmon /root/results/$LOGDIR
mv /tmp/*.log /root/results/$LOGDIR
cp /root/HiBench/report/hibench.report /root/results/$LOGDIR
echo "Running nmonchart against nmon files."
nmonchart /root/results/$LOGDIR/worker-01.nmon
nmonchart /root/results/$LOGDIR/worker-02.nmon
nmonchart /root/results/$LOGDIR/worker-03.nmon
nmonchart /root/results/$LOGDIR/worker-04.nmon
nmonchart /root/results/$LOGDIR/manager-01.nmon
echo "Testing complete.  Logs are located in the directory below:"
echo "/root/results/"$LOGDIR
```

reset-testbed.sh

```
#!/bin/sh
echo " "
echo "Stopping Manager and Worker Spark"
echo " "
/root/stop-spark.sh
sleep 2
echo " "
echo "Dropping caches on Manager"
echo " "
sync; echo 3 > /proc/sys/vm/drop_caches
sleep 2
echo " "
echo "Dropping caches on Workers."
echo " "
ssh -t worker-01 'sync; echo 3 > /proc/sys/vm/drop_caches'
ssh -t worker-02 'sync; echo 3 > /proc/sys/vm/drop_caches'
ssh -t worker-03 'sync; echo 3 > /proc/sys/vm/drop_caches'
ssh -t worker-04 'sync; echo 3 > /proc/sys/vm/drop_caches'
sleep 2
echo " "
echo "Cleaning up /tmp folder on Workers"
echo " "
ssh -t worker-01 'rm -rf /tmp/*'
ssh -t worker-02 'rm -rf /tmp/*'
ssh -t worker-03 'rm -rf /tmp/*'
ssh -t worker-04 'rm -rf /tmp/*'
sleep 2
echo " "
echo "Cleaning up /tmp folder on Manager"
echo " "
rm -rf /tmp/*
sleep 2
echo " "
echo "Dismounting and formatting drives on Workers"
echo " "
ssh -t worker-01 '/root/dismount-format-drives.sh'
ssh -t worker-02 '/root/dismount-format-drives.sh'
ssh -t worker-03 '/root/dismount-format-drives.sh'
ssh -t worker-04 '/root/dismount-format-drives.sh'
```

stop-spark.sh

```
echo "Stopping Spark on Worker-01."
ssh worker-01 '~/stop-spark.sh'
echo "Stopping Spark on Worker-02."
ssh worker-02 '~/stop-spark.sh'
echo "Stopping Spark on Worker-03."
ssh worker-03 '~/stop-spark.sh'
echo "Stopping Spark on Worker-04."
ssh worker-04 '~/stop-spark.sh'
echo "Stopping Spark on Manager."
/opt/yarn/hadoop-3.3.0/bin/hdfs --daemon stop namenode
sleep 2
/opt/yarn/hadoop-3.3.0/bin/hdfs --daemon stop secondarynamenode
sleep 2
/opt/yarn/hadoop-3.3.0/bin/yarn --daemon stop resourcemanager
sleep 2
/opt/yarn/hadoop-3.3.0/bin/yarn --daemon stop nodemanager
sleep 2
/opt/yarn/spark-3.1.2-bin-hadoop3.2/sbin/stop-master.sh
sleep 2
```

start-spark.sh

```
#!/bin/sh
echo "Formatting HDFS filesystem"
hdfs namenode -format
sleep 5
echo "Starting Spark on Manager"
/opt/yarn/hadoop-3.3.0/bin/hdfs --daemon start namenode
/opt/yarn/hadoop-3.3.0/bin/hdfs --daemon start secondarynamenode
/opt/yarn/hadoop-3.3.0/bin/yarn --daemon start resourcemanager
/opt/yarn/hadoop-3.3.0/bin/yarn --daemon start nodemanager
/opt/yarn/spark-3.1.2-bin-hadoop3.2/sbin/start-master.sh
echo "Manager has started. Pausing for 10 seconds before starting Workers."
sleep 10
echo "Starting Spark on Workers"
sleep 1
echo "Starting Spark on Worker-01"
ssh worker-01 '/root/start-spark.sh'
echo "Starting Spark on Worker-02"
ssh worker-02 '/root/start-spark.sh'
echo "Starting Spark on Worker-03"
ssh worker-03 '/root/start-spark.sh'
echo "Starting Spark on Worker-04"
ssh worker-04 '/root/start-spark.sh'
echo "Workers have started."
sleep 2
echo "Systems are ready for testing"
```

Read the report at https://facts.pt/RzLQcyo ▶

This project was commissioned by AMD.

Principled Technologies®

Facts matter.®