The science behind the report:

Servers powered by Intel Xeon Platinum 8160 processors yielded better performance on a mixed cloud workload than those powered by AMD EPYC 7601 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 "Servers powered by Intel Xeon Platinum 8160 processors yielded better performance on a mixed cloud workload than those powered by AMD EPYC 7601 processors."

We concluded our hands-on testing on January 11, 2019. 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 12, 2018 or earlier. Unavoidably, these configurations may not represent the latest versions available when this report appears.

Our results

We tested the simultaneous workload three times for each environment. This allowed us to ensure that results were repeatable and consistent. Once we had three runs from each environment, we needed a way to find the median run from each set so we could compare results. We chose to take the geometric mean of each workload result in order to create an aggregate score for each run. When calculating the cube root, we inverted the Apache® Spark™ score because it decreases with improved performance. For example, we calculated the geometric mean of Intel Run 1 by multiplying 1,537 with 245,554, dividing that result by 318, and finally taking the cube root. Our results and geometric mean calculations are in the table below.

	Total Webserver Transactions/second	Apache Cassandra® Operations/second	Apache Spark Time to complete (seconds)	Geometric mean
Intel® Xeon® Platinum 8160 processor-based servers				
Run 1	1,537	245,554	318	105.9
Run 2	1,511	233,921	321	103.3
Run 3	1,530	240,392	302	106.8
AMD EPYC™ 7601 processor-based servers				
Run 1	1,388	190,612	319	94.0
Run 2	1,318	194,819	329	92.1
Run 3	1,398	190,578	336	92.6

Notes on pricing

For our comparison, we chose servers using processors that were roughly comparable in terms of price. The follow table summarizes the pricing research we conducted on February 28, 2019. All prices are in USD.

Processor	Price source	Pricing
AMD EPYC 7601 processor	AMD website https://www.amd.com/en/shop/us/Server%20Processors?keyword=7601&sort_by=vision_date	"from $4,408.99"
Intel Xeon Platinum 8160 processor	Intel website https://www.intel.com/content/www/us/en/products/processors/xeon/scalable/platinum-processors/platinum-8160.html	"$from $4,825"

System configuration information

The table below presents detailed information on the systems we tested.

Server configuration information	Intel Server System R2224WFTZS	Dell EMC™ PowerEdge™ R7425
BIOS name and version	SE5C620.86B.00.01.0014	Dell 1.4.3
Non-default BIOS settings	Virtualization enabled	None
Operating system name and version/build number	Ubuntu 18.04 4.15.0-39-generic	Ubuntu 18.04 4.15.0-39-generic
Date of last OS updates/patches applied	11/12/2018	11/12/2018
Power management policy	Balanced	Balanced
Processor		
Number of processors	2	2
Vendor and model	Intel Xeon Platinum 8160	AMD EPYC 7601
Core count (per processor)	24	32
Core frequency (GHz)	2.10	2.20
Stepping	4	2
Memory module(s)		
Total memory in system (GB)	384	512
Number of memory modules	12	16
Vendor and model	Hynix® HMA84GR7AFR4N-VK	Hynix HMA84GR7AFR4N-VK
Size (GB)	32	32
Type	PC4-2666V	PC4-2666V
Speed (MHz)	2,666	2,666
Speed running in the server (MHz)	2,666	2,666
SATA storage		
Number of drives	1	1
Drive vendor and model	Intel SSDSCKGW180A4	Intel SSDSCKGW180A4
Drive size (GB)	180	180
Drive information (speed, interface, type)	M.2 SATA SSD	M.2 SATA SSD

Server configuration information	Intel Server System R2224WFTZS	Dell EMC™ PowerEdge™ R7425
NVMe storage		
Number of drives	4	4
Drive vendor and model	Intel P4600	Intel P4600
Drive size (TB)	1.6	1.6
Drive information (speed, interface, type)	NVMe SSD	NVMe SSD
Network adapter		
Vendor and model	Intel XL710	Intel XL710
Number and type of ports	2 x 40GbE SFP+	2 x 40GbE SFP+
Driver version	i40e-2.7.12	i40e-2.7.12
Cooling fans		
Vendor and model	Intel FR2UFAN60HSW	Dell N5T36
Number of cooling fans	6	6
Power supplies		
Vendor and model	SoluM PSSF132202A	Dell 0CMPGM
Number of power supplies	2	2
Wattage of each (W)	750	1100

The table below provides detailed configuration information for the network switches we used.

Network switch configuration information	Dell EMC Networking S4048-ON
Firmware revision	9.9 (0.0P9)
Number and type of ports	48 x SFP+ 10GbE, 6 x QSFP 40GbE
Number and type of ports used in test	2 x SFP+ 10GbE, 6 x QSFP 40GbE
Non-default settings used	N/A

How we tested

We ran all three workloads simultaneously to simulate a cloud environment hosting multiple types of applications with varying sizes of VMs.

The table below shows the resource allocation we used for testing.

	Small VMs (Webserver workload)	Medium-sized VMs (Cassandra workload)	Large VMs (Spark workload)
Number of virtual CPUs	2	4	8
Storage (GB)	20	40	60
Memory (GB)	8	16	32
VMs per server (Intel Xeon Platinum 8160 processor-based servers)	6	9	3
VMs per server (AMD EPYC 7601 processor-based servers)	8	12	4

Configuring the servers under test

We left all BIOS settings at default except that we enabled virtualization on the Intel servers. Each server had a 1Gb connection to an infrastructure network and a 40Gb connection to a testing network. We installed Ubuntu 18.04 with default packages onto the M.2 SSD in each server, leaving the four NVMe SSDs for VM volumes.

Installing Ubuntu

1. Boot the server to the Ubuntu 18.04 installation media.
2. Select English as the preferred language, and press Enter.
3. Select English (US) as the keyboard layout, and press Enter.
4. Select Install Ubuntu, and press Enter.
5. Ensure the 1Gb adapter has DHCP and the 40Gb adapter has no address, and press Enter.
6. Press Enter to skip the proxy address page.
7. Press Enter to accept the default mirror address.
8. Select Use an Entire Disk, and press Enter.
9. Select the M.2 SSD, and press Enter.
10. Select Done, and press Enter.
11. Select Continue, and press Enter.
12. Enter a name, hostname, username, and password. Select Done, and press Enter.
13. Select Done, and press Enter.
14. When the installation completes, remove the installation media, and reboot the server.

Updating and configuring the operating system

1. Log into the operating system as the user created in the previous section.
2. Disable the system firewall by running the following command:

```
sudo ufw disable
```

3. Update the operating system and reboot by running the following commands:

```
sudo apt -y upgrade
sudo apt -y update
sudo reboot
```

4. Configure password-less SSH:
   a. On one of the hosts, configure SSH by running the following commands:

```
rm -rf ~/.ssh
mkdir -p ~/.ssh
chmod 700 ~/.ssh
cd ~/.ssh
ssh-keygen -t rsa -q
cp id_rsa.pub authorized_keys
echo "StrictHostkeyChecking=no" > config
echo > known_hosts
cp -rp ~/.ssh ~/clean_ssh
```

   b. Copy keys to root user:

```
sudo su
rm -rf /root/.ssh
cp -rp ../clean_ssh /root/.ssh
chown -R root:root /root/.ssh
```

   c. Copy to other hosts:

```
ssh <remotehost> 'mkdir ~/.ssh'
scp -rp ~/.clean_ssh/* remotehost:~/.ssh
```

   d. SSH to each host, and copy keys to root user:

```
ssh remotehost
cd ~
sudo su
rm -rf /root/.ssh
cp -rp .ssh /root/.ssh
chown -R root:root /root/.ssh
exit
exit
```

Configuring the NVMe SSDs

1. Create a physical volume using the four NVMe SSDs by running the following command:

```
sudo pvcreate /dev/nvme0n1 /dev/nvme1n1 /dev/nvme2n1 /dev/nvme3n1
```

2. Create a volume group:

```
sudo vgcreate vgdata /dev/nvme0n1 /dev/nvme1n1 /dev/nvme2n1 /dev/nvme3n1
```

Installing the hypervisor

1. Install the required packages by running the following command:

```
sudo apt -y install qemu-kvm libvirt-bin
```

2. Edit the file /etc/netplan/50-cloud-init.yaml to reflect the following (replace 192.168.0.21/24 with the desired host IP address and replace adapter names if necessary):

```
network:
    version: 2
    renderer: networkd
    ethernets:
        eno1:
            addresses: []
            dhcp4: true
            optional: true
        eno2:
            addresses: []
            dhcp4: true
            optional: true
        eno3:
            addresses: []
            dhcp4: true
        eno4:
            addresses: []
```

```
            dhcp4: true
            optional: true
        enp225s0f0:
            addresses: []
            dhcp4: true
            optional: true
        enp225s0f1:
            addresses: []
            dhcp4: true
    bridges:
        br0:
            dhcp4: false
            addresses: [192.168.0.21/24]
            interfaces:
              - enp225s0f1
```

3. Apply the networking configuration by running the following command:

```
sudo netplan apply
```

4. Create a file name br0.xml and type the following to configure the bridge network:

```
<network>
  <name>br0</name>
  <forward mode='bridge'/>
  <bridge name='br0'/>
</network>
```

5. Define and start the bridge network with the following commands:

```
sudo virsh net-define ~/br0.xml
sudo virsh net-start br0
sudo virsh net-autostart br0
```

6. Download the Ubuntu 18.04 ISO by running the following command:

```
wget
http://www.cdimage.ubuntu.com/ubuntu/releases/18.04/release/ubuntu-18.04.1-server-amd64.iso
```

Creating the VMs

We used a separate server with Ubuntu 16.04 to host Virtual Machine Manager (VMM). We updated the operating system, disabled the firewall, and configured password-less SSH.

Installing Virtual Machine Manager and connecting the hosts

1. Install Ubuntu Desktop by running the following commands:

```
sudo apt -y install ubuntu-desktop
sudo reboot
```

2. Install VMM by running the following command:

```
sudo apt -y install virt-manager
```

3. Open VMM, and click File→Add Connection.
4. Select QEMU/KVM as the hypervisor.
5. Check the Connect to remote host checkbox.
6. Select SSH as the connection method.
7. Enter the host username and hostname.
8. Check the Autoconnect checkbox.
9. Click Connect.

Creating the storage pool for the installation media

1. Right-click the desired host, and click Details.
2. Click the Storage tab.
3. Click the plus sign for add pool.
4. Enter a name for the pool, and click Forward.
5. Enter the file location of the Ubuntu 18.04 ISO, and click Finish.

Creating the storage pool on the NVMe SSDs for VM volumes

1. Right-click the desired host, and click Details.
2. Click the Storage tab.
3. Click the plus sign for add pool.
4. Enter a name for the pool, select logical: LVM Volume Group, and click Forward.
5. Enter `/dev/vgdata` for the Target Path, and click Finish.

Creating the VM

1. Select the desired host, and click New VM.
2. Click Forward.
3. Click Browse.
4. Select the installation media storage pool in the left pane, then select the Ubuntu 18.04 ISO in the right pane, and click Choose Volume.
5. Click Forward.
6. Enter the memory amount and number of CPUs for the VM. (We used the memory and CPU counts in the table on page 11.) Click Forward.
7. Select the radio button for Select or create custom storage, and click Manage.
8. Select the NVMe storage pool in the left pane, and click the Create new volume button.
9. Enter a name for the volume and a Max Capacity to match the VM storage required. Click Finish.
10. Click Choose Volume.
11. Click Forward.
12. Enter a name for the VM, select Virtual network 'br0': Bridge network for the network selection, and click Finish.
13. Right-click the VM, and click Open.
14. Click Show virtual hardware details.
15. In the left pane, select IDE Disk 1. Click Advanced options, and select VirtIO for the Disk bus. Click Apply.
16. In the left pane, select NIC. Select virtio for the Device model, and click Apply.
17. In the left pane, select the Sound Controller, and click Remove.
18. In the left pane, select Display, and select Spice server for Type, All interfaces for Address, and en-us for Keymap. Click Apply.
19. Power on the VM, and follow the steps in the OS configuration section to install Ubuntu 18.04, update the OS, disable the firewall, and configure password-less SSH.

Installing the webserver workload

We created six webserver VMs on each server in the Intel group, and eight webserver VMs on each server in the AMD group. We installed a LAMP stack and WordPress on each VM and configured the default WordPress® test page as a target website. Our client VM existed on a separate server and used Apache JMeter™ to send webpage requests to our webserver VMs.

Installing WordPress on the target VMs

1. Install Apache, MySQL, and PHP by running the following command:

```
sudo apt -y install apache2 mysql-server php libapache2-mod-php php-mysql
```

2. Edit /etc/apache2/mods-enabled/dir.conf to reflect the following:

```
<IfModule mod_dir.c)
    DirectoryIndex index.php index.html index.cgi index.pl index.xhtml index.htm
</IfModule>
```

3. Restart Apache by running the following command:

```
sudo systemctl restart apache2
```

4. Enter the MySQL command line by running the following command:

```
sudo mysql -u root
```

5. Create the database for WordPress and grant permissions by running the following commands:

```
mysql> CREATE DATABASE wordpress DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;
mysql> GRANT ALL ON wordpress.* TO 'wordpressuser'@'localhost' IDENTIFIED BY 'Password1';
mysql> FLUSH PRIVILEGES;
mysql> EXIT;
```

6. Install additional PHP software by running the following command:

```
sudo apt -y install php-curl php-gd php-mbstring php-xml php-xmlrpc php-soap php-intl php-zip
```

7. Restart Apache by running the following command:

```
sudo systemctl restart apache2
```

8. Copy the default site configuration file by running the following commands:

```
cd /etc/apache2/sites-available
cp 000-default.conf wordpress.conf
```

9. Edit /etc/apache2/sites-available/000-default.conf to reflect the following:

```
DocumentRoot /var/www/wordpress
```

10. Edit /etc/apache2/sites-available/wordpress.conf to reflect the following:

```
<Directory /var/www/wordpress/>
    AllowOverride All
</Directory>
```

11. Apply the changes and restart Apache by running the following commands:

```
a2enmod rewrite
systemctl restart apache2
```

12. Install WordPress by running the following commands:

```
cd ~
wget https://wordpress.org/latest.tar.gz
tar xzvf latest.tar.gz
touch ~/wordpress/.htaccess
cp ~/wordpress/wp-config-sample.php ~/wordpress/wp-config.php
mkdir ~/wordpress/wp-content/upgrade
cp -a ~/wordpress/. /var/www/wordpress

sudo chown -R www-data:www-data /var/www/wordpress/
sudo find /var/www/wordpress/ -type d -exec chmod 750 {} \;
sudo find /var/www/wordpress/ -type f -exec chmod 640 {} \;
```

13. Edit /var/www/wordpress/wp-config.php to reflect the following:

```
define('AUTH_KEY',          'Password1');
define('SECURE_AUTH_KEY',   'Password1');
define('LOGGED_IN_KEY',     'Password1');
define('NONCE_KEY',         'Password1');
define('AUTH_SALT',         'Password1');
define('SECURE_AUTH_SALT',  'Password1');
define('LOGGED_IN_SALT',    'Password1');
define('NONCE_SALT',        'Password1');

define('DB_NAME', 'wordpress');

/** MySQL database username */
define('DB_USER', 'wordpressuser');

/** MySQL database password */
define('DB_PASSWORD', 'Password1');

define('FS_METHOD', 'direct');
```

14. Open a browser on the client VM and navigate to the webserver IP address.
15. Log in with the admin credentials and follow the wizard to complete the WordPress installation.
16. After cloning the VM, correct the mismatched IP address in MySQL by running the following commands:

```
sudo mysql -u root

mysql> use wordpress;
mysql> UPDATE wp_options SET option_value = "http://<NEW IP ADDRESS>" WHERE option_value =
'http://<OLD IP ADDRESS>';
```

Installing the workload driver on the client VMs

1. Download and install Apache Jmeter and the Plugin Manger by running the following commands:

```
wget http://ftp.naz.com/apache//jmeter/binaries/apache-jmeter-5.0.tgz

tar -zxf apache-jmeter-5.0.tgz

wget https://jmeter-plugins.org/get/
mv plugins-manager.jar apache-jmeter-5.0/lib/ext/
```

2. Install Java by running the following commands:

```
sudo add-apt-repository ppa:webupd8team/java
sudo apt update
sudo apt -y install oracle-java8-installer
```

3. Create a file in the home directory with a list of the webserver VM IP addresses, for example:

```
192.168.0.11
192.168.0.12
.
.
.
192.168.0.36
```

4. Open the console of the client VM and run Apache JMeter.
5. Click Options→Plugin Manager.
6. Check the checkbox for Custom Thread Groups and click Apply Changes and Restart JMeter.
7. Click the New button to create a new test plan.
8. Right-click the test plan and click Add→Threads (Users)→jp@gc Ultimate Thread Group.
9. Enter the following values for the thread group:
   a. Start Threads Count: `100`
   b. Initial Delay, sec: `0`
   c. Startup Time, sec: `0`
   d. Hold Load For, sec: `600`
   e. Shutdown Time: `0`
10. Right-click the thread group and click Add→Config Element→CSV Data Set Config.
11. Click CSV Data Set Config and enter the location of the IP address list created in step 3.
12. Enter a variable such as hostname for the Variable Names (comma-delimited) field.
13. Right-click the thread group and click Add→Config Element→HTTP Request Defaults.
14. Click HTTP Request Defaults and enter `${hostname}` in the Server Name or IP field.
15. Right-click the thread group and click Add→Sampler→HTTP Request.
16. Click HTTP Request and enter `/` for the Path field.
17. Save the test plan with a name such as webserver.jmx and close Apache JMeter.

Installing the Apache Cassandra workload

We created nine Cassandra VMs on each of the three Intel Xeon Platinum 8160 processor-based servers and 12 Cassandra VMs on each of the three AMD EPYC 7601 processor-based servers. Because Cassandra is a distributed database, users have the option of putting all VMs in a single large cluster or splitting them into multiple smaller clusters. We used the first option. We used a bare-metal server with threaded cassandra-stress instances to drive the Cassandra workload.

Installing Cassandra on the target VMs

1. Install Cassandra by running the following commands:

```
sudo add-apt-repository ppa:webupd8team/java
sudo apt -y update
sudo apt -y install oracle-java8-installer

echo "deb http://www.apache.org/dist/cassandra/debian 311x main" | sudo tee -a /etc/apt/sources.
list.d/cassandra.sources.list

curl https://www.apache.org/dist/cassandra/KEYS | sudo apt-key add -

sudo apt -y update
sudo apt -y install cassandra cassandra-tools
```

2. Edit /etc/cassandra/cassandra.yaml to reflect the following:

```
seeds: "<IP address of VM1>,<IP address of VM2>,<etc>"
start_rpc: true
# rpc_address: localhost
rpc_interface: ens3
# listen_address: localhost
listen_interface: ens3
```

3. Apply the new Cassandra configuration by running the following commands:

```
sudo systemctl stop cassandra
sudo rm -rf /var/lib/cassandra/*
sudo systemctl start cassandra
```

Installing the workload driver on the client server

1. Install the Cassandra toolset by running the following commands:

```
sudo add-apt-repository ppa:webupd8team/java
sudo apt -y update
sudo apt -y install oracle-java8-installer

echo "deb http://www.apache.org/dist/cassandra/debian 311x main" | sudo tee -a /etc/apt/sources.
list.d/cassandra.sources.list

curl https://www.apache.org/dist/cassandra/KEYS | sudo apt-key add -

sudo apt -y update
sudo apt -y install cassandra cassandra-tools
```

2. Create a file named cluster.<Intel or AMD> with a comma-separated list of the Cassandra node IP addresses, for example:

```
192.168.0.101,192.168.0.102,192.168.0.103,<etc>
```

3. We used a script to run eight instances of cassandra-stress, spread evenly across NUMA nodes. We used the script below, which we named numarun.sh:

```
rm ~/casstest*

count=$1
cpus=$[<number of CPU cores>/$count]

echo count=$count
echo cpus=$cpus

for i in  $(seq 1 1 $count)
do
        numactl -C $[($i-1)*$cpus]-$[$i*$cpus-2] -l cassandra-stress write no-warmup duration=600s
-rate threads=1000 -schema 'replication(factor=3)' -node $(cat $2) > casstest$i.txt &
done
```

Installing the Apache Spark workload

We created three Spark DataNode VMs on each of the three Intel Xeon Platinum 8160 processor-based servers and three Spark DataNode VMs on each of the three AMD EPYC 7601 processor-based servers. Because Spark is a distributed service, users have the option of putting all VMs in a single large cluster or splitting them into multiple smaller clusters. We used the first option. We used virtual machines with the HiBench benchmark to drive the workload.

Installing Apache Hadoop® YARN on the target VMs

1. Install Hadoop by running the following commands:

```
sudo add-apt-repository ppa:webupd8team/java
sudo apt -y update
sudo apt -y install oracle-java8-installer

wget http://mirrors.advancedhosters.com/apache/hadoop/common/hadoop-2.8.5/hadoop-2.8.5.tar.gz

tar -zxf hadoop-2.8.5.tar.gz
mv hadoop-2.8.5 hadoop
```

2. Edit ~/.profile to add the following lines:

```
PATH=~/hadoop/bin:~/hadoop/sbin:~/spark/bin:~/spark/sbin:$PATH
export HADOOP_CONF_DIR=~/hadoop/etc/hadoop
export SPARK_HOME=~/spark
export LD_LIBRARY_PATH=~/hadoop/lib/native:$LD_LIBRARY_PATH
```

3. Set the changes to ~/.profile by running the following command:

```
source ~/.profile
```

4. Edit ~/hadoop/etc/hadoop-env.sh to reflect the following:

```
export JAVA_HOME=/usr/lib/jvm/java-8-oracle/jre
```

5. Edit ~/hadoop/etc/hadoop/core-site.xml to reflect the following:

```
<property>
<name>dfs.namenode.name.dir</name>
<value>~/hadoop/data/nameNode</value>
</property>

<property>
<name>dfs.datanode.data.dir</name>
<value>~/hadoop/data/dataNode</value>
</property>

<property>
<name>dfs.replication</name>
<value>3</value>
</property>
```

6. Edit ~/hadoop/etc/hadoop/hdfs-site.xml to reflect the following:

```
<property>
<name>dfs.namenode.name.dir</name>
<value>~/hadoop/data/nameNode</value>
</property>

<property>
<name>dfs.datanode.data.dir</name>
<value>~/hadoop/data/dataNode</value>
</property>

<property>
<name>dfs.replication</name>
<value>3</value>
</property>
```

7. Edit ~/hadoop/etc/hadoop/mapred-site.xml to reflect the following:

```
<property>
<name>mapreduce.framework.name</name>
<value>yarn</value>
</property>

<property>
<name>yarn.app.mapreduce.am.resource.mb</name>
<value>8192</value>
</property>

<property>
<name>mapreduce.map.memory.mb</name>
<value>4096</value>
</property>

<property>
<name>mapreduce.reduce.memory.mb</name>
<value>4096</value>
</property>
```

8. Edit ~/hadoop/etc/hadoop/yarn-site.xml to reflect the following:

```
<property>
<name>yarn.acl.enable</name>
<value>0</value>
</property>

<property>
<name>yarn.resourcemanager.hostname</name>
<value>namenode-<Intel or AMD></value>
</property>

<property>
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle</value>
</property>

<property>
<name>yarn.nodemanager.resource.memory-mb</name>
<value>24576</value>
</property>

<property>
<name>yarn.scheduler.maximum-allocation-mb</name>
<value>24576</value>
</property>

<property>
<name>yarn.scheduler.minimum-allocation-mb</name>
<value>2048</value>
</property>

<property>
<name>yarn.nodemanager.vmem-check-enabled</name>
<value>false</value>
</property>
```

9. Place the IP addresses of each Spark datanode in ~/hadoop/etc/hadoop/slaves, for example:

```
192.168.0.71
192.168.0.72
192.168.0.73
192.168.0.74
192.168.0.81
192.168.0.82
192.168.0.83
192.168.0.84
192.168.0.91
192.168.0.92
192.168.0.93
192.168.0.94
```

10. Format the HDFS filesystem by running the following command on the NameNode:

```
hdfs namenode -format
```

11. Start HDFS and Yarn by running the following commands on the NameNode:

```
start-dfs.sh
start-yarn.sh
```

Installing Spark on the target VMs

1. Install Spark by running the following commands:

```
sudo apt -y install scala
wget http://mirrors.advancedhosters.com/apache/spark/spark-2.3.2/spark-2.3.2-bin-hadoop2.7.tgz
tar -zxf spark-2.3.2-bin-hadoop2.7.tgz
mv spark-2.3.2-bin-hadoop2.7 spark
```

2. Copy the Spark configuration file template to create the Spark configuration file by running the following command:

```
mv $SPARK_HOME/conf/spark-defaults.conf.template $SPARK_HOME/conf/spark-defaults.conf
```

3. Edit ~/spark/conf/spark-defaults.conf to reflect the following:

```
spark.master    yarn
spark.driver.memory    8192m
spark.yarn.am.memory    8192m
spark.executor.memory    8192m
```

4. Copy the Hadoop slaves file to the Spark configuration directory:

```
cp ~/hadoop/etc/hadoop/slaves ~/spark/conf/slaves
```

5. Start the Spark master and slave services by running the following commands on the namenode:

```
start-master.sh
start-slaves.sh
```

Installing HiBench on the client VMs

1. Install Hadoop prerequisites by running the following commands:

```
sudo add-apt-repository ppa:webupd8team/java
sudo apt -y update
sudo apt -y install oracle-java8-installer

wget http://mirrors.advancedhosters.com/apache/hadoop/common/hadoop-2.8.5/hadoop-2.8.5.tar.gz

tar -zxf hadoop-2.8.5.tar.gz
mv hadoop-2.8.5 hadoop
```

2. Edit ~/.profile to add the following lines:

```
PATH=~/hadoop/bin:~/hadoop/sbin:$PATH
```

3. Set the changes to ~/.profile by running the following command:

```
source ~/.profile
```

4. Edit ~/hadoop/etc/hadoop-env.sh to reflect the following:

```
export JAVA_HOME=/usr/lib/jvm/java-8-oracle/jre
```

5. Edit ~/hadoop/etc/hadoop/core-site.xml to reflect the following:

```
<property>
<name>fs.default.name</name>
<value>namenode-<Intel or AMD></value>
</property>
```

6. Edit ~/hadoop/etc/hadoop/yarn-site.xml to reflect the following:

```
<property>
<name>yarn.resourcemanager.hostname</name>
<value>namenode-<Intel or AMD></value>
</property>
```

7. Install HiBench by running the following commands:

```
sudo apt -y install python
git clone https://github.com/intel-hadoop/HiBench
cd HiBench
mvn -Dspark=2.1 -Dscala=2.11 clean package
sudo cp conf/hadoop.conf.template hadoop.conf
```

8. Edit ~/HiBench/conf/hadoop.conf to reflect the following:

```
hibench.hadoop.home     ~/hadoop
hibench.hdfs.master     hdfs://<namenode IP>:9000/hibench
```

9. Copy the Spark configuration template by running the following command:

```
cp ~/HiBench/conf/spark.conf.template ~/HiBench/conf/spark.conf
```

10. Edit ~/HiBench/conf/spark.conf to reflect the following:

```
hibench.spark.home      ~/spark
hibench.spark.master    spark://<namenode IP>:7077
hibench.yarn.executor.num     24
hibench.yarn.executor.cores   4
spark.executor.memory  8g
spark.driver.memory    4g
```

11. Edit ~/HiBench/conf/workloads/ml/kmeans.conf to reflect the following:

```
hibench.kmeans.huge.num_of_clusters             5
hibench.kmeans.huge.dimensions                  50
hibench.kmeans.huge.num_of_samples              100000000
hibench.kmeans.huge.samples_per_inputfile       20000000
hibench.kmeans.huge.max_iteration               50
hibench.kmeans.huge.k                           50
hibench.kmeans.huge.convergedist                0.5
```

12. Edit ~/HiBench/conf/hibench.conf to reflect the following:

```
hibench.scale.profile                 huge
```

13. Prepare the cluster for the Kmeans workload by running the following command:

```
~/HiBench/bin/workloads/ml/kmeans/spark/prepare.sh
```

Running the simultaneous workload

We ran all three workloads simultaneously to simulate a cloud environment hosting multiple types of applications with varying sizes of VMs. We performed a reset procedure before every test run to ensure repeatable results. We cleaned the Cassandra database and ensured the cluster was balanced. The webserver workload did not require a database reset. The HiBench benchmark automatically performs a reset before starting the workload.

After cleaning the Cassandra database, we shut down all VMs, rebooted the hosts, and powered the VMs back on. We then ensured all services were running and waited five minutes to ensure all startup processes had finished.

We configured the HiBench benchmark to perform a workload that lasted at least five minutes. We started all workloads simultaneously and collected the webserver and Cassandra results from the period that Spark was running.

Cleaning the Cassandra database

1. Create a file named cassandra-vms.<Intel or AMD>.list with each Cassandra node IP address on a separate line.
2. Clean the Cassandra database by running the following commands and waiting for each to complete:

```
for host in  $(cat cassandra-vms.<Intel or AMD>.list); do ssh $host 'nodetool flush ; nodetool cleanup
; sync'; done

cqlsh --request-timeout=120 -e "drop keyspace keyspace1" <IP address of any Cassandra node>

for host in  $(cat cassandra-vms.<Intel or AMD>.list); do ssh $host 'nodetool flush ; sleep 10 ;
nodetool cleanup ; nodetool clearsnapshot ; nodetool flush ; nodetool cleanup ; sync'; done
```

3. Clean the Cassandra lib directory by running the following commands:

```
sudo systemctl stop cassandra
sudo rm -rf /var/lib/cassandra/*
```

Running the simultaneous workload

1. Shut down all target VMs.
2. Reboot each KVM host.
3. Power on all target VMs.
4. Wait five minutes or until CPU usage is idle on all target VMs.

5. Perform the following actions, ensuring that each benchmark begins before starting the next one.
   a. From the webserver client, start the workload by running the following command:

```
./apache-jmeter-5.0/bin/jmeter -n -f -t ./apache-jmeter-5.0/webserver.jmx
```

   b. From the Cassandra client, start the workload by running the following command:

```
numarun.sh 8 cluster.<Intel or AMD>
```

   c. From the HiBench client, start the workload by running the following command:

```
~/HiBench/bin/workloads/ml/kmeans/spark/run.sh
```

Read the report at http://facts.pt/bwnqd4l ▶

This project was commissioned by Intel.