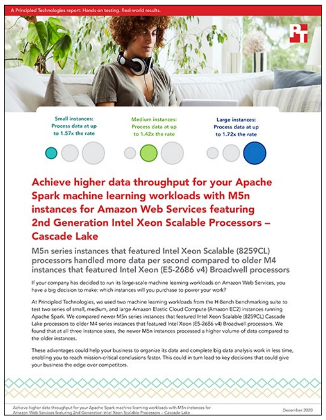
A Principled Technologies report: Hands-on testing. Real-world results.

Small instances: Process data at up to 1.57x the rate

Medium instances: Process data at up to 1.42x the rate

Large instances: Process data at up to 1.72x the rate

Achieve higher data throughput for your Apache Spark machine learning workloads with M5n instances for Amazon Web Services featuring 2nd Generation Intel Xeon Scalable Processors – Cascade Lake

M5n series instances that featured Intel Xeon Scalable (8259CL) processors handled more data per second compared to older M4 instances that featured Intel Xeon (E5-2686 v4) Broadwell processors

If your company has decided to run its large-scale machine learning workloads on Amazon Web Services, you have a big decision to make: which instances will you purchase to power your work?

At Principled Technologies, we used two machine learning workloads from the HiBench benchmarking suite to test two series of small, medium, and large Amazon Elastic Cloud Compute (Amazon EC2) instances running Apache Spark. We compared newer M5n series instances that featured Intel Xeon Scalable (8259CL) Cascade Lake processors to older M4 series instances that featured Intel Xeon (E5-2686 v4) Broadwell processors. We found that at all three instance sizes, the newer M5n instances processed a higher volume of data compared to the older instances.

These advantages could help your business to organize its data and complete big data analysis work in less time, enabling you to reach mission-critical conclusions faster. This could in turn lead to key decisions that could give your business the edge over competitors.

Achieve higher data throughput for your Apache Spark machine learning workloads with M5n instances for Amazon Web Services featuring 2nd Generation Intel Xeon Scalable Processors – Cascade Lake

December 2020

The science behind the report:

Achieve higher data throughput for your Apache Spark machine learning workloads with M5n instances for Amazon Web Services 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 Achieve higher data throughput for your Apache Spark machine learning workloads with M5n instances for Amazon Web Services 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 20, 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) Amazon EC2 instances running Apache Spark. The m5n.2xlarge instance featured Cascade Lake processors, while the m4.2xlarge instance featured Broadwell processors.

Small instances (8 vCPUs)			
Naive Bayesian classification	m4.2xlarge	m5n.2xlarge	m5n advantage
Throughput (MB/s)	29,637,445	36,396,624	1.22x
Time (seconds)	2,528.090	2,058.601	
k-means clustering algorithm	m4.2xlarge	m5n.2xlarge	m5n advantage
Throughput (MB/s)	27,619,090	43,377,951	1.57x
Time (seconds)	8,647.882	5,506.176	

Table 2: Comparison of results for Naive Bayesian classification and k-means clustering algorithm tests from the HiBench benchmarking suite on medium (16vCPU) Amazon EC2 instances running Apache Spark. The m5n.4xlarge instance featured Cascade Lake processors, while the m4.4xlarge instance featured Broadwell processors.

Medium instances (16 vCPUs)			
Naive Bayesian classification	m4.4xlarge	m5n.4xlarge	m5n advantage
Throughput (MB/s)	85,855,636	105,513,559	1.22x
Time (seconds)	872.699	710.109	
k-means clustering algorithm	m4.4xlarge	m5n.4xlarge	m5n advantage
Throughput (MB/s)	66,696,052	95,013,003	1.42x
Time (seconds)	3,581.121	2,513.831	

Table 3: Comparison of results for Naive Bayesian classification and k-means clustering algorithm tests from the HiBench benchmarking suite on large (64vCPU) Amazon EC2 instances running Apache Spark. The m5n.16xlarge instance featured Cascade Lake processors, while the m4.16xlarge instance featured Broadwell processors.

Large instances (64 vCPUs)			
Naive Bayesian classification			m5n advantage
Throughput (MB/s)	338,019,726	529,270,148	1.56x
Time (seconds)	221.662	141.565	
k-means clustering algorithm			m5n advantage
Throughput (MB/s)	141,121,360	243,951,314	1.72x
Time (seconds)	1,692.491	979.075	

System configuration information

Table 4: Detailed information for the Intel Broadwell processor-based VMs.

VM configuration information	m4.2xlarge	m4.4xlarge	m4.16xlarge
Tested by	Principled Technologies	Principled Technologies	Principled Technologies
Test date	12/01/2020	12/01/2020	12/01/2020
CSP/region	us-east1-f	us-east1-f	us-east1-f
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
WL specific parameters	In test script below	In test script below	In test script below
Iterations and result choice	3 runs, median	3 runs, median	3 runs, median
Server platform	m4.2xlarge	m4.4xlarge	M4.16xlarge
BIOS name and version	Xen 4.2.amazon, 8/24/2006	Xen 4.2.amazon, 8/24/2006	Xen 4.2.amazon, 8/24/2006
Operating system name and version/build number	CentOS 8.2	CentOS 8.2	CentOS 8.2
Date of last OS updates/ patches applied	11/15/2020	11/15/2020	11/15/2020
Processor			
Number of processors	1	1	2
Vendor and model	Intel® Xeon® CPU E5-2686 v4	Intel Xeon CPU E5-2686 v4	Intel Xeon CPU E5-2686 v4
Core count (per processor)	4	8	16
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	High	High	10 Gb
Storage bandwidth per VM	1,000 Mbps	2,000 Mbps	10,000 Mbps

VM configuration information	m4.2xlarge	m4.4xlarge	m4.16xlarge
Local storage (OS)			
Number of drives	1	1	1
Drive size (GB)	50	50	50
Drive information	gp2	gp2	gp2
Local storage (data drive)			
Number of drives	1	1	1
Drive size (GB)	600	600	600
IOPS	2,000	2,000	2,000
Drive information	io1	io1	io1
Local storage (log drive)			
Number of drives	1	1	1
Drive size (GB)	100	100	100
IOPs	2,000	2,000	2,000
Drive information	io1	io1	io1
Network adapter			
Vendor and model	Intel 82599	Intel 82599	Amazon Elastic Network Adapter
Number and type of ports	1x 10Gb	1x 10Gb	1x 25Gb

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

VM configuration information	m5n.2xlarge	m5n.4xlarge	m5n.16xlarge
Tested by	Principled Technologies	Principled Technologies	Principled Technologies
Test date	12/01/2020	12/01/2020	12/01/2020
CSP/Region	us-east1-f	us-east1-f	us-east1-f
Workload and 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
Workload-specific parameters	In test script below	In test script below	In test script below
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	CentOS 8.2	CentOS 8.2	CentOS 8.2
Date of last OS updates/ patches applied	11/15/2020	11/15/2020	11/15/2020
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)	4	8	16

VM configuration information	m5n.2xlarge	m5n.4xlarge	m5n.16xlarge
Core frequency (GHz)	2.50	2.50	2.50
Stepping	7	7	7
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 memory 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	Up to 25 Gb	Up to 25 Gb	75 Gb
Storage bandwidth per VM	Up to 4,750 Mbps	4,750 Mbps	13,600 Mbps
Local storage (OS)			
Number of drives	1	1	1
Drive size (GB)	50	50	50
Drive information	gp2	gp2	gp2
Local storage (data drive)			
Number of drives	1	1	1
Drive size (GB)	600	600	600
IOPS	2,000	2,000	2,000
Drive information	io1	io1	io1
Local storage (log drive)			
Number of drives	1	1	1
Drive size (GB)	100	100	100
IOPS	2,000	2,000	2,000
Drive information	io1	io1	io1
Network adapter			
Vendor and model	Amazon Elastic Network Adapter	Amazon Elastic Network Adapter	Amazon Elastic Network Adapter
Number and type of ports	1x 25Gb	1x 25Gb	1x 75Gb

How we tested

Testing overview

For this project, we tested AWS instances featuring older Intel 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 AWS and navigate to the AWS Management Console.
2. Click EC2.
3. Click Launch instance. To open the Launch Instance wizard, select Launch instance from the drop-down menu.
4. In the search window, enter Centos 8 and press Enter.
5. On the AWS Marketplace tab, click the Select button next to Centos 8 base by Amazon Web Services.
6. On the Choose Instance Type tab, select t2.medium, and 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: 30GB
    - Volume Type: gp2
    - Delete on Termination: Checked
    - Encryption: Not Encrypted
10. Click Next: Add Tags
11. On the Add Tags tab, set the following:
    - ProjectName: Gyasi
    - Study: 2
12. Click Next: Configure Security Group
13. On the Configure Security Group tab, leave defaults.
14. Click Review and Launch.
15. On the Review Tab, click Launch.
16. Choose the appropriate option for the key pair, then click Launch Instances.

Configuring Centos 8 and installing Apache Hadoop and Spark

1. Via ssh, log into the postgresql instance.
2. Log into your VM.
3. Change the root password.

```
sudo passwd root
```

4. Switch to the root user.

```
su -
```

5. 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
```

6. Set the hostname by typing the following command:

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

7. To add your hostname to your IP address, modify your hosts file.
8. Turn off and disable your firewall:

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

Edit your selinux to disable its enforcing:

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

Update your OS:

```
yum upgrade -y
```

Install the prerequisites via yum:

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

Download Apache Hadoop and Apache 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
```

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
```

Reboot your system.

9. Add in the hadoop users

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

10. Create default hadoop directories:

```
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
```

11. 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
```

12. 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
```

13. Navigate into the hadoop configuration directory:

```
cd etc/hadoop/
```

14. 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
```

Power off the instance:

```
poweroff
```

Creating an AMI of your basline VM

1. Log into AWS, and navigate to the AWS Management Console.
2. Click EC2.
3. Click on Running instances.
4. Place a checkmark next to the instance you wish to create an image from.
5. Click the Action drop-down and select Image -> Create Image.
6. Enter the Image name and click Create Image.
7. Navigate to Images → AMI's in the menu on the left side of the page to see your new image.

Creating your instances with the baseline image

Creating the worker VMs from your image

1. Log into AWS, and navigate to the AWS Management Console.
2. Click EC2.
3. Click Images → AMIs.
4. Check the box next to the image you created in the previous step, and click Launch.
5. On the Choose Instance Type tab, select your VM size, and click Next: Configure Instance Details.
6. On the Configure Instance tab, set the following:
   - Number of instances: 4
   - Purchasing option: Leave unchecked.
   - Network: Default VPC.
   - Subnet: Choose the region you are 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
7. Click Next: Add Storage.

8. On the Add Storage tab, set the following:
    - Size: 50GB
    - Volume Type: (We chose gp2.)
    - Delete on Termination: Unchecked.
    - Encryption: Not Encrypted.

    - Size: 600GB
    - Volume Type: (We chose io1 with 2,000 IOPS.).
    - Delete on Termination: Unchecked.
    - Encryption: Not Encrypted.

    - Size: 100GB
    - Volume Type: (We chose io1 with 2,000 IOPS.).
    - Delete on Termination: Unchecked
    - Encryption: Not Encrypted.
9. Click Next: Add Tags
10. On the Add Tags tab, set the following:
    - ProjectName: Gyasi
    - Study: 2
11. Click Next: Configure Security Group
12. On the Configure Security Group tab, add your security group.
13. Click Review and Launch.
14. On the Review Tab, click Launch.
15. Choose the appropriate option for the key pair, and click Launch Instances.

Configuring and starting the cluster

1. To set the hostname on the master and each of the worker Instances, edit /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 SSH into each instance without a password.
4. On the worker instances, create an XFS file system 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.
14. 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
```

15. Navigate to your home directory and download HiBench:

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

16. Install HiBench for Apache Spark 3.0:

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

17. 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. The benchmark is started 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. Results are automatically saved in the ~/results directory of the manager node.

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

5. 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
```

Determining CPU vulnerability mitigation

The information below shows the Intel processor mitigation settings on the AWS instances.

m4.16xlarge

```
CVE-2017-5753: OK (Mitigation: usercopy/swapgs barriers and __user pointer sanitization)
CVE-2017-5715: OK (Full retpoline is mitigating the vulnerability)
CVE-2017-5754: OK (Mitigation: PTI)
CVE-2018-3640: VULN (an up-to-date CPU microcode is needed to mitigate this vulnerability)
CVE-2018-3639: VULN (Your CPU doesn't support SSBD)
CVE-2018-3615: OK (your CPU vendor reported your CPU model as not vulnerable)
CVE-2018-3620: OK (Mitigation: PTE Inversion)
CVE-2018-3646: OK (this system is not running a hypervisor)
CVE-2018-12126: VULN (Your kernel supports mitigation, but your CPU microcode also needs to be
updated to mitigate the vulnerability)
CVE-2018-12130: VULN (Your kernel supports mitigation, but your CPU microcode also needs to be
updated to mitigate the vulnerability)
CVE-2018-12127: VULN (Your kernel supports mitigation, but your CPU microcode also needs to be
updated to mitigate the vulnerability)
CVE-2019-11091: VULN (Your kernel supports mitigation, but your CPU microcode also needs to be
updated to mitigate the vulnerability)
CVE-2019-11135: VULN (Vulnerable: Clear CPU buffers attempted, no microcode; SMT Host state unknown)
CVE-2018-12207: OK (this system is not running a hypervisor)
```

m5n.16xlarge

```
CVE-2017-5753: OK (Mitigation: usercopy/swapgs barriers and __user pointer sanitization)
CVE-2017-5715: OK (Full retpoline is mitigating the vulnerability)
CVE-2017-5754: OK (Mitigation: PTI)
CVE-2018-3640: VULN (an up-to-date CPU microcode is needed to mitigate this vulnerability)
CVE-2018-3639: VULN (Your CPU doesn't support SSBD)
CVE-2018-3615: OK (your CPU vendor reported your CPU model as not vulnerable)
CVE-2018-3620: OK (Mitigation: PTE Inversion)
CVE-2018-3646: OK (this system is not running a hypervisor)
CVE-2018-12126: VULN (Your kernel supports mitigation, but your CPU microcode also needs to be
updated to mitigate the vulnerability)
CVE-2018-12130: VULN (Your kernel supports mitigation, but your CPU microcode also needs to be
updated to mitigate the vulnerability)
CVE-2018-12127: VULN (Your kernel supports mitigation, but your CPU microcode also needs to be
updated to mitigate the vulnerability)
CVE-2019-11091: VULN (Your kernel supports mitigation, but your CPU microcode also needs to be
updated to mitigate the vulnerability)
CVE-2019-11135: OK (your CPU vendor reported your CPU model as not vulnerable)
CVE-2018-12207: OK (this system is not running a hypervisor)
```

Read the report at http://facts.pt/3Kjn66x ▶

This project was commissioned by Intel.