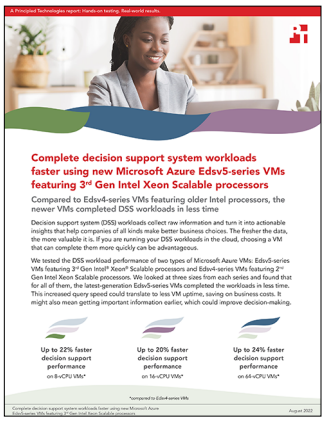

The science behind the report:

Complete decision support system workloads faster using new Microsoft Azure Edsv5-series VMs featuring 3rd Gen Intel Xeon Scalable 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 Complete decision support system workloads faster using new Microsoft Azure Edsv5-series VMs featuring 3rd Gen Intel Xeon Scalable processors.

We concluded our hands-on testing on July 10, 2022. 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 July 8, 2022 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: Results of our testing.

10-node VM cluster	Number of executors	Executor cores	Executor memory (GB)	Query time (hrs)	Edsv5-series advantage (percent faster)
E8ds_v4	10	7	54	2.075	N/A
E8ds_v5	10	7	54	1.688	22%
E16ds_v4	10	15	108	1.233	N/A
E16ds_v5	10	15	108	1.027	20%
E64ds_v4	30	10	50	0.872	N/A
E64ds_v5	30	10	50	0.700	24%

System configuration information

Table 2: Detailed information on the Microsoft Azure Edsv4-series VMs we tested.

System configuration information	10x E8ds_v4	10x E16ds_v4	10x E64ds_v4
Tested by	Principled Technologies	Principled Technologies	Principled Technologies
Test date	7/10/2022	7/9/2022	7/8/2022
CSP / Region	Azure / East US	Azure / East US	Azure / East US
Workload & version	Spark v3.2.0 TPC-DS-like	Spark v3.2.0 TPC-DS-like	Spark v3.2.0 TPC-DS-like
WL specific parameters	1000 scale, 10 executors, 7 executor cores, 54g executor memory	1000 scale, 10 executors, 15 executor cores, 108g executor memory	1000 scale, 30 executors, 10 executor cores, 50g executor memory
Iterations and result choice	3 runs, median	3 runs, median	3 runs, median
Server platform	E8ds_v4	E16ds_v4	E64ds_v4
BIOS name and version	Hyper-V UEFI Release v4.1 05/09/2022	Hyper-V UEFI Release v4.1 05/09/2022	Hyper-V UEFI Release v4.1 05/09/2022
Operating system name and version/ build number	Ubuntu 20.04.4 LTS, kernel 5.13.0-1031-azure	Ubuntu 20.04.4 LTS, kernel 5.13.0-1031-azure	Ubuntu 20.04.4 LTS, kernel 5.13.0-1031-azure
Date of last OS updates/patches applied	03/01/2022	03/01/2022	03/01/2022
Processor			
Number of processors	1	1	2
Vendor and model	Intel Xeon Platinum 8272CL	Intel Xeon Platinum 8272CL	Intel Xeon Platinum 8272CL
Core count (per processor)	26	26	26
Core frequency (GHz)	2.6	2.6	2.6
Stepping	7	7	7
Hyper-threading	Yes	Yes	Yes
Turbo	Yes	Yes	Yes
Number of vCPU per VM	8	16	64
Memory module(s)			
Total memory in system (GB)	64	128	504
NVMe® memory present?	No	No	No
Total memory (GB)	64	128	504
General HW			
Storage: NW or Direct Att / Instance	NW	NW	NW
Network BW / Instance	12,500 Mbps	12,500 Mbps	30,000 Mbps
Storage BW / Instance	1,536 Mbps	3,072 Mbps	9,600 Mbps

System configuration information	10x E8ds_v4	10x E16ds_v4	10x E64ds_v4
Local storage			
OS			
Number of drives	1	1	1
Drive size (GB)	128	128	128
Drive information	Standard SSD LRS, E10, 500 IOPS	Standard SSD LRS, E10, 500 IOPS	Standard SSD LRS, E10, 500 IOPS
Data drive			
Number of drives	1	1	1
Drive size (GB)	1,024	1,024	1,024
Drive information	Premium SSD LRS, P30, 5,000 IOPS	Premium SSD LRS, P30, 5,000 IOPS	Premium SSD LRS, P30, 5,000 IOPS
Temporary drive			
Number of drives	1	1	1
Drive size (GB)	300	600	2,400
Network adapter			
Vendor and model	Microsoft Hyper-V Network Adapter	Microsoft Hyper-V Network Adapter	Microsoft Hyper-V Network Adapter
Number and type of ports	1 x 12,500 Mbps	1 x 12,500 Mbps	1 x 30,000 Mbps

Table 3: Detailed information on the Microsoft Azure Edsv5-series VMs we tested.

System configuration information	10x E8ds_v5	10x E16ds_v5	10x E64ds_v5
Tested by	Principled Technologies	Principled Technologies	Principled Technologies
Test date	7/10/2022	7/9/2022	7/8/2022
CSP / Region	Azure / East US	Azure / East US	Azure / East US
Workload & version	Spark v3.2.0 TPC-DS-like	Spark v3.2.0 TPC-DS-like	Spark v3.2.0 TPC-DS-like
WL specific parameters	1000 scale, 10 executors, 7 executor cores, 54g executor memory	1000 scale, 10 executors, 15 executor cores, 108g executor memory	1000 scale, 30 executors, 10 executor cores, 50g executor memory
Iterations and result choice	3 runs, median	3 runs, median	3 runs, median
Server platform	E8ds_v5	E16ds_v5	E64ds_v5
BIOS name and version	Hyper-V UEFI Release v4.1 05/09/2022	Hyper-V UEFI Release v4.1 05/09/2022	Hyper-V UEFI Release v4.1 05/09/2022
Operating system name and version/ build number	Ubuntu 20.04.4 LTS, kernel 5.13.0-1031-azure	Ubuntu 20.04.4 LTS, kernel 5.13.0-1031-azure	Ubuntu 20.04.4 LTS, kernel 5.13.0-1031-azure
Date of last OS updates/patches applied	03/01/2022	03/01/2022	03/01/2022

System configuration information	10x E8ds_v5	10x E16ds_v5	10x E64ds_v5
Processor			
Number of processors	1	1	2
Vendor and model	Intel Xeon Platinum 8370c	Intel Xeon Platinum 8370c	Intel Xeon Platinum 8370c
Core count (per processor)	32	32	32
Core frequency (GHz)	2.8	2.8	2.8
Stepping	6	6	6
Hyper-threading	Yes	Yes	Yes
Turbo	Yes	Yes	Yes
Number of vCPU per VM	8	16	64
Memory module(s)			
Total memory in system (GB)	64	128	512
NVMe memory present?	No	No	No
Total memory (GB)	64	128	512
General HW			
Storage: NW or Direct Att / Instance	NW	NW	NW
Network BW / Instance	12,500 Mbps	12,500 Mbps	30,000 Mbps
Storage BW / Instance	2,320 Mbps	4,800 Mbps	13,880 Mbps
Local storage			
OS			
Number of drives	1	1	1
Drive size (GB)	128	128	128
Drive information	Standard SSD LRS, E10, 500 IOPS	Standard SSD LRS, E10, 500 IOPS	Standard SSD LRS, E10, 500 IOPS
Data drive			
Number of drives	1	1	1
Drive size (GB)	1024	1024	1024
Drive information	Premium SSD LRS, P30, 5,000 IOPS	Premium SSD LRS, P30, 5,000 IOPS	Premium SSD LRS, P30, 5,000 IOPS
Temporary drive			
Number of drives	1	1	1
Drive size (GB)	300	600	2,400
Network adapter			
Vendor and model	Microsoft Hyper-V Network Adapter	Microsoft Hyper-V Network Adapter	Microsoft Hyper-V Network Adapter
Number and type of ports	1 x 12,500 Mbps	1 x 12,500 Mbps	1 x 30,000 Mbps

How we tested

Testing overview

We ran a TPC-DS-like test using Spark SQL to query our database, which resided on a Hadoop HDFS cluster. We set up our cluster with one manager node and 10 worker nodes and installed Hadoop to spread our data across the 10 worker nodes. We then installed Spark so that our workers could query the database with Spark SQL, using Apache Hive as the SQL translator and YARN as the resource manager for our cluster. We selected a 1TB dataset because it is the smallest scale factor defined for TPC-DS-like tests, which we felt was the best fit across all three VM sizes to push the CPU, as that dataset mostly fit into RAM at the smallest VM size and fully fit into RAM at the largest size. We tested the Edsv4- and Edsv5-series VMs, scaling from 8 vCPUs to 16 vCPUs, and then finally 64 vCPUs. For each VM type, we used a 1TB P30 premium SSD on each worker VM and the manager VM. We ran our tests three times for each configuration and collected the median run.

Creating the Ubuntu 20.04 baseline image

Creating the baseline image VM

1. Log into the Azure Portal, and navigate to the Virtual Machines service.
2. Click Create → Azure virtual machine.
3. On the Basics tab, set the following:
4. Choose your Subscription.
5. Choose your Resource group.
6. Name the Virtual Machine.
7. Choose your Region.
8. Leave the Availability options set to No infrastructure redundancy required.
9. For Image, choose Ubuntu Server 20.04 LTS - Gen2.
10. Leave Azure Spot VM set to No.
11. Select the VM size you wish to use. We used Standard B4ms.
12. For the Administrator account, create a new username and password. We used ubuntu.
13. Leave Public inbound ports set to Allow selected ports.
14. For Select inbound ports, choose SSH (22).
15. On the Disks tab, set the following:
16. For the OS disk type, choose Standard HDD.
17. Leave the default Encryption type.
18. Under Data disks for test, click Create and attach a new disk.
19. Name the disk.
20. Leave the Source type as None (empty disk).
21. Select 1024 GiB for the size. This should be a P30 Premium SSD.
22. Click OK.
23. Next to the disk just created, under Delete with VM, select the checkbox.
24. 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.
25. On the Management tab, set the following:
    a. Choose your Diagnostics storage account.
    b. Leave the rest set to their defaults.
26. On the Advanced tab, leave all defaults.
27. On the Tags tab, add any tags you wish to use.
28. On the Review + create tab, review your settings, and click Create.

Configuring Ubuntu 20.04 and installing Apache Hadoop and Spark

1. Log into your VM.
2. Set the hostname by editing `/etc/hostname`.
3. Modify your hosts file at `/etc/hosts` and add manager and worker host names and IP addresses.
4. Turn off and disable your firewall:

```
sudo ufw disable
sudo ufw stop
```

5. Update your OS:

```
sudo apt update
sudo apt upgrade
```

6. Install Java 8:

```
sudo apt install openjdk-8-jdk
```

7. Reboot.
8. Log in, and create a directory for the Hadoop setup and test staging:

```
mkdir /home/hadoop
```

9. Change directories to the one you just created:

```
cd /home/hadoop
```

10. Download Hadoop, Spark, and Hive:

```
wget https://dlcdn.apache.org/hadoop/common/hadoop-3.3.1/hadoop-3.3.1.tar.gz
wget https://archive.apache.org/dist/spark/spark-3.2.0/spark-3.2.0-bin-hadoop3.2.tgz
wget https://downloads.apache.org/hive/hive-3.1.2/apache-hive-3.1.2-bin.tar.gz
```

11. Create an XFS filesystem on the Hadoop HDFS drive:

```
sudo mkfs.xfs /dev/<DEVICE>
```

12. Mount the Hadoop HDFS drive:

```
sudo mount /dev/<DEVICE> /home/hadoop/hdfs
```

13. To enable the drive to automatically mount after a reboot, add the following to your fstab file:

```
echo "/dev/<DEVICE> /home/hadoop/hdfs defaults 0 0" | sudo tee -a /etc/fstab
```

14. Create default Hadoop directories:

```
mkdir -p /home/hadoop/hdfs/namenode
mkdir -p /home/hadoop/hdfs/datanode
mkdir -p /home/hadoop/hdfs/tmp
```

15. Extract the Hadoop, Spark, and Hive compressed files:

```
tar -xzf /home/hadoop/hadoop-3.3.1.tar.gz
tar -xzf /home/hadoop/spark-3.2.0-bin-hadoop3.2.tgz
tar -xzf /home/hadoop/apache-hive-3.1.2-bin.tar.gz
```

16. Set up the Hadoop environment by editing `~/.profile` according to the Scripts we used for testing section.
17. Load the Hadoop environment:

```
source ~/.profile
```

18. Edit `core-site.xml`, `hadoop-env.sh`, `hdfs-site.xml`, and `mapred-site.xml` in $HADOOP_HOME/etc/hadoop according to the Scripts we used for testing section.
19. Edit `spark-defaults.conf` according to the Scripts we used for testing section.
20. Uncomment the JAVA_HOME line in hadoop-env.sh, and edit to match the line below:

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

21. Shut down the VM.

Creating a snapshot of your baseline VM

1. In your Azure portal, navigate to the Snapshots service.
2. To open the Snapshot wizard, click Add.
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 the 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 VMs with the baseline 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 VM 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. 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. Power on the baseline image. This will be the manager VM.
2. Create and attach two more 1TB gp3 drives to the manager VM using the following steps:
    a. Click the Manager VM.
    b. Select the Disks tab.
    c. Click Create and attach a new disk.
    d. Give the disk a name.
    e. Leave the storage type as Premium SSD.
    f. Set the size to 1024 GiB.
    g. Click Save.
    h. Repeat steps a through g for the second disk.
3. Mount one of the drives to the HDFS tmp directory:

```
sudo mount /dev/<TEMP DRIVE> /home/hadoop/hdfs/tmp
```

4. Set the hostname on the manager and each of the worker VMs by editing `/etc/hostname`.
5. Add the FQDN, hostname, and IP address of each VM to the /etc/hosts file on the manager and worker VMs.
6. Verify that you can do passwordless SSH into each VM.
7. Run the `format_hdfs.sh` script in the Scripts we used for testing section to format the HDFS filesystem on the manager node and each of the worker nodes.
8. Run the `start_all.sh` script in the Scripts we used for testing section to start the Hadoop namenode, secondary namenode, YARN resource manager, Spark master on the manager node, and the Hadoop datanode and YARN nodemanager on each of the worker VMs.

Generating the 1TB dataset and creating the TPC-DS-like database

1. On the manager VM, create a directory for the dataset creation and the IBM db toolkit:

```
mkdir /home/hadoop/db
```

2. Create a directory for the dataset:

```
mkdir /home/hadoop/db/data
```

3. Create an XFS filesystem on the remaining data drive attached to the manager VM:

```
sudo mkfs.xfs /dev/<DATA DRIVE>
```

4. Mount the data drive to the data directory you created in the steps above:

```
sudo mount /dev/<DATA DRIVE> /home/hadoop/db/data
```

5. Download the TPC-DS toolkit: https://www.tpc.org/tpc_documents_current_versions/download_programs/tools-download-request5.asp?bm_type=TPC-DS&bm_vers=3.2.0&mode=CURRENT-ONLY
6. Place the file you just downloaded in `/home/hadoop/db`, and unzip it.
7. Navigate into the tools subdirectory of the toolkit directory you just unzipped:

```
cd DSGen-software-code-3.2.0rc1/tools
```

8. Install prerequisites:

```
sudo apt install -y gcc make
```

9. Create the dataset:

```
dsdgen -scale 1000 -dir /home/hadoop/db/data -parallel 4 -child 1 &
dsdgen -scale 1000 -dir /home/hadoop/db/data -parallel 4 -child 2 &
dsdgen -scale 1000 -dir /home/hadoop/db/data -parallel 4 -child 3 &
dsdgen -scale 1000 -dir /home/hadoop/db/data -parallel 4 -child 4 &
```

10. Clone the GitHub repository containing the IBM toolkit:

```
git clone https://github.com/IBM/spark-tpc-ds-performance-test.git
```

11. Change directories into the toolkit:

```
cd /home/hadoop/db/spark-tpc-ds-performance-test
```

12. Copy the necessary files into the working directory:

```
cp src/properties/log4j.properties work/
cp src/ddl/create_database.sql work/
cp src/ddl/create_tables.sql work/
cp src/queries/* work/
```

13. Create the following HDFS directory for Hive data:

```
hdfs dfs -mkdir /user/hadoop/warehouse
```

14. Change into the working directory:

```
cd work/
```

15. Edit `log4j.properties`, `create_database.sql`, and `create_tables.sql` according to the Scripts we used for testing section.
16. Go back up a directory to the toolkit home:

```
cd ../
```

17. Load the dataset data into the HDFS filesystem by running the `load_hdfs.sh` script in the Scripts we used for testing section.
18. To create the database, run the following command:

```
spark-sql \
--driver-memory 6g \
--driver-java-options -Dlog4j.configuration=file:///$PWD/work/log4j.properties \
--executor-cores 2 \
--executor-memory 32g \
--conf \ spark.executor.extraJavaOptions=Dlog4j.configuration=file:///$PWD/work//log4j.properties \
-f $PWD/work/create_database.sql
```

19. To create the tables, run the following command:

```
spark-sql \
--driver-memory 6g \
--driver-java-options -Dlog4j.configuration=file:///$PWD/work/log4j.properties \
--executor-cores 2 \
--executor-memory 6g \
--conf \ spark.executor.extraJavaOptions=Dlog4j.configuration=file:///$PWD/work//log4j.properties \
-f $PWD/work/create_tables.sql
```

Running the tests

8-vCPU VM

1. On the manager VM, navigate to the scripts directory.
2. Stop Hadoop, Spark, and YARN by running the `stop_all.sh` script in the Scripts we used for testing section.
3. Edit the YARN resource manager configuration by editing $HADOOP_HOME/etc/hadoop/yarn-site.xml. We used the following:
   - yarn.nodemanager.resource.memory-mb: 60000
   - yarn.scheduler.maximum-allocation-mb: 60000
   - yarn.nodemanager.resource.cpu-vcores: 8
   - yarn.scheduler.maximum-allocation-vcores: 8
4. Start Hadoop, Spark, and YARN by running the `start_all.sh` script in the Scripts we used for testing section.
5. Navigate to the testing toolkit:

```
cd /home/hadoop/db/spark-tpc-ds-performance-test
```

6. Run the run.sh script in the Scripts we used for testing section, and enter the appropriate configuration values. We used the following:
   - Date: <DATE>
   - Run: <RUN #>
   - Number of executors: 10
   - Executor cores: 7
   - Executor memory (k, m, g): 54g
7. Repeat steps 1 through 6 two more times for a total of three runs.

16-vCPU VM

1. On the manager VM, navigate to the scripts directory.
2. Stop Hadoop, Spark, and YARN by running the `stop_all.sh` script in the Scripts we used for testing section.
3. Edit the YARN resource manager configuration by editing $HADOOP_HOME/etc/hadoop/yarn-site.xml. We used the following:
   - yarn.nodemanager.resource.memory-mb: 126976
   - yarn.scheduler.maximum-allocation-mb: 124928
   - yarn.nodemanager.resource.cpu-vcores: 16
   - yarn.scheduler.maximum-allocation-vcores: 16
4. Start Hadoop, Spark, and YARN by running the `start_all.sh` script in the Scripts we used for testing section.
5. Navigate to the testing toolkit:

```
cd /home/hadoop/db/spark-tpc-ds-performance-test
```

6. Run the run.sh script in the Scripts we used for testing section, and enter the appropriate configuration values. We used the following:
   - Date: <DATE>
   - Run: <RUN #>
   - Number of executors: 10
   - Executor cores: 15
   - Executor memory (k, m, g): 108g
7. Repeat steps 1 through 6 two more times for a total of three runs.

64-vCPU VM

1. On the manager VM, navigate to the scripts directory.
2. Stop Hadoop, Spark, and YARN by running the `stop_all.sh` script we have in the Scripts we used for testing section.
3. Edit the YARN resource manager configuration by editing $HADOOP_HOME/etc/hadoop/yarn-site.xml. We used the following:
   - yarn.nodemanager.resource.memory-mb: 507904
   - yarn.scheduler.maximum-allocation-mb: 499712
   - yarn.nodemanager.resource.cpu-vcores: 64
   - yarn.scheduler.maximum-allocation-vcores: 64
4. Start Hadoop, Spark, and YARN by running the `start_all.sh` script we have in the Scripts we used for testing section.
5. Navigate to the testing toolkit:

```
cd /home/hadoop/db/spark-tpc-ds-performance-test
```

6. Run the run.sh script in the Scripts we used for testing section, and enter the appropriate configuration values. We used the following:
   - Date: <DATE>
   - Run: <RUN #>
   - Number of executors: 30
   - Executor cores: 10
   - Executor memory (k, m, g): 50g
7. Repeat steps 1 through 6 two more times for a total of three runs.

Scripts we used for testing

Configuration files

~/.profile

```
# ~/.profile: executed by the command interpreter for login shells.
# This file is not read by bash(1), if ~/.bash_profile or ~/.bash_login
# exists.
# see /usr/share/doc/bash/examples/startup-files for examples.
# the files are located in the bash-doc package.

# the default umask is set in /etc/profile; for setting the umask
# for ssh logins, install and configure the libpam-umask package.
#umask 022

# if running bash
if [ -n "$BASH_VERSION" ]; then
    # include .bashrc if it exists
    if [ -f "$HOME/.bashrc" ]; then
      . "$HOME/.bashrc"
    fi
fi

# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
fi

# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/.local/bin" ] ; then
    PATH="$HOME/.local/bin:$PATH"
fi

export SPARK_HOME=/home/hadoop/spark-3.2.0-bin-hadoop3.2

export HADOOP_HOME=/home/hadoop/hadoop-3.3.1
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
export HADOOP_INSTALL=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib/native"

export HIVE_HOME=/home/hadoop/apache-hive-3.1.2-bin

export YARN_HOME=$HADOOP_HOME

export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin:$HIVE_HOME/bin:$SPARK_HOME/bin
```

core-site.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
<name>fs.defaultFS</name>
<value>hdfs://[Manager IP Address]:9000</value>
</property>
<property>
<name>hadoop.tmp.dir</name>
<value>/home/hadoop/hdfs/tmp</value>
</property>
</configuration>
```

hdfs-site.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
<name>dfs.replication</name>
<value>3</value>
</property>
<property>
<name>dfs.name.dir</name>
<value>file:///home/hadoop/hdfs/namenode</value>
</property>
<property>
<name>dfs.data.dir</name>
<value>file:///home/hadoop/hdfs/datanode</value>
</property>
<property>
  <name>dfs.client.read.shortcircuit</name>
  <value>true</value>
</property>
<property>
  <name>dfs.domain.socket.path</name>
  <value>/home/hadoop/hdfs/socket</value>
</property>
</configuration>
```

mapred-site.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

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

spark-defaults.conf

```
#
# Licensed to the Apache Software Foundation (ASF) under one or more
# contributor license agreements.  See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The ASF licenses this file to You under the Apache License, Version 2.0
# (the "License"); you may not use this file except in compliance with
# the License.  You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#

# Default system properties included when running spark-submit.
# This is useful for setting default environmental settings.

# Example:
# spark.master                     spark://master:7077
# spark.eventLog.enabled           true
# spark.eventLog.dir               hdfs://namenode:8021/directory
# spark.serializer                 org.apache.spark.serializer.KryoSerializer
# spark.driver.memory              5g
# spark.executor.extraJavaOptions  -XX:+PrintGCDetails -Dkey=value -Dnumbers="one two three"

spark.master                 yarn
spark.eventLog.enabled               true
spark.eventLog.dir                   hdfs://[Manager IP Address]:9000/hadoop/logs
spark.serializer                     org.apache.spark.serializer.KryoSerializer
spark.sql.warehouse.dir              /user/hadoop/warehouse
```

workers

```
worker-1
worker-2
worker-3
worker-4
worker-5
worker-6
worker-7
worker-8
worker-9
worker-10
```

format_hdfs.sh

```
#!/bin/bash

rm -rf /home/hadoop/hdfs/tmp/*
sudo umount /dev/<TEMP DRIVE>
rm -rf /home/hadoop/hdfs/*
mkdir /home/hadoop/hdfs/namenode
mkdir /home/hadoop/hdfs/datanode
mkdir /home/hadoop/hdfs/tmp
sudo mount /dev/<TEMP DRIVE> /home/hadoop/hdfs/tmp
sudo chown -R ubuntu:ubuntu /home/hadoop/hdfs
/home/hadoop/hadoop-3.3.1/bin/hdfs namenode -format

for i in $(cat workers);
do
      echo "Formatting HDFS on ${i}"
      echo ""
      ssh ${i} 'rm -rf /home/hadoop/hdfs/*'
      ssh ${i} 'mkdir /home/hadoop/hdfs/namenode'
    ssh ${i} 'mkdir /home/hadoop/hdfs/datanode'
    ssh ${i} 'mkdir /home/hadoop/hdfs/tmp'
      ssh ${i} '/home/hadoop/hadoop-3.3.1/bin/hdfs namenode -format'
      echo ""
done
```

start_all.sh

```
#!/bin/bash

$HADOOP_HOME/bin/hdfs --daemon start namenode
$HADOOP_HOME/bin/hdfs --daemon start secondarynamenode
$HADOOP_HOME/bin/yarn --daemon start resourcemanager
$SPARK_HOME/sbin/start-master.sh

for i in $(cat workers);
do
      ssh ${i} '/home/hadoop/hadoop-3.3.1/bin/hdfs --daemon start datanode'
      ssh ${i} '/home/hadoop/hadoop-3.3.1/bin/yarn --daemon start nodemanager'
      ssh ${i} '/home/hadoop/spark-3.2.0-bin-hadoop3.2/sbin/start-worker.sh spark://[Manager IP or
hostname]:7077'
done
```

stop_all.sh

```
#!/bin/bash


for i in $(cat workers);
do
      ssh ${i} "/home/hadoop/hadoop-3.3.1/bin/hdfs --daemon stop datanode"
      ssh ${i} "/home/hadoop/hadoop-3.3.1/bin/yarn --daemon stop nodemanager"
      ssh ${i} "/home/hadoop/spark-3.2.0-bin-hadoop3.2/sbin/stop-worker.sh spark://[Manager IP or
hostname]:7077"
done

$SPARK_HOME/sbin/stop-master.sh
$HADOOP_HOME/bin/yarn --daemon stop resourcemanager
$HADOOP_HOME/bin/hdfs --daemon stop secondarynamenode
$HADOOP_HOME/bin/hdfs --daemon stop namenode
```

run_query.sh

```
#!/bin/bash


date=${1}
run=${2}
num_executors=${3}
exec_cores=${4}
exec_mem=${5}

results=/home/hadoop/results

#Cleanup previous runs
pkill nmon
rm *.nmon

for i in {1..10};
do
        ssh worker-$i 'pkill nmon'
      ssh worker-$i 'rm /home/hadoop/*.nmon'
done

#Make results directory
mkdir -p $results/$date/$run

#Start nmon on Manager node
nmon -f -t -s 5 -c 1656

#Start nmon on Worker nodes
for i in {1..10};
do
      ssh worker-$i 'nmon -f -t -s 5 -c 1656'
done

sleep 10

#Run queries
for i in {01..99};
do
      echo "query $i"
      spark-sql --driver-memory 4g --driver-java-options -Dlog4j.configuration=file:///$PWD/work/log4j.
properties --executor-memory ${exec_mem} --executor-cores ${exec_cores} --num-executors ${num_executors}
--conf spark.executor.extraJavaOptions=-Dlog4j.configuration=file:///${PWD}/work/log4j.properties --conf
spark.sql.crossJoin.enabled=true -database tpcds -f work/query${i}.sql > work/query${i}.res 2>&1
done

sleep 10

#Stop nmon on Manager node
pkill nmon
```

```
#Stop nmon on Worker nodes
for i in {1..10};
do
        ssh worker-$i 'pkill nmon'
done

#Copy nmon files, query results, and config details to results folder
mv ./*.nmon $results/$date/$run/

for i in {01..99};
do
      cp ./work/query$i.res $results/$date/$run/
done

for i in {1..10};
do
      scp worker-$i:/home/hadoop/*.nmon $results/$date/$run/
      ssh worker-$i 'rm /home/hadoop/*.nmon'
done

mkdir $results/$date/$run/hadoop_config
mkdir $results/$date/$run/spark_config
cp $HADOOP_HOME/etc/hadoop/* $results/$date/$run/hadoop_config
cp $SPARK_HOME/conf/* $results/$date/$run/spark_config
cp /home/hadoop/tpcds/spark-tpc-ds-performance-test/run_query.sh $results/$date/$run/

touch $results/$date/$run/config.txt
config=$results/$date/$run/config.txt
echo "Date: $date" >> $config
echo "Run: $run" >> $config
echo "Number of executors: $num_executors" >> $config
echo "Executor cores: $exec_cores" >> $config
echo "Executor memory: $exec_mem" >> $config
```

run.sh

```
#!/bin/bash

read -p 'Date: ' date
read -p 'Run number: ' run
read -p 'Number of executors: ' num_executors
read -p 'Executor cores: ' exec_cores
read -p 'Executor memory (k,m,g): ' exec_mem


nohup ./run_query.sh $date $run $num_executors $exec_cores $exec_mem &
```

Read the report at http://facts.pt/WmJJyE9 ▶

This project was commissioned by Intel.

Principled Technologies®

Facts matter.®