
Gain more k-means clustering data analysis performance per dollar with 3rd Gen AMD EPYC 75F3 processor-powered Dell EMC PowerEdge R6525 servers
Compared to the same server powered by 2nd Gen AMD EPYC 7542 processors
Analyze data in 28.5% less time*
Handle up to 40% more data per hour*
Get 24.5% better performance for every dollar*†

The science behind the report:

Gain more k-means clustering data analysis performance per dollar with 3rd Gen AMD EPYC 75F3 processor-powered Dell EMC PowerEdge R6525 servers

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 Gain more k-means clustering data analysis performance per dollar with 3rd Gen AMD EPYC 75F3 processor-powered Dell EMC PowerEdge R6525 servers.

We concluded our hands-on testing on April 7, 2021. 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 March 31, 2021 or earlier. Unavoidably, these configurations may not represent the latest versions available when this report appears.

Our results

Table 1: The results of our testing.

	AMD EPYC™ 7542 processor-powered Dell EMC™ PowerEdge™ R6525	AMD EPYC 75F3 processor-powered Dell EMC PowerEdge R6525
Time to complete the Spark-Bench k-means workload on an 811GB dataset (hours)	2.8	2.0
Percentage less time	-	28.57%
Processing rate (MB/hour)	289,642	405,500
Percentage higher rate	-	40.00%
Hardware and support price (USD)	$57,510.01	$64,650.01
Performance per dollar (MB/hour per dollar)	5.036	6.272
Percentage higher performance per dollar	-	24.53%

CPU utilization charts

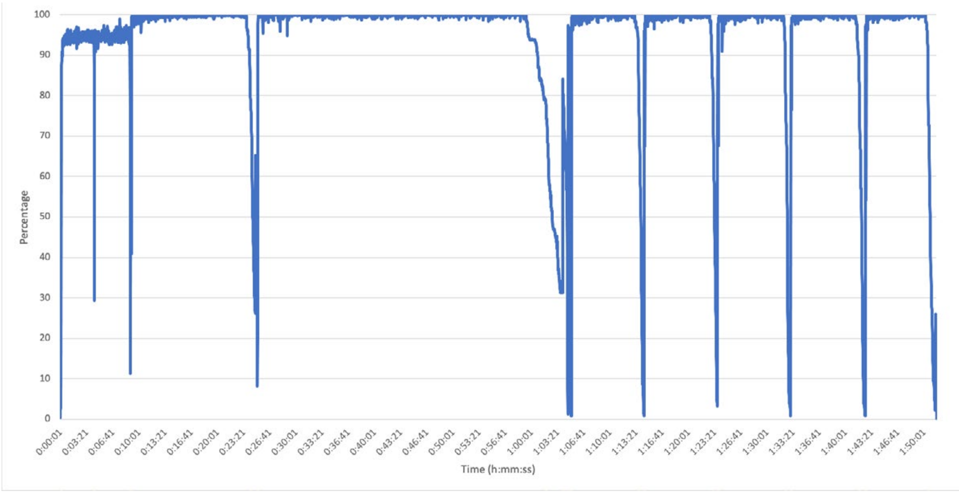

Figure 1: CPU utilization for the Dell EMC PowerEdge R6525 server with AMD EPYC 75F3 processors for the duration of the k-means clustering workload.
Source: Principled Technologies.

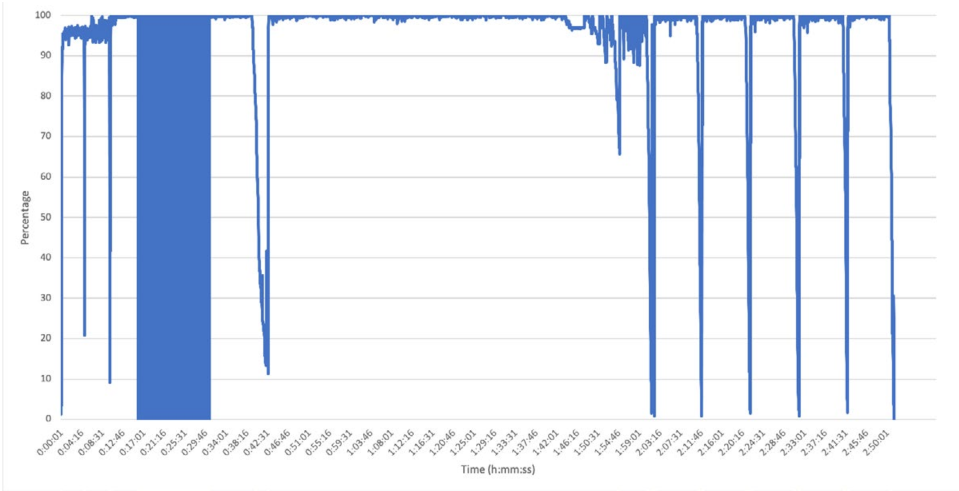

Figure 2: CPU utilization for the Dell EMC PowerEdge R6525 server with AMD EPYC 7542 processors for the duration of the k-means clustering workload.
Source: Principled Technologies.

System configuration information

Table 2: Detailed information on the system we tested.

System configuration information	Dell EMC PowerEdge R6525	
BIOS name and version	Dell 2.0.3	
Non-default BIOS settings	N/A	
Operating system name and version/build number	RHEL 8.3 (Kernel 4.18.0-240.15.1.el8_3.x86_64)	
Date of last OS updates/patches applied	4/2/21	
Power management policy	Performance	
Processor	3rd Gen	2nd Gen
Number of processors	2	2
Vendor and model	AMD EPYC 75F3	AMD EPYC 7542
Core count (per processor)	32	32
Core frequency (GHz)	2.95	2.90
Memory module(s)		
Total memory in system (GB)	1,024	
Number of memory modules	16	
Vendor and model	Hynix HMAA8GR7AJR4N-XN	
Size (GB)	64	
Type	PC4-3200	
Speed (MHz)	3,200	
Speed running in the server (MHz)	3,200	
Storage controller 1		
Vendor and model	Dell BOSS-S1 Adapter	
Cache size	0	
Firmware version	2.5.13.3024	
Storage controller 2		
Vendor and model	Dell PERC S150 Controller	
Cache size	0	
Firmware version	6.0.3-0005	
Local storage (OS)		
Number of drives	2	
Drive vendor and model	Intel® SSDSCKKB480G8R	
Drive size (GB)	480	
Drive information	6Gbps SATA M.2 SSD	

System configuration information	Dell EMC PowerEdge R6525
Local storage (data)	
Number of drives	4
Drive vendor and model	Samsung® MZ-WLJ1T90
Drive size (GB)	1,920
Drive information	NVMe™ PCIe® Gen4 SSD
Network adapter	
Vendor and model	Broadcom® BCM5720
Number and type of ports	2x 1Gb Ethernet Adapter
Firmware version	21.60.2
Cooling fans	
Vendor and model	Foxconn PIH040M12P
Number of cooling fans	6
Power supplies	
Vendor and model	Dell L1400E-S0
Number of power supplies	2
Wattage of each (W)	1,400

How we tested

We assessed which of two systems would take less time complete a k-means algorithm from the Spark-Bench benchmark suite. The solutions we tested are as follows:

- Dell EMC PowerEdge R6525 powered by AMD EPYC 75F3 processors
- Dell EMC PowerEdge R6525 powered by AMD EPYC 7542 processors

We installed Red Hat® Enterprise Linux® 8.3 (RHEL 8.3) on each solution and ran a k-means clustering algorithm from the Spark-Bench benchmark suite on an 811GB dataset. We hosted this dataset on a Linux software RAID10 we built from four 1.92TB PCIe Gen4 NVMe drives.

Installing Spark on RHEL 8.3

We installed RHEL 8.3. 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:

```
#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:

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

6. Determine and set your JAVA home:

```
export JAVA_HOME="/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.282.b08-2.el8_3.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-1.8.0.282.b08-2.el8_3.x86_64/jre PATH=$PATH:$HOME/
bin:$JAVA_HOME
export PATH
```

7. Download the Spark files:

```
cd /home/
wget http://www.gtlib.gatech.edu/pub/apache/spark/spark-2.4.7/spark-2.4.7-bin-hadoop2.7.tgz
tar -xvf spark-2.4.7-bin-hadoop2.7.tgz
```

8. Navigate to the Spark directory, and start Spark:

```
#Start the controller server
./sbin/start-master.sh
#Verify that the server is running by navigating to http://[localhost]:8080
#Start the worker 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:

```
#Unzip the file using
tar -xvfz 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 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. Start the test:

```
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 811GB 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://hspark:7077"
     }
   workload-suites = [
      {
        descr = "KMean data generator"
        benchmark-output = "console"
        workloads = [
          {
           name = "data-generation-kmeans"
           rows = 450000000
           cols = 99
           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. We used 31 executors.
- We assigned a total of 992 GB of memory to the executors for both servers.
- The exec_mem is 992 divided by the number of executors. We used 32GB.

[sparkbench install]/examples/KMeans_run.conf

```
spark-bench = {
 spark-home = "/opt/spark"
  spark-submit-config = [{
   spark-args = {
      master = "spark://hspark:7077"
      num-executors = 31
      executor-cores = 4
      executor-memory = 32g
    }
   workload-suites = [
      {
        descr = "KMean data generator"
        benchmark-output = "console"
        workloads = [
          {
            name = "kmeans"
            input = "/stor/kmeans-data.csv"
            rows = 450000000
            cols = 99
            scaling = 1.6
            partitions = 10
            output = "/home/kmeans/results/results.csv"
            k = 1200
            maxiterations = 4
          }
        ]
      }
    ]
  }]
}
```

Read the report at http://facts.pt/RRQ3nvZ ▶

This project was commissioned by Dell Technologies.