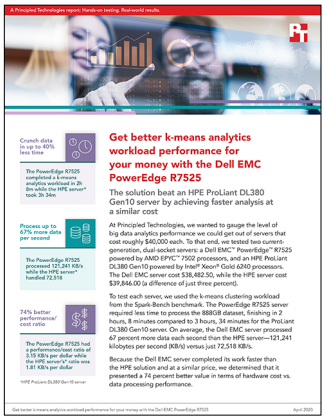

The science behind the report:

Get better k-means analytics workload performance for your money with the Dell EMC PowerEdge R7525

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 Get better k-means analytics workload performance for your money with the Dell EMC PowerEdge R7525.

We concluded our hands-on testing on February 26, 2020. During testing, we determined the appropriate hardware and software configurations and applied updates as they became available. The results in this report reflect configurations that we finalized on February 21, 2020 or earlier. Unavoidably, these configurations may not represent the latest versions available when this report appears.

Our results

Table 1: Time to complete the k-means workload on an 888GB file

	Dell EMC PowerEdge R7525	HPE ProLiant DL380 Gen10
Run 1	2 hours, 9 minutes	3 hours, 33 minutes
Run 2	2 hours, 1 minute	3 hours, 36 minutes
Run 3	2 hours, 8 minutes	3 hours, 34 minutes
Median	**2 hours, 8 minutes**	**3 hours, 34 minutes**

Table 2: Data processing rate, hardware cost, and value

	Dell EMC PowerEdge R7525	HPE ProLiant DL380 Gen10
Processing rate (KB/s)	121,241	72,518
Hardware cost (USD)	$38,482.50	$39,846.00
Value (KB/s per dollar)	3.15	1.81

CPU utilization for each server under test

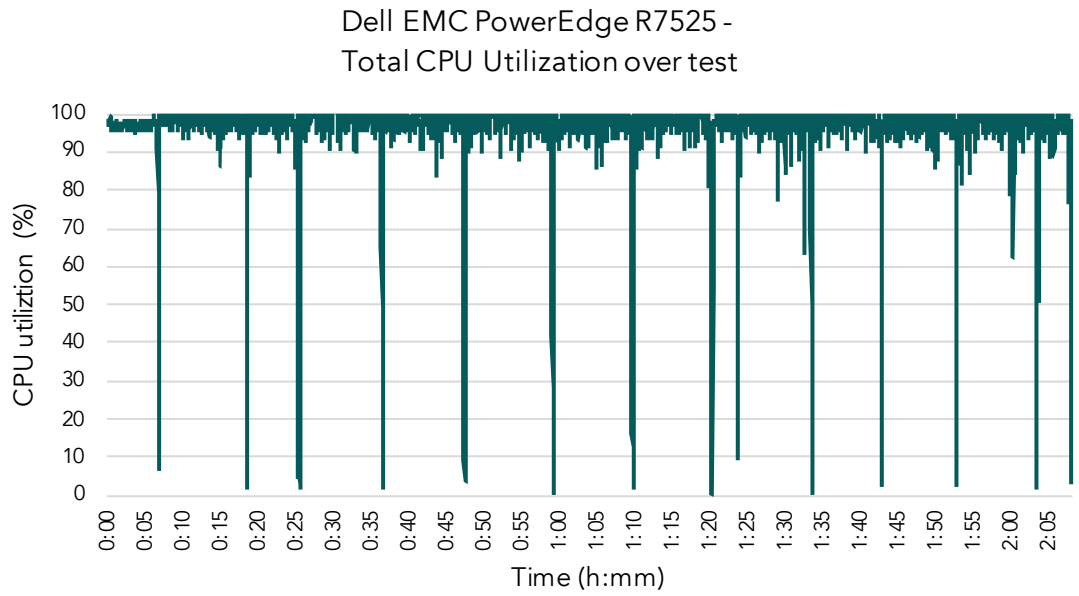

Figure 1: Graph of the Dell EMC PowerEdge R7525 server's CPU utilization for the duration of the k-means clustering workload. Source: Principled Technologies.

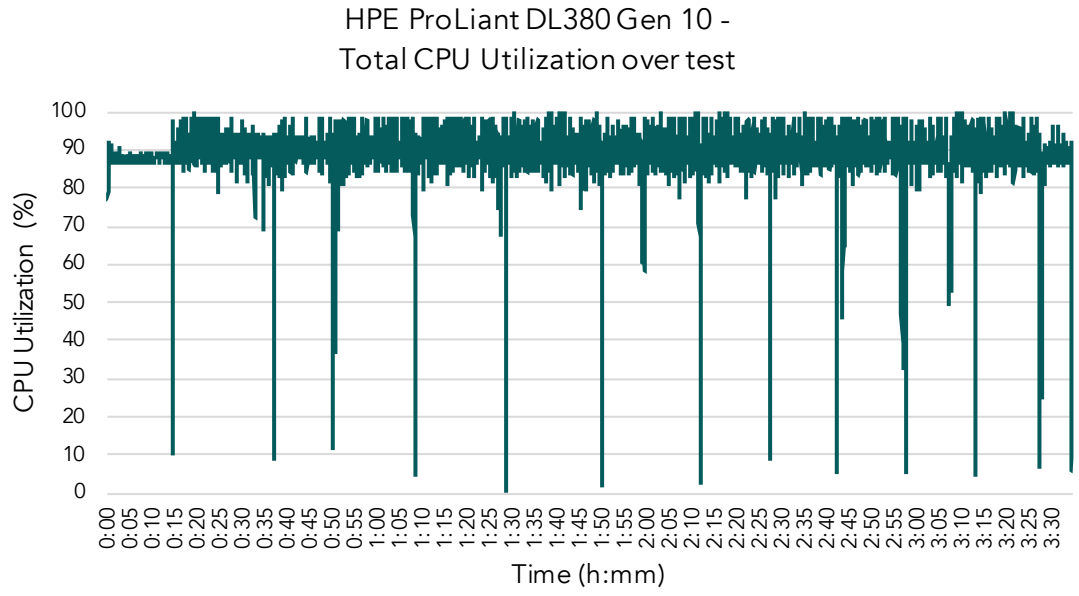

Figure 2: Graph of the HPE ProLiant DL380 Gen10 server's CPU utilization for the duration of the k-means clustering workload. Source: Principled Technologies.

System configuration information

Table 3: Detailed information on the systems we tested.

Server configuration information	Dell EMC PowerEdge R7525	HPE ProLiant DL380 Gen10
BIOS name and version	Dell 1.2.9	U30 v2.10
Non-default BIOS settings	System Profile Settings > System Profile: Performance	Workload Profile – High Performance Computer (HPC)
Operating system name and version/build number	RHEL 8.1	RHEL 8.1
Date of last OS updates/patches applied	02/07/2020	02/07/2020
Processor		
Number of processors	2	2
Vendor and model	AMD EPYC 7502	Intel® Xeon® Gold 6240
Core count (per processor)	32	18
Core frequency (GHz)	2.5	2.6
Stepping	U0	B1
Memory module		
Total memory in system (GB)	512	512
Number of memory modules	16	16
Vendor and model	SK Hynix HMA84GR7CJR4N-XN	SK Hynix HMA84GR7CJR4N-WM
Size (GB)	32	32
Type	PC4-3200	PC4-2933Y
Speed (MHz)	3,200	2,933
Speed running in the server (MHz)	3,200	2,933
Local storage		
Number of drives	5	5
Drive vendor and model	Samsung® PM1725b 1.6TB SFF	Samsung PM1725b 1.6TB SFF
Drive size (TB)	1.6	1.6
Drive information (speed, interface, type)	PCIe Gen 3 x4/dual x2	PCIe Gen 3 x4/dual x2
Software RAID usage	RAID10 using 4 disks	RAID10 using 4 disks
Network adapters		
Vendor and model	Broadcom Gigabit Ethernet BCM5720	HPE Ethernet 1Gb 4-port 331i Adapter
Number and type of ports	4 x 1GbE	4 x 1GbE
Cooling fans		
Vendor and model	DC Brushless PPPTM-X30	Nidec UltraFlo V60E12BS1M3
Number of cooling fans	6	6
Power supplies		

Server configuration information	Dell EMC PowerEdge R7525	HPE ProLiant DL380 Gen10
Vendor and model	DellEMC D2400E-S1	HP 865414-B21
Number of power supplies	2	2
Wattage of each (W)	2400	800

How we tested

We assessed which of two dual-socket systems could more quickly complete a k-means algorithm from the Spark-Bench benchmark suite. The servers we tested are as follows:

- Dell EMC PowerEdge R7525 powered by AMD EPYC 7502 processors
- HPE ProLiant DL380 Gen10 server powered by Intel Xeon Gold 6240 processors

For each system, we selected processors with a comparable price point.

We installed Red Hat Enterprise Linux 8.1 (RHEL 8.1) on both servers and ran a k-means cluster-finding algorithm from the Spark-Bench benchmark suite on an 888GB dataset. We hosted this dataset on a Linux software RAID10 we built from four NVMe drives.

Installing Spark on RHEL 8.1

We installed RHEL 8.1 to a single disk volume. During installation, we disabled kdump, enabled the Ethernet port, and changed the hostname to accommodate our environment.

1. After installing RHEL, use the subscription manager to register the operating system, update the software, and install mdadm and vim:

```
subscription-manager register --username * --password * --auto-attach
yum upgrade -y
yum install mdadm vim -y
```

2. Disable the firewall, and disable SELinux:

```
sudo systemctl stop firewalld
sudo systemctl disable firewalld
sudo setenforce 0
#Edit the selinux config file
vi /etc/selinux/config
…
SELINUX = disabled
...
```

3. Prepare each of the four drives you need for the software RAID. We used lsblk to determine which drives to include. Perform the following commands on each individual disk:

```
parted
#Select the target disk
select /dev/nvme*n1
#Clear and create a new partition table.
         mklabel gpt
         #Create new primary partition
         mkpart primary ext4 0 1.5T
```

4. Create the RAID10 using the following commands:

```
#Create a RAID10 from the 4 target NVME drive"'s partitions. List each of the target partitions for
each NVMe
mdadm --create /dev/md3 --level=10 --raid-devices=4 /dev/nvme*n1p1 /dev/nvme*n1p1 /dev/nvme*n1p1 /
dev/nvme*n1p1
#Define filesystem
mkfs.ext4 /dev/md3
#Mount the RAID
mkdir /stor
sudo mount /dev/md3 /stor
#add the disk to fstab so it mounts on reboot
vim /etc/fstab
/dev/md3  /stor  ext4  defaults  0  2
```

5. Download the Java JDK, and install it using the following command:

```
yum install tar wget java-1.8.0-openjdk -y
```

6. Determine and set your JAVA home using the following commands:

```
export JAVA_HOME="/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.242.b08-0.el8_1.x86_64/jre"
#Edit the bash_profile
vi ~/.bash_profile
            ...
            # User specific environment and startup programs
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.242.b08-0.el8_1.x86_64/jre
            PATH=$PATH:$HOME/bin:$JAVA_HOME

            export PATH
```

7. Download the Spark files using the following command:

```
cd /home/
wget http://www.gtlib.gatech.edu/pub/apache/spark/spark-2.4.4/spark-2.4.4-bin-hadoop2.7.tgz
tar -xvf spark-2.4.4-bin-hadoop2.7.tgz
```

8. Navigate to the Spark directory, and start Spark using the following commands:

```
#Start the master server
./sbin/start-master.sh
#Verify that the server is running by navigating to http://[localhost]:8080
#Start the slave server
./sbin/start-slave.sh spark://[local machine IP]:7077
```

9. Download and extract the Spark-Bench package from https://github.com/CODAIT/spark-bench. We downloaded spark-bench_2.3.0_0.4.0-RELEASE_99.tgz, and used SCP to copy it to our target server at /home/.
10. Set up Spark-Bench using the following commands:

```
#Unzip the file using
tar -xvf spark-bench_2.3.0_0.4.0-RELEASE_99.tgz
#create a symbolic link to the spark home directory
ln -s /home/spark-2.4.4-bin-hadoop2.7 /opt/spark
```

11. To set up environment variables for Spark-Bench, add the following lines of code to the end of /root/bashrc:

```
vi /root/.bachrc
export SPARK_HOME=/opt/spark
export PATH=$SPARK_HOME/bin:$PATH
```

12. In the Spark-Bench folder, under examples, create the workload files KMeans_generator.conf and KMeans_run.conf. (We provide the text for these files at the end of this document.)
13. To start the test, run the following commands:

```
cd spark-bench_2.3.0_0.4.0-RELEASE
bin/spark-bench.sh examples/KMeans_generator.conf
bin/spark-bench.sh examples/KMeans_run.conf
```

Workload files

Generating the dataset

We used the following configuration file to generate an 888GB dataset for the Spark-Bench k-means clustering workload. Note that:

- **rows:** The number of rows to generate for the dataset
- **cols:** The number of rows and columns to generate for the dataset
- **k:** The number of clusters the workload generates
- **scaling:** The scaling factor of the dataset
- **partitions:** The number of partitions in the dataset

[sparkbench install]/examples/KMeans_generator.conf

```
spark-bench = {
  spark-home = "/opt/spark"
   spark-submit-config = [{
    spark-args = {
       master = "spark://[localhost]:7077"
     }
   workload-suites = [
      {
        descr = "KMean data generator"
        benchmark-output = "console"
        workloads = [
          {
           name = "data-generation-kmeans"
           rows = 2002597965
           cols = 24
           output = "/stor/kmeans-data.csv"
           k = 2000
           scaling = 1.6
           partitions = 10
          }
        ]
      }
    ]
 }]
}
```

Running the k-means clustering workload

We used the following configuration file to run the Spark-Bench k-means workload. Note that:

- The number of executors is based on the processor's core count.
  For the AMD server, we used 30 executors. For the Intel server, we used 17.
- We assigned a total of 510 GB of memory to the executors for both servers.
- The exec_mem is 510 divided by the number of executors.
  For the AMD server, we used 17GB, and for the Intel server, we used 30GB memory.

[sparkbench install]/examples/KMeans_run.conf

```
spark-bench = {
  spark-home = "/opt/spark"
  spark-submit-config = [{
   spark-args = {
      master = "spark://[localhost]:7077"
      #The number of executors is based on the processor"'s core count.
      #For the AMD server, we used 30 executors.
      #For the Intel server, we used 17.
      num-executors = [executors]
      executor-cores = 4
      #Total memory assigned to executors is 510 GB for both servers. The exec_mem is 510
      divided by the number of executors.
```

```
      #For the AMD server, we used 17g
      #For the Intel server, we used 30g
      executor-memory = [exec_mem]
    }
   workload-suites = [
      {
        descr = "KMean data generator"
        benchmark-output = "console"
        workloads = [
          {
            name = "kmeans"
            input = "/stor/kmeans-data.csv"
            rows = 2002597965
            cols = 24
            scaling = 1.6
            partitions = 10
            output = /home/kmeans/results/results.csv
            k = 4
            maxiterations = 4
          }
        ]
      }
    ]
  }]
}
```

Read the report at http://facts.pt/0jyo64h ▶

This project was commissioned by Dell Technologies.