The science behind the report:

Analyze more data per second on Apache Spark clusters with new Microsoft Azure VMs featuring 2nd Generation Intel Xeon Scalable Processors – Cascade Lake

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 Analyze more data per second on Apache Spark clusters with new Microsoft Azure VMs featuring 2nd Generation Intel Xeon Scalable Processors – Cascade Lake.

We concluded our hands-on testing on December 3, 2020. During testing, we determined the appropriate hardware and software configurations and applied updates as they became available. The results in this report reflect configurations that we finalized on November 18, 2020 or earlier. Unavoidably, these configurations may not represent the latest versions available when this report appears.

Our results

Table 1: Comparison of results for Naive Bayesian classification and k-means clustering algorithm tests from the HiBench benchmarking suite on small (8vCPU) Microsoft Azure VMs running Apache Spark. The Dd8s_v4 VM featured Cascade Lake processors, while the D8s_v3 VM featured Broadwell processors. Source: Principled Technologies.

Small VMs (8 vCPUs, 32 GB database)			
Naive Bayesian classification	D8s_v3	D8ds_v4	D8ds_v4 advantage
Throughput (MB/s)	29,721,409	36,816,927	1.23x
Time (seconds)	2520.948	2035.100	
k-means clustering algorithm	D8s_v3	D8ds_v4	D8ds_v4 advantage
Throughput (MB/s)	32,168,290	39,824,790	1.23x
Time (seconds)	7,424.909	5,997.436	

Table 2: Comparison of results for Naive Bayesian classification and k-means clustering algorithm tests from the HiBench benchmarking suite on medium (16vCPU) Microsoft Azure VMs running Apache Spark. The Dd16s_v4 VM was featured Cascade Lake processors, while the D16s_v3 VM featured Broadwell processors. Source: Principled Technologies.

Medium VMs (16 vCPUs, 64GB database)			
Naive Bayesian classification	D16s_v3	D16ds_v4	D16ds_v4 advantage
Throughput (MB/s)	103,923,482	152,625,461	1.46x
Time (seconds)	720.974	490.915	
k-means clustering algorithm	D16s_v3	D16ds_v4	D16ds_v4 advantage
Throughput (MB/s)	74,789,806	98,020,935	1.31x
Time (seconds)	3,193.572	2,436.690	

Table 3: Comparison of results for Naive Bayesian classification and k-means clustering algorithm tests from the HiBench benchmarking suite on large (64vCPU) Microsoft Azure VMs running Apache Spark. The Dd64s_v4 VM featured Cascade Lake processors, while the D64s_v3 VM featured Broadwell processors. Source: Principled Technologies.

Large VMs (64 vCPUs, 256GB database)			
Naive Bayesian classification	D64s_v3	D64ds_v4	D64ds_v4 advantage
Throughput (MB/s)	336,728,484	522,399,049	1.55x
Time (seconds)	222.512	143.427	
k-means clustering algorithm	D64s_v3	D64ds_v4	D64ds_v4 advantage
Throughput (MB/s)	218,673,759	333,852,790	1.52x
Time (seconds)	1,092.251	715.425	

System configuration information

Table 4: Detailed information for the Broadwell based VMs.

VM configuration information	D8s_v3	D16s_v3	D64s_v3
Tested by	Principled Technologies	Principled Technologies	Principled Technologies
Test date	12/1/2020	12/1/2020	12/1/2020
CSP/region	Microsoft Azure East US (Zone 1)	Microsoft Azure East US (Zone 1)	Microsoft Azure East US (Zone 1)
Workload & version	HiBench 7.1; Java 1.8.0; Apache Spark 3.0.1; Hadoop: 3.2.1	HiBench 7.1; Java 1.8.0; Apache Spark 3.0.1; Hadoop: 3.2.1	HiBench 7.1; Java 1.8.0; Apache Spark 3.0.1; Hadoop: 3.2.1
Workload-specific parameters	Bayes, bigdata, 90% Reserved Memory Kmeans, bigdata, 90% Reserved Memory	Bayes, bigdata, 90% Reserved Memory Kmeans, bigdata, 90% Reserved Memory	Bayes, bigdata, 90% Reserved Memory Kmeans, bigdata, 90% Reserved Memory
Iterations and result choice	3 runs, median	3 runs, median	3 runs, median
Server platform	D8s_v3	D16s_v3	D64s_v3
BIOS name and version	SMBIOS 2.4	SMBIOS 2.4	SMBIOS 2.4
Operating system name and version/build number	CentOS Linux release 8.2.2004 (Core) 4.18.0-193.19.1.el8_2.x86_64	CentOS Linux release 8.2.2004 (Core) 4.18.0-193.19.1.el8_2.x86_64	CentOS Linux release 8.2.2004 (Core) 4.18.0-193.19.1.el8_2.x86_64
Date of last OS updates/ patches applied	10/06/2020	10/06/2020	10/06/2020
Processor			
Number of processors	8	16	64
Vendor and model	Intel® Xeon® CPU E5-2673 v4	Intel Xeon CPU E5-2673 v4	Intel Xeon CPU E5-2673 v4
Core count (per processor)	4	4	4
Core frequency (GHz)	2.30	2.30	2.30
Stepping	1	1	1
Hyperthreading?	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™ module present?	No	No	No
Total memory (DDR+NVMe RAM)	32	64	256
General hardware			
Storage: Network or Direct attached	Network-attached	Network-attached	Network-attached
Network bandwidth per VM	4,000 Mbps	8,000 Mbps	30,000 Mbps
Storage bandwidth per VM	192 MB/s	384 MB/s	1,200 MB/s

VM configuration information	D8s_v3	D16s_v3	D64s_v3
Local storage (OS)			
Number of drives	1	1	1
Drive size (GB)	30	30	30
Drive information	Standard HDD	Standard HDD	Standard HDD
Local storage (data drive)			
Number of drives	1	1	1
Drive size (GB)	1,024	512	512
Drive information	Premium SSD 200 MB/sec 5,000 IOPS	Ultra SSD 384 MB/sec 6,000 IOPS	Ultra SSD 1,200 MB/sec 8,000 IOPS

Table 5: Detailed configuration information for the Cascade Lake VMs.

VM configuration information	D8ds_v4	D16ds_v4	D64ds_v4
Tested by	Principled Technologies	Principled Technologies	Principled Technologies
Test date	12/1/2020	12/1/2020	12/1/2020
CSP/Region	Microsoft Azure East US (Zone 1)	Microsoft Azure East US (Zone 1)	Microsoft Azure East US (Zone 1)
Workload & version	HiBench 7.1; Java 1.8.0; Apache Spark 3.0.1	HiBench 7.1; Java 1.8.0; Apache Spark 3.0.1	HiBench 7.1; Java 1.8.0; Apache Spark 3.0.1
WL specific parameters	Bayes, bigdata, 90% Reserved Memory Kmeans, bigdata, 90% Reserved Memory	Bayes, bigdata, 90% Reserved Memory Kmeans, bigdata, 90% Reserved Memory	Bayes, bigdata, 90% Reserved Memory Kmeans, bigdata, 90% Reserved Memory
Iterations and result choice	3 runs, median	3 runs, median	3 runs, median
Server platform	D8ds_v4	D16ds_v4	D64ds_v4
BIOS name and version	SMBIOS 3.1.1	SMBIOS 3.1.1	SMBIOS 3.1.1
Operating system name and version/build number	CentOS Linux release 8.2.2004 (Core) 4.18.0-193.19.1.el8_2.x86_64	CentOS Linux release 8.2.2004 (Core) 4.18.0-193.19.1.el8_2.x86_64	CentOS Linux release 8.2.2004 (Core) 4.18.0-193.19.1.el8_2.x86_64
Date of last OS updates/ patches applied	10/06/2020	10/06/2020	10/06/2020
Processor			
Number of processors	8	16	64
Vendor and model	Intel® Xeon® Platinum 8272CL	Intel Xeon Platinum 8272CL	Intel Xeon Platinum 8272CL
Core count (per processor)	4	4	4
Core frequency (GHz)	2.60	2.60	2.60
Stepping	7	7	7
Hyperthreading?	Yes	Yes	Yes
Turbo?	Yes	Yes	Yes
Number of vCPUs per VM	8	16	64

VM configuration information	D8ds_v4	D16ds_v4	D64ds_v4
Memory module(s)			
Total memory in system (GB)	32	64	256
NVMe module present?	No	No	No
Total memory (DDR+NVMe RAM)	32	64	256
General hardware			
Storage: Network or Direct attached	Direct attached	Direct attached	Direct attached
Network bandwidth per VM	4,000 Mbps	8,000 Mbps	30,000 Mbps
Storage bandwidth per VM	192 MB/s	384 MB/s	1,200 MB/s
Local storage (OS)			
Number of drives	1	1	1
Drive size (GB)	30	30	30
Drive information (speed, interface, type)	Standard HDD	Standard HDD	Standard HDD
Local storage (data drive)			
Number of drives	1	1	1
Drive size (GB)	1,024	512	512
Drive information (speed, interface, type)	Premium SSD 200 MB/sec 5,000 IOPS	Ultra SSD 384 MB/sec 6,000 IOPS	Ultra SSD 1,200 MB/sec 8,000 IOPS

How we tested

Testing overview

For this project, we tested Microsoft Azure VMs featuring older Intel Broadwell processors vs. Cascade Lake versions. We ran the kmeans and bayes tests from the HiBench suite to show a performance increase in terms of time to complete each test and total throughput. Our results reflect what customers can expect to see using the newer instance series vs. the older.

Using our methodology to aid your own deployments

While the methodology below describes in great detail how we accomplished our testing, it is not a deployment guide. However, because we include many basic installation steps for operating systems and testing tools, reading our methodology may help with your own installation.

Creating the Centos 8 baseline image

This section contains the steps we took to create our baseline image.

Creating the baseline image VM

1. Log into the Azure Portal, and navigate to the Virtual Machines service.
2. To open the Add VM wizard, click Add.
3. On the Basics tab, set the following:
    a. Choose your Subscription.
    b. Choose your Resource group.
    c. Name the Virtual Machine.
    d. Choose your Region.
    e. Leave the Availability options set to No infrastructure redundancy required.
    f. For Image, choose CentOS-based 8.2.
    g. Leave Azure Spot instance set to No.
    h. Select the instance size you wish to use. We used Standard B4ms.
    i. For the Administrator account, create a new username and password.
    j. Leave Public inbound ports set to Allow selected ports.
    k. For Select inbound ports, choose SSH (22).
4. On the Disks tab, set the following:
    a. For the OS disk type, choose Standard HDD.
    b. Leave the default Encryption type.
5. On the Networking tab, set the following:
    a. Choose your Virtual network.
    b. To create a new Public IP, choose Create new.
    c. Leave the rest of the settings as default.
6. On the Management tab, set the following:
    a. Choose your Diagnostics storage account.
    b. Leave the rest set to their defaults.
7. On the Advanced tab, leave all defaults.
8. On the Tags tab, add any tags you wish to use.
9. On the Review + create tab, review your settings, and click Create.

Configuring Centos 8 and installing Apache Hadoop and Spark

1. Log into your VM.
2. Change the root password:

```
sudo passwd root
```

3. Switch to the root user:

```
su -
```

4. Modify SSH to allow a pre-shared key login:

```
mkdir -p /root/.ssh
chmod 700 /root/.ssh
cd /root/.ssh
ssh-keygen -t rsa -q
cp id_rsa.pub authorized_keys
echo "StrictHostKeyChecking=no" > config
```

5. Set the hostname:

```
hostnamectl set-hostname [HOSTNAME]
```

6. Modify your hosts file to add your hostname to your IP address.
7. Turn off and disable your firewall:

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

8. Edit your selinux to disable its enforcing:

```
setenforce 0
vi /etc/selinux/config (modify "enforcing" to "disabled" in the file)
```

9. Update your OS:

```
yum upgrade -y
```

10. Install the prerequisites via yum:

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

11. Download Hadoop and Spark:

```
wget http://www.gtlib.gatech.edu/pub/apache/spark/spark-3.0.1/spark-3.0.1-bin-hadoop3.2.tgz
wget http://www.gtlib.gatech.edu/pub/apache/hadoop/common/hadoop-3.2.1/hadoop-3.2.1.tar.gz
```

12. Modify your bash profile and add the following lines:

```
JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.265.b01-0.el8_2.x86_64/jre
PATH=$PATH:$HOME/bin:/opt/yarn/hadoop-3.2.1/bin
```

13. Reboot your system.
14. Add the hadoop users:

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

15. 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
```

16. Extract the hadoop and spark compressed files:

```
cd /opt/yarn
tar xvzf /root/hadoop-3.2.1.tar.gz
tar -xvzf ~/spark-3.0.1-bin-hadoop3.2.tgz
```

17. Move into the hadoop directory and make a yarn directory:

```
cd hadoop-3.2.1/
mkdir logs
chmod g+w logs
chown yarn:hadoop . -R
```

18. Navigate into the hadoop configuration directory:

```
cd etc/hadoop/
```

19. 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 HOSTNAME HERE]</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
Uncomment the JAVA_HOME line and add the following information:

```
JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.265.b01-0.el8_2.x86_64/jre
```

20. Power off the instance:

```
poweroff
```

Creating a snapshot of your baseline VM

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

Creating the baseline image

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

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

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

Creating your instances with the baseline image

Creating the worker VMs from your image

In this section, we list the steps required to create a VM from the image we created previously.

1. Open the Azure Portal, and navigate to the Share image galleries service.
2. Click the Shared image gallery you created.
3. Navigate to the image version you created (we used 1.0.0 above), and click Create VM.
4. On the Basics tab, set the following:
    a. Choose your Subscription.
    b. Choose your Resource group.
    c. Enter a Virtual machine name.
    d. Choose Availability Zone, and set the Zone you desire.
    e. Select the instance size you want.
    f. Under Authentication type, select Password.
    g. Leave the rest as defaults.
5. On the Disks tab, set the following:
    a. Change the OS disk type to Standard HDD.
    b. Click Create and attach a new disk.
    c. In the Create a new disk wizard, click Change size, and pick the size and settings of Premium/Ultra SSD that matches your instance type.
    d. Leave the rest as defaults, and click OK.
6. Skip the Networking, Management, and Advanced tabs.
7. On the Tags tab, assign any tags you wish to use.
8. On the Review + create tab, review your settings, and click Create.

Configuring and starting the cluster

1. Set the hostname on the master and each of the worker Instances by editing /etc/hostname.
2. Add the FQDN, hostname, and IP address of each VM to the /etc/hosts file on the manager and worker instances.
3. Verify that you can do passwordless SSH into each instance.
4. On the worker instances, create an XFS filesystem on the data and temp drive:

```
mkfs.xfs /dev/<DATA DRIVE> && mkfs.xfs /dev/<TEMP DRIVE>
```

5. On the worker instances, remove the hadoop data subdirectories:

```
rm -rf /var/data/hadoop/*
```

6. On the worker instances, mount the data drive:

```
mount /dev/<DATA DRIVE> /var/data/hadoop
```

7. On the worker instances, create the hadoop subdirectories for the namenode, secondary namenode, and datanode data and set the 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
```

8. On each of the instances, format the hdfs filesystem and create the temp location:

```
hdfs namenode -format
mkdir -p /var/data/hadoop/hdfs/tmp
```

9. On the worker instances, mount the temp drive.

```
mount /dev/<TEMP DRIVE> /var/data/hadoop/hdfs/tmp
```

10. 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
```

11. Start the Hadoop services and spark on the manager node.

```
/opt/yarn/hadoop-3.2.1/bin/hdfs --daemon start namenode
/opt/yarn/hadoop-3.2.1/bin/hdfs --daemon start secondarynamenode
/opt/yarn/hadoop-3.2.1/bin/yarn --daemon start resourcemanager
/opt/yarn/hadoop-3.2.1/bin/yarn --daemon start nodemanager
/opt/yarn/spark-3.0.1-bin-hadoop3.2/sbin/start-master.sh
```

12. Start the hadoop services and spark on each of the worker nodes.

```
/opt/yarn/hadoop-3.2.1/bin/hdfs --daemon start datanode
/opt/yarn/spark-3.0.1-bin-hadoop3.2/sbin/start-slave.sh spark://[MANAGER IP ADDRESS]:7077
```

13. Perform the following steps on the manager node to install and configure HiBench:

- Create the directories you will use for HiBench by typing the following commands:

```
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 by typing the following commands:

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

- Install HiBench for spark 3.0 by typing the following commands:

```
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-3.2.1

# 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-3.0.1-bin-hadoop3.2/

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

Running the tests

In this section, we list the steps to run the bayes and kmeans benchmark on the VMs under test. First, you will start the benchmark from the manager node using a script that automates the entire process at each instance size.

1. Log into the manager node via SSH.
2. Create a results directory.

```
mkdir ~/results
```

3. Navigate to the directory with your scripts.

```
cd ~/scripts
```

4. Run the benchmark script substituting the VM Instance size and CPU codename. E.g., clx for Cascade Lake, or bdw for Broadwell.

```
./run_test.sh <Instance size> <CPU codename>
```

5. Results are automatically saved in the ~/results directory of the manager node.
6. Run the script at each instance size, and collect the data.

Test scripts

run_test.sh

```
#!/bin/bash
vcpu_count=${1}vCPU
platform=${2}
results_dir=~/results

mkdir -p ${results_dir}/${vcpu_count}
mkdir -p ${results_dir}/${vcpu_count}/bayes
mkdir -p ${results_dir}/${vcpu_count}/kmeans

timestamp=$(date '+%Y%m%d_%H%M%S')
bayes_results=${results_dir}/${vcpu_count}/bayes
kmeans_results=${results_dir}/${vcpu_count}/kmeans

#Start Apache Hadoop & Spark
~/scripts/start-spark.sh
sleep 120

#Delete input and output directories from prior testing
/opt/yarn/hadoop-3.2.1/bin/hadoop --config /opt/yarn/hadoop-3.2.1/etc/hadoop fs -rm -r -skipTrash
hdfs://spark1.gyasi.local:9000/HiBench/Bayes/Input
/opt/yarn/hadoop-3.2.1/bin/hadoop --config /opt/yarn/hadoop-3.2.1/etc/hadoop fs -rm -r -skipTrash
hdfs://spark1.gyasi.local:9000/HiBench/Bayes/Output
/opt/yarn/hadoop-3.2.1/bin/hadoop --config /opt/yarn/hadoop-3.2.1/etc/hadoop fs -rm -r -skipTrash
hdfs://spark1.gyasi.local:9000/HiBench/Kmeans/Input
/opt/yarn/hadoop-3.2.1/bin/hadoop --config /opt/yarn/hadoop-3.2.1/etc/hadoop fs -rm -r -skipTrash
hdfs://spark1.gyasi.local:9000/HiBench/Kmeans/Output
sleep 3

#Clear the memory cache on each worker node
~/scripts/reset-testbed.sh
sleep 30

#Change memory used by Spark based on instance size
if [ ${1} == 8 ]; then
    sed -i '/spark.executor.memory/ c\spark.executor.memory  26g' ~/Hibench/conf/spark.conf
    sed -i '/spark.driver.memory/ c\spark.driver.memory    26g' ~/Hibench/conf/spark.conf
elif [ ${1} == 16 ]; then
    sed -i '/spark.executor.memory/ c\spark.executor.memory  52g' ~/Hibench/conf/spark.conf
    sed -i '/spark.driver.memory/ c\spark.driver.memory    52g' ~/Hibench/conf/spark.conf
elif [ ${1} == 64 ]; then
    sed -i '/spark.executor.memory/ c\spark.executor.memory  205g' ~/Hibench/conf/spark.conf
    sed -i '/spark.driver.memory/ c\spark.driver.memory    205g' ~/Hibench/conf/spark.conf
```

```
else
    echo "Incompatible vCPU count. Please try again with either 8, 16, or 64 vCPUs."
    exit
fi
#Change dataset size to bigdata
sed -i '3s/gigantic/bigdata/' ~/Hibench/conf/hibench.conf

#Run Bayes test and copy results to Bayes subdirectory within the results directory
for i in {1..3}
do
~/scripts/bayes-test.sh
mkdir ${bayes_results}/run$i
    for j in {1..5}
    do
    cp -fv /tmp/spark$j.nmon ${bayes_results}/run$i/${platform}_bayes_node${j}_run${i}_${timestamp}.
nmon
    done
cp -fv ~/Hibench/report/hibench.report ${bayes_results}/run$i/${platform}_bayes_hibench_run${i}.
report
~/scripts/reset-testbed.sh
sleep 30
done

#Remove input and output directories generated for Bayes data
/opt/yarn/hadoop-3.2.1/bin/hadoop --config /opt/yarn/hadoop-3.2.1/etc/hadoop fs -rm -r -skipTrash
hdfs://spark1.gyasi.local:9000/HiBench/Bayes/Input
/opt/yarn/hadoop-3.2.1/bin/hadoop --config /opt/yarn/hadoop-3.2.1/etc/hadoop fs -rm -r -skipTrash
hdfs://spark1.gyasi.local:9000/HiBench/Bayes/Output
sleep 3

#Run Kmeans test and copy results to Kmeans subdirectory within the results directory
for i in {1..3}
do
~/scripts/kmeans-test.sh
mkdir ${kmeans_results}/run$i
    for j in {1..5}
    do
    cp -fv /tmp/spark$j.nmon ${kmeans_results}/run$i/${platform}_kmeans_node${j}_
run${i}_${timestamp}.nmon
    done
cp -fv ~/Hibench/report/hibench.report ${kmeans_results}/run$i/${platform}_kmeans_hibench_run${i}.
report
~/scripts/reset-testbed.sh
sleep 30
done

#Remove input and output directories generated for Kmeans data
/opt/yarn/hadoop-3.2.1/bin/hadoop --config /opt/yarn/hadoop-3.2.1/etc/hadoop fs -rm -r -skipTrash
hdfs://spark1.gyasi.local:9000/HiBench/Kmeans/Input
/opt/yarn/hadoop-3.2.1/bin/hadoop --config /opt/yarn/hadoop-3.2.1/etc/hadoop fs -rm -r -skipTrash
hdfs://spark1.gyasi.local:9000/HiBench/Kmeans/Output
sleep 3

#Stop Spark & Hadoop
~/scripts/stop-spark.sh

#Poweroff worker nodes
for i in {2..5}
do
ssh spark$i poweroff
done
```

start_spark.sh (manager node)

```
/opt/yarn/hadoop-3.2.1/bin/hdfs --daemon start namenode
/opt/yarn/hadoop-3.2.1/bin/hdfs --daemon start secondarynamenode
/opt/yarn/hadoop-3.2.1/bin/yarn --daemon start resourcemanager
/opt/yarn/hadoop-3.2.1/bin/yarn --daemon start nodemanager
/opt/yarn/spark-3.0.1-bin-hadoop3.2/sbin/start-master.sh
sleep 60
ssh spark2 '~/start-spark.sh'
ssh spark3 '~/start-spark.sh'
ssh spark4 '~/start-spark.sh'
ssh spark5 '~/start-spark.sh'
```

start_spark.sh (worker nodes)

```
/opt/yarn/hadoop-3.2.1/bin/hdfs --daemon start datanode
/opt/yarn/spark-3.0.1-bin-hadoop3.2/sbin/start-slave.sh spark://[MANAGER IP ADDRESS]:7077
```

stop_spark.sh (manager node)

```
ssh spark2 '~/stop-spark.sh'
ssh spark3 '~/stop-spark.sh'
ssh spark4 '~/stop-spark.sh'
ssh spark5 '~/stop-spark.sh'
/opt/yarn/hadoop-3.2.1/bin/hdfs --daemon stop namenode
/opt/yarn/hadoop-3.2.1/bin/hdfs --daemon stop secondarynamenode
/opt/yarn/hadoop-3.2.1/bin/yarn --daemon stop resourcemanager
/opt/yarn/hadoop-3.2.1/bin/yarn --daemon stop nodemanager
/opt/yarn/spark-3.0.1-bin-hadoop3.2/sbin/stop-master.sh
```

stop_spark.sh (worker nodes)

```
/opt/yarn/hadoop-3.2.1/bin/hdfs --daemon stop datanode
/opt/yarn/spark-3.0.1-bin-hadoop3.2/sbin/stop-slave.sh
```

reset_testbed.sh

```
#!/bin/sh
sync; echo 3 > /proc/sys/vm/drop_caches
ssh -t spark2 'sync; echo 3 > /proc/sys/vm/drop_caches'
ssh -t spark3 'sync; echo 3 > /proc/sys/vm/drop_caches'
ssh -t spark4 'sync; echo 3 > /proc/sys/vm/drop_caches'
ssh -t spark5 'sync; echo 3 > /proc/sys/vm/drop_caches'
```

bayes-test.sh

```
#!/bin/sh
echo "Preparing Bayes test"
echo " "
echo " "
sleep 5
~/Hibench/bin/workloads/ml/bayes/prepare/prepare.sh
sleep 60

nmon -F /tmp/spark1.nmon -s1 -c3600 -J -t

for i in {2..5}
do
ssh spark$i nmon -F /tmp/spark$i.nmon -s1 -c3600 -J -t
done

echo "Running Bayes test"
echo " "
echo " "
sleep 5
time ~/Hibench/bin/workloads/ml/bayes/spark/run.sh
sleep 5
```

```
pkill nmon

for i in {2..5}
do
ssh spark$i pkill nmon
done

for i in {2..5}
do
scp spark$i:/tmp/spark$i.nmon /tmp/
done
```

kmeans-test.sh

```
#!/bin/sh

echo "Preparing Kmeans test"
echo " "
echo " "
sleep 5
~/Hibench/bin/workloads/ml/kmeans/prepare/prepare.sh
sleep 60

nmon -F /tmp/spark1.nmon -s1 -c3600 -J -t

for i in {2..5}
do
ssh spark$i nmon -F /tmp/spark$i.nmon -s1 -c3600 -J -t
done

echo "Running Kmeans test"
echo " "
echo " "
sleep 5
time ~/Hibench/bin/workloads/ml/kmeans/spark/run.sh
sleep 5

pkill nmon

for i in {2..5}
do
ssh spark$i pkill nmon
done

for i in {2..5}
do
scp spark$i:/tmp/spark$i.nmon /tmp/
done
```


This project was commissioned by Intel.