Deploying Apache Spark and testing big data applications on servers powered by the AMD EPYC 7601 processor

Processed a 210GB Bayesian classification problem in

3m 22s

Processed a 224GB Kmeans dataset in

9m 13s

Counted the words in a 1,530GB database in

5m 47s

Your company has access to troves of data on your customers and how they interact with your services. To take advantage of that raw data and turn it into meaningful returns for your business requires the right big data hardware and software solution. You may have begun looking for servers capable of running intense big data software solutions, such as Apache Spark™.

Earlier this year, the AMD EPYC™ series of server processors entered the market. We conducted a proof-of-concept study using one of these servers with Apache Spark running the HiBench big data benchmarking tool. In this document, we lead you through the process of setting up such a solution, share the results of our HiBench testing, and look at the AMD Zen architecture.

Apache Spark helps you reap the value of big data

As the IT marketplace, and especially the cloud sector, continues to evolve rapidly, the volume of data in existence has exploded and will continue to expand for the foreseeable future. Alongside traditional devices and databases, a host of new Internet of things (IoT) platforms, mobile technologies, and apps are also generating information. In fact, the International Data Corporation (IDC) predicts "digital data will grow at a compound annual growth rate (CAGR) of 42% through 2020." Put differently, the world's digital data "will grow from about 1ZB in 2010 to about 50ZB in 2020."[2]

Alongside this data growth, increasing disk, networking, and compute speeds for servers have converged to enable hardware technologies that can quickly process much larger amounts of data than was previously possible. A variety of software tools have emerged to help analysts make sense of this big data. Among these is Apache Spark, a distributed, massively parallelized data processing engine that data scientists can use to query and analyze large amounts of data.

While Apache Spark is often paired with traditional Hadoop® components, such as HDFS for file system storage, it performs its real work in memory, which shortens analysis time and accelerates value for customers. Companies across the industry now use Apache Spark in applications ranging from real-time monitoring and analytics to consumer-focused recommendations on ecommerce sites.

In this study, we used Apache Spark with AMD EPYC-powered servers to demonstrate data analysis performance using several subtests in the industry-standard HiBench tool.

The configuration we used for this proof-of-concept study

Hardware

We set up a Hortonworks Data Platform (HDP) cluster for Apache Spark using six servers. We set up one server as an infrastructure or utility server, two servers as name nodes, and the remaining three servers (configured as data nodes) as the systems under test. We networked all systems together using 25GbE LinkX™ cables, connecting to a Mellanox® SN2700 Open Ethernet switch.

The infrastructure server ran VMware® vSphere® and hosted the Ambari host VM and the Apache Spark client driver VM. We used a pair of two-socket AMD servers for name nodes. Each of these three supporting servers had one 25GbE connection to the network switch via a Mellanox ConnectX®-4 Lx NIC.

About the Mellanox SN2700 Open Ethernet switch

The Mellanox SN2700 100GbE switch we used is a Spectrum-based, 32-port, ONIE (Open Network Install Environment)-based platform on which you can mount a variety of operating systems. According to Mellanox, the switch lets you use 25, 40, 50 and 100GbE in large scale without changing power infrastructure facilities.

Learn more at http://www.mellanox.com/related-docs/prod_eth_switches/PB_SN2700.pdf

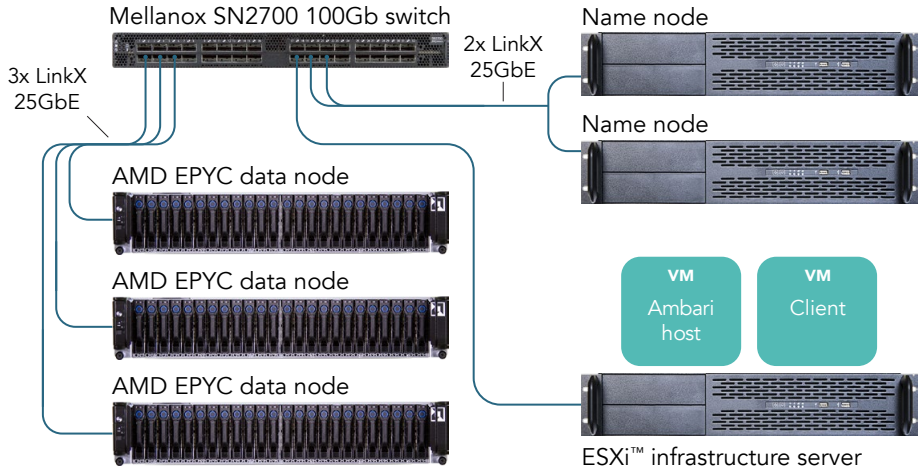
Our three systems under test (the data nodes in the cluster) were dual-socket AMD EPYC processor-powered servers. They each had two AMD EPYC 7601 processors, 512 GB of DDR4-2400 RAM, and 24 Samsung® PM863 solid-state drives. We connected each server to the network switch with one 25GbE connection via a Mellanox ConnectX-4 Lx NIC. We tuned the Mellanox cards using the high-throughput setting. We left BIOS settings as default, and we placed all disks in HBA mode.

About Samsung PM863a solid-state drives

Samsung PM863a 2.5-inch SATA SSDs leverage the company's 3D (V-NAND) technology to provide storage capacities of up to 3.84 TB without increasing the physical footprint of the drive. Note that the server we tested used the earlier version, the PM863.

Learn more at http://www.samsung.com/semiconductor/minisite/ssd/product/enterprise/pm863a.html

The following diagram shows the physical setup of our testbed:



Software

Operating system and prerequisites

We installed Red Hat® Enterprise Linux 7.3 on our infrastructure, name node, and data node systems, and updated the software through June 15, 2017. We disabled `selinux` and the firewalls on all servers, and set `tuned` to network-latency, high performance with a focus on low network latency. We also installed the Mellanox OFED 4.0-2.0.0.1-rhel7.3 driver. Finally, we installed OpenJDK 1.8.0 on all hosts.

Assembling the solution

We configured Red Hat Enterprise Linux to run HDP and HiBench. We set up passwordless SSH among all hosts in the testbed and configured each server's NTP client to use a common NTP server. Additionally, we configured a hosts file for all hosts in the testbed and copied it to all systems. On the Ambari VM, we installed Ambari server and used it to distribute HDP across all servers. On the client host, we installed HiBench. Finally, we ran the HiBench tests on the cluster.

Using HiBench to examine big data capabilities

Basic overview

HiBench is a benchmark suite that helps evaluate big data frameworks in terms of speed, throughput, and system resource utilization. It includes Hadoop, Apache Spark, and streaming workloads.[3] For our tests in this study, we ran the following three tests:

- Bayes (big data dataset)
- Kmeans (big data dataset)
- Wordcount (big data dataset)

Running the workload

While HiBench allows users to select which of its multiple workloads to run, we used a general format that runs any of them. We executed the tests against the cluster from the client system. We ran our tests using the following general process:

1. Clear the PageCache, dentries, and inodes on all systems under test.
2. Navigate to the HiBench directory for the current test (e.g., `~/HiBench/bin/workloads/kmeans`).
3. Run the prepare script to initialize the data used in the test.
4. Run the Apache Spark run script to perform the test (in Kmeans, that would be to sort the data we created using the Kmeans grouping algorithm).

Tuning information

On the data nodes, we found the following HiBench settings yielded the best results for the Apache Spark workloads we tested:

- hibench.conf
  - hibench.default.map.parallelism 1080
  - hibench.default.shuffle.parallelism 1080
- spark.conf
  - hibench.yarn.executor.num 90
  - hibench.yarn.executor.cores 4
  - spark.executor.memory 13g
  - spark.driver.memory 8g
  - hibench.streambench.spark.storageLevel 0

Sizing information

During testing, we observed that AMD EPYC processor-powered servers operated at their peak when the workload neared or exceeded the memory in the server. The large number of memory channels, combined with the SSDs in the system directly controlled by the processors, can potentially help the EPYC processors perform well in high-memory-usage situations.

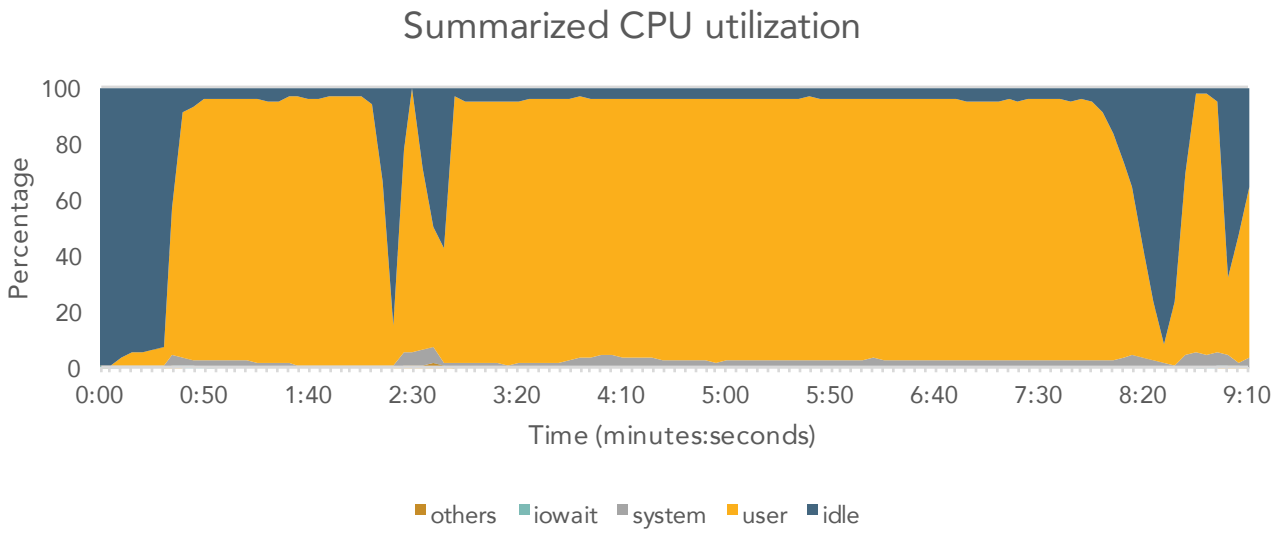
Results from testing

We ran three of the more popular HiBench Spark tests with the settings we mentioned previously. In the table below, we summarize the time and throughput results for each test. We then discuss each set of results in greater detail.

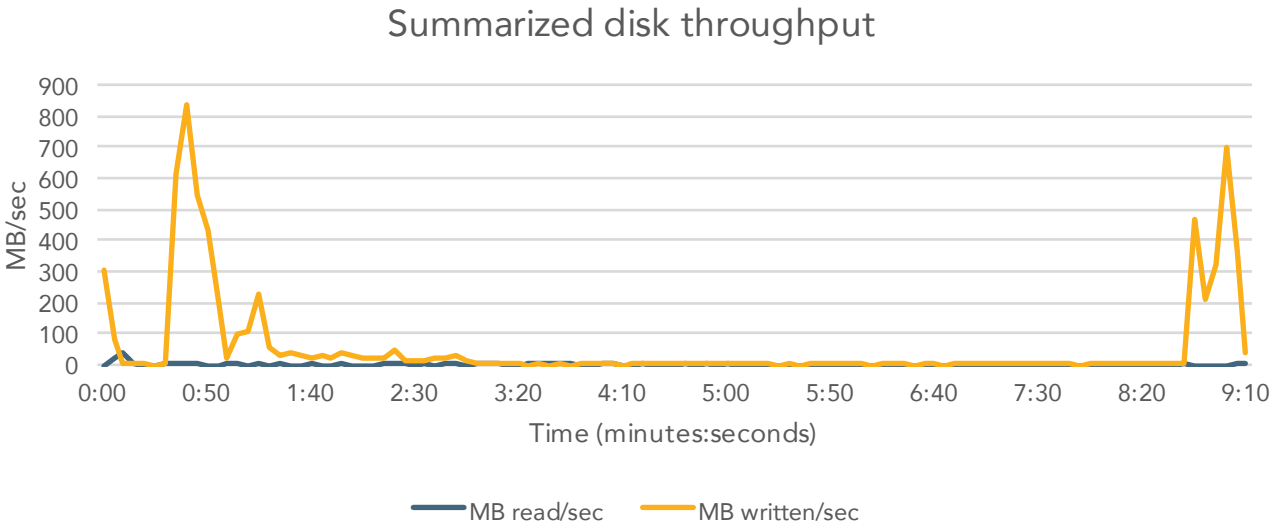
Benchmark	Time to complete (seconds)	Throughput (bytes per second)
Bayes	552.873	408.702
Kmeans	207.245	1,162.787
Wordcount	346.843	4,735.449

Bayes

The Bayesian classification test, which HiBench refers to as Bayes, uses the Naïve Bayesian trainer to classify data. Like many Apache Spark workloads, it has one large disk-usage spike in the beginning and another at the end, but primarily taxes a server's CPU. It is also RAM intensive, which means that as database size increases, the large number of AMD EPYC memory channels comes into play. The graph below, from the HiBench report, shows the CPU usage during the test.

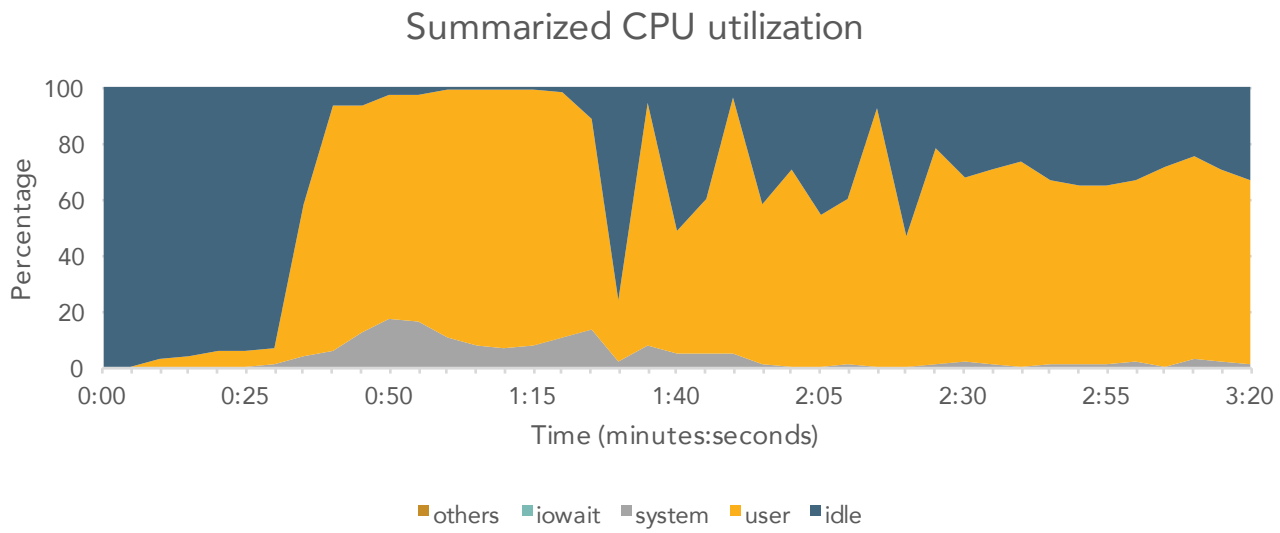


The graph below shows the disk usage during the test.

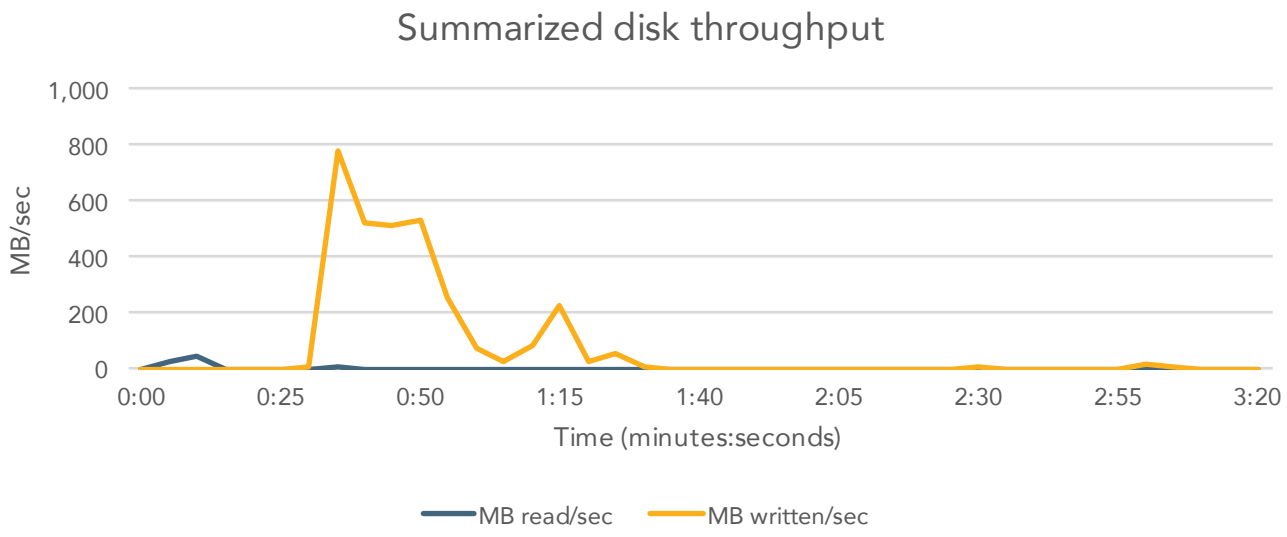


Kmeans

Many companies use the K-means clustering algorithm, which HiBench refers to as Kmeans, to help separate data into useful categories. Companies can use this algorithm to help segment the market for targeted advertising, or to generate product suggestions on websites based on a user's habits. Like the Bayes test, it is very CPU intensive. Unlike Bayes, after performing the initial analysis, Kmeans runs subsequent, smaller checks to eventually arrive at its conclusion, as the graph below shows.

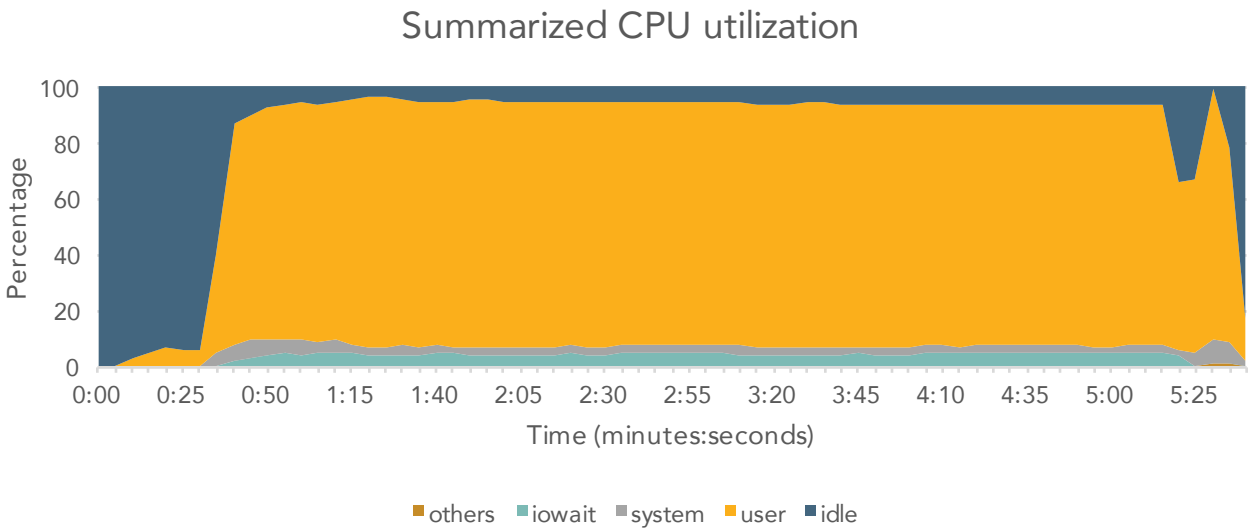


Kmeans has even less disk activity than Bayes, involving only a very large spike in the beginning as the workload distributes itself among the nodes. The graph below illustrates this.

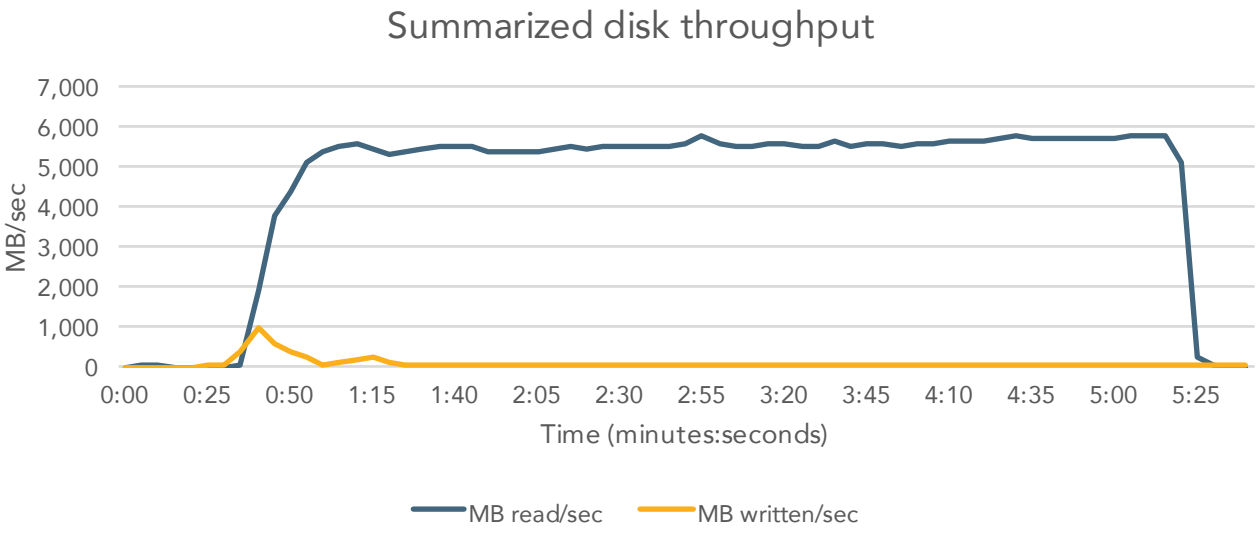


Wordcount

Wordcount is the simplest HiBench test we ran. One real-world application of Wordcount is to create word clouds. These are a way of visually representing the number of times a given body of text, such as a website or set of documents, uses terms. The more frequently a term occurs in the text, the larger and more prominently the term appears in the word cloud. The Wordcount benchmark counts the number of times each word appears in a randomized dataset. While Wordcount remains CPU intensive, when the database size exceeds the RAM on the systems, the number of memory channels and the storage bandwidth of the AMD systems can come into play. The chart below shows CPU utilization during the Wordcount run.



As the chart below shows, while the Wordcount benchmark runs, a large amount of read I/O occurs in the system.



AMD EPYC architecture overview

The AMD EPYC processor has up to up to 32 cores, 64 threads, and can run with 2 TB of DDR4 memory capacity spread across eight channels. This high core density contributed to the ability to process data quickly in Apache Spark.

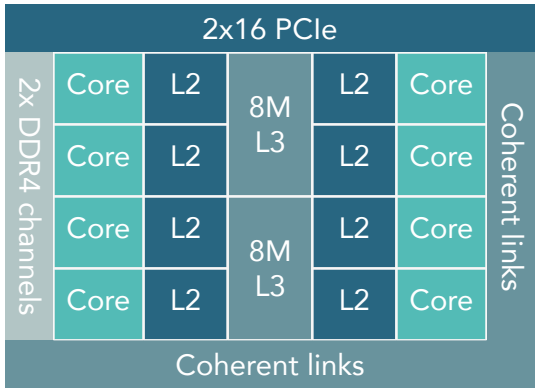
To better understand this high core density, let's look at the components that go into it. Note that the information in this section comes from publicly available materials from the AMD website.

Building blocks of the EPYC 7601

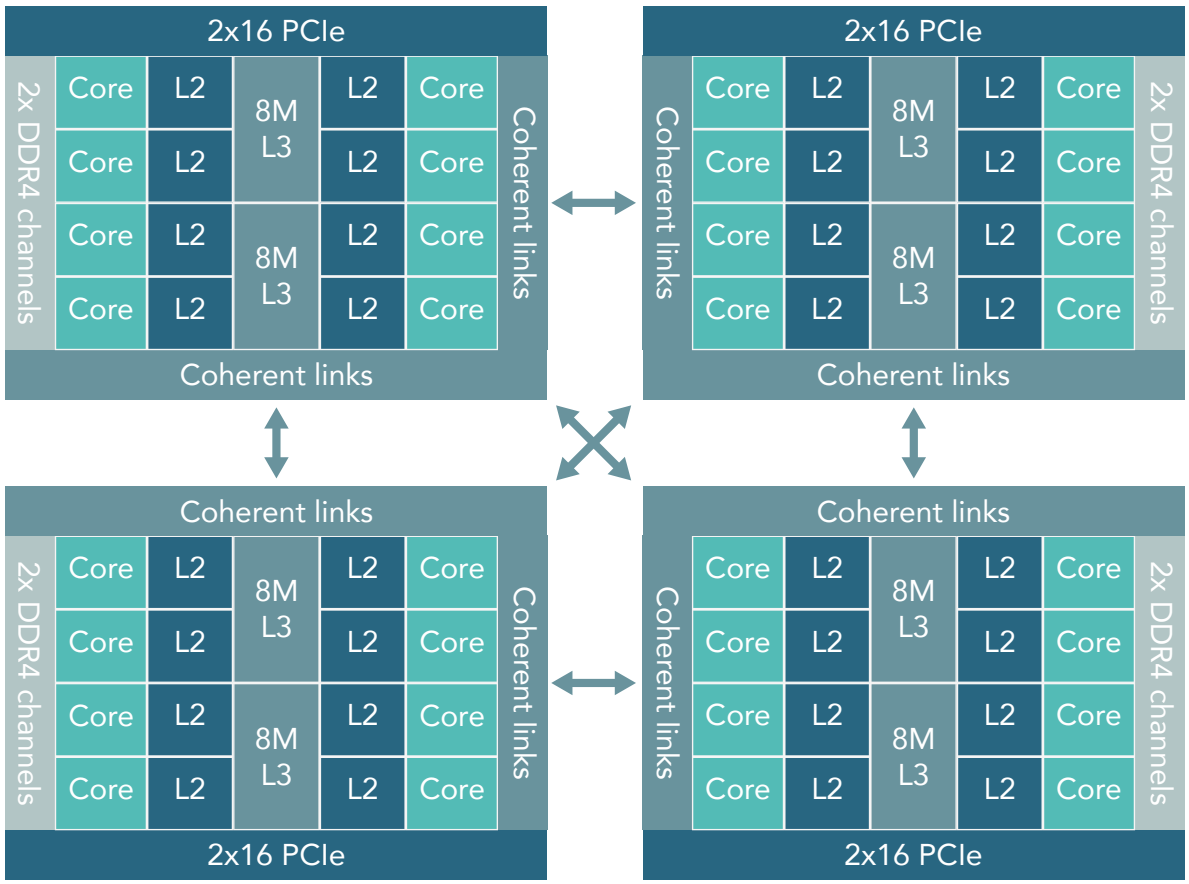
The primary building block is the Zen core. Each Zen core has a 64KB L1 instruction cache and a 32KB L2 data cache, as well as access to a 512KB instruction and data (I+D) L2 cache and a shared L3 cache.

Four Zen cores, along with 8 MB of L3 caching, make up a single CPU Complex (CCX).

A Zeppelin die comprises two CCXs, meaning that each Zeppelin die has eight Zen cores.



On a 7000-series processor, four Zeppelin dies connect to each other with AMD Infinity Fabric (IF) interconnect to form the AMD EPYC processor. As we show below, one EPYC processor comprises eight CCXes, 32 cores, and 64 threads.

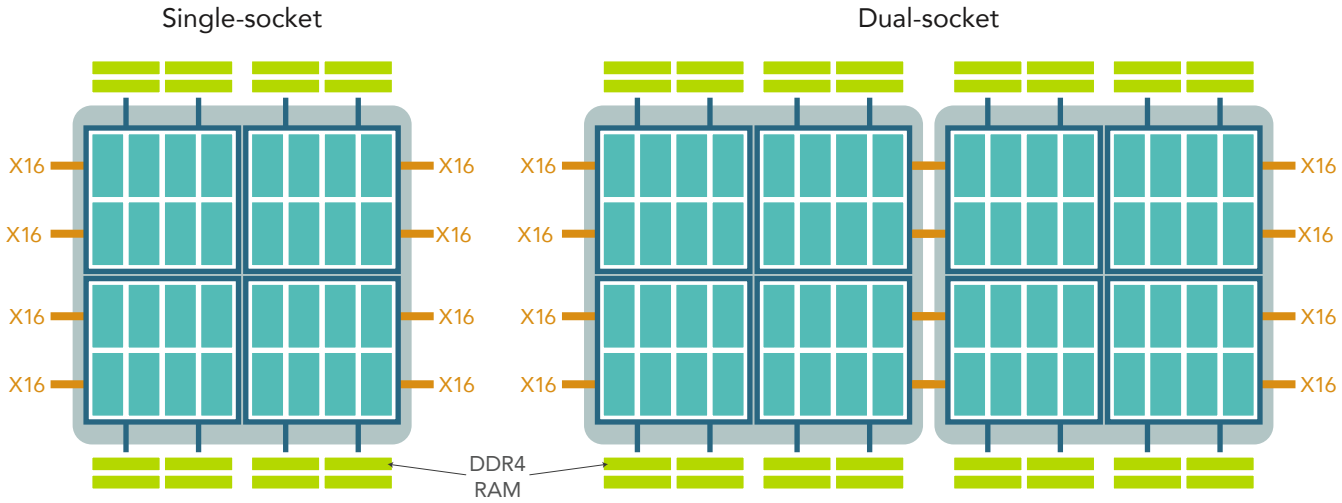


Single socket

In a single-socket configuration, the EPYC chip has eight DDR4 channels and 128 PCIe lanes.

Dual socket

In a dual-socket configuration, half of the PCIe lanes are for the IF between both sockets. This configuration has 16 DDR4 channels and 128 PCIe lanes.



Performance and power management

EPYC chips have several features for performance and power management. Performance Determinism sets consistent performance across all CPUs, though the power draw of each processor may vary slightly with manufacturing variance. Power Determinism sets consistent power usage across all processors, though the performance of each processor may also vary. Being able to manage power usage can be useful in datacenters where it is necessary to control the power draw of servers.

Security and encrypted memory

Though we did not test them in our datacenter, the EPYC family of processors contains several security features:

- AMD Secure Root-of-Trust technology prevents booting any software that is not cryptographically signed.
- AMD Secure Run Technology encrypts software and data in memory.
- AMD Secure Move technology lets you securely migrate virtual machines and containers between EPYC processor-based servers.

The EPYC family supports encrypted memory features:

- Transparent mode encrypts all memory pages. You can enable transparent mode in the BIOS without having to change the OS or individual applications.
- In Secure Encrypted Virtualization (SEV) mode, you can encrypt specific virtual machines without having to change individual applications.

Memory capacity

Each AMD EPYC processor supports 2 TB of memory, equivalent to 4 TB per two-socket server. The closest competitor supports 1.5 TB of memory per processor, equivalent to 3 TB per two-socket server. This means that supporting 12 TB of memory would require three AMD EPYC processor-powered two-socket servers and four of the competitor's two-socket servers.

Conclusion

To get the most out of the heaps of data your company is sitting on, you'll need a platform such as Apache Spark to sort through the noise and get meaningful conclusions you can use to improve your services. If you need to get results from such an intense workload in a reasonable amount of time, your company should invest in a solution whose power matches your level of work.

This proof of concept has introduced you to a new solution based on the AMD EPYC line of processors. Based on the new Zen architecture, the AMD EPYC line of processors offers resources and features worth considering. In the Principled Technologies datacenter, we set up a big data solution consisting of whitebox servers powered by the AMD EPYC 7601—the top-of-the-line offering from AMD. We ran an Apache Spark workload and tested the solution with three components of the HiBench benchmarking suite. The AMD systems maintained a consistent level of performance across these tests.

---

1 Core count per CCX decreases for lower-bin processors in the EPYC line.

2 Source: "Digital Data Storage is Undergoing Mind-Boggling Growth," accessed July 25, 2017, http://www.eetimes.com/author.asp?section_id=36&doc_id=1330462

3 Source: "HiBench benchmark," accessed July 26, 2017, https://github.com/intel-hadoop/HiBench

On July 5, 2017, we finalized the hardware and software configurations we tested. Updates for current and recently released hardware and software appear often, so unavoidably these configurations may not represent the latest versions available when this report appears. For older systems, we chose configurations representative of typical purchases of those systems. We concluded hands-on testing on July 25, 2017.

Appendix A: System configuration information

Server configuration information	
BIOS name and version	AMD WSW7802N_PLAT-
Non-default BIOS settings	NA
Operating system name and version/build number	Red Hat Enterprise Server 7.3 / 3.10.0-514.16.1.el7.x86_64
Date of last OS updates/patches applied	06/15/2017
Power management policy	High Performance
Processor	
Number of processors	2
Vendor and model	AMD EPYC 7601
Core count (per processor)	32
Core frequency (GHz)	2.20
Stepping	1
Memory module(s)	
Total memory in system (GB)	512
Number of memory modules	32
Vendor and model	Samsung M393A2K40BB1-CRC0Q
Size (GB)	16
Type	PC4-2400T
Speed (MHz)	19200
Speed running in the server (MHz)	19200
Storage controller	
Vendor and model	AMD FCH SATA Controller
Driver version	3.0
Local storage	
Number of drives	24
Drive vendor and model	Samsung PM863
Drive size (GB)	120
Drive information (speed, interface, type)	6Gb, SATA, SSD

Server configuration information	
Network adapter	
Vendor and model	Mellanox ConnectX-4 Lx
Number and type of ports	2 x 25GbE
Driver version	4.0-2.0.0.1
Cooling fans	
Vendor and model	Delta Electronics GFC0812DS
Number of cooling fans	4
Power supplies	
Vendor and model	LITEON PS-2112-5Q
Number of power supplies	2
Wattage of each (W)	1200

Appendix B: How we tested

We performed the following steps to install and configure the testbed for Hortonworks Data Platform and HiBench. Your environment may differ slightly, so keep that in mind when setting IPs and naming hosts.

Installing the operating system

1. Insert the Red Hat Enterprise Linux 7.3 installation media, and power on the system.
2. Select Install Red Hat Enterprise Linux 7.3, and press Enter.
3. Select English, and click Continue.
4. Click NETWORK & HOST NAME.
5. Click ON for the 25Gb Ethernet connection you are using.
6. Select the 25Gb Ethernet connection you are using, and click Configure.
7. Click IPv4 Settings.
8. Change Method to Manual.
9. Click Add, and type your IP address, netmask, and gateway.
10. Click Save.
11. Change your host name to an appropriate host name based on the role of your system, then click Apply.
12. Click Done to exit the network configuration.
13. Click INSTALLATION DESTINATION.
14. Select I will configure partitioning, and click Done.
15. Select Standard Partition, and then click Click here to create them automatically.
16. Modify the swap to 4 GB, then click Done.
17. When the summary of changes appears, click Accept Changes.
18. Click Begin Installation.
19. Click ROOT PASSWORD.
20. Type the root password, and confirm it. Click Done.
21. When the installation finishes, click Reboot.
22. Repeat steps 1–21 for the remaining servers in the test bed.

Configuring the operating system

1. Log in to the Red Hat console.
2. Disable selinux by typing the following commands:

```
setenforce 0
sed -i 's/SELINUX=.*/SELINUX=disabled/' /etc/selinux/config
```

3. Disable the firewall by typing the following commands:

```
systemctl stop firewalld
systemctl disable firewalld
```

4. Configure default file permissions by typing the following command:

```
echo umask 0022 >> /etc/profile
```

5. Either register your system with Red Hat or configure your yum patch repositories to operate off of a locally accessible repository.
6. Remove chrony from your system by typing the following command:

```
yum remove -y chrony
```

7. Update the system by typing the following command:

```
yum update -y
```

8. Reboot the system.
9. Install the prerequisites for Ambari and HDP by typing the following command:

```
yum install -y ntp time xfsprogs tuned wget vim nfs-utils openssh-clients man zip unzip numactl
sysstat bc lzop xz libhugetlbfs python numpy blas64 lapack64 gtk2 atk cairo gcc-gfortran tcsh lsof
tcl tk java-1.8.0-openjdk-devel perl-Data-Dumper.x86_64
```

10. Enable NTP by typing the following commands:

```
sed -i '/^server [^ ]* iburst/d' /etc/ntp.conf
echo "server [your NTP server IP address] iburst" >> /etc/ntp.conf
systemctl enable ntpd
systemctl start ntpd
```

11. Reboot the system.
12. Download the latest version of the Mellanox OFED from the Mellanox website.

13. Install the Mellanox OFED by typing the following commands:

```
tar -xf MLNX_OFED_LINUX-4.0-2.0.0.1-rhel7.3-x86_64.tgz
cd MLNX_OFED_LINUX-4.0-2.0.0.1-rhel7.3-x86_64
./mlnxofedinstall --all --force
```

14. Clear the SSH settings by typing the following commands:

```
mkdir -p /root/.ssh
chmod 700 /root/.ssh
cd /root/.ssh
mv * *.orig
```

15. Create an SSH private key for all hosts by typing the following commands:

```
ssh-keygen -t rsa -q
cp id_rsa.pub authorized_keys
echo "StrictHostKeyChecking=no" > config
```

16. Copy the key to the other hosts.
17. Create a hosts file containing a FQDN, nickname, and IP address of every host you plan to use in the cluster. Copy the file out to all of your hosts.
18. On the hosts that will be used as data nodes, run the following script to format and prepare the drives you are using for HDFS, making sure to modify the SKIP_LETTER variable for the drive your operating system is running on:

```
#!/bin/bash
FS=xfs
COUNT=0
SKIP_LETTER="w"
umount /grid/*
rmdir /grid/*
rmdir /grid
mkdir -p /grid
for i in {a..x};
do
  if [ "$SKIP_LETTER" = "$i" ]; then
    continue
  fi
  DEV=/dev/sd$i
  GRID=/grid/$COUNT
  echo $DEV $COUNT
  mkdir -p $GRID
  #dd if=/dev/zero of=$DEV bs=1M count=10 oflag=direct
  sync
  mkfs.${FS} $DEV
  echo -e "`blkid -p $DEV | awk '{print $2}'`\t${GRID} \t${FS}\tdefaults,noatime,nodiratime,nofail,x-
systemd.device-timeout=60\t0 0" >> /etc/fstab
  ((COUNT++))
done
```

19. On the data nodes, run `vim /etc/default/grub` and modify the `GRUB_CMDLINE_LINUX` setting by adding `iommu=pt` to the end of the line.
20. Configure the data nodes to run in high throughput performance mode by typing the following command:

```
tuned-adm profile throughput-performance
```

21. Configure the data nodes to run the Mellanox cards in high performance mode by typing the following command (you will need to repeat this each time you reboot):

```
mlnx_tune -p HIGH_THROUGHPUT` Installing Ambari
```

22. Log into your Ambari host.
23. Add the Ambari repository to your yum repos by typing the following command:

```
wget -nv http://public-repo-1.hortonworks.com/ambari/centos7/2.x/updates/2.5.0.3/ambari.repo -O /etc/
yum.repos.d/ambari.repo
```

24. Navigate to your OpenJDK directory and start the Ambari server setup by typing the following command:

    ```
    ambari-server setup
    ```

25. To accept the default of not customizing a user account for the ambari-server daemon, press Enter.
26. When prompted, choose a custom JDK by typing `3` and pressing Enter.
27. To accept the default of not entering advanced database configuration, press Enter.
28. Type `ambari-server start` to start up the Ambari host.

Installing Hortonworks Data Platform on the cluster

1. Open a web browser, and navigate to the Ambari host's website.
2. Log into the Ambari host website.
3. In Welcome to Apache Ambari, click Launch Install Wizard.
4. In the Get Started page, name your cluster, and click Next.
5. In Select Version, select the latest version, check Use Public Repository, and click Next.
6. In Install Options, enter the hosts you plan to use. For example:

    ```
    namenode[1-2].hdp.local
    amd[1-3].hdp.local
    client1.hdp.local
    ```

7. Copy the contents of the SSH private key you created previously into the host registration information. Write root as the User Account and 22 as the port number.
8. Click Register and Confirm.
9. When the Host name pattern expressions window appears, verify that all of your hosts are there, and click OK.
10. In the Confirm Hosts window, verify that all of the hosts you intend to install on are there, and click Next.
11. In Choose Services, uncheck the following services:

    - Accumulo
    - Atlas
    - Falcon
    - Flume
    - HBase
    - Sqoop
    - Oozie

    Leave the rest of the services at their defaults, and click Next.
12. In Assign Masters, assign services to the following master servers:

    - namenode1
        - NameNode
        - ZooKeeper Server
        - DRPC Server
        - Storm UI Server
        - Nimbus
        - Infra Solr Instance
        - Metrics Collector
        - Grafana
        - Kafka Broker
        - Knox Gateway
        - HST Server
        - Activity Explorer
        - Activity Analyzer
        - Spark History Server
        - Spark2 History Server
    - namenode2
        - SNameNode
        - App Timeline Server
        - ResourceManager
        - History Server
        - WebHCat Server

- Hive Metastore
- HiveServer2
- ZooKeeper Server

13. In Assign Slaves and Clients, apply the slave and client components to the following hosts:
    - amd1-3
      - DataNode
      - NodeManager
      - Supervisor
    - vclient1
      - Client
14. In Customize Services, perform the following actions:
    a. Enter a database password for the Hive Metastore.
    b. Enter a password for the Grafana Admin.
    c. Enter a password for the Knox Master Secret.
    d. Enter a password for the admin in SmartSense.
    e. In the HDFS section, add the following line to the DataNode directories textbox:

```
/grid/0,/grid/1,/grid/2,/grid/3,/grid/4,/grid/5,/grid/6,/grid/7,/grid/8,/grid/9,/grid/10,/
grid/11,/grid/12,/grid/13,/grid/14,/grid/15,/grid/16,/grid/17,/grid/18,/grid/19,/grid/20,/
grid/21,/grid/22
```

15. When warned in the Configurations window, click Proceed Anyway.
16. In Review, verify your settings and click Deploy.
17. In Install, Start and Test, click Next when the deployment completes.
18. In Summary, click Complete.

Installing HiBench

1. Log into your client host.
2. Install the prerequisite packages to the client by typing the following commands:

```
yum install -y maven git vim numpy blas64 lapack64
echo "/usr/hdp/current/hadoop-client/lib/native" > /etc/ld.so.conf.d/hadoop-client-native.conf
```

3. Change to the HDFS user, create the HiBench directory in the dfs, and set permissions by typing the following commands:

```
su - hdfs
hdfs dfs -mkdir /HiBench
hdfs dfs -chown -R root:hadoop /HiBench
hdfs dfs -mkdir /user/root
hdfs dfs -chown root /user/root
exit
```

4. Edit the bash profile to automatically include the HiBench prerequisites by typing the following commands:

```
vim ~/.bash_profile
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk
export HADOOP_HOME=/usr/hdp/current/hadoop-client
export SPARK_HOME=/usr/hdp/current/spark2-client
export KAFKA_HOME=/usr/hdp/current/kafka-broker
export LD_LIBRARY_PATH=/usr/hdp/current/hadoop-client/lib/native
```

5. Download the HiBench benchmark from GitHub by typing the following commands:

```
cd /root
```

```
git clone https://github.com/intel-hadoop/HiBench.git
```

6. Change to the newly downloaded HiBench directory and type the following command:

```
mvn -Dspark=2.1 -Dscala=2.11 clean package | tee hibench_build.log
```

7. Modify the `99-user_defined_properties.conf` file by changing the following lines:

```
hibench.hadoop.home                                 /usr/hdp/current/hadoop-client
hibench.spark.home                                 /usr/hdp/current/spark2-client
hibench.hadoop.mapreduce.home        /usr/hdp/current/hadoop-mapreduce-client
```

Configuring Hortonworks Data Platform

We made the following changes to the settings to the services via the Ambari console:

- HDFS
  - NameNode
    - NameNode Java heap size - 5.25 GB
    - NameNode Server threads - 1800
  - DataNode
    - DataNode directories - each of the drives on the server
    - DataNode failed disk tolerance - 2
  - Advanced
    - NameNode new generation size - 672 MB
    - NameNode maximum new generation size - 672 MB
- YARN
  - Memory
    - Memory allocated for all YARN containers on a node - 480 GB
    - Maximum Container Size (Memory) - 491520 MB
  - CPU
    - Percentage of physical CPU allocated for all containers on a node - 100%
    - Number of virtual cores - 128
    - Maximum Container Size (VCores) - 16

Configuring HiBench

We used the following settings across the tests we ran:

hibench.conf	
hibench.default.map.parallelism	1080
hibench.default.shuffle.parallelism	1080
hibench.scale.profile	bigdata
spark.conf	
hibench.yarn.executor.num	90
hibench.yarn.executor.cores	4
spark.executor.memory	13g
spark.driver.memory	8g
hibench.streambench.spark.storageLevel	0

We modified the Bayes test to better take advantage of the memory in the AMD servers by increasing the size of the workload to 60 million pages and 60,000 classes.

hibench.bayes.bigdata.pages	60000000
hibench.bayes.bigdata.classes	60000

Running HiBench

1. Log into the first data node.
2. Clear the PageCache, dentries, and inodes by typing the following command:
   ```
   echo 3 > /proc/sys/vm/drop_caches
   ```
3. Repeat steps 1–2 for the remaining data nodes.
4. Log into the client host.
5. Navigate to the bin/workload/[benchmark] directory in the HiBench directory.
6. Initialize the data by typing the following command:
   ```
   ./prepare/prepare.sh
   ```
7. After initializing the data, wait 5 minutes.
8. Run the benchmark by typing the following command:
   ```
   ./spark/run.sh
   ```
9. When the test completes, record the results.

This project was commissioned by AMD.