BIG DATA TECHNOLOGY ON RED HAT ENTERPRISE LINUX: OPENJDK VS. ORACLE JDK

Red Hat® Enterprise Linux® and OpenJDK®: driving Hadoop® 2.0
OpenJDK
Oracle JDK
Red Hat Enterprise Linux
Red Hat Enterprise Linux
Delivered comparable Hadoop performance to Oracle® JDK in our hands-on tests
Certified for Hortonworks 2.0
redhat

Organizations that use Apache™ Hadoop to process very large amounts of data demand the best performance possible from their infrastructures. The Java Virtual Machine (JVM) you choose to pair with your operating system can affect how your Hadoop deployment performs, and can also create or alleviate management headaches for administrators.

In the Principled Technologies labs, we compared the performance of a Hadoop deployment running on Red Hat Enterprise Linux using both OpenJDK and Oracle JDK.[1] We found that the open-source OpenJDK performed comparably to Oracle JDK on all Hadoop performance tests we ran. OpenJDK is distributed alongside the operating system and requires no additional actions by administrators, making it a natural choice for organizations wishing to standardize on open source software deployments that include Red Hat Enterprise Linux and Hadoop.

[1] OpenJDK is a trademark of Oracle, Inc.

MARCH 2014
A PRINCIPLED TECHNOLOGIES TEST REPORT
Commissioned by Red Hat, Inc.

APACHE HADOOP PERFORMANCE AND THE JVM

Apache Hadoop is an open-source, highly scalable, and reliable distributed data storage and analytics engine used for storing and processing massive amounts of information. Hadoop is highly available and performs advanced business analytics on data to help organizations make decisions. Core Hadoop components are mostly written in Java, and run on top of the JVM. In addition, the components include *HDFS,* the Hadoop Distributed File System; *MapReduce*, a functional data processing framework for share-nothing distributed data algorithms; and a number of *Common* utilities to integrate HDFS and MapReduce and support workload and job processing. Because Hadoop performance is closely tied to the performance of the underlying JVM, optimizing the JVM for lower latency and better execution speed is crucial. The choice of JVM can greatly affect the performance of the entire Hadoop deployment.

In our hands-on tests, we explored the effect this JVM choice has on Hadoop performance, focusing on two market leaders, OpenJDK 7 and Oracle JDK 7. We evaluated their relative performance on a reference Hortonworks Hadoop Data Platform (HDP) 2.0 deployment running on Red Hat Enterprise Linux 6.5 on a six-node reference implementation (see Appendix A for system configuration information).

We tested two areas within Hadoop where performance is critical and that are therefore of interest to both IT managers and Big Data scientists wishing to optimize standard Hadoop deployments:

- Core infrastructure
- Machine Learning (ML), essential for Big Data Analytics platforms

To measure Hadoop performance, we used the Intel® HiBench Sort and TeraSort benchmarks for measuring infrastructure performance as well as the Mahout Bayes classification and K-means clustering benchmarks for measuring performance of machine learning tasks.

Because our goal was to measure only the relative performance of the JVMs for each test that we performed, we used near out-of-box configurations for both the operating system and Hadoop. In addition, we did not configure the HiBench workloads to saturate the resources of the servers, but we instead chose workloads that used 60 to 80 percent of the CPU resources. We expect additional operating system, Hadoop, and workload tunings to yield comparable performance boosts for both OpenJDK and Oracle JDK. For details about the hardware we used, see Appendix A. For step-by-step details on how we tested, see Appendix B.

OpenJDK: Similar performance, better for management

For the Red Hat Enterprise Linux and Hortonworks HDP configurations we tested, we found that Hadoop workloads performed about the same with either the OpenJDK JVM or Oracle JVM. We expected this behavior based on other performance comparison tests we have reported on in the past.[2] From a manageability perspective, we found OpenJDK convenient for Red Hat Enterprise Linux administrators to use in their Hadoop deployments as it updates automatically as a component of the operating system.

WHAT WE FOUND

Figure 1 shows the Hadoop performance results we obtained for four HiBench benchmarks. They are plotted relative to the OpenJDK results for easy comparison. OpenJDK and Oracle JVM performed similarly across the four Hadoop workloads with all differences being under 8 percent. (See Figure 9 for details on the HiBench workloads.)

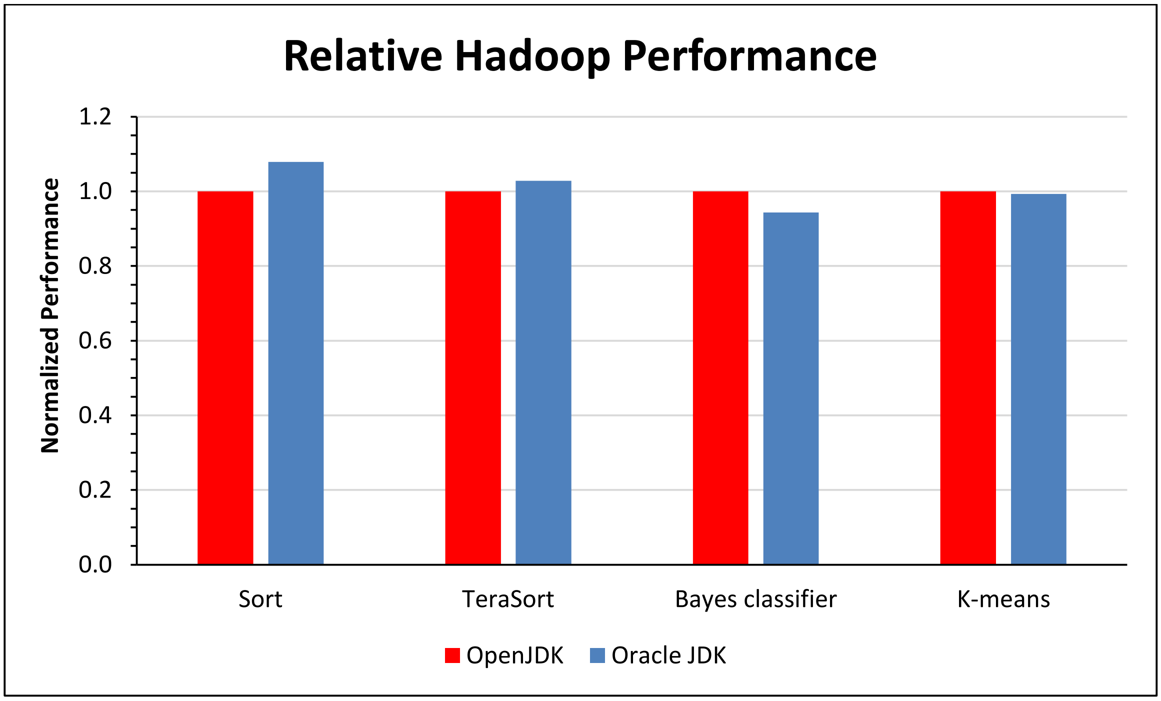

Figure 1: Relative comparison of OpenJDK and Oracle JDK on four Hadoop workloads: Sort, TeraSort, Bayes Classification, and K-means Clustering (lower is better)

[2] See our report "Performance Comparison of Multiple JVM implementations with Red Hat Enterprise Linux 6" available at www.principledtechnologies.com/Red%20Hat/RHEL6_rhj_0613.pdf

Using statistics for both JVMs, Figures 2 through 5 shows server-resource usage, averaged over the four data nodes, during the median run of the Sort benchmark. We chose to present the resource usage for this test because the difference in benchmark duration was the greatest.

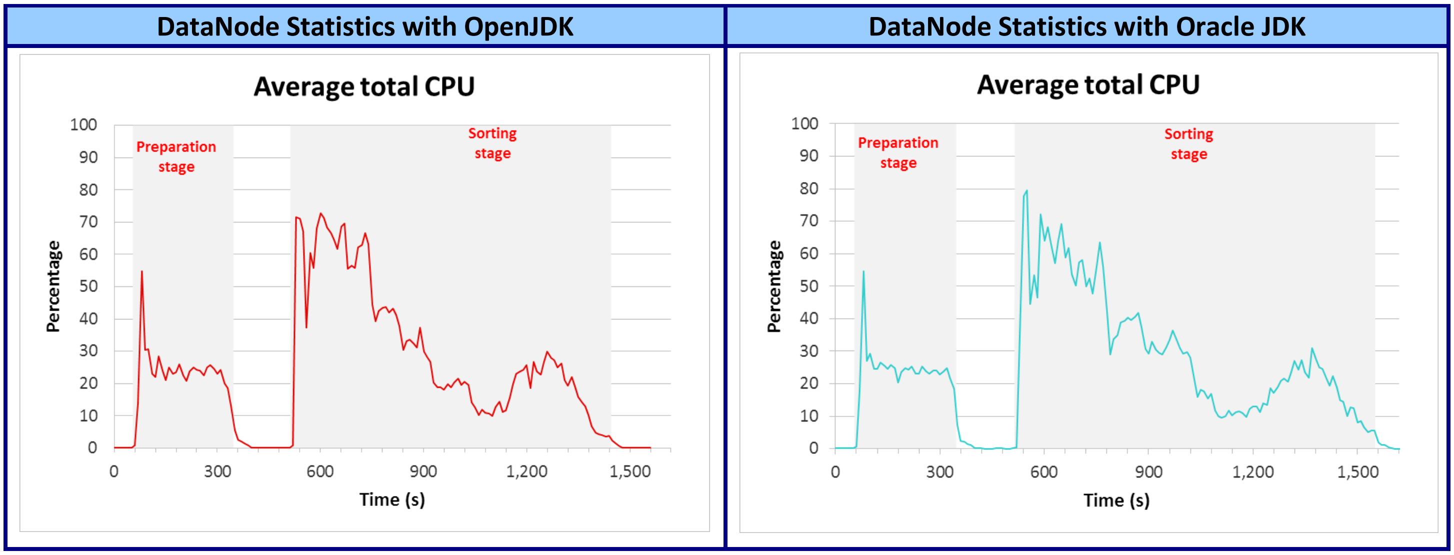

Figure 2: Comparison of average CPU usage during the Sort test for OpenJDK and Oracle JDK.

The charts compare average CPU, average total disk reads, average total disk writes, and average outgoing network I/O. As can be observed, all performance graphs have similar shapes for both JDKs.

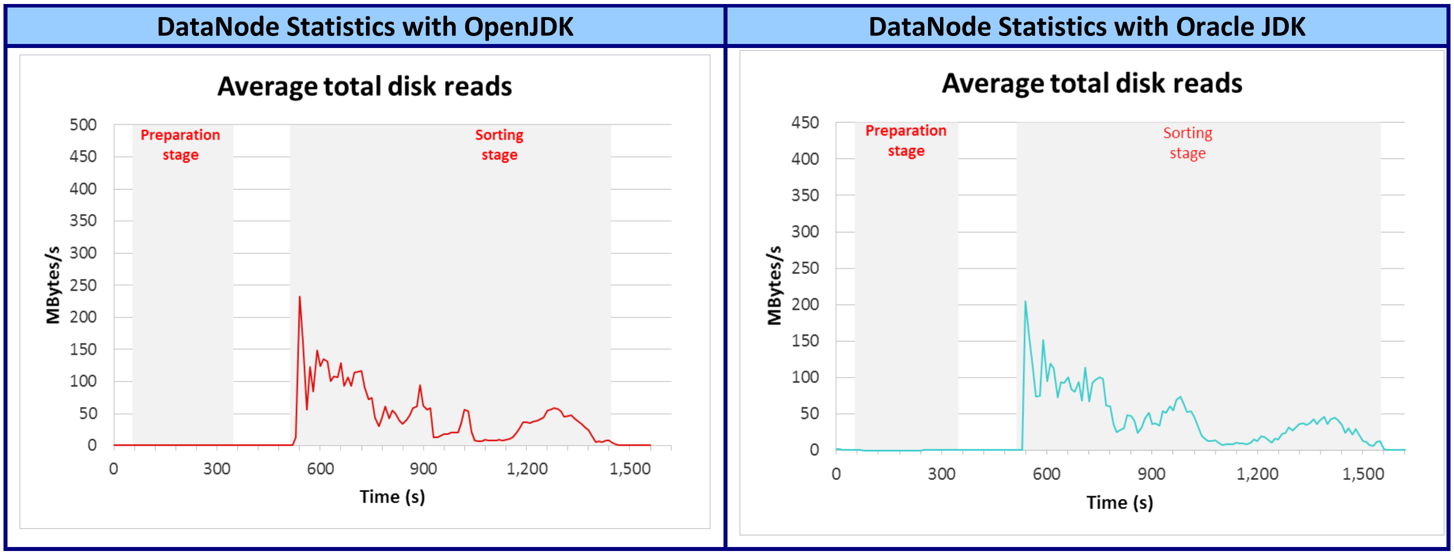

Figure 3: Comparison of average disk-reads in megabytes per second during the Sort test for OpenJDK and Oracle JDK.

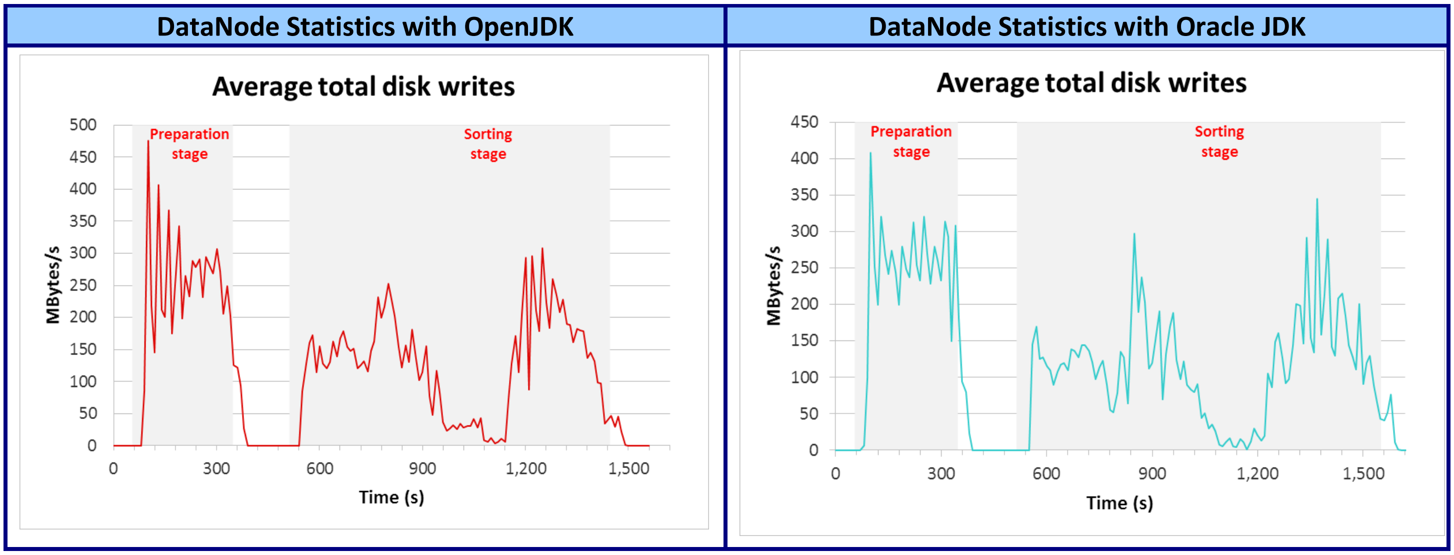

Figure 4: Comparison of average disk-writes in megabytes per second during the Sort test for OpenJDK and Oracle JDK.

We have marked the two computational stages for this workload: the data-preparation/generation stage and the data-transformation/sorting stage. See section "Running one HiBench test" in Appendix B for more detail. On the other three workloads, the use of OpenJDK or Oracle JDK in conjunction with Red Hat Enterprise Linux resulted in similar performance across these Hadoop workloads.

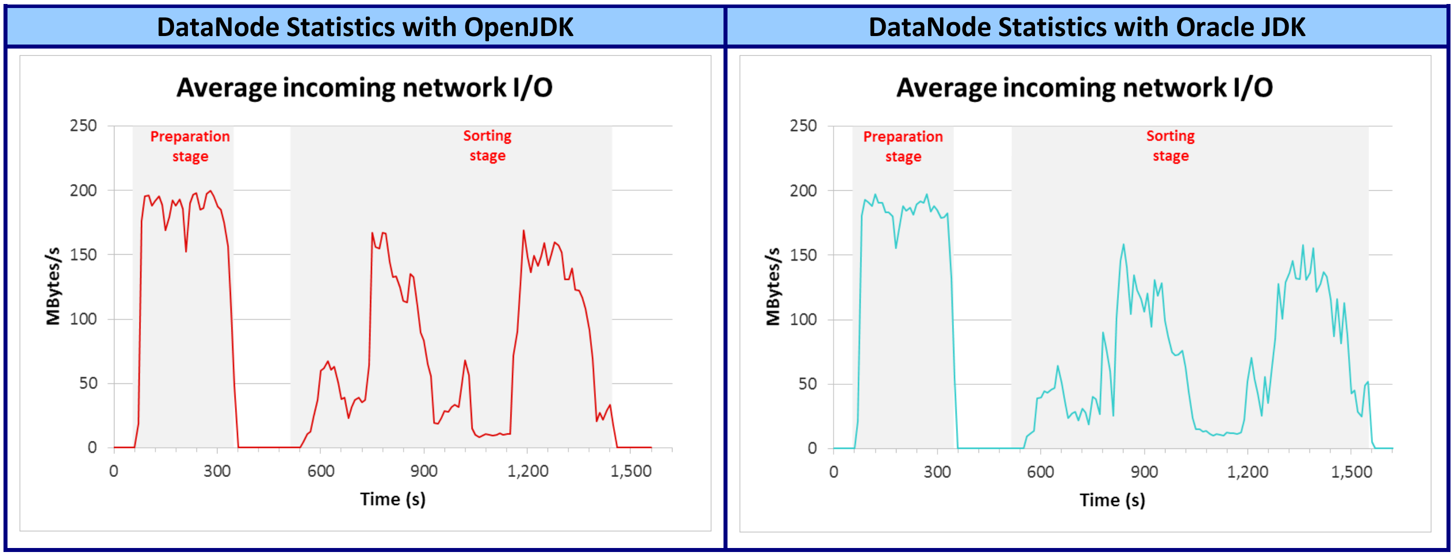

Figure 5: Comparison of average incoming network I/O in megabytes per second during the Sort test for OpenJDK and Oracle JDK.

WHAT WE TESTED

For our tests, we used the YARN version of the Intel HiBench 2.2 suite to gauge performance of a HDP 2.0 deployment. At this time, only six of the nine benchmarks have been ported to YARN (that is, for Hadoop 2.0 and MapReduce version 2). Of these, we tested Sort, TeraSort, Bayes Classifier and K-means clustering.[3] To test the core infrastructure we used Sort and TeraSort. Both of these tests are typical real-world MapReduce jobs that transform data from one representation into another but differ in the volume of data transformed. The *Sort* test processes large amounts of textual data produced by the `RandomTextWriter` text data generator available in standard Hadoop. *TeraSort* is a Big Data version of Sort. It sorts 10 billion 100-byte records produced by the `TeraGen` generator program, also part of the standard Hadoop distribution. The other two tests we chose *Bayes Classification* and *K-means Clustering*, part of the Apache Mahout program suite, represent key machine learning tasks in Big Data Analytics that use the MapReduce framework. Bayes Classification implements the training component of Naïve Bayes, a classification algorithm used in data mining and discovery applications. It takes as input a large set of randomly generated text documents. K-means clustering implements the highly used and well-understood K-means clustering algorithm operating on a large set of randomly generated numerical multidimensional vectors with specific statistical distributions.

IN CONCLUSION

OpenJDK is an efficient foundation for distributed data processing and analytics using Apache Hadoop. In our testing of a Hortonworks HDP 2.0 distribution running on Red Hat Enterprise Linux 6.5, we found that Hadoop performance using OpenJDK was comparable to the performance using Oracle JDK. Comparable performance paired with automatic updates means that OpenJDK can benefit organizations using Red Hat Enterprise Linux -based Hadoop deployments.

[3] The YARN versions of Intel HiBench benchmarks are available in Intel's GitHub HiBench source code repository. You can download them from www.github.com/intel-hadoop/HiBench/tree/yarn

APPENDIX A – SYSTEM CONFIGURATION INFORMATION

Figure 6 provides detailed configuration information for the test systems.

System	Dell™ PowerEdge™ R710 (Hadoop Master servers)	Dell PowerEdge R710 (Hadoop Slave servers)
Power supplies		
Total number	2	2
Vendor and model number	Dell N870P-S0	Dell N870P-S0
Wattage of each (W)	870	870
Cooling fans		
Total number	5	5
Vendor and model number	Dell RK 385-A00	Dell RK 385-A00
Dimensions (h × w) of each	2″ × 2″ × 1.5″	2″ × 2″ × 1.5″
Volts	12	12
Amps	1.6	1.6
General		
Number of processor packages	2	2
Number of cores per processor	6	6
Number of hardware threads per core	2	2
CPU		
Vendor	Intel	Intel
Name	Xeon®	Xeon
Model number	X5670	X5670
Stepping	02	02
Socket type	FCLGA1366	FCLGA1366
Core frequency (GHz)	2.93	2.93
Bus frequency (MHz)	3,200	3,200
L1 cache (KB)	192	192
L2 cache (KB)	1526	1526
L3 cache (KB)	12,288	12,288
Platform		
Vendor and model number	Dell PowerEdge R710	Dell PowerEdge R710
Motherboard model number	00NH4P	00NH4P
Motherboard chipset	Intel 5520	Intel 5520
BIOS name and version	6.4.0	6.4.0
BIOS settings	Defaults + Power Setting to Maximum Performance	Defaults + Power Setting to Maximum Performance
Memory module(s)		
Total RAM in system (GB)	96	48
Vendor and model number	Hynix HMT31GR7BFR4A-H9	Samsung M393B1K70BH1-CH9
Type	PC3-10600	PC3-10600
Speed (MHz)	1,333	1,333
Speed running in the system (MHz)	1,333	1,333
Timing/Latency (tCL-tRCD-tRP-tRASmin)	9-9-9-36	9-9-9-36

System	Dell™ PowerEdge™ R710 (Hadoop Master servers)	Dell PowerEdge R710 (Hadoop Slave servers)
Size (GB)	8	8
Number of RAM module(s)	12	6
Chip organization	Double-sided	Double-sided
Rank	Dual	Dual
Hard disk		
Vendor and model number	Seagate ST9146852SS	Western Digital WD300BKHG-18A29V0
Number of disks in system	6	8
Size (GB)	146	300
Buffer size (MB)	16	16
RPM	15K	10K
Type	SAS	SAS
Disk Controller		
Vendor and model	Dell PERC 6/i	Dell PERC 6/i
Controller cache (MB)	256	256
Controller Driver (Module)	megaraid_sas	megaraid_sas
Controller Driver Version	06.700.06.00-rh1	06.700.06.00-rh1
Controller firmware	6.3.3.0002	6.3.3.0002
RAID configuration	3 RAID1 volumes of 2 disks each	1 RAID1 volume of 2 disks; 6 un-RAIDed disks
Operating system		
Name	Red Hat Enterprise Linux 6.5	Red Hat Enterprise Linux 6.5
File system	ext4	ext4
Kernel	2.6.32-431.3.1.el6.x86_64	2.6.32-431.3.1.el6.x86_64
Language	English	English
Graphics		
Vendor and model number	Matrox® MGA G200eW WPCM450	Matrox MGA G200eW WPCM450
Graphics memory (MB)	8	8
Ethernet		
Vendor and model number	4 × Broadcom® 5709C NetXtreme® II t 1GigE	4 × Broadcom 5709C NetXtreme II t 1GigE
Type	PCI Express	PCI Express
Driver (Module)	bnx2	bnx2
Driver Version	2.2.3	2.2.3
Optical drive(s)		
Vendor and model number	TEAC DVD-ROM DV28SV	TEAC DVD-ROM DV28SV
Type	SATA	SATA
USB ports		
Number	4 external	4 external
Type	2.0	2.0

Figure 6: System configuration information for the test systems.

APPENDIX B – HOW WE TESTED

This section captures the installation and configuration of a six-node Hadoop cluster that has two master nodes and four data nodes. We followed closely the Red Hat Reference Architecture for running HDP 2.0 on Red Hat Enterprise Linux 6 with OpenJDK.[4]

One master node (named “master1”) ran the HDFS NameNode service. The second master node (named “master2”) was a back-up NameNode that ran the YARN resource manager and MapReduce job-history manager.

The four data nodes (named “data1”, “data2”, “data3”, and “data4”) ran the HDFS DataNode service and the YARN distributed NodeManager.

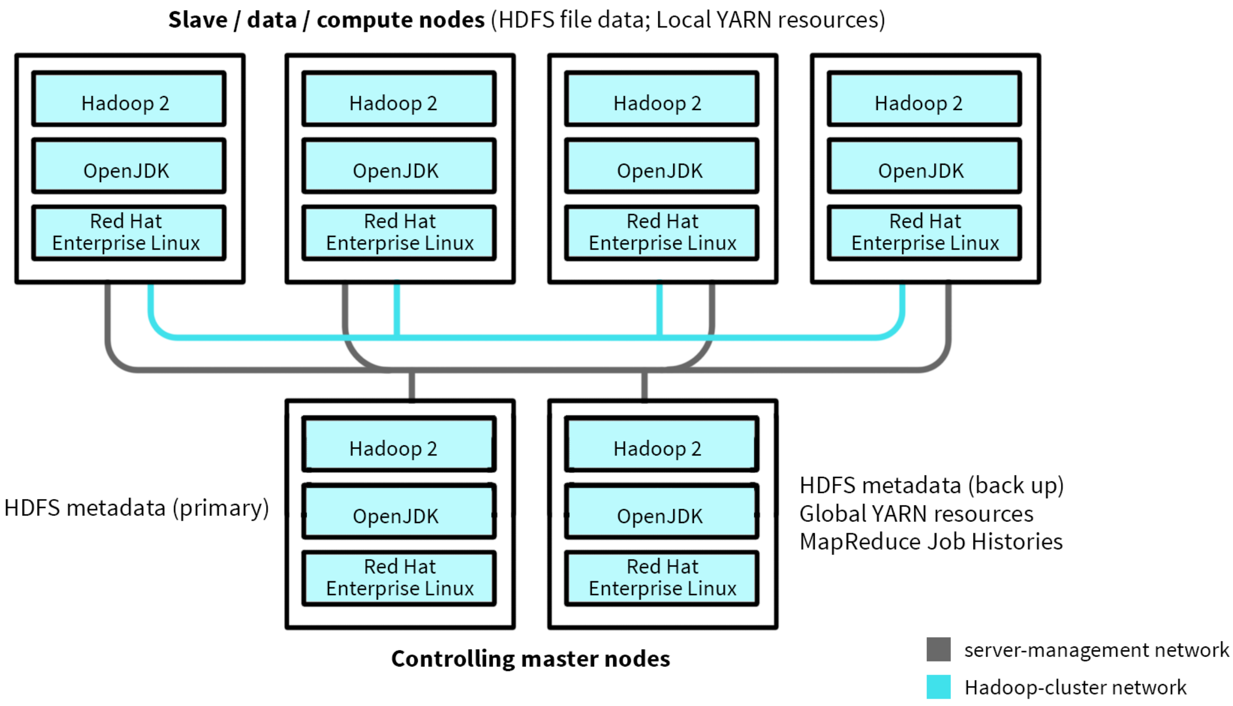

Figure 7: Schematic of the six-node Hadoop cluster used in our tests. The light blue lines denote the network connections for the Hadoop-cluster network. Similarly, the grey lines denote the server-management network.

Figure 8 shows the hardware we used in our testing. We used six Dell PowerEdge R710 servers – two master nodes and four data nodes. Each server had six or eight 300GB SAS disks, a PowerEdge Raid Controller (PERC), two Intel Xeon processors X5670, one four-port GbE NIC, and 48GB RAM.

The resources required by the two master nodes differed from those of the data nodes, so we discussed those separately.

[4] "Exploring the next generation of Big Data solutions with Hadoop 2: Deploying Hortonworks Data Platform 2.0 on Red Hat Enterprise Linux 6 with OpenJDK", version 1.2.

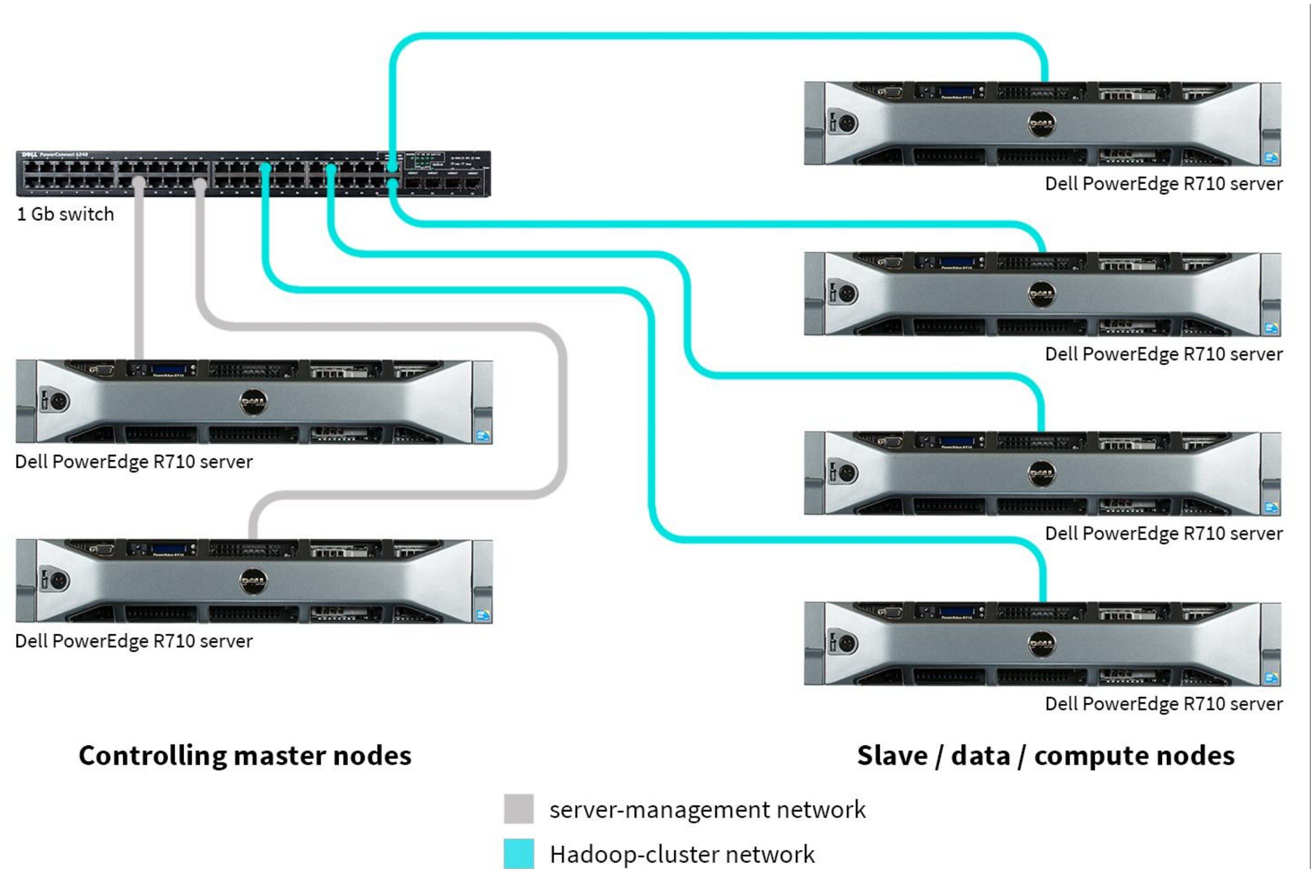
1 Gb switch
Dell PowerEdge R710 server
Dell PowerEdge R710 server
Dell PowerEdge R710 server
Dell PowerEdge R710 server
Dell PowerEdge R710 server
Dell PowerEdge R710 server
Controlling master nodes
Slave / data / compute nodes
server-management network
Hadoop-cluster network

Figure 8: The physical hardware used for the six-node Hadoop cluster in our tests.

Configuring the master nodes

Each master node had 96GB RAM and six disks. The operating system was installed on a mirrored pair of disks (RAID 1). The remaining four disks were configured as two mirrored pairs because the Hadoop metadata stored therein must be protected from loss.

Configuring the data nodes

Each data node had 48GB RAM and eight disks. The operating system was installed on a mirrored pair of disks (RAID 1). We presented the remaining six disks to the system as JBODs (without RAID) to increase I/O by providing the maximum number of spindles for the configuration.

Configuring networking

We used two subnets: one for cluster traffic and one for access to the servers. Consequently, we used two network ports on each server. In our tests, the cluster subnet was 192.168.100.0/24 and the data subnet was 192.168.200.0/24.

Installing Red Hat Enterprise Linux 6

The first step in setting up the six-node Hortonworks Data Platform 2.0 cluster is to install the Red Hat Enterprise Linux 6.5 operating system on the servers.

For each server, master and slave, install the Red Hat Enterprise Linux 6.5 operating system. Prior to starting the installation, you should have a designated list of host names and IP addresses.

For definiteness, this section instructs how to install Red Hat Enterprise Linux 6.5 from the full installation DVD, but you could also install the operating system via PXE, kickstart file, or satellite server.

1. Insert the Red Hat Enterprise Linux 6.5 installation DVD into the server's DVD drive.
2. On the Welcome to Red Hat Enterprise Linux 6.5 screen, press Enter to boot the installer.
3. On the Welcome to Red Hat Enterprise Linux for x86_64 screen, select SKIP, and press Enter.
4. On the Red Hat Enterprise Linux 6 screen, click Next.
5. On the installation-language screen, select your language (this guide uses English), and click Next.
6. On the keyboard-selection screen, select the correct format, and click Next.
7. On the Storage Devices screen, select Basic Storage Devices, and click Next.
8. If a pop-up window appears with a Storage Device Warning, click Yes, discard any data.
9. On the Name This Computer screen, enter the server's name, and click Configure Network.
10. On the Network Connections pop-up screen, select your management interface (e.g., eth0), and click Edit.
11. On the Editing eth0 pop-up screen, check the Connect automatically box, and select the IPv4 Settings tab. Change Method to Manual. Click Add, and enter the IP address, Netmask, and Gateway. Enter the IP address of your DNS server. Click Apply to close the pop-up screen.
12. Back on the Network Connections pop-up screen, select the cluster-interconnect interface (e.g., eth1), and click Edit.
13. On the Editing eth1 pop-up screen, check the Connect automatically box, and select the IPv4 Settings tab. Change Method to Manual. Click Add, and enter the IP address, Netmask, and Gateway. Enter the IP address of your DNS server. Click Apply to close the pop-up screen.
14. On the Network Connections pop-up screen, click Close, and click Next.
15. On the Time Zone screen, select your time zone, and click Next.
16. On the administrator's password page, enter the root password, and click Next.
17. On the Installation Type screen, keep the default installation type, check the Review and modify partitioning layout box, and click Next.
18. On the Device assignment screen, select the server's OS disk from the list in the left column, and click the upper arrow to designate the disk as the target device. Click Next.
19. On the Review and Modify Partitioning screen, delete the /home logical volume and assign the resulting free space to the root partition. When finished, click Next.
20. On the Format Warnings pop-up screen, click Format.
21. On the Writing storage configuration to disk screen, click Write changes to disk.
22. On the Boot loader selection screen, keep the defaults, and click Next.
23. On the Red Hat Enterprise Linux software installation screen, select Basic Server, and click Next.
24. When the installation completes, click Reboot.

Configuring the hosts

This section shows how to perform post-installation configuration steps for all hosts at once by running a sequence of bash commands from one of the servers rather than configuring them one at a time. The post-installation configurations include:

- Configuring NTP
- Configuring the file systems
- Disabling unneeded services
- Configuring the tuned package
- Installing OpenJDK

- Adding the IP address and hostnames to /etc/hosts

Configuring NTP

1. The servers' clocks must be synchronized to a common time source in order for the cluster to function properly. Modify /etc/ntp.conf to add the IP address of your local NTP server (for example, 192.168.200.20) as in the following line:

```
server 192.168.200.20 iburst
```

2. Start the NTP daemon on each server; for example,

```
chkconfig ntpd on
service ntpd start
```

Configuring the file system

Partition, format, and mount the data disks on each server. The servers' operation and application filesystem is on disk /dev/sda. The Hadoop data disks on the master servers are /dev/sdb and /dev/sdc, and on the slave servers are /dev/sdb, /dev/sdc, /dev/sdd, /dev/sde, /dev/sdf, and /dev/sdg.

1. On each node, partition and format the first data disk (sdb), and mount its filesystem on directory /grid01.

```
parted /dev/sdb mklabel gpt
parted /dev/sdb mkpart primary "1 -1"
mkfs.ext4 /dev/sdb1
mkdir /grid01
mount /dev/sdb1 /grid01
echo "/dev/sdb1 /grid01 ext4 defaults 0 0" >> /etc/fstab
/dev/sdc1 /grid02 ext4 defaults 0 0
```

2. On the master nodes, repeat step 1 for the second data disk (sdc).
3. On each slave node, repeat step 1 for the remaining data disks (sdc, sdd, sde, sdf, and sdg).

Disabling unneeded services

Disable unneeded services on each server.

```
for i in autofs cups nfslock portreserve postfix rpcbind rpcgssd iptables
ip6tables; do
  chkconfig $i off
  service $i stop
done
```

Configuring the tuned package

On each server, install and configure the *tuned* package to use the enterprise-storage profile.

```
yum install tuned
tuned-adm profile enterprise-storage
```

Installing OpenJDK

To install HDP 2.0, we installed the OpenJDK platform. Instructions for installing the Oracle JDK for the performance test are in the section "Changing the Java platform from OpenJDK to Oracle JDK."

1. On each server, install OpenJDK 1.7.0 b51.

```
yum install java-1.7.0-openjdk java-1.7.0-openjdk-devel
```

2. On each server, link the OpenJDK directory to /usr/java/default, the HDP default for JAVA_HOME.

```
mkdir /usr/java
ln -s /usr/lib/jvm/java-1.7.0-openjdk-1.7.0.*.x86_64 /usr/java/default
```

Adding the IP address and hostnames to /etc/hosts

We used the hosts file rather than a DNS server to provide name resolution. On each server, add the following to the end of the file /etc/hosts.

```
# cluster network
192.168.100.51    master1     # name node
192.168.100.52    master2     # secondary name node and YARN master
192.168.100.61    data1       # data node
192.168.100.62    data2       # data node
192.168.100.63    daat3       # data node
192.168.100.64    data4       # data node
# management network
192.168.200.51    master1m    # name node
192.168.200.52    master2m    # secondary name node and YARN master
192.168.200.61    data1m      # data node
192.168.200.62    data2m      # data node
192.168.200.63    daat3m      # data node
192.168.200.64    data4m      # data node
```

Installing Hortonworks Data Platform 2.0 on the servers

For further detail on installing and configuring Hortonworks Data Platform, see "Installing HDP Manually" (chapters 2-4).[5] While it is possible to install Hortonworks Data Platform in other ways, the method we used has the advantage that the RPM installation creates the Hadoop user accounts and groups along with system configuration files controlling system resources.

Installing Hadoop software

1. Download the HDP yum repo file from Hortonworks and copy it to each of the servers.

```
wget http://public-repo-1.hortonworks.com/HDP/centos6/2.x/updates/2.0.6.0/hdp.repo
```

2. Install the Hadoop software (HDFS, YARN, MapReduce, Mahout, compression and SSL libraries) on each server.

```
yum install \
  hadoop hadoop-hdfs hadoop-libhdfs hadoop-client\
  hadoop-yarn   hadoop-mapreduce mahout\
  openssl snappy snappy-devel lzo lzo-devel hadoop-lzo hadoop-lzo-native
ln -sf /usr/lib64/libsnappy.so /usr/lib/hadoop/lib/native/.
```

Server modifications for HDFS, YARN, and MapReduce

1. If necessary, delete the previous Hadoop data and log directories by running these commands on each server.

```
rm -rf /grid0?/hadoop/
rm -rf /var/log/hadoop/hdfs
rm -rf /var/log/hadoop/yarn
rm -rf /var/log/hadoop/mapred
rm -rf /var/run/hadoop/hdfs
rm -rf /var/run/hadoop/yarn
rm -rf /var/run/hadoop/mapred
```

2. On each server, create the first set of HDFS metadata directories under /grid01.

```
mkdir -p /grid01/hadoop/hdfs/nn
mkdir -p /grid01/hadoop/hdfs/snn
chown -R hdfs:hadoop /grid01/hadoop/hdfs/nn
chown -R hdfs:hadoop /grid01/hadoop/hdfs/snn
chmod -R 755 /grid01/hadoop/hdfs/nn
chmod -R 755 /grid01/hadoop/hdfs/snn
```

3. On each slave server, create the HDFS and YARN data directories under /grid01.

[5] docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.0.9.0/bk_installing_manually_book/content/index.html

```
mkdir -p /grid01/hadoop/hdfs/dn
mkdir -p /grid01/hadoop/yarn/local
mkdir -p /grid01/hadoop/yarn/logs
chown -R hdfs:hadoop /grid01/hadoop/hdfs/dn
chown -R yarn:hadoop /grid01/hadoop/yarn/local
chown -R yarn:hadoop /grid01/hadoop/yarn/logs
chmod -R 750 /grid01/hadoop/hdfs/dn
chmod -R 750 /grid01/hadoop/yarn/local
chmod -R 750 /grid01/hadoop/yarn/logs
```

4. On each master server, repeat step 2 for the second data directory, /grid02.
5. On each slave server, repeat steps 2 and 3 for the remaining data directory, /grid02, /grid03, /grid04, /grid05, /grid06.
6. On the second master server, create the directories for the YARN resource manager.

```
mkdir -p /grid01/hadoop/yarn/local
mkdir -p /grid01/hadoop/yarn/logs
mkdir -p /grid02/hadoop/yarn/local
mkdir -p /grid02/hadoop/yarn/logs
chown -R yarn:hadoop /grid0?/hadoop/yarn/lo*
chmod -R 750 /grid0?/hadoop/yarn/lo*
```

7. On all servers, add the YARN, and mapred users to the hdfs group.

```
usermod -G hdfs yarn
usermod -G hdfs mapred
```

8. On each server, correct the file ownership and permissions for a YARN health-check command.

```
chown -R root:hadoop /usr/lib/hadoop-yarn/bin/container-executor
chmod -R 6050 /usr/lib/hadoop-yarn/bin/container-executor
```

Configuring Hadoop

The Hadoop components, HDFS, YARN, and MapReduce, require configuration to specify the nodes and their roles as well as the location of each node's HDFS files. Hortonworks provides a sample set of configuration files that you can readily modify for your cluster. You can always find the up-to-date Internet location of the tar archive for these helper files in Hortonworks HDP documentation; see *Hortonworks Data Platform: Installing HDP Manually*, section 1.8.

1. Download Companion Files.[6] We used version 2.0.6.101 of the helper files. Download the helper files from Hortonworks.

```
wget http://public-repo-1.hortonworks.com/HDP/tools/2.0.6.0/\
hdp_manual_install_rpm_helper_files-2.0.6.101.tar.gz
```

2. Extract the files.

```
tar zxf hdp_manual_install_rpm_helper_files-2.0.6.101.tar.gz
```

3. On each server, delete the contents of the Hadoop core configuration directory.

```
rm /etc/hadoop/conf/*
```

4. On one server, copy the contents of the core Hadoop configuration files to /etc/hadoop/conf.

```
cd hdp_manual_install_rpm_helper_files-2.0.6.101/configuration_files/core_hadoop
cp * /etc/hadoop/conf
cd /etc/hadoop/conf
```

5. In the file `core-site.xml`, replace every occurrence of the token `TODO-NAMENODE-HOSTNAME:PORT` with `master1:8020.`

[6] docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.0.9.0/bk_installing_manually_book/content/rpm-chap1-9.html

6. In the file hdfs-site.xml, perform this sequence of changes:
    a. Replace every occurrence of the token TODO-NAMENODE-HOSTNAME with master1.
    b. Replace every occurrence of the token TODO-NAMENODE- SECONDARYNAMENODE with master2.
    c. Replace every occurrence of the token TODO-DFS-DATA-DIR with file:///grid01/hadoop/hdfs/dn,file:///grid02/hadoop/hdfs/dn,file:///grid03/hadoop/hdfs/dn,file:///grid04/hadoop/hdfs/dn,file:///grid05/hadoop/hdfs/dn,file:///grid06/hadoop/hdfs/dn
    d. Replace every occurrence of the token TODO-DFS-NAME-DIR with grid01/hadoop/hdfs/nn,/grid02/hadoop/hdfs/nn.
    e. Replace every occurrence of the token TODO-DFS-NAME-DIR with /grid01/hadoop/hdfs/nn,/grid02/hadoop/hdfs/nn
7. In the file yarn-site.xml, replace every occurrence of the token TODO-RMNODE-HOSTNAME with master2.
8. In the file mapred-site.xml, replace every occurrence of the token TODO-JOBHISTORYNODE-HOSTNAME with master2.
9. In the files container-executor.cfg and yarn-site.xml, replace every occurrence of the token TODO-YARN-LOCAL-DIR with /grid01/hadoop/yarn/local,/grid02/hadoop/yarn/local al.
10. In the file container-executor.cfg, replace every occurrence of the token TODO-YARN-LOG-DIR with /grid01/hadoop/yarn/logs,/grid02/hadoop/yarn/logs.
11. In the file yarn-site.xml, replace every occurrence of the token TODO-YARN-LOCAL-LOG-DIR with /grid01/hadoop/yarn/logs,/grid02/hadoop/yarn/logs.
12. Copy the updated Hadoop configuration (HDFS, YARN, and MapReduce) to the remaining servers with scp. For example, for a server named NODE.
```
scp /etc/hadoop/conf/* NODE:/etc/hadoop/conf/
```

Starting Hadoop for the first time

Initializing HDFS

1. Log onto the namenode master as user hdfs.
2. Format HDFS.
```
hdfs namenode -format
```
3. Start HDFS by running the commands in steps 1 through 4 in the “Starting HDFS” section.

Initializing YARN and the Job History service

1. With HDFS running, create and configure directories on HDFS for the job-history service.
2. Log onto the second master server as user hdfs, and run the following commands.
```
hdfs dfs -mkdir –p /mr-history/tmp
hdfs dfs -mkdir -p /mr-history/done
hdfs dfs -mkdir -p /app-logs
hdfs dfs -chmod -R 1777 /mr-history/tmp
hdfs dfs -chmod -R 1777 /mr-history/done
hdfs dfs -chmod -R 1777 /app-logs
hdfs dfs -chown -R mapred:hdfs /mr-history
hdfs dfs -chown yarn /app-logs
```
3. Start YARN by running the commands in steps 1 through 3 of the Starting YARN section.

Starting Hadoop

Hadoop is started by first starting HDFS, pausing for 30 seconds, and then starting YARN. See the next two sections for instructions for starting HDFS and YARN.

Starting HDFS

1. Log onto each of the servers as user hdfs.
2. Run the following command on the namenode master.
   ```
   /usr/lib/hadoop/sbin/hadoop-daemon.sh start namenode
   ```
3. Run the following command on the secondary-namenode master.
   ```
   /usr/lib/hadoop/sbin/hadoop-daemon.sh start secondarynamenode
   ```
4. Run the following command on each slave server.
   ```
   /usr/lib/hadoop/sbin/hadoop-daemon.sh start datanode
   ```

Starting YARN

HDFS must be running before starting YARN. See the previous section for instructions for starting HDFS.

1. Log onto the second master server as user yarn and run the following commands on the secondary master server.
   ```
   export HADOOP_LIBEXEC_DIR=/usr/lib/hadoop/libexec
   /usr/lib/hadoop-yarn/sbin/yarn-daemon.sh start resourcemanager
   ```
2. Log off the second master server and login again as user mapred to run the following commands.
   ```
   export HADOOP_LIBEXEC_DIR=/usr/lib/hadoop/libexec
   /usr/lib/hadoop-mapreduce/sbin/mr-jobhistory-daemon.sh start historyserver
   ```
3. Log onto the slave servers as user yarn in order to start the local YARN resource node-managers, and run the following commands on each.
   ```
   export HADOOP_LIBEXEC_DIR=/usr/lib/hadoop/libexec
   /usr/lib/hadoop-yarn/sbin/yarn-daemon.sh start nodemanager"'
   ```

Stopping Hadoop

Hadoop is stopped by stopping YARN and then stopping HDFS. See the next two sections for instructions for stopping HDFS and YARN.

Stopping YARN

1. Log onto the slave servers as user yarn and run the following commands on each.
   ```
   export HADOOP_LIBEXEC_DIR=/usr/lib/hadoop/libexec
   /usr/lib/hadoop-yarn/sbin/yarn-daemon.sh stop nodemanager
   ```
2. Log onto the second master server as user mapred to run the following commands.
   ```
   export HADOOP_LIBEXEC_DIR=/usr/lib/hadoop/libexec
   /usr/lib/hadoop-mapreduce/sbin/mr-jobhistory-daemon.sh stop historyserver
   ```
3. Log off the second master server and login again as user yarn to run the following commands.
   ```
   export HADOOP_LIBEXEC_DIR=/usr/lib/hadoop/libexec
   /usr/lib/hadoop-yarn/sbin/yarn-daemon.sh stop resourcemanager
   ```

Stopping HDFS

1. Log onto the servers as user hdfs.
2. On the slave servers, run the following command.
   ```
   usr/lib/hadoop/sbin/hadoop-daemon.sh stop datanode
   ```
3. On the secondary-namenode master server, run the following command.
   ```
   /usr/lib/hadoop/sbin/hadoop-daemon.sh stop secondarynamenode
   ```
4. On the namenode master server, run the following command.
   ```
   /usr/lib/hadoop/sbin/hadoop-daemon.sh stop namenode
   ```

Installing HiBench

1. Download the HiBench Suite from github.com/intel-hadoop/HiBench/tree/yarn.

2. Copy the HiBench ZIP archive to one of the master nodes.

3. On the master node, unzip the HiBench archive under */opt*.

   ```
   cd /opt
   unzip /tmp/HiBench-yarn.zip
   ```

4. Modify the default HiBench configuration for HDP 2 in the file */opt/HiBench-yarn/bin/hibench-config.sh* : change HADOOP_CONF_DIR= to HADOOP_CONF_DIR=/etc/hadoop/conf, and change HADOOP_EXAMPLES_JAR= to HADOOP_EXAMPLES_JAR=/usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples-2.2.0.2.0.6.0-101.jar.

5. Change the ownership of the HiBench files to user hdfs.

   ```
   chown -R hdfs /opt/HiBench-yarn
   ```

Running HiBench Sort, TeraSort, Bayes and K-means tests

The HiBench tests are performed as user hdfs from the namenode. The default dataset sizes were modified to use more of the cluster's resources.

The changes to the configurations are as follows:

1. For the Sort test, the dataset size is increased by a factor of 10 to 24,000,000,000 bytes: In the file `/opt/HiBench-yarn/sort/conf/configure.sh`, change the line `DATASIZE=2400000000 to DATASIZE=24000000000`.

2. For the Bayes test, the number of document pages in the dataset is increased by a factor of 2.5 to 100,000: : In the file `/opt/HiBench-yarn/bayes/conf/configure.sh`, change the line `PAGES=40000` to `PAGES=100000`.

3. For the TeraSort test, the number of document pages in the dataset is increased by a factor of 2.5 to 100,000: : In the file `/opt/HiBench-yarn/terasort/conf/configure.sh`, change the line `DATASIZE=100000000` to `DATASIZE=2000000000`, and change the line `NUM_MAPS=96 to NUM_MAPS=180`.
4. For the K-means test, modify the file `/opt/HiBench-yarn/kmenas/conf/configure.sh` as follows: change the line `NUM_OF_CLUSTERS=5` to `NUM_OF_CLUSTERS=32`, the line `NUM_OF_SAMPLES=3000000` to `NUM_OF_SAMPLES=200000000`, the line `SAMPLES_PER_INPUTFILE=600000` to `SAMPLES_PER_INPUTFILE=40000000`, the line `DIMENSIONS=20` to `DIMENSIONS=4`.

Changing the Java platform from OpenJDK to Oracle JDK

The Hadoop configuration used the OpenJDK and can be used directly for performance test. To run performance tests with the Oracle JDK, we removed OpenJDK, installed the Oracle JDK, and rebuilt HDFS.

1. On all servers, stop YARN by following the instruction in section “Stopping YARN.”
2. On all servers, stop HDFS by following the instructions in section “Stopping HDFS.”
3. On all servers, remove OpenJDK.
   ```
   yum remove yum install java-1.7.0-openjdk java-1.7.0-openjdk-devel
   rm /usr/java/default
   ```
4. On all servers, install Oracle JDK 1.7.0 b51 from the Red Hat Supplementary repository.
   ```
   yum install java-1.7.0-oracle-1.7.0.51-1jpp.1.el6_5.x86_64\
               java-1.7.0-oracle-devel1.7.0.51-1jpp.1.el6_5.x86_64
   ln -s /usr/lib/jvm/java-1.7.0-oracle-1.7.0.51.x86_64  /usr/java/default
   ```
5. Destroy and recreate the Hadoop directories and log files by performing steps 1 through 6 in section “Server modifications for HDFS, YARN, and MapReduce.”
6. Reformat HDFS and YARN by following the instructions in sections “Initializing HDFS and “Initializing YARN and

the Job History service.”

7. Start YARN following the instructions in section “Starting YARN.”

Running one HiBench test

The following bash script, run-hibench.sh, was used to perform complete runs for the HiBench tests. In each case, the script performs the following steps:

1. Before the HiBench data is created, the page cache is cleared on the six servers by running an auxiliary script included at the end of this section. This script must be run as root. The sudo command was used to affect this.
2. The HiBench data is prepared.
3. The page cache is cleared again before the HiBench algorithm is started.
4. Pause for 120 seconds.
5. Run the HiBench algorithm.
6. Print the HiBench statistics.

The bash script *sort-run.sh* for running the full Sort test follows.

```
#!/bin/bash
# run-hibench.sh
## run the HiBench Sort test, specified by the argument
if [ "$#" -ne 1 ]; then
  echo "Usage: $0 <name of test>"; exit 1
fi
TEST=$1
HIBENCH_DIR=/opt/HiBench-yarn
echo "Run of HiBench test $TEST"; date; echo " clearing the cache"
sudo sh /root/scripts/clear-cache.sh
echo; echo "Pausing for 50 seconds"; sleep 50
echo; echo Starting data preparation...; date; echo
$HIBENCH_DIR/$TEST/prepare.sh; echo; date; echo
echo Data prep completed
echo " clearing the cache before the main run"
sudo sh /root/scripts/clear-cache.sh
echo; echo "Pausing for 120 seconds"; sleep 120 ;
echo; echo Starting data-transformation...; date; echo
MAHOUT_HOME=/usr/lib/mahout \
      $HIBENCH_DIR/$TEST/bin/run.sh; echo; date; echo
echo Run completed; echo
echo HiBench statistics
head -n 1 $HIBENCH_DIR/hibench.report
tail -n 1 $HIBENCH_DIR/hibench.report
echo Done
```

The bash script *clear-cache.sh*, used by *sort-run.sh* and *bayes-run.sh*, follows.

```
#!/bin/bash
# clear-cache.sh
## clear the page cache on the Hadoop nodes
for i in master1 master2 data1 data2 data3 data4 ; do
  ssh $i 'sync; echo 3 > /proc/sys/vm/drop_caches ; sync'
done
```

Detailed Hadoop results for OpenJDK and Oracle JDK on four HiBench workloads

Figure 9 provides the detailed results we obtained for all runs of the HiBench benchmarks, on both OpenJDK and Oracle JVMs. For each run, we show the input size in KB, the duration of the run in seconds, and the total throughput per node in KB per second. These items show the problem size, its processing time and the corresponding average data-volume movement per node, respectively. For each benchmark and JDK permutation, we selected a median value across all runs. Those runs are highlighted in yellow. The variation in HiBench scores for the various tests was similar (about half of one percent) with the exception of the Sort test with Oracle JDK. We found the variation of Sort scores with Oracle JDK was about 10 percent over 12 runs as compared to 2 percent for Sort with OpenJDK. For this reason, and to ensure a more representative median run, we tested seven runs for Oracle JDK versus five runs for OpenJDK on the Sort test.

	OpenJDK			Oracle JDK		
Test	**Input size (KB)**	**Duration (seconds)**	**Throughput/node (KBps)**	**Input size (KB)**	**Duration (seconds)**	**Throughput/node (KBps)**
Sort						
Run 1	96,239,051	916.7	26,245.2	96,239,145	1,261.0	19,079.3
Run 2	96,239,051	936.2	25,699.7	96,239,047	973.3	24,719.5
Run 3	96,239,051	1,212.3	19,846.8	96,239,188	1,133.7	21,223.2
Run 4	96,239,124	962.3	25,002.0	96,239,121	1,127.7	21,334.4
Run 5	96,239,081	1,142.9	21,052.3	96,239,271	925.2	26,004.0
Run 6				96,239,020	1,038.4	23,169.4
Run 7				96,239,126	863.1	27,877.0
TeraSort						
Run 1	195,312,500	2,307.5	21,160.3	195,312,500	2,267.7	21,532.0
Run 2	195,312,500	2,094.7	23,310.8	195,312,500	2,349.6	20,781.5
Run 3	195,312,500	2,284.7	21,371.5	195,312,500	2,375.5	20,555.0
Bayes						
Run 1	437,075	1,046.3	104.4	437,075	987.4	110.7
Run 2	437,075	1,043.1	104.8	437,075	972.0	112.4
Run 3	437,075	1,093.0	100.0	437,075	990.7	110.3
K-means						
Run 1	50,000,007	2,364.2	5,287.2	50,000,007	2,342.9	5,335.4
Run 2	50,000,007	2,355.6	5,306.6	50,000,007	2,359.1	5,298.7
Run 3	50,000,007	2,368.4	5,277.9	50,000,007	2,348.8	5,321.9

Figure 9: Detailed Hibench results for Sort, TeraSort, Bayes Classification, and K-means Clustering.

ABOUT PRINCIPLED TECHNOLOGIES

Principled Technologies, Inc.
1007 Slater Road, Suite 300
Durham, NC, 27703
www.principledtechnologies.com

We provide industry-leading technology assessment and fact-based marketing services. We bring to every assignment extensive experience with and expertise in all aspects of technology testing and analysis, from researching new technologies, to developing new methodologies, to testing with existing and new tools.

When the assessment is complete, we know how to present the results to a broad range of target audiences. We provide our clients with the materials they need, from market-focused data to use in their own collateral to custom sales aids, such as test reports, performance assessments, and white papers. Every document reflects the results of our trusted independent analysis.

We provide customized services that focus on our clients' individual requirements. Whether the technology involves hardware, software, Web sites, or services, we offer the experience, expertise, and tools to help our clients assess how it will fare against its competition, its performance, its market readiness, and its quality and reliability.

Our founders, Mark L. Van Name and Bill Catchings, have worked together in technology assessment for over 20 years. As journalists, they published over a thousand articles on a wide array of technology subjects. They created and led the Ziff-Davis Benchmark Operation, which developed such industry-standard benchmarks as Ziff Davis Media's Winstone and WebBench. They founded and led eTesting Labs, and after the acquisition of that company by Lionbridge Technologies were the head and CTO of VeriTest.

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.