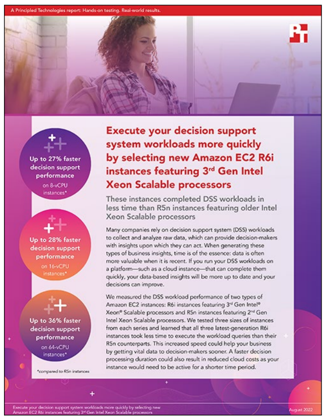

A Principled Technologies report: Hands-on testing. Real-world results.

Execute your decision support system workloads more quickly by selecting new Amazon EC2 R6i instances featuring 3rd Gen Intel Xeon Scalable processors

These instances completed DSS workloads in less time than R5n instances featuring older Intel Xeon Scalable processors

Many companies rely on decision support system (DSS) workloads to collect and analyze raw data, which can provide decision-makers with insights upon which they can act. When generating these types of business insights, time is of the essence: data is often more valuable when it is recent. If you run your DSS workloads on a platform—such as a cloud instance—that can complete them quickly, your data-based insights will be more up to date and your decisions can improve.

We measured the DSS workload performance of two types of Amazon EC2 instances: R6i instances featuring 3rd Gen Intel® Xeon® Scalable processors and R5n instances featuring 2nd Gen Intel Xeon Scalable processors. We tested three sizes of instances from each series and learned that all three latest-generation R6i instances took less time to execute the workload queries than their R5n counterparts. This increased speed could help your business by getting vital data to decision-makers sooner. A faster decision processing duration could also result in reduced cloud costs as your instance would need to be active for a shorter time period.

Up to 27% faster decision support performance on 8-vCPU instances*

Up to 28% faster decision support performance on 16-vCPU instances*

Up to 36% faster decision support performance on 64-vCPU instances*

*compared to R5n instances

Execute your decision support system workloads more quickly by selecting new Amazon EC2 R6i instances featuring 3rd Gen Intel Xeon Scalable processors

August 2022

The science behind the report:

Execute your decision support system workloads more quickly by selecting new Amazon EC2 R6i instances 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 Execute your decision support system workloads more quickly by selecting new Amazon EC2 R6i instances featuring 3rd Gen Intel Xeon Scalable processors.

We concluded our hands-on testing on July 11, 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 instance cluster	Number of executors	Executor cores	Executor memory (GB)	Query time (hrs)	R6i series advantage (percent faster)
r5n.2xlarge	10	7	54	2.072	N/A
r6i.2xlarge	10	7	54	1.625	27%
r5n.4xlarge	10	15	108	1.285	N/A
r6i.4lxarge	10	15	108	1.003	28%
r5n.16xlarge	30	10	50	0.919	N/A
r6i.16xlarge	30	10	50	0.674	36%

System configuration information

Table 2: Detailed information on the systems we tested.

System configuration information	r5n.2xlarge	r5n.4xlarge	r5n.16xlarge
Tested by	Principled Technologies	Principled Technologies	Principled Technologies
Test date	7/10/2022	7/9/2022	7/8/2022
CSP / Region	AWS / us-east-1	AWS / us-east-1	AWS / us-east-1
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	1,000 scale, 10 executors, 7 executor cores, 54g executor memory	1,000 scale, 10 executors, 15 executor cores, 108g executor memory	1,000 scale, 30 executors, 10 executor cores, 50g executor memory
Iterations and result choice	3 runs, median	3 runs, median	3 runs, median
Server platform	r5n.2xlarge	r5n.4xlarge	r5n.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	Ubuntu 20.04.4 LTS, kernel 5.13.0-1031-aws	Ubuntu 20.04.4 LTS, kernel 5.13.0-1031-aws	Ubuntu 20.04.4 LTS, kernel 5.13.0-1031-aws
Date of last OS updates/patches applied	04/20/2022	04/20/2022	04/20/2022
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)	24	24	24
Core frequency (GHz)	2.5	2.5	2.5
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	512
NVMe memory present?	No	No	No
Total memory (DDR+NVMe RAM)	64	128	512
General HW			
Storage: NW or Direct Att / Instance	NW	NW	NW
Network BW / Instance	Up to 25 Gbps	Up to 25 Gbps	75 Gbps
Storage BW / Instance	Up to 3.5 Gbps	3.5 Gbps	10 Gbps

System configuration information	r5n.2xlarge	r5n.4xlarge	r5n.16xlarge
Local storage			
OS			
Number of drives	1	1	1
Drive size (GB)	100	100	100
Drive information (speed, interface, type)	gp3, EBS, 125 throughput/3,000 IOPs	gp3, EBS, 125 throughput/3,000 IOPs	gp3, EBS, 125 throughput/3,000 IOPs
Data drive			
Number of drives	1	1	1
Drive size (GB)	1,024	1,024	1,024
Drive information (speed, interface, type)	gp3, EBS, 1,000 throughput/16,000 IOPs	gp3, EBS, 1,000 throughput/16,000 IOPs	gp3, EBS, 1,000 throughput/16,000 IOPs
Network adapter			
Vendor and model	Amazon Elastic Network Adapter	Amazon Elastic Network Adapter	Amazon Elastic Network Adapter
Number and type of ports	1 x 25 Gbps	1 x 25 Gbps	1 x 75 Gbps

Table 3: Detailed information on the systems we tested.

System configuration information	r6i.2xlarge	r6i.4xlarge	r6i.16xlarge
Tested by	Principled Technologies	Principled Technologies	Principled Technologies
Test date	7/10/2022	7/9/2022	7/8/2022
CSP / Region	AWS / us-east-1	AWS / us-east-1	AWS / us-east-1
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	1,000 scale, 10 executors, 7 executor cores, 54g executor memory	1,000 scale, 10 executors, 15 executor cores, 108g executor memory	1,000 scale, 30 executors, 10 executor cores, 50g executor memory
Iterations and result choice	3 runs, median	3 runs, median	3 runs, median
Server platform	r6i.2xlarge	r6i.4xlarge	r6i.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	Ubuntu 20.04.4 LTS, kernel 5.13.0-1031-aws	Ubuntu 20.04.4 LTS, kernel 5.13.0-1031-aws	Ubuntu 20.04.4 LTS, kernel 5.13.0-1031-aws
Date of last OS updates/patches applied	04/20/2022	04/20/2022	04/20/2022
Processor			
Number of processors	1	1	2
Vendor and model	Intel Xeon Platinum 8375c	Intel Xeon Platinum 8375c	Intel Xeon Platinum 8375c
Core count (per processor)	32	32	32
Core frequency (GHz)	2.9	2.9	2.9
Stepping	6	6	6
Hyper-Threading	Yes	Yes	Yes
Turbo	Yes	Yes	Yes
Number of vCPU per VM	8	16	64

System configuration information	r6i.2xlarge	r6i.4xlarge	r6i.16xlarge
Memory module(s)			
Total memory in system (GB)	64	128	512
NVMe memory present?	No	No	No
Total memory (DDR+NVMe RAM)	64	128	512
General HW			
Storage: NW or Direct Att / Instance	NW	NW	NW
Network BW / Instance	Up to 12.5 Gbps	Up to 12.5 Gbps	25 Gbps
Storage BW / Instance	Up to 10 Gbps	Up to 10 Gbps	20 Gbps
Local storage			
OS			
Number of drives	1	1	1
Drive size (GB)	100	100	100
Drive information (speed, interface, type)	gp3, EBS, 125 throughput/3,000 IOPs	gp3, EBS, 125 throughput/3,000 IOPs	gp3, EBS, 125 throughput/3,000 IOPs
Data drive			
Number of drives	1	1	1
Drive size (GB)	1,024	1,024	1,024
Drive information (speed, interface, type)	gp3, EBS, 1,000 throughput/16,000 IOPs	gp3, EBS, 1,000 throughput/16,000 IOPs	gp3, EBS, 1,000 throughput/16,000 IOPs
Network adapter			
Vendor and model	Amazon Elastic Network Adapter	Amazon Elastic Network Adapter	Amazon Elastic Network Adapter
Number and type of ports	1 x 12.5 Gbps	1 x 12.5 Gbps	1 x 25 Gbps

How we tested

Testing overview

To show the gen-over-gen improvement moving from a Cascade Lake instance to an Ice Lake instance in AWS, 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 1 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. We felt this was the best choice to push the CPU across all three instance sizes because it mostly fit into RAM at the smallest instance size and fully fit into RAM at the largest size. We tested the R5n and R6i instance types, scaling first from 8 vCPUs to 16 vCPUs, and then to 64 vCPUs. For each instance type, we used a 1TB P30 premium SSD on each worker instance and the manager instance. We ran our tests three times for each configuration and collected the median run.

Creating the Ubuntu 20.04 baseline image

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

Create the baseline image instance

1. Log into AWS and navigate to the AWS Management Console.
2. Click EC2.
3. Click Launch instance, then Launch instance in the drop-down menu to open the Launch Instance wizard.
4. Give your instance a name, and add any necessary tags.
5. Click Ubuntu, and select 20.04 from the Application and OS Images drop-down menu.
6. From the Instance type drop-down menu, select t3.medium as the instance type.
7. From the Key pair drop-down menu, select an existing key pair or create a new one.
8. In the Network settings drop-down menu, leave the network settings as defaults.
9. In the Configure storage drop-down menu, set the size to 100 GiB and the volume type to gp3.
10. Leave the number of instances at 1 in the Summary drop-down menu, and click Launch instance.
11. In the left-hand navigation pane, click Volumes under Elastic Block Store.
12. Click Create volume.
13. For the volume type, select gp3.
14. For the size, enter 1024.
15. Set the IOPs to 16,000 and the throughput to 1,000.
16. Set the Availability zone to match the Instance and add any necessary tags.
17. Click Create volume.
18. Click the volume you just created.
19. Click Actions → Attach volume.
20. Select the baseline instance and leave the Device name as default.
21. Click Attach volume.

Configuring Ubuntu 20.04 and installing Apache Hadoop and Spark

1. Log into your instance.
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. Add the following to your FSTAB file so the drive will automatically mount after a reboot:

```
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` to match what is in the Scripts we used for testing section.
17. Load the Hadoop environment:

```
source ~/.profile
```

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

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

21. Shut down the instance.

Creating an AMI of your baseline instance

1. Log into AWS and navigate to the AWS Management Console.
2. Click EC2.
3. Click the baseline instance.
4. Next to the instance from which you wish to create an image, place a checkmark.
5. Click the Action dropdown and select Image → Create Image.
6. Enter the Image name and click Create Image.
7. In the menu on the left side of the screen, navigate to Images → AMIs to see your new image.

Creating your instances with the baseline image

Creating the worker instances from your image

1. Log into AWS and navigate to the AWS Management Console.
2. Click EC2.
3. Click Images → AMIs.
4. Next to the image you created in the previous step, check the box and click Launch instance from AMI.
5. Name the instance Worker.
6. Select the instance type.
7. Select the Key pair used by the baseline instance.
8. Next to Select existing security group, select the radio button.
9. In the number of instances field, enter 10.
10. Click Launch instance.

Configuring and starting the cluster

1. Power on the baseline image which will serve as the manager instance.
2. Create and attach two more 1TB gp3 drives to the manager instance following the same steps in the baseline image creation.
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 Instances by editing `/etc/hostname`.
5. Add the FQDN, hostname, and IP address of each instance to the /etc/hosts file on the manager and worker instances.
6. Verify that you can do passwordless SSH into each instance.
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 node manager on each of the worker instances.

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

1. On the manager instance, 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 instance:

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

4. Mount the data drive to the data directory you created earlier:

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

5. Download the TPC-DS toolkit you need to generate the dataset from here: 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. Move 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 to the working directory:

```
cd work/
```

15. Edit `log4j.properties`, `create_database.sql`, and `create_tables.sql` to match what is in 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. Create the database:

```
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. Create the tables:

```
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 instance

1. On the manager instance, 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 instance

1. On the manager instance, 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: 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 instance

1. On the manager instance, 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: 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 https://facts.pt/Tx8rPBZ ▶

This project was commissioned by Intel.

Principled Technologies®

Facts matter.®

Principled Technologies is a registered trademark of Principled Technologies, Inc.
All other product names are the trademarks of their respective owners.

DISCLAIMER OF WARRANTIES; LIMITATION OF LIABILITY:
Principled Technologies, Inc. has made reasonable efforts to ensure the accuracy and validity of its testing, however, Principled Technologies, Inc. specifically disclaims any warranty, expressed or implied, relating to the test results and analysis, their accuracy, completeness or quality, including any implied warranty of fitness for any particular purpose. All persons or entities relying on the results of any testing do so at their own risk, and agree that Principled Technologies, Inc., its employees and its subcontractors shall have no liability whatsoever from any claim of loss or damage on account of any alleged error or defect in any testing procedure or result.

In no event shall Principled Technologies, Inc. be liable for indirect, special, incidental, or consequential damages in connection with its testing, even if advised of the possibility of such damages. In no event shall Principled Technologies, Inc.'s liability, including for direct damages, exceed the amounts paid in connection with Principled Technologies, Inc.'s testing. Customer's sole and exclusive remedies are as set forth herein.