Big Data Performance on VMware Cloud on AWS

Spark Machine Learning and IoT Analytics Performance On-premises and in the Cloud

Performance Study - August 16, 2018

Table of Contents

Executive Summary

Both traditional on-premises virtualized private clouds and the new VMware Cloud on AWS provide users the benefits of server consolidation, manageability, and flexibility that VMware brings to the table. To gauge the relative merits of on-premises vs. VMware Cloud on AWS from a Big Data performance point of view, similarly configured 8-server clusters were built in both environments, and the same set of Spark benchmarks were run on both. Performance was very similar, leaving customers with two excellent options for running their Big Data workloads.

Introduction

In addition to its traditional on-premises, private cloud virtualization offerings, VMware now offers the ability to run the same workloads in the public cloud. VMware Cloud on AWS provides users the ability to use dedicated physical servers in Amazon Web Services data centers to run the VMware Software-Defined Datacenter (SDDC) stack with the exact same management software as on-premises. The SDDC consists of VMware's extremely popular vSphere software for server virtualization, together with vSAN for storage virtualization and NSX for network virtualization. VMware Cloud on AWS is deployed with an as-a-service model, with clusters of four or more virtualized servers being quickly configured with vSAN datastores and virtualized networks, with a vCenter Server interface for the customer to access. Workloads may be migrated back and forth from on-premises to VMware Cloud on AWS as capacity requirement and/or data privacy requires. As shown in Figure 1, having the user's workload on VMware Cloud on AWS facilitates the use of cloud services such as Amazon Simple Storage Service (S3). For more details on VMware Cloud on AWS see the VMware Cloud on AWS documentation [1].

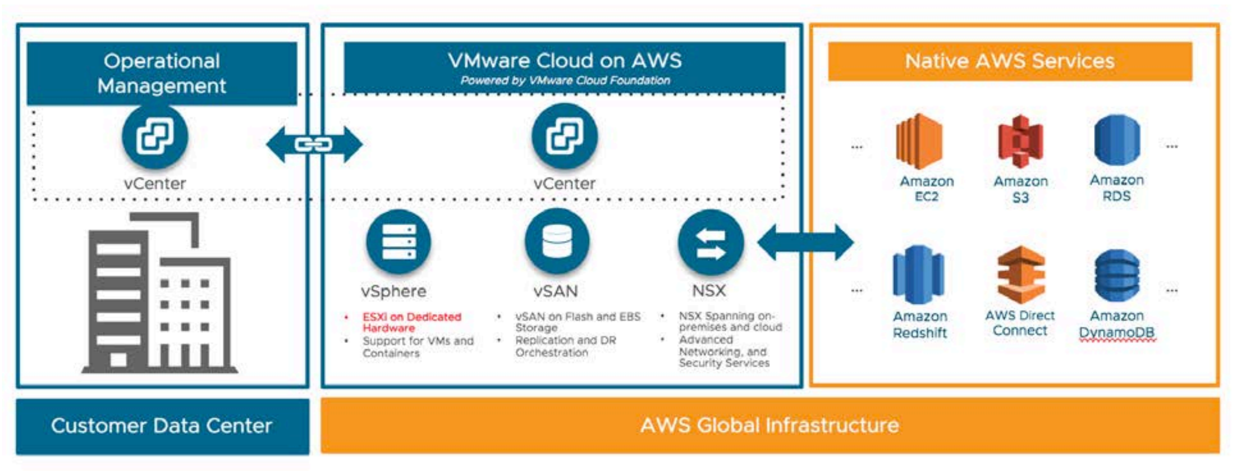

Operational Management
vCenter
Customer Data Center
VMware Cloud on AWS
Powered by VMware Cloud Foundation
vCenter
vSphere
ESXi on Dedicated Hardware
Support for VMs and Containers
vSAN
vSAN on Flash and EBS Storage
Replication and DR Orchestration
NSX
NSX Spanning on-premises and cloud
Advanced Networking, and Security Services
Native AWS Services
Amazon EC2
Amazon S3
Amazon RDS
Amazon Redshift
AWS Direct Connect
Amazon DynamoDB
AWS Global Infrastructure

Figure 1: VMware Cloud on AWS

A recent set of white papers has shown that vSphere in the private cloud is an excellent platform for running Big Data workloads such as Hadoop or Spark, with performance equal to or better than running on bare metal on those same on-premises servers. (See Fast Virtualized Hadoop and Spark on All-Flash Disks [2] for an example). To gauge the performance impact of running those workloads on VMware Cloud on AWS, similarly configured 8-server clusters were built on-premises and in VMware Cloud on AWS, and a set of Spark programs were run on both.

In addition to a set of standard benchmarks exercising key Spark Machine Learning Library (Spark MLlib) algorithms, a new VMware-developed open source benchmark, IoT Analytics Benchmark, was employed. This workload models data analytics being performed on an internet-of-things data stream, for example, factory sensors being monitored for impending failure conditions. IoT analytics is a great example of an application that bridges the on-premises and cloud worlds. For example, factory sensor data may be collected in S3 over time and labeled as predicting a failure or not. When enough such training data is collected, a cluster may be spun up in VMware Cloud on AWS for a week or two of model development using that S3 data. When the model is accurate enough, the cluster can be deleted (keeping costs to a minimum) and the model deployed to on-site edge nodes or an on-premises data center in the factory.

In the next section, the on-premises and VMware Cloud on AWS clusters are documented, followed by details of the vSphere/vSAN configurations and Hadoop tuning parameters. Following that, the Spark MLlib and IoT Analytics benchmarks are discussed in detail. Finally, test results from the on-premises and VMware Cloud on AWS environments are shown and analyzed.

Test Environments

On-premises and VMware Cloud on AWS Cluster Configurations

Eight servers were used in both the on-premises and VMware Cloud on AWS clusters. Both sets of servers were configured with two Intel Xeon v4 Processors ("Broadwell"), with the servers in VMware Cloud on AWS slightly faster (running at 2.30 GHz vs. 2.10 GHz) and with more cores (18 vs. 16 cores). Hyperthreading was enabled on all servers so the VMware Cloud on AWS servers showed 72 logical processors or hyperthreads, while the on-premises ones showed 64. All servers had 512 GiB of memory.

Each on-premises server was configured with an SD card, two 1.2 TB spinning disks, four 800 GB NVMe SSDs connected to the PCI bus, and twelve 800 GB SAS SSDs connected through the RAID controller.

ESXi 6.7.0 was installed on the SD card on each on-premises server. The 4 NVMEs were configured as vSAN cache disks and the 12 SSDs as vSAN capacity disks. Each VM's operating system was installed on the vSAN datastore, as well as all the data files (more details on the vSAN configuration follow).

Each on-premises server had one 1 GbE NIC and four 10 GbE NICs. Two distributed vSwitches were created from the four 10GbE NICs, one for vSAN traffic and one for inter-VM traffic. The maximum transmission unit (MTU) of each virtual NIC was set to 9000 bytes to handle jumbo Ethernet frames.

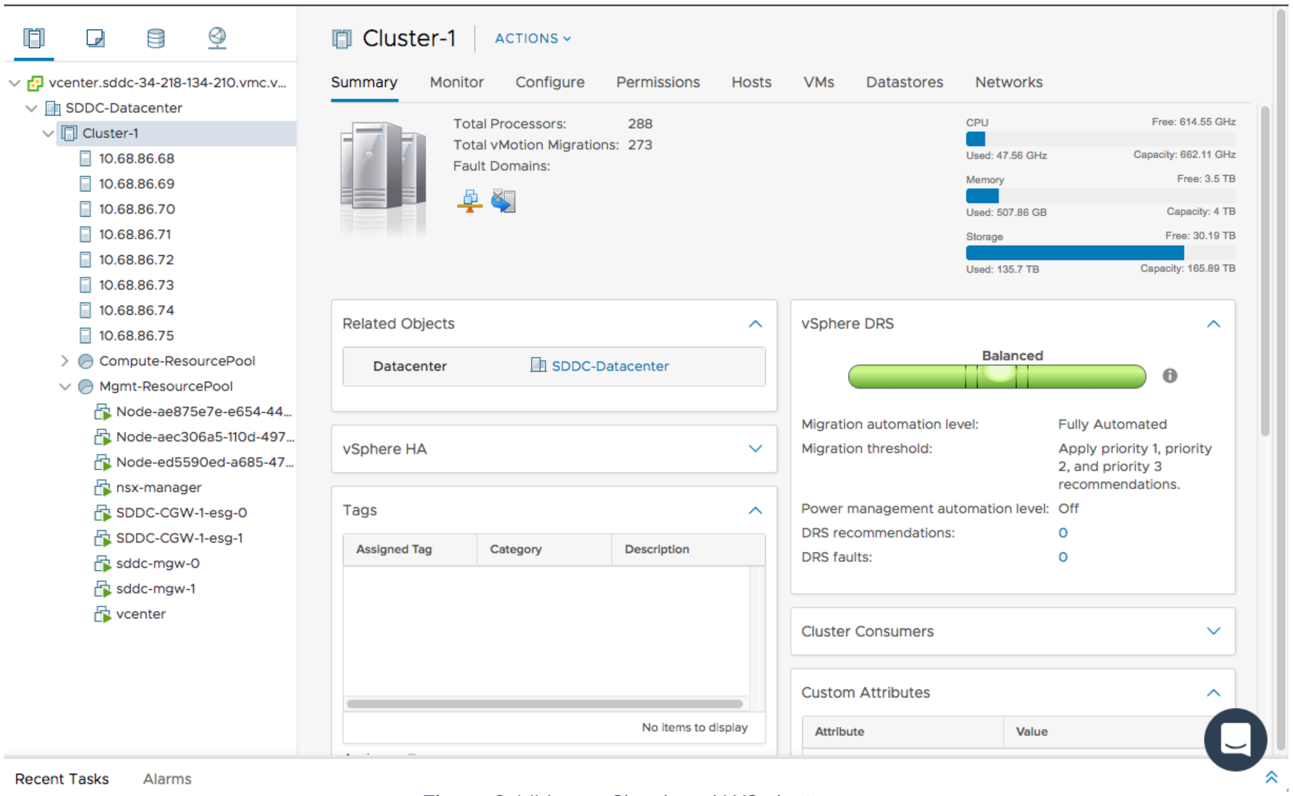
The VMware Cloud on AWS servers came configured with ESXi 6.8.0 and eight 1.73 TB NVMe drives. The pre-created vSAN datastore was built on the eight drives, with two of the drives serving as cache tier disks and the remaining six as the capacity tier. A distributed switch for both vSAN and VM traffic was created using one 25 GbE NIC on each server.

Details of both configurations are found in Table 1.

Feature	On-premises	VMware Cloud on AWS
Hypervisor	ESXi 6.7.0, 8169922	ESXi 6.8.0, 8493082
Servers	8	8
Processors per host	2x Intel® Xeon® CPU E5-2683 v4 @ 2.10 GHz	2x Intel® Xeon® CPU E5-2686 v4 @ 2.30GHz
Logical processors per host	64	72
Memory per host	512 GB	512 GB
NICs per host	4x 10 GbE	1x 25 GbE
Flash disks per host	4x NVMe (800 GB) 12x SSD (800 GB)	8x NVMe (1.73 TB)
Workload VMs per host	2	2
vCPUs per Workload VM	16	16
Memory per Workload VM	200 GB	200 GB

Table 1: Big Data on VMware Cloud on AWS and on-premises configurations

The VMware Cloud on AWS cluster, including the eight hosts with their IP addresses, is shown in Figure 2.


vcenter.sddc-34-218-134-210.vmc.v...
SDDC-Datacenter
Cluster-1
10.68.86.68
10.68.86.69
10.68.86.70
10.68.86.71
10.68.86.72
10.68.86.73
10.68.86.74
10.68.86.75
Compute-ResourcePool
Mgmt-ResourcePool
Node-ae875e7e-e654-44...
Node-aec306a5-110d-497...
Node-ed5590ed-a685-47...
nsx-manager
SDDC-CGW-1-esg-0
SDDC-CGW-1-esg-1
sddc-mgw-0
sddc-mgw-1
vcenter

Cluster-1 ACTIONS

Summary Monitor Configure Permissions Hosts VMs Datastores Networks

Total Processors: 288
Total vMotion Migrations: 273
Fault Domains:

CPU Free: 614.55 GHz
Used: 47.56 GHz Capacity: 662.11 GHz
Memory Free: 3.5 TB
Used: 507.86 GB Capacity: 4 TB
Storage Free: 30.19 TB
Used: 135.7 TB Capacity: 165.89 TB

Related Objects

Datacenter	SDDC-Datacenter

vSphere HA

Tags

Assigned Tag	Category	Description

No items to display

vSphere DRS

Balanced

Migration automation level:	Fully Automated
Migration threshold:	Apply priority 1, priority 2, and priority 3 recommendations.
Power management automation level:	Off
DRS recommendations:	0
DRS faults:	0

Cluster Consumers

Custom Attributes

Attribute	Value

Recent Tasks Alarms

Figure 2: VMware Cloud on AWS cluster

vSphere/vSAN Configuration

For both the on-premises cluster and the VMware Cloud on AWS cluster, two VMs were installed on each server. Whereas previous tests have set the total number of vCPUs on each host equal to the number of logical processors or hyperthreads (64 or 72 in this case), for vSAN it has been found that setting the number of vCPUs equal to the number of physical cores (32 or 36) provides the best performance. To enable a fair comparison, the lower number was chosen. The 32 vCPUs were evenly distributed to the VMs, 16 each. For vSAN, experimentation showed that 20% of server memory needs to be provided for ESXi, so the remaining 400 GiB was divided equally between the two VMs in both cases. A full discussion of vSAN memory usage is in VMware KB 2113954 [3].

Each virtual machine was installed with the CentOS 7.5 operating system from CentOS-7-x86_64-Everything-1804.iso, available from the CentOS download page [4]. This includes the latest Spectre/Meltdown patches as of the time of this writing.

It should be noted that the VMware Cloud on AWS cluster includes nine lightweight management VMs that provide functionality for vCenter, NSX, and the distributed switch, collected in the Mgmt-ResourcePool as shown in Figure 2. The Distributed Resource Scheduler in VMware Cloud on AWS places these VMs on hosts that have the most available resources to minimize their impact on workload performance, but nevertheless they consume resources on the VMware Cloud on AWS cluster but not on the on-premises cluster, where vCenter and other support services run elsewhere.

To build the vSAN storage cluster on the on-premises cluster, first a distributed switch was built using two of the 10 GbE NICs on each of the eight servers. The NICs were trunked together as a Link Aggregation Group (LAG) for bandwidth and redundancy. A VMkernel network adapter enabling vSAN service was added, as well as one enabling vMotion. All eight servers were then added to this distributed switch.

A second distributed switch was created using the remaining two 10 GbE NICs on each server, also in a LAG group, to carry inter-VM traffic.

A single vSAN datastore was created on the eight hosts, with four disk groups per host. Each disk group used an NVMe for the cache tier and three SSDs for the capacity tier, resulting in a datastore of 69.86 TB. Previous performance studies have shown that configuring the maximum number of disk groups yields the best vSAN performance for Big Data [5].

The cluster was configured with the default settings for Deduplication, Compression, and Encryption—all were Disabled.

As mentioned, the VMware Cloud on AWS cluster comes pre-configured with a workload vSAN datastore for the user's application as well as a management datastore for the management VMs. The workload datastore consisted of two disk groups per host, with each disk group using one NVMe for the cache tier and three for the capacity tier, resulting in a datastore of 82.95 TB.

Similar vSAN storage policies were created on the two clusters. In both cases Checkum was disabled (since the Hadoop Distributed Filesystem (HDFS) provides checksumming of data blocks) and the Stripe Width was set equal to the number of capacity disks in the host, 12 on-premises and 6 on VMware Cloud on AWS. Deduplication, Compression, and Encryption were disabled on VMware Cloud on AWS to be consistent with the on-premises case. Failures to Tolerate (FTT) was set to 1 in both cases, meaning there were two copies of each data block for redundancy.

In both clusters, a single 100 GB VMware disk (VMDK) was created on each VM for that VM's operating system. Additionally, the master VMs received one additional 100 GB data VMDK, while the worker VMs received six additional data VMDKs, 350 GB each on-premises and 200 GB on VMware Cloud on AWS (where part of the datastore was previously consumed by other workloads).

All VMDKs were connected through virtual SCSI controllers using the VMware Paravirtual SCSI driver. The OS disk and data disks were spread evenly over the four SCSI controllers.

The VMDKs on each VM were formatted using the ext4 filesystem, and the resulting disks were used to create the Hadoop filesystem. Since vSAN with FTT=1 provides data block redundancy, the HDFS block redundancy (`dfs.replication`) was reduced from the default of 3 to 2 for the FTT=1 tests.

vSAN storage configuration parameters are summarized in Table 2. For more details on vSAN see the VMware vSAN product page [6].

Feature	On-premises	VMware Cloud on AWS
Flash disks per host	4x NVMe (800 GB) 12x SSD (800 GB)	8x NVMe (1.73 TB)
Disk group configuration	1x NVMe (cache) + 3x SSD (capacity)	1x NVMe (cache) + 3x NVMe (capacity)
Disk groups per host	4	2
Capacity disks per host	12	6
vSAN Stripe Width	12	6
vSAN features – Checksum, Deduplication, Compression	Disabled	Disabled
vSAN Failures to Tolerate	1	1
vSAN storage capacity	69.86 TB	82.95 TB

Table 2: vSAN storage configurations

Hadoop/Spark Configuration

In a Hadoop cluster, there are three kinds of servers or nodes (see Table 3). One or more gateway servers act as client systems for Hadoop applications and provide a remote access point for users of cluster applications. Master servers run the Hadoop master services such as the Hadoop Distributed File System (HDFS) NameNode and the Yet Another Resource Negotiator (YARN) ResourceManager and their associated services (JobHistory Server, etc.), as well as other Hadoop services such as Hive, Oozie, and Hue. Worker nodes run only the resource-intensive HDFS DataNode role and the YARN NodeManager role (which also serve as Spark executors).

For these tests, three VMs on each cluster were used to manage the Hadoop cluster. One VM hosted the gateway node, running the cluster manager and several other Hadoop functions as well as the gateways for the HDFS, YARN, Spark, and Hive services. Two VMs hosted master functions, the active and passive NameNode and ResourceManager components and associated services. The active NameNode and active ResourceManager ran on different VMs for best distribution of CPU load, with the standby of each on the opposite master node. This also guarantees the highest cluster availability. For NameNode and ResourceManager high availability, at least three ZooKeeper services and three HDFS JournalNodes are required. These ran on the gateway VM and the two master VMs.

The other twelve VMs ran only the worker services, HDFS DataNode and YARN NodeManager. Spark executors ran on the YARN NodeManagers.

The full assignment of roles is shown in Table 3.

Node	Roles
Gateway	Cluster Manager, ZooKeeper Server, HDFS JournalNode, HDFS gateway, YARN gateway, Hive gateway, Spark gateway, Spark History Server, Hive Metastore Server, Hive Server2, Hive WebHCat Server, Hue Server, Oozie Server
Master1	HDFS NameNode (active), YARN ResourceManager (standby), ZooKeeper Server, HDFS JournalNode, HDFS Balancer, HDFS FailoverController, HDFS HttpFS, HDFS NFS gateway
Master2	HDFS NameNode (standby), YARN ResourceManager (active), ZooKeeper Server, HDFS JournalNode, HDFS FailoverController, YARN JobHistory Server
Workers (12)	HDFS DataNode, YARN NodeManager, Spark Executor

Table 3: Hadoop/Spark roles

In tuning for Hadoop, the two key cluster parameters that need to be set by the user are `yarn.nodemanager.resource.cpu-vcores` and `yarn.nodemanager.resource.memory-mb`, which tell YARN how much CPU and memory resources, respectively, can be allocated to task containers in each worker node.

For CPU resources, the vCPUs in each worker VM were exactly committed to YARN containers; that is, the number of vcores, `yarn.nodemanager.resource.cpu-vcores` was set equal to the number of vCPUs in each VM, 16. For memory, about 40 GiB of the VM's 200 GiB needs to be set aside for the operating system and the JVMs running the DataNode and NodeManager processes, leaving 160 GiB for `yarn.nodemanager.resource.memory-mb`. Thus, for the 12 worker nodes, the total vcores available was 192 and the total memory was 1920 GiB.

A few additional parameters were changed from their default values. The buffer space allocated on each mapper to contain the input split while being processed (`mapreduce.task.io.sort.mb`) was raised to its maximum value, 2047 MiB (about 2 GiB) to take advantage of the large amount of available memory. The amount of memory dedicated to the Application Master process, `yarn.app.mapreduce.am.resource.mb`, was raised from 1 GiB to 4 GiB. The parameter `yarn.scheduler.increment-allocation-mb` was lowered from 512 GiB to 256 MiB to allow finer grained specification of task sizes. The log levels of all key processes were turned down from the default of INFO to WARN for production use, but the much lower levels of log writes did not have a measurable impact on application performance.

These global parameters are summarized in Table 4.

Parameter	Default	Configured
yarn.nodemanager.resource.cpu-vcores	-	16
yarn.nodemanager.resource.memory-mb	-	160 GiB
mapreduce.task.io.sort.mb	256 MiB	2047 MiB
yarn.app.mapreduce.am.resource.mb	1 GiB	4 GiB
yarn.scheduler.increment-allocation-mb	512 MiB	256 MiB
Log Level on HDFS, YARN, Hive	INFO	WARN
Note: MiB = 2**20 (1048576) bytes, GiB = 2**30 (1073741824) bytes		

Table 4: Key Hadoop/Spark cluster parameters used in tests

More details on general Hadoop/YARN tuning are available in Fast Virtualized Hadoop and Spark on All-Flash Disks [2].

Workloads

Several Spark benchmarks that exercise the key components of a Big Data cluster were used for this test. These benchmarks may be used by customers as a starting point for characterizing their Big Data clusters, but their own applications will provide the best guidance for choosing the correct architecture.

Three standard analytic programs from the Spark machine learning library (MLlib), K-means clustering, Logistic Regression classification, and Random Forest decision trees, were driven using spark-perf [7]. In addition, a new, VMware-developed benchmark, IoT Analytics Benchmark, which models real-time machine learning on Internet-of-Things data streams, was used in the comparison. The benchmark is available from GitHub [8].

K-Means Clustering

Clustering is used for analytic tasks such as customer segmentation for purposes of ad placement or product recommendations. K-Means groups input into a specified number, k, of clusters in a multi-dimensional space. The test code groups a large training set into the specified number of clusters and uses this to build a model to quickly place a real input set into one of the groups.

Two K-Means tests were run, each with 5 million examples. The number of clusters was set to 20 in each. The number of features was varied, with 5,750 and 11,500 features generating dataset sizes of 500 GB and 1 TB. The training time reported by the benchmark kit was recorded. Four runs at each size were performed, with the first one being discarded and the remaining three averaged to give the reported elapsed time.

Logistic Regression Classification

Logistic regression (LR) is a binary classifier used in tools such as credit card fraud detection and spam filters. Given a training set of credit card transaction examples with, say, 20 features, (date, time, location, credit card number, amount, etc.) and whether that example is valid or not, LR builds a numerical model that is used to quickly determine if subsequent (real) transactions are fraudulent.

Two LR tests were run, each with 5 million examples. The number of features was varied, with 5,750 and 11,500 features generating dataset sizes of 500 GB and 1 TB. The training time reported by the benchmark kit was recorded. Four runs at each size were performed, with the first one being discarded and the remaining three averaged to give the reported elapsed time.

Random Forest Decision Trees

Random Forest automates any kind of decision making or classification algorithm by first creating a model with a set of training data, with the outcomes included. Random Forest runs an ensemble of decision trees in order to reduce the risk of overfitting the training data.

Two Random Forest tests were run, each with 5 million examples. The number of trees was set to 10 in each. The number of features was varied with 7,500 and 15,000 features generating dataset sizes of 500 GB and 1 TB. The training time reported by the benchmark kit was recorded. Four runs at each size were performed, with the first one discarded and the remaining three averaged to give the reported elapsed time.

The Spark MLlib code enables the specification of the number of partitions that each Spark resilient distributed dataset (RDD) employs. For these tests, the number of partitions was set equal to the number of Spark executors times the number of cores in each.

IoT Analytics Benchmark

The IoT Analytics Benchmark is a simulation of data analytics being run on a stream of sensor data, for example, factory machines being monitored for impending failure conditions.

IoT Analytics Benchmark consists of 3 Spark programs:

- iotgen generates synthetic training data files using a simple randomized model. Each row of sensor values is preceded by a label, either 1, or 0, indicating whether that set of values would trigger the failure condition
- iottrain uses the pre-labeled training data to train a Spark machine learning library model using logistic regression
- iotstream applies that model to a stream of incoming sensor values using Spark Streaming, indicating when the impending failure conditions need attention.

In these tests iotgen was used to generate datasets of 250, 500, and 750 GB, and then iottrain ran against these datasets to train the machine learning models used by iotstream. Because iotstream is a very lightweight process, it doesn't generate useful data for test results, so it wasn't used in this capacity.

In each case, four tests were run, with the first one discarded, and the last three averaged for the reported results.

Performance Results

Spark MLlib Results

The three Spark MLlib benchmarks were controlled by configuration files exposing many Spark and algorithm parameters. A few parameters were modified from their default values. From experimentation, it was found that the three programs ran fastest with 2 vcores and 20 GiB for each of 95 executors, using up most of the 192 vcores and 1,920 GiB available in the cluster. The 20 GiB was specified as 16 GiB `spark.executor.memory` plus 4 GiB `spark.yarn.executor.memoryOverhead`. The number of resilient distributed dataset (RDD) partitions was set to the number of executors times the number of cores per executor, or 190, so there would be one partition per core. 20 GiB was assigned to the Spark driver process (`spark.driver.memory`).

All three MLlib applications were tested with training dataset sizes of 500 GB and 1 TB. The cluster memory was sufficient to contain all datasets. For each test, first a training set of the specified size was created. Then the machine learning component was executed and timed, with the training set ingested and used to build the mathematical model to be used to classify real input data. The training times of four runs were recorded, with the first one discarded and the average of the remaining three values reported here.

Complete Spark MLlib test parameters are shown in Table 5.

Parameter	k-Means	Logistic Regression	Random Forest
# examples	5,000,000	5,000,000	5,000,000
500 GB # features	5,750	5,750	7,500
1 TB # features	11,500	11,500	15,000
# executors	95	95	95
Cores per executor	2	2	2
# partitions	190	190	190
Spark driver memory	20 GiB	20 GiB	20 GiB
Executor memory	16 GiB	16 GiB	16 GiB
Executor overhead memory	4 GiB	4 GiB	4 GiB

Table 5: Spark Machine Learning program parameters

The Spark Machine Learning test results are shown in Table 6–Table 8, and plotted in Figure 3. Since these programs mainly consume CPU and memory resources, there is very little (less than 10%) performance difference between the on-premises results and those from VMware Cloud on AWS. The larger datasets, which use more disk for caching intermediate results, run relatively slightly worse on VMware Cloud on AWS, which has half the number of disk groups than the on-premises cluster.

Platform	500 GB k-Means Elapsed Time (sec)	1 TB k-Means Elapsed Time (sec)
On-premises	78.7	167.1
VMware Cloud on AWS	79.8	182.9
Performance advantage, On-premises over VMware Cloud on AWS	1.4%	9.5%

Table 6: Spark k-Means performance results – smaller is better

Platform	500 GB Logistic Regression Elapsed Time (sec)	1 TB Logistic Regression Elapsed Time (sec)
On-premises	22.3	36.8
VMware Cloud on AWS	21.2	36.9
Performance advantage, On-premises over VMware Cloud on AWS	-5.2%	0.3%

Table 7: Spark Logistic Regression performance results – smaller is better

Platform	500 GB Random Forest Elapsed Time (sec)	1 TB Random Forest Elapsed Time (sec)
On-premises	124.2	219.7
VMware Cloud on AWS	124.5	236.2
Performance advantage, On-premises over VMware Cloud on AWS	0.3%	7.5%

Table 8: Spark Random Forest performance results – smaller is better

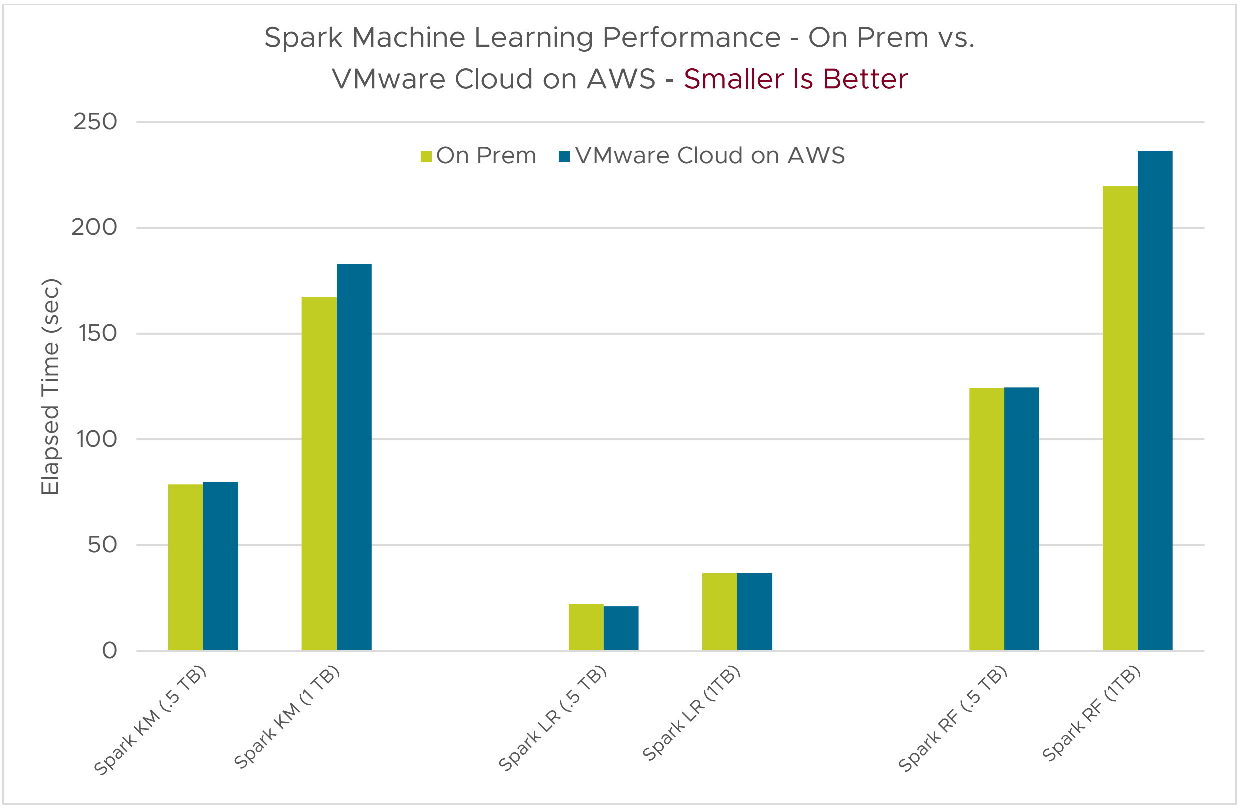


Figure 3: Spark MLlib performance

IoT Analytics Benchmark

The IoT Analytics Benchmark parameters are fully documented in the benchmark's GitHub site. As seen in Table 9, the programs were run using the standard spark-submit command, with Spark parameters immediately following, and the specific benchmark parameters at the end.

Single vcore executors are optimum for the write-based data generation (iotgen) program. 191 such executors were run (leaving 1 container available to YARN for the Application Master), each using a total of 10 GiB (8 GiB executor memory plus 2 GiB overhead). The parameters following `iotstream_2.10-0.0.1.jar` specify the number of rows, sensors per row, and partitions, and then the storage protocol, folder and file name of the output file. The final parameter (25215000) is used to control the percentage of rows that were coded to be "True" for model training.

For the model training (iottrain) program, 4 cores per executor were found to be optimal. Thus, 47 such executors were run (consuming a total of 188 vcores) each using a total of 40 GiB (32 GiB executor memory plus 8 GiB overhead). The parameters following `iotstream_2.10-0.0.1.jar` specify the storage protocol, folder, and file name of the training data file, and the name of the output file containing the trained model.

IoT Analytics Benchmark Commands
250 GB Data Generation
spark-submit --num-executors 191 --executor-cores 1 --executor-memory 8g --conf spark.yarn.executor.memoryOverhead=2048 \ --name iotgen_lr --class com.iotstream.iotgen_lr iotstream_2.10-0.0.1.jar 3356825 10000 191 HDFS sd \ sensor_data3356825_10000_10_191_1 25215000
500 GB Data Generation
spark-submit --num-executors 191 --executor-cores 1 --executor-memory 8g --conf spark.yarn.executor.memoryOverhead=2048 \ --name iotgen_lr --class com.iotstream.iotgen_lr iotstream_2.10-0.0.1.jar 6713459 10000 191 HDFS sd \ sensor_data6713459_10000_10_191_1 25215000
750 GB Data Generation
spark-submit --num-executors 191 --executor-cores 1 --executor-memory 8g --conf spark.yarn.executor.memoryOverhead=2048 \ --name iotgen_lr --class com.iotstream.iotgen_lr iotstream_2.10-0.0.1.jar 10070475 10000 191 HDFS sd \ sensor_data10070475_10000_10_191_1 25215000
250 GB Model Training
spark-submit --num-executors 47 --executor-cores 4 --executor-memory 32g --conf spark.yarn.executor.memoryOverhead=8192 \ --name iottrain_lr --class com.iotstream.iottrain_lr iotstream_2.10-0.0.1.jar HDFS sd sensor_data3356825_10000_10_191_1 lr10K_1
500 GB Model Training
spark-submit --num-executors 47 --executor-cores 4 --executor-memory 32g --conf spark.yarn.executor.memoryOverhead=8192 \ --name iottrain_lr --class com.iotstream.iottrain_lr iotstream_2.10-0.0.1.jar HDFS sd sensor_data6713459_10000_10_191_1 lr10K_2
750 GB Model Training
spark-submit --num-executors 47 --executor-cores 4 --executor-memory 32g --conf spark.yarn.executor.memoryOverhead=8192 \ --name iottrain_lr --class com.iotstream.iottrain_lr iotstream_2.10-0.0.1.jar HDFS sd sensor_data10070475_10000_10_191_1 lr10K_3

Table 9: IoT Analytics Benchmark commands

The IoT Analytics Benchmark performance results are shown in Table 10 and Table 11 and plotted in Figure 4. The iotgen data generation component, which is an extremely distributed, CPU-heavy application, with only a short write to HDFS, runs slightly better on VMware Cloud on AWS, with its faster CPUs. On the other hand, iottrain, which employs an iterative optimization algorithm that results in a high degree of inter-VM traffic, appears to benefit from the higher network capacity (4 NICs to 1) of the on-premises cluster and runs significantly slower in the cloud (employing all VMware Cloud on AWS vCPUs as additional Spark executors would reduce this deficit).

Platform	250 GB iotgen Elapsed Time (sec)	500 GB iotgen Elapsed Time (sec)	750 GB iotgen Elapsed Time (sec)
On-premises	367.8	728.3	1095.2
VMware Cloud on AWS	328.7	701.9	1039.2
Performance advantage, On-premises over VMware Cloud on AWS	-10.6%	-3.6%	-5.1%

Table 10: IoT Analytics Benchmark data generation performance results – smaller is better

Platform	250 GB iottrain Elapsed Time (sec)	500 GB iottrain Elapsed Time (sec)	750 GB iottrain Elapsed Time (sec)
On-premises	152.3	305.3	469.1
VMware Cloud on AWS	186.4	348.7	604.8
Performance advantage, On-premises over VMware Cloud on AWS	22.4%	14.2%	28.9%

Table 11: IoT Analytics Benchmark model training performance results – smaller is better

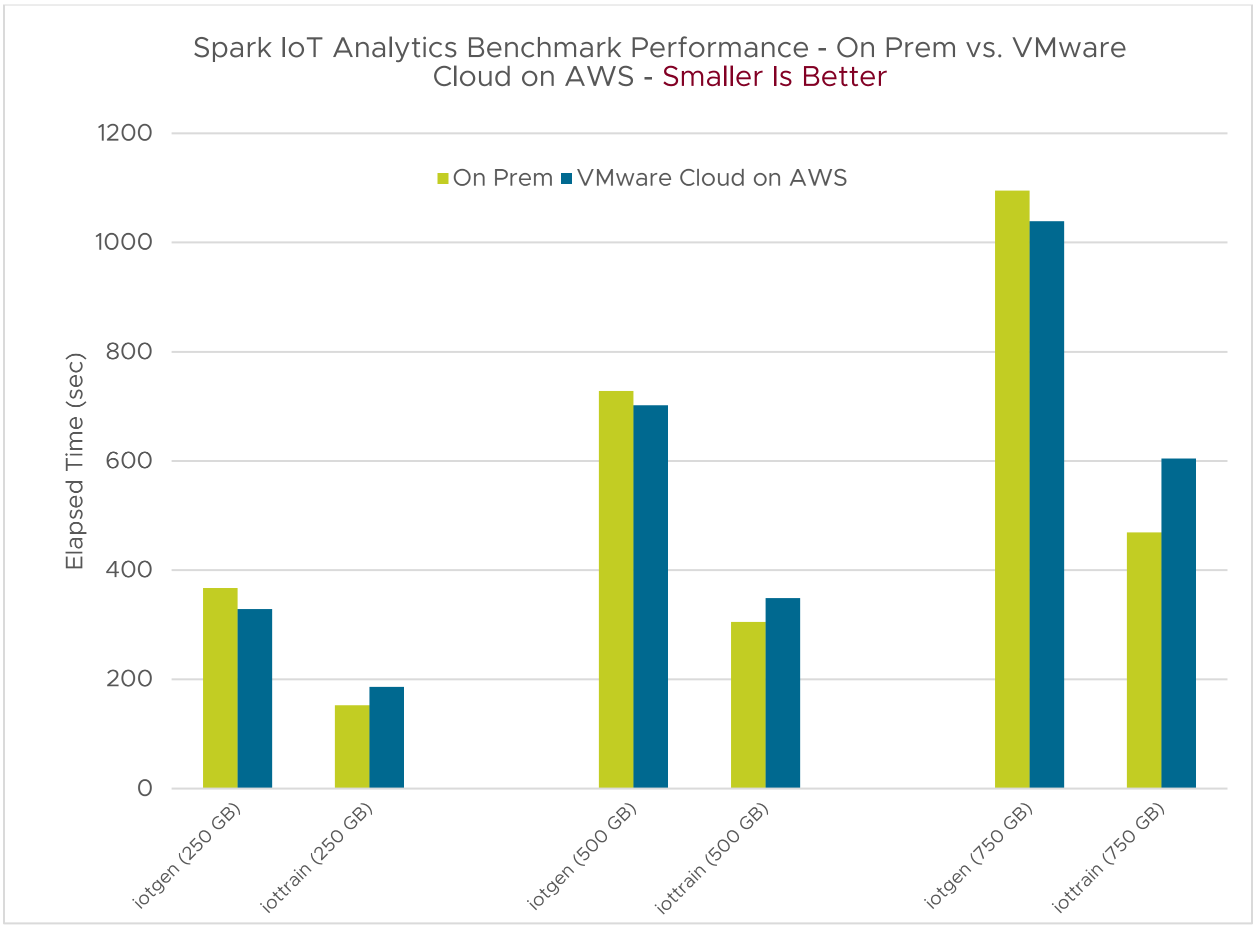


Figure 4: Spark IoT Analytics Benchmark performance

Conclusion

The decision to deploy a workload in a private cloud or public cloud has many factors including economic ones (long-term investment in a datacenter vs. short-term rental of on-line resources), security (data privacy, for example) and access to other services. One factor that should not enter the equation, however, is performance. As this work has shown for Big Data (and other studies have shown for other business-critical applications), performance in VMware Cloud on AWS is very comparable to that of an on-premises private cloud. So, whichever way a customer chooses to go, they will find VMware an excellent platform for Big Data.

References

[1] VMware. (2018) VMware Cloud on AWS documentation.
https://docs.vmware.com/en/VMware-Cloud-on-AWS/index.html

[2] VMware. (2017, August) Fast Virtualized Hadoop and Spark on All-Flash Disks.
https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/performance/bigdata-vsphere65-perf.pdf

[3] VMware. (2018, July) Understanding vSAN memory consumption in ESXi 6.5.0d/ 6.0 U3 and later (2113954).
https://kb.vmware.com/s/article/2113954

[4] The CentOS Project. (2018) CentOS download page.
https://www.centos.org/download/

[5] Intel. (2017) Ushering in a New Era of HyperConverged Big Data Using Hadoop with All-Flash VMware vSAN.
https://builders.intel.com/docs/storagebuilders/Hyper-Converged_big_data_using_Hadoop_with_All-Flash_VMware_vSAN.pdf

[6] VMware. (2018) VMware vSAN product page.
https://www.vmware.com/products/vsan.html

[7] Databricks. (2015, December) spark-perf: Spark Performance Tests.
https://github.com/databricks/spark-perf

[8] Dave Jaffe. (2018) IoT Analytics Benchmark.
https://github.com/vmware/iot-analytics-benchmark

About the Authors

Dave Jaffe is a staff engineer at VMware, specializing in Big Data performance.

Acknowledgements

The author would like to acknowledge Chen Wei and Amitabha Banerjee for consultation on vSAN, David Morse for VMware Cloud on AWS access and assistance, and Justin Murray for a never-ending discussion of how to best deliver virtualized Big Data to our customers.