PERFORMANCE STUDY – AUGUST 2017

FAST VIRTUALIZED HADOOP AND SPARK ON ALL-FLASH DISKS

Best Practices for Optimizing Virtualized Big Data Applications on VMware vSphere 6.5

vmware®

Table of Contents

Executive Summary

Best practices are described for optimizing Big Data applications running on VMware vSphere®. Hardware, software, and vSphere configuration parameters are documented, as well as tuning parameters for the operating system, Hadoop, and Spark. The Hewlett Packard Enterprise ProLiant DL380 Gen9 servers used in the test featured fast Intel processors with a large number of cores, large memory (512 GiB), and all-flash disks. Test results are shown from two MapReduce and three Spark applications running on three different configurations of vSphere (with 1, 2, and 4 VMs per host) as well as directly on the hardware. Among the virtualized clusters, the fastest configuration was 4 VMs per host due to NUMA locality and best disk utilization. The 4 VMs per host platform was faster than bare metal for all tests with the exception of a large (10 TB) TeraSort test where the the bare metal advantage of larger memory overcame the disadvantage of NUMA misses.

Introduction

Server virtualization has brought its advantages of rapid deployment, ease of management, and improved resource utilization to many data center applications, and Big Data is no exception. IT departments are being tasked to provide server clusters to run Hadoop, Spark, and other Big Data programs for a variety of different uses and sizes. There might be a requirement, for example, to run test/development, quality assurance, and production clusters at the same time, or different versions of the same software. Virtualizing Hadoop on vSphere avoids the need to dedicate a set of hardware to each different requirement, thus reducing costs. And with the need for many identical worker nodes, the automation tools provided by vSphere can provide for rapid deployment of new clusters.

The current work is the latest in a series of VMware studies into the optimum method to run Big Data workloads on vSphere, comparing those results to running the same workloads on similarly configured bare metal servers.

A 2015 paper, Virtualized Hadoop Performance with VMware vSphere 6 [1] showed how Hadoop Map Reduce version 1 (MRv1) workloads running on highly tuned vSphere implementations can approach and sometimes even exceed native performance. This was followed in 2016 by a study, Big Data Performance on vSphere 6 [2], that applied some of the learning from the 2015 study to a more typical customer environment, one which utilized tools such as Cloudera Manager and YARN [3] to deploy and manage a Hadoop cluster in a highly available fashion. Yet Another Resource Negotiator (YARN) replaced the JobTracker/TaskTracker resource management of MRv1 with a more general resource management process that can support other workloads such as Spark, in addition to MapReduce. Using YARN, the performance of the cluster was measured both with the standard Hadoop MapReduce benchmark suite of TeraGen, TeraSort, and TeraValidate, and the TestDFSIO Hadoop Distributed Filesystem (HDFS) stress tool, as well as a set of Spark machine learning programs that represent some of the leading edge uses of Big Data technology. The conclusion of that study was that a properly virtualized cluster ran both the Hadoop and Spark workloads the same as or faster than the same cluster running the worker servers on bare metal.

The current study expands upon the previous work using a new cluster equipped with the latest hardware: Intel v4 processors, large server memory, Non-Volatile Memory Express (NVMe) and solid state disk (SSD) storage, running the current version of vSphere, 6.5. Additionally, three different virtualization configurations (1, 2, and 4 VMs per host) were tested along with bare metal. Finally, a newer Spark machine learning benchmark, spark-perf [4], from Databricks, Inc., the developer of Spark, was used for the Spark tests.

This paper will show how to best deploy and configure the underlying vSphere infrastructure, as well as the Hadoop cluster, in such an environment. Best practices for all layers of the stack will be documented and their implementation in the test cluster described. The performance of the cluster will be shown both with the TeraSort suite, TestDFSIO HDFS stress tool, and the new Spark machine learning benchmarks.

Best Practices

Hardware Selection

The server hardware used in this study reflected general recommendations for Big Data as listed in the previous paper in this series [2]: large memory, especially for Spark (512 GiB), fast processors (Intel Xeon Processors E5-2683 v4 @ 2.10 GHz) with a high core count (16 cores per CPU), solid state storage (NVMe and SSD), and 10 GbE networking.

*Note: In this document, notation such as "GiB" refers to binary quantities such as gibibytes (2**30 or 1,073,741,824) while "GB" refers to gigabytes (10**9 or 1,000,000,000).*

The number of servers should be determined by workload size, number of concurrent users, and types of application. For workload size, it is necessary to include in the calculation the number of data block replicas created by the HDFS for availability in the event of disk or server failure (3 by default), as well as the possibility that the application needs to keep a copy of the input and output at the same time. Once sufficient capacity is put in place, it might be necessary to increase the cluster's computation resources to improve application performance or handle many simultaneous users.

Software Selection

Hadoop can be deployed directly from open source Apache Hadoop, but many production Hadoop users employ a distribution such as Cloudera, Hortonworks, or MapR, which provide deployment and management tools, performance monitoring, and support. These tests used the Cloudera Distribution of Hadoop 5.10.0

Most recent Linux operating systems are supported for Hadoop including RedHat/CentOS 6 and 7, SUSE Linux Enterprise Server 11 and 12, and Ubuntu 12 and 14. Java Development Kit versions 1.7 and 1.8 are supported. For any use other than a small test/development cluster, it is recommended to use a standalone database for management and for the Hive Metastore. MySQL, PostgreSQL, and Oracle are supported. Check the distribution for details.

Virtualizing Hadoop

On a virtualized server, some memory needs to be reserved for ESXi for its own code and data structures. With vSphere 6.5, about 5−6% of total server memory should be reserved for ESXi, with the remainder used for virtual machines.

Contemporary multiprocessor servers employ a design where each processor is directly connected to a portion of the overall system memory, as shown in Figure 1. This results in non-uniform memory access (NUMA), where a processor's access to its local memory is faster than to remote memory attached to other processors. For best performance with vSphere, it is recommended to size the VMs to each fit within a single NUMA node (the processor and its local memory), by creating one or more VMs per NUMA node. By doing so, the vSphere scheduler will keep each virtual machine on the NUMA node to which it was originally assigned, hence optimizing memory locality. So, for a two processor system, creating 2 or 4 VMs per host will give the fastest memory access.

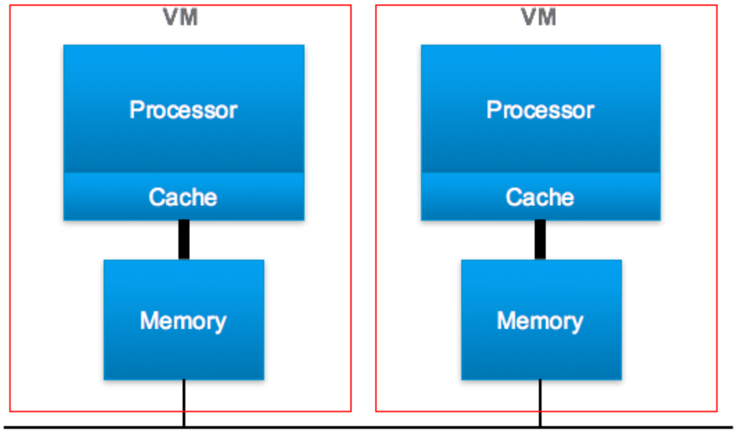


Figure 1. NUMA locality in a 2-processor server. The red lines show the virtual machine boundaries.

To maximize the utilization of each data disk, the number of disks per DataNode should be limited: 4 to 6 is a good starting point. (See Virtualized Hadoop Performance with VMware vSphere 6 [1] for a full discussion). Disk areas need to be cleared before writing. By choosing the eager-zeroed thick (**Thick Provision Eager Zeroed**) format for the virtual machine filesystem's (VMFS) virtual machine disks (VMDKs), this step will be accomplished before use, thereby increasing production performance. The file system used inside the guest should be ext4 or xfs. Ext4 provides fast large block sequential I/O—the kind used in HDFS—by mapping contiguous blocks (up to 128 MB) to a single descriptor or "extent." Xfs is a newer file system that provides quicker disk formatting times.

The VMware® Paravirtual SCSI (pvscsi) driver for disk controllers provides the best disk performance. There are four virtual SCSI controllers available in vSphere 6.5; the data disks should be distributed among all four.

Virtual network interface cards (NICs) should be configured with MTU=9000 for jumbo Ethernet frames and use the vmxnet3 network driver; the virtual switches and physical switches to which the host attaches should be configured to enable jumbo frames.

Tuning

Operating System

For both virtual machines and bare metal servers, the following operating system parameters are recommended for Hadoop (see, for example, the Cloudera documentation for Performance Management) [5].

The Linux kernel parameter `vm.swappiness` controls the aggressiveness of memory swapping, with higher values increasing the chance that an inactive memory page will be swapped to disk. This can cause problems for Hadoop clusters due to lengthy Java garbage collection pauses for critical system processes. It is recommended this be set to 0 or 1.

The use of transparent hugepage compaction should be disabled by setting the value of `/sys/kernel/mm/transparent_hugepage/defrag` to **never**. For example, the command

```
        echo never > /sys/kernel/mm/transparent_hugepage/defrag
```

can be placed in `/etc/rc.local` where it will be executed upon system startup. The value of `/sys/kernel/mm/transparent_hugepage/enabled` should be set similarly.

Jumbo Ethernet frames should be enabled on NICs (both in the bare metal and guest OS) and on physical network switches.

Hadoop

The developers of Hadoop have exposed a large number of configuration parameters to the user. In this work, a few key parameters were addressed.

A YARN-based Hadoop cluster has two primary processes: the NameNode and the ResourceManager. The NameNode manages the Hadoop Distributed Filesystem (HDFS), communicating with DataNode processes on each worker node to read and write data to HDFS. The YARN ResourceManager manages NodeManager processes on each worker node, which manage containers to run the map and reduce tasks of MapReduce or the executors used by Spark.

For HDFS, the two most important parameters involve size and replication number of the blocks making up the content stored in the HDFS filesystem. The block size `dfs.blocksize` controls the number of Hadoop input splits. Higher values result in fewer blocks (and map tasks) and thus more efficiency except for very small workloads. For most workloads, the block size should be raised from the default 128 MiB to 256 MiB. This can be made a global parameter that can be overridden at the command line if necessary. To accommodate the larger block size, `mapreduce.task.io.sort.mb` (a YARN parameter) should be raised to 400 MiB. This is the buffer space allocated on the mapper to contain the input split while being processed. Raising it prevents unnecessary spills to disk. The number of replicas, `dfs.replication`, defaults to 3 copies of each block for availability in the case of disk, server, or rack loss. These are general recommendations; as will be shown for the Hadoop applications tested here, raising these values significantly higher took the best advantage of the large memory servers.

YARN and MapReduce2 do away with the concept of map and reduce slots from MapReduce v1 and instead dynamically assign resources to applications through the allocation of task containers, where a task is a map or reduce task or Spark executor. The user specifies available CPU and memory resources as well as minimum and maximum allocations for container CPU and memory, and YARN will allocate the resources to tasks. This is the best approach for general cluster use, in which multiple applications are running simultaneously. For greater control of applications running one at a time, task container CPU and memory requirements can be specified to manually allocate cluster resources per application. This latter approach was used in the tests described here.

For CPU resources, the key parameter is `yarn.nodemanager.resource.cpu-vcores`. This tells YARN how many virtual CPU cores (vcores) it can allocate to containers. In these tests, all available virtual CPUs (vSphere vCPUs) or logical cores (bare metal) were allocated. This will be discussed in more detail later after the hardware configurations are described.

The application container CPU requirements for map and reduce tasks are specified by `mapreduce.map.cpu.vcores` and `mapreduce.reduce.cpu.vcores`, which are specified in the cluster configuration and can be overridden in the command line, per application.

Similar to CPU resources, cluster memory resources are specified by `yarn.nodemanager.resource.memory-mb`, which should be defined in the cluster configuration parameters, while `mapreduce.map.memory.mb` and `mapreduce.reduce.memory.mb` can be used to specify container memory requirements that differ from the default of 1 GiB. The java virtual machine (JVM) that runs the task within the YARN container needs some overhead; the parameter `mapreduce.job.heap.memory-mb.ratio` specifies what percentage of the container memory to use for the JVM heap size. The default value of 0.8 was used in these tests. Details of the memory allocation for the virtualized and bare metal clusters follow.

In general, it is not necessary to specify the number of maps and reduces in an application because YARN will allocate sufficient containers based on the calculations described above. However, in certain cases (such as TeraGen, an all-map application that writes data to HDFS for TeraSort to sort), it is necessary to instruct YARN how many tasks to run in the command line. To fully utilize the cluster, the total number of available tasks minus one for the primary task that controls the application can be specified. The parameters controlling the number of tasks are `mapreduce.job.maps` and `mapreduce.job.reduces`.

The correct sizing of task containers—larger versus more containers—and the correct balance between map and reduce tasks are fundamental to achieving high performance in a Hadoop cluster. A typical MapReduce job starts with all map tasks ingesting and processing input data from files (possibly the output of a previous MapReduce job). The result of this processing is a set of key-value pairs which the map tasks then partition and sort by key and write (or spill) to disk. The sorted output is then copied to other nodes where reduce tasks merge the files by key, continue the processing, and (usually) write output to HDFS. The map and reduce phases can overlap, so it is necessary to review the above choices in light of the fact that each DataNode/NodeManager node will generally be running both kinds of tasks simultaneously.

Spark on YARN

Spark is a memory-based execution engine that in many ways is replacing MapReduce for Big Data applications. It can run standalone on Big Data clusters or as an application under YARN. For these tests, the latter was used. The parameters `spark.executor.cores` and `spark.executor.memory` play the same role for Spark executors as map/reduce task memory and vcore assignment do for MapReduce. Additionally, the parameter `spark.yarn.executor.memoryOverhead` can be set if the default (10% of `spark.executor.memory`) is insufficient (if out-of-memory issues are seen). In general, Spark runs best with fewer executors containing more memory.

Workloads

Several standard 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.

MapReduce

Two industry-standard MapReduce benchmarks, the TeraSort suite and TestDFSIO, were used for measuring performance of the cluster.

TeraSort Suite

The TeraSort suite (TeraGen/TeraSort/TeraValidate) is the most commonly used Hadoop benchmark and ships with all Hadoop distributions. By first creating a large dataset, then sorting it, and finally validating that the sort was correct, the suite exercises many of Hadoop's functions and stresses CPU, memory, disk, and network.

TeraGen generates a specified number of 100 byte records, each with a randomized key occupying the first 10 bytes, creating the default number of replicas as set by `dfs.replication`. In these tests 10, 30, and 100 billion records were specified resulting in datasets of 1, 3, and 10 TB. TeraSort sorts the TeraGen output, creating one replica of the sorted output. In the first phase of TeraSort, the map tasks read the dataset from HDFS. Following that is a CPU-intensive phase where map tasks partition the records they have processed by a computed key range, sort them by key, and spill them to disk. At this point, the reduce tasks take over, fetch the files from each mapper corresponding to the keys associated with that reducer, and then merge the files for each key (sorting them in the process) with several passes, and finally write to disk. TeraValidate, which validates that the TeraSort output is indeed in sorted order, is mainly a read operation with a single reduce task at the end.

TestDFSIO

TestDFSIO is a write-intensive HDFS stress tool also supplied with every distribution. It generates a specified number of files of a specified size. In these tests 1,000 files of size 1 GB, 3 GB, or 10 GB files were created for total size of 1, 3, and 10 TB.

Spark

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 [4].

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 code tested takes a large training set with a known grouping for each example and uses this to build a model to quickly place a real input set into one of the groups.

Three K-means tests were run, each with 5 million examples. The number of groups was set to 20 in each. The number of features was varied with 11,500, 23,000, and 34,500 features generating dataset sizes of 1 TB, 2 TB, and 3 TB. The training time reported by the benchmark kit was recorded. Six runs at each size were performed, with the first one being discarded and the remaining five 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.

Three LR tests were run, each with 5 million examples. The number of features were varied with 11,500, 23,000, and 34,500 features generating dataset sizes of 1 TB, 2 TB, and 3 TB. The training time reported by the benchmark kit was recorded. Six runs at each size were performed, with the first one being discarded and the remaining five 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.

Three 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 15,000, 30,000, and 45,000 features generating dataset sizes of 1 TB, 2 TB, and 3 TB. The training time reported by the benchmark kit was recorded. Six runs at each size were performed, with the first one discarded and the remaining five 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 initially set equal to the number of Spark executors times the number of cores in each, but was increased in certain configurations as necessary.

Test Environment

Hardware

Thirteen Hewlett Packard Enterprise (HPE) ProLiant DL380 Gen9 servers were used in the test as shown in Figure 2. The servers were configured identically, with two Intel Xeon Processors E5-2683 v4 (“Broadwell”) running at 2.10 GHz with 16 cores each and 512 GiB of memory. Hyperthreading was enabled so each server showed 64 logical processors or hyperthreads.

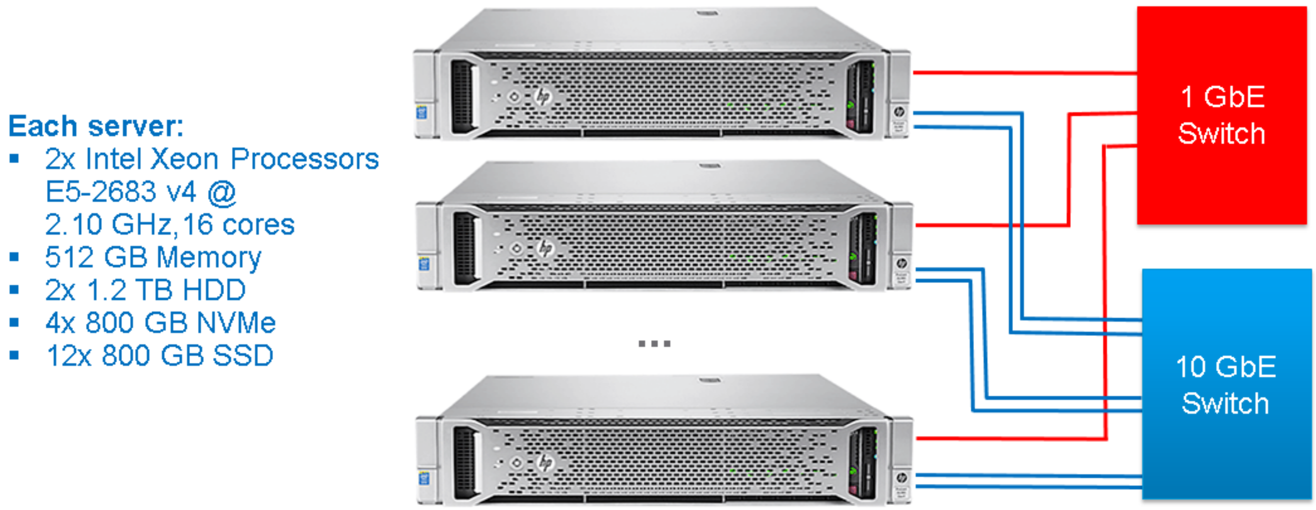


Figure 2. Big Data Cluster – 13 HPE ProLiant DL380 Gen9 servers

For the virtualized servers, all 64 logical processors were assigned to virtual CPUs (vCPUs), with the 1, 2, and 4 VMs per host platform having 64, 32, and 16 vCPUs per VM, respectively. To enable the NUMA calculation to include both physical and hyperthreaded logical cores, it is necessary to set `Numa.PreferHT` to `True` in vSphere on each host. Then, for example, the ESXi scheduler will calculate that a 32 vCPU VM in the 2 VMs per host platform will indeed fit within a single NUMA node since 32 vCPUs will fit within that processor’s cores if hyperthreads are counted.

Each 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 HPE Smart Array P840ar/2 GB RAID controller.

ESXi 6.5.0 was installed on the SD card on each server. The two internal 1.2 TB disks in each server were mirrored in a RAID1 set and used for the operating system (for bare metal) or as a virtual machine file system (VMFS) datastore upon which the operating system disks for the VMs on that host were stored. The 16 flash drives attached to each worker server were used as data drives for the NodeManager and DataNode processes. With the NVMe storage providing the highest random read/write I/Os per second, the 4 NVMes in each server were assigned to handle the NodeManager temporary data, which consisted of Hadoop map spills to disk and reduce shuffles. SAS SSDs provide very high speed sequential reads and writes so the 12 SSDs in each server were assigned to the DataNode traffic, consisting of reads and writes of permanent HDFS data. As shown later on in Table 3, the drives are evenly distributed among VMs: for the 1 VM per host platform (as well as bare metal), all 4 NVMEs and 12 SSDs are assigned to the VM/OS, while at the other extreme, the 4 VMs per host platform had 1 NVMe and 3 SSDs per VM.

For the bare metal cluster, the drives were formatted and mounted as an ext4 filesystem. The virtualized clusters were set up as VMFS datastores with one 745 GB virtual machine disk (VMDK) created on each, which was then formatted and mounted as ext4 using the same parameters as in the bare metal case. For best performance, these VMDKs were initialized with eager-zeroed thick formatting, which means the disk was pre-formatted so writes did not have to format the disk during production.

Each server had one 1 GbE NIC and four 10 GbE NICs (of which only two were used in this test). One virtual switch was created on each ESXi host using the 1 GbE NIC and served as the management network for the host as well as the external virtual machine network for the guests. A second virtual switch, created on two 10 GbE NICs bonded together, served as the internal network between VMs/bare metal hosts for all Hadoop traffic. By setting the bond load balancing policy to **Route based on originating virtual port**, both failover and performance throughput enhancement were achieved because the VMs attached to that NIC bond had different originating ports. The maximum transmission unit (MTU) of the bonded 10 GbE NIC was set to 9000 bytes to handle jumbo Ethernet frames. For the bare metal hosts, NIC bonding was achieved with adaptive load balancing.

An optimized vSphere driver for the HPE Smart Array P840ar/2 GB Controller, VMW-ESX-6.5.0-nhpsa-2.0.10-4593811.zip, was downloaded from Hewlett Packard Enterprise's site and installed on each vSphere host.

Hardware details are summarized in Table 1.

COMPONENT	QUANTITY/TYPE
Server	HPE ProLiant DL380 Gen9
CPU	2x Intel Xeon Processors E5-2683 v4 @ 2.10 GHz w/16 cores each
Logical Processors (including hyperthreads)	64
Memory	512 GiB (16x 32 GiB DIMMs)
NICs	2x 1 GbE ports + 4 x 10 GbE ports
Hard Drives	2x 1.2 TB 12 Gb/s SAS 10K 2.5in HDD – RAID 1 for OS
NVMes	4x 800 GB NVMe PCIe – NodeManager traffic
SSDs	12x 800 GB 12G SAS SSD – DataNode traffic
RAID Controller	HPE Smart Array P840ar/2G controller
Remote Access	HPE iLO Advanced

Table 1. Server configuration. In this document notation such as "GiB" refers to binary quantities such as gibibytes (2**30 or 1,073,741,824) while "GB" refers to gigabytes (10**9 or 1,000,000,000).

Hadoop Cluster Configuration

In a Hadoop cluster, there are three kinds of servers or nodes (see Table 2). One or more gateway servers act as client systems for Hadoop applications and provide a remote access point for users of cluster applications. Primary servers run the Hadoop primary services such as HDFS NameNode and 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 NameNode and ResourceManager high availability, at least three ZooKeeper services and three HDFS JournalNodes are required; these can run on the gateway and two primary servers.

For these tests, three of the servers were virtualized with VMware vSphere 6.5 and ran infrastructure VMs to manage the Hadoop cluster. On the first server a VM hosted the gateway node, running Cloudera Manager and several other Hadoop functions as well as the gateways for the HDFS, YARN, Spark, and Hive services. The second and third servers each hosted a primary VM, on which the active and passive NameNode and ResourceManager components and associated services ran. The active NameNode and active ResourceManager ran on different nodes for best distribution of CPU load, with the standby of each on the opposite primary node. ZooKeeper, running on all three VMs, provided synchronization of the distributed processes. Placing the gateway and two primary VMs on different physical servers guaranteed the highest cluster availability.

The other ten servers (either virtualized or bare metal) ran only the worker services, HDFS DataNode, and YARN NodeManager. Spark executors ran on the YARN NodeManagers.

In a production vSphere cluster, the three infrastructure hosts dedicated to the gateway and primary nodes also host additional VMs supporting services such as VMware vCenter Server® and VMware management tools such as VMware vRealize® Operations Manager™ appliance and VMware vRealize® Log Insight™. These services can be protected and load-balanced through vSphere high availability (HA) and distributed resource scheduler (DRS), in addition to the protection offered by NameNode and ResourceManager high availability. The gateway VM is the only server that needs direct network access to the outside world, so it can be protected with a firewall and access limited with a tool such as Kerberos. (The other nodes may need Internet access during installation depending on the method used; that can be achieved by IP forwarding through the gateway).

The full assignment of roles is shown in Table 2. Key software component versions are shown in Table 4.

NODE	ROLES
Gateway	Cloudera 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
Primary1	HDFS NameNode (active), YARN ResourceManager (standby), ZooKeeper Server, HDFS JournalNode, HDFS Balancer, HDFS FailoverController, HDFS HttpFS, HDFS NFS gateway
Primary2	HDFS NameNode (standby), YARN ResourceManager (active), ZooKeeper Server, HDFS JournalNode, HDFS FailoverController, YARN JobHistory Server,
Workers (10)	HDFS DataNode, YARN NodeManager, Spark Executor

Table 2. Hadoop/Spark roles

During these tests, the three virtualized servers controlling the cluster were left unchanged, while the configuration of the worker servers, which determine application performance, was changed for each platform. It is a common practice in production clusters to have a dedicated infrastructure (comprising less expensive servers) managing several clusters of more richly configured worker servers.

For the virtualized platforms, vSphere 6.5 was installed on each server. The 64 vCPUs on each host (corresponding to the 64 logical processors with hyperthreading enabled) were evenly committed to the VMs (see Table 3 for per-platform details). Following best practices, 32 GiB of each server's 512 GiB of memory (6%) was provided for ESXi, and the remaining 480 GiB was divided equally between the VMs. On all VMs, the virtual NIC connected to the bonded 10 GbE NICs was assigned an IP address internal to the cluster. The vmxnet3 driver was used for all network connections.

Each virtual machine was installed with its CentOS 7.3 operating system on the RAID1 datastore for that host, one or more NVMes assigned as NodeManager disks, and three or more SSDs assigned as DataNode disks. All disks 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 virtualized cluster is shown in Figure 3.

For the bare metal performance tests, the worker servers were reinstalled with CentOS 7.3. Each of the 10 bare metal worker servers used the full complement of 64 logical processors, 512 GiB, one RAID1 pair of 1.2 TB disks for its operating system, four 800 GB NVMes for NodeManager data, and twelve 800 GB SSDs for DataNode data. The two 10 GbE NICs were bonded together using Linux bonding mode adaptive load balancing to provide both failover and enhanced throughput similar to that employed with ESXi. In neither the virtualized nor the bare metal case was the network switch to which the NICs attach configured with any special configuration for the NIC bonding. The bare metal cluster is shown in Figure 4.

The worker node configurations for virtualized and bare metal are compared in Table 3. The important YARN cluster parameters `yarn.nodemanager.resource.memory-mb` (container memory) and `yarn.nodemanager.resource.cpu-vcores` (container CPUs) are discussed in more detail following.

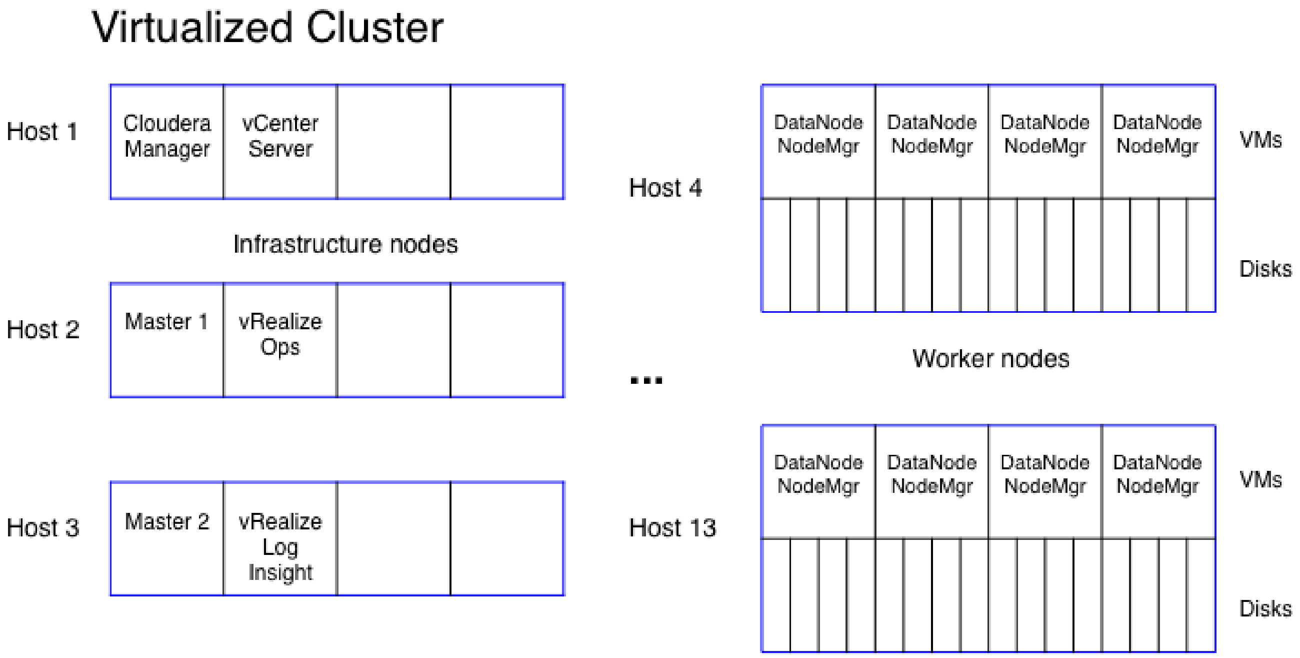


Figure 3. Big Data cluster – virtualized – 4 VMs per host platform shown

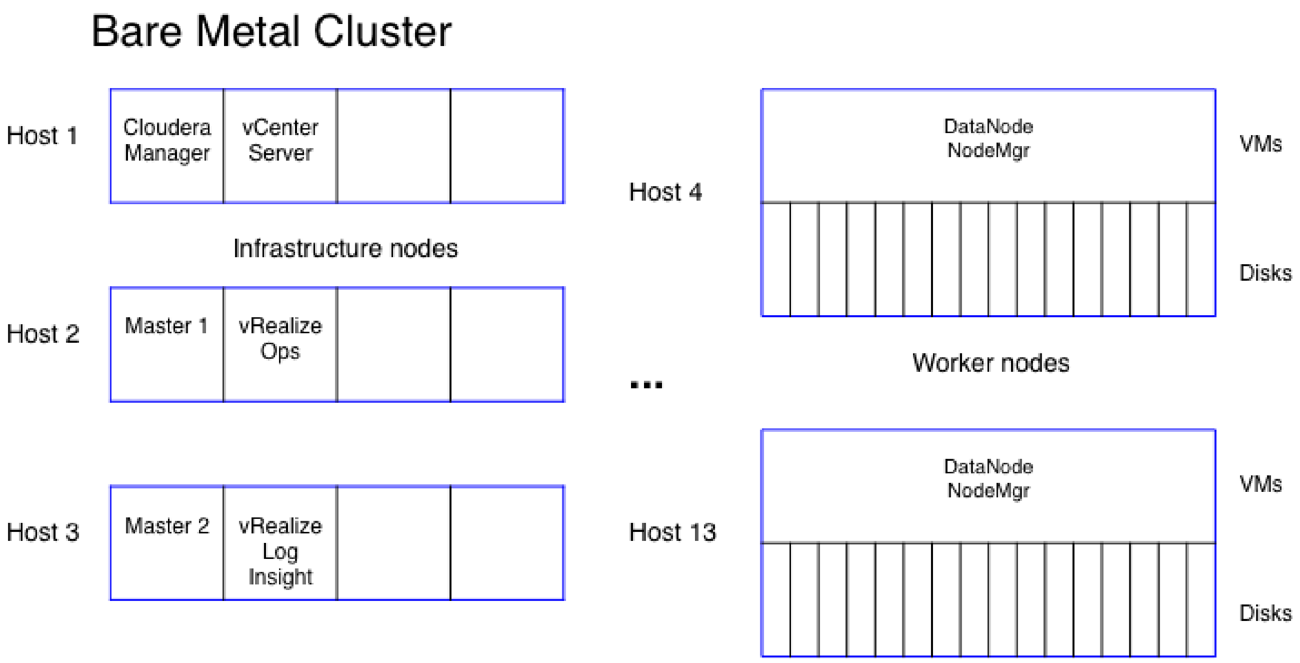


Figure 4. Big Data cluster - bare metal

COMPONENT	1 VM PER HOST	2 VMS PER HOST	4 VMS PER HOST	BARE METAL
Virtual CPUs	64	32	16	64
Memory	480 GiB	240 GiB	120 GiB	512 GiB
Container memory	432 GiB	208 GiB	104 GiB	448 GiB
Container vcores	64	32	16	64
Virtual NICs	Bonded 10 GbE NICs – ESX bonding	Bonded 10 GbE NICs – ESX bonding	Bonded 10 GbE NICs – ESX bonding	Bonded 10 GbE NICs – Balance ALB
OS drives	100 GB HDD	100 GB HDD	100 GB HDD	1.1 TB HDD
NodeManager drives	4x 740 GB NVMe	2x 740 GB NVMe	1x 740 GB NVMe	4x 740 GB NVMe
DataNode drives	12x 741 GB SSD	6x 741 GB SSD	3x 741 GB SSD	12x 741 GB SSD

Table 3. Worker node configuration per VM or bare metal server

Software

The software stack employed on each VM is shown in Table 4. The combination of Cloudera Distribution of Hadoop (CDH) 5.10.0 on CentOS 7.3 was used. The Oracle JDK 1.8.0_65 was installed on each node and then employed by CDH. MySQL 5.7 Community Release was installed on the gateway node for the Hive metastore and Oozie databases.

COMPONENT	VERSION
vSphere/ESXi	6.5.0, 4564106
Guest operating system	CentOS 7.3
Cloudera distribution of Hadoop	5.10.0
Cloudera Manager	5.10.0
Hadoop	2.6.0+cdh5.10.0+2102
HDFS	2.6.0+cdh5.10.0+2102
YARN	2.6.0+cdh5.10.0+2102
MapReduce2	2.6.0+cdh5.10.0+2102
Hive	1.1.0+cdh5.10.0+859
Spark	1.6.0+cdh5.10.0+457
ZooKeeper	3.4.5+cdh5.10.0+104
Java	Oracle 1.8.0_111-b14
MySQL	5.6.35 community server

Table 4. Key software components

Hadoop/Spark Configuration

The Cloudera Distribution of Hadoop (CDH) was installed using Cloudera Manager and the parcels method (see the Cloudera documentation [6] for details). HDFS, YARN, ZooKeeper, Spark, Hive, Hue, and Oozie were installed with the roles deployed as shown in Table 2. Once the Cloudera Manager and primary nodes were installed on the infrastructure servers, they were left in place for all three virtualized configurations, as well as the bare metal cluster, with the worker nodes being redeployed between each set of tests. The Cloudera Manager home screen is shown in Figure 5.

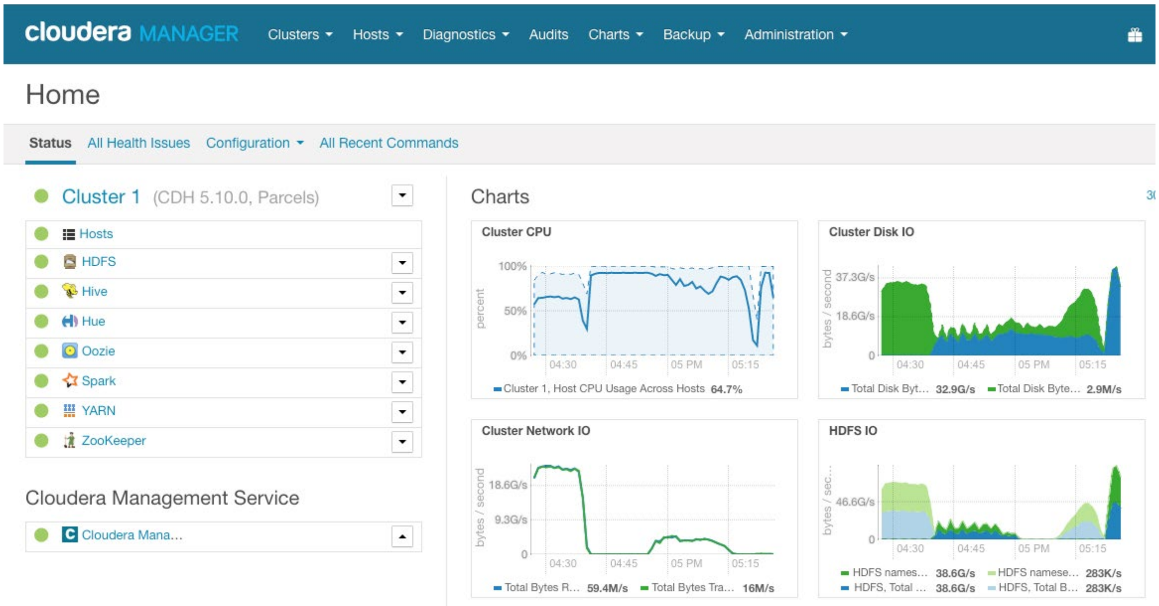
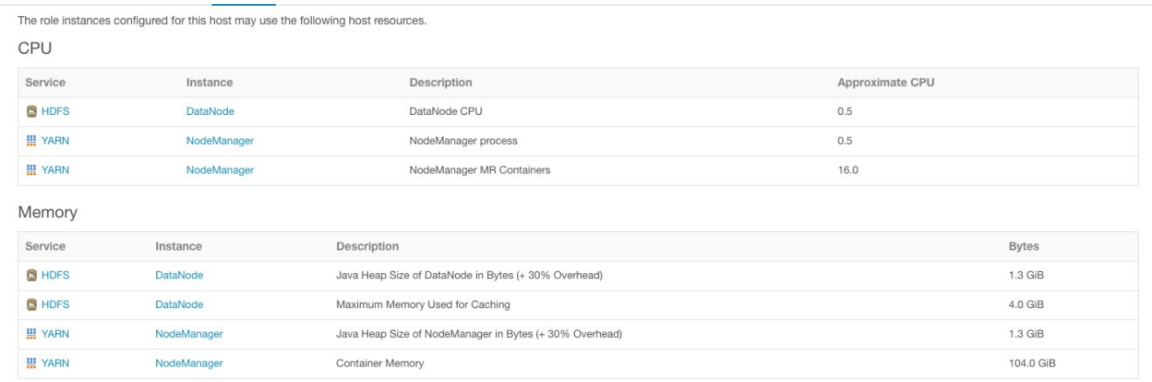


Figure 5. Cloudera Manager screenshot showing 10 Terabyte TeraGen/TeraSort/TeraValidate resource utilization. Where the CPU% drops marks the beginning of the next application.

As described previously, 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 the virtualized case, the vCPUs in each worker VM were exactly committed to YARN containers, that is, `yarn.nodemanager.resource.cpu-vcores` was set equal to the number of vCPUs in each VM. As seen in Figure 6, this actually overcommits the CPUs slightly since, in addition to the 16 vcores for containers (4 VMs per host case), a total of 1 vcore was required for the DataNode and NodeManager processes that run on each worker node. For the bare metal case, all 64 logical CPUs were allocated to containers, again slightly overcommitting the CPU resources.

The memory calculation was a little more involved. Taking the 4 VMs per host platform as an example (Figure 6), each of the 4 VMs have 120 GiB of memory after the 32 GiB of the 512 GiB host memory was set aside for ESXi. 9.4 GiB of that 120 GiB was set aside for the guest operating system. Of the remaining 110.6 GiB, 1.3 GiB each were required for the JVMs running the DataNode and NodeManager processes, plus 4 GiB for cache, leaving 104 GiB for containers (`yarn.nodemanager.resource.memory-mb`). The amount of memory for the operating system was under Cloudera's target of 20%. Since no swapping was seen on the worker nodes, the warning from Cloudera Manager was suppressed. (The 20% target could have been lowered as well).



The role instances configured for this host may use the following host resources.

CPU

Service	Instance	Description	Approximate CPU
HDFS	DataNode	DataNode CPU	0.5
YARN	NodeManager	NodeManager process	0.5
YARN	NodeManager	NodeManager MR Containers	16.0

Memory

Service	Instance	Description	Bytes
HDFS	DataNode	Java Heap Size of DataNode in Bytes (+ 30% Overhead)	1.3 GiB
HDFS	DataNode	Maximum Memory Used for Caching	4.0 GiB
YARN	NodeManager	Java Heap Size of NodeManager in Bytes (+ 30% Overhead)	1.3 GiB
YARN	NodeManager	Container Memory	104.0 GiB

Figure 6. DataNode/NodeManager resources – 4 VMs/host case

The memory calculation for the bare metal servers is similar. Setting aside 57.4 GiB of the 512 GiB server memory for the OS plus the 6.6 GiB for the DataNode and NodeManager processes and cache leaves 448 GiB for containers. Again, no swapping was seen with this amount of OS memory.

Totaling the available container memory and vcores for each cluster, all platforms have the same number of vcores available (640) as shown in Table 5. But note that between the ESXi overhead (32 GiB) and the fact that each VM requires one set of JVMs for the DataNode, NodeManager, and associated cache (requiring 6.6 GiB total) versus just one set total on bare metal, the total container memory of the virtualized platforms is lower than that available on the bare metal server. However, as will be shown in the results, this advantage does not make up for the virtualization advantages of NUMA locality and better disk utilization on most applications.

COMPONENT	1 VM PER HOST	2 VMS PER HOST	4 VMS PER HOST	BARE METAL
YARN container memory per VM or bare metal server	432 GiB	208 GiB	104 GiB	448 GiB
YARN container vcores per VM or bare metal server	64	32	16	64
Number of VMs or servers per cluster	10	20	40	10
YARN container memory per cluster	4320 GiB	4160 GiB	4160 GiB	4480 GiB
YARN container vcores per cluster	640	640	640	640

Table 5. Available YARN container memory and vcores per VM and per cluster by platform

These cluster parameters define the total memory and number of cores available to applications. To achieve the correct balance between the number and size of containers, much experimentation was done, particularly with TeraSort, factoring in the available memory vs. vcores. In general, the best performance on Hadoop was found by utilizing all YARN vcores but not all cluster memory. For Spark, the best performance was achieved by leaving both memory and cores underutilized. The exact settings per application per platform are shown in the Performance Results section.

A few additional parameters were changed from their default values on all platforms. 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 accommodate the very large block size that was used in the TeraSort suite (see below). The amount of memory dedicated to the primary application 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 Cloudera 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 6.

PARAMETER	DEFAULT	CONFIGURED
`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 6. Key Hadoop/Spark cluster parameters used in tests

Performance Results – Virtualized vs Bare Metal

TeraSort Results

The commands to run the three components of the TeraSort suite (TeraGen, TeraSort, and TeraValidate) are shown in Table 7. The key application-dependent parameters such as blocksize, map and reduce task container size (memory and cores) and, in some cases, number of tasks, are shown in Table 8 for each platform.

The `dfs.blocksize` was set at 1 GiB to take advantage of the large memory available to YARN, and, as mentioned previously, `mapreduce.task.io.sort.mb` was consequently set to the largest possible value, 2047 MiB, to minimize spills to disk during the map processing of each HDFS block.

It was found that the TeraGen map tasks for all platforms ran faster with 2 vcores. With 640 total cores available for each platform, 320 2-vcore map tasks could run simultaneously. However, a vcore must be set aside to run the primary application process, leaving 319 map tasks. Similarly, the TeraSort reduce task runs best with 1 vcore, so 639 are assigned.

The optimum map and reduce task memory allocations were determined experimentally. In all cases, less than the total cluster memory was assigned. For example, for the bare metal cluster, with 448 GiB of container memory available per server, 64 reduce tasks could be as large as 7 GiB (7168 MiB). However, it was found that a value of 6400 MiB, for a total consumption of 400 GiB, was found to give the best performance. As can be seen in Table 8 these values vary for each platform.

TERASORT COMMANDS

`time hadoop jar $examples_jar teragen -Ddfs.blocksize=$blocksize -Dmapreduce.job.maps=$n_maps_tg \`
`-Dmapreduce.map.memory.mb=$map_mem_tg -Dmapreduce.map.cpu.vcores=$map_vcores_tg \`
`10000000000/30000000000/100000000000 terasort<size>_input`

`time hadoop jar $examples_jar terasort -Ddfs.blocksize=$blocksize -Dmapreduce.job.reduces=$n_reds_ts \`
`-Dmapreduce.map.memory.mb=$map_mem_ts -Dmapreduce.reduce.memory.mb=$red_mem_ts \`
`-Dmapreduce.map.cpu.vcores=$map_vcores_ts -Dmapreduce.reduce.cpu.vcores=$red_vcores_ts terasort<size>_input \`
`terasort<size>_output`

`time hadoop jar $examples_jar teravalidate -Dmapreduce.map.memory.mb=$map_mem_tv terasort<size>_output \`
`terasort1TB_validate`

`where $examples_jar = /opt/cloudera/parcels/CDH/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar`

Table 7. TeraSort suite commands for 1, 3, and 10 TB input – see below for parameters

PARAMETER	ALL	1 VM PER HOST	2 VMS PER HOST	4 VMS PER HOST	BARE METAL
blocksize	1342177280				
n_maps_tg	319				
map_mem_tg (MiB)		13824	12800	13312	12800
map_vcores_tg	2				
n_reds_ts	639				
map_mem_ts (MiB)		6912	6400	6656	6400
red_mem_ts (MiB)		6912	6400	6656	6400
map_vcores_ts	1				
red_vcores_ts	1				
map_mem_tv (MiB)		6912	6400	6656	6400

Table 8. TeraSort suite parameters per platform

The TeraSort results are shown in Table 9–Table 11 and plotted in Figure 7. Focusing first on the optimum virtualized platform, 4 VMs per host, virtualized TeraGen was faster than bare metal due to the smaller number of disks per data node. This is consistent as the dataset created grows from 1 TB to 3 TB to 10 TB. Virtualized TeraSort starts at 7.5% faster than bare metal largely due to the benefits of NUMA locality (the NUMA miss rate as measured by the Linux numastat tool is 0% for the VMs, 21% for bare metal), but decreases to 1.1% faster at 3 TB, and finally goes slower than bare metal as the extra memory available in bare metal dominates. Virtualized TeraValidate, mainly reads, was faster than bare metal for the smaller dataset sizes but ended up about the same as bare metal at 10 TB.

Examining the relative performance of the three virtualized platforms, 4 VMs per host consistently is faster than 2 VMs per host due to its better match to number of disks per DataNode (4 vs. 8). 1 VM per host, which suffers from both NUMA non-locality and vSphere overhead, is the worst of the three virtualized configurations and is, in fact, worse than bare metal.

The resource utilization of the three applications is shown in the Cloudera Manager screenshot in Figure 5 (10 TB TeraSort suite). During TeraGen, about 35.3 GB/s of cluster disk writes occur accompanied by 23.7 GB/s of cluster network I/O as replicas are copied to other nodes. TeraSort starts with the CPU-intensive map sort, with disk writes occurring as sorted data is spilled to disk. In the reduce copy phase of TeraSort, network I/O ticks up, followed by disk I/O as the sorted data gets written to HDFS by the reducers. Finally, TeraValidate is a high disk read, low CPU operation.

PLATFORM	TERAGEN ELAPSED TIME (SEC)	TERASORT ELAPSED TIME (SEC)	TERAVALIDATE ELAPSED TIME (SEC)
1 VM/host	211.4	472.5	52.2
2 VMs/host	127.4	290.2	47.3
4 VMs/host	101.2	274.7	46.3
Bare metal	145.4	295.2	52.6
Performance advantage, 4 VMs/host over bare metal	43.7%	7.5%	13.7%

Table 9. TeraSort performance results showing vSphere vs. bare metal – 1 TB (smaller is better)

PLATFORM	TERAGEN ELAPSED TIME (SEC)	TERASORT ELAPSED TIME (SEC)	TERAVALIDATE ELAPSED TIME (SEC)
1 VM/host	586.1	1371.9	108.8
2 VMs/host	318.4	949.9	100.9
4 VMs/host	242.5	722.1	94.4
Bare metal	382.9	729.6	101.8
Performance advantage, 4 VMs/host over bare metal	57.9%	1.0%	7.8%

Table 10. TeraSort performance results showing vSphere vs. bare metal – 3 TB (smaller is better)

PLATFORM	TERAGEN ELAPSED TIME (SEC)	TERASORT ELAPSED TIME (SEC)	TERAVALIDATE ELAPSED TIME (SEC)
1 VM/host	1877.8	5216.4	305.1
2 VMs/host	991.6	3186.2	283.0
4 VMs/host	748.3	2519.8	261.5
Bare metal	1221.1	2318.7	264.3
Performance advantage, 4 VMs/host over bare metal	63.2%	-8.0%	1.1%

Table 11. TeraSort performance results showing vSphere vs. bare metal – 10 TB (smaller is better)

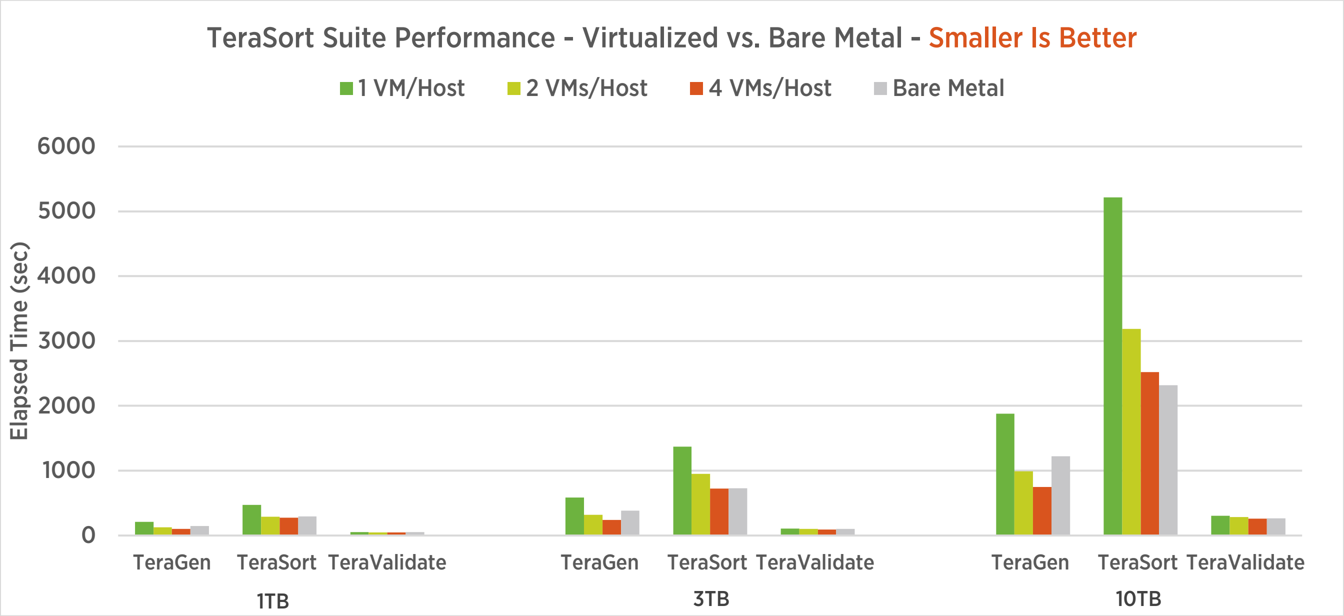


Figure 7. TeraSort suite performance showing vSphere vs. bare metal

TestDFSIO Results

TestDFSIO was run as shown in Table 12 to generate output of 1, 3, and 10 TB by writing 1,000 files of increasing size. As in TeraSort, the map memory size was adjusted experimentally for best performance for each platform, as shown in Table 13. There is a short reduce phase at the end of the test which was found to run best with 2 cores per reduce task.

TESTDFSIO COMMAND

`time hadoop jar $test_jar TestDFSIO -Ddfs.blocksize=$blocksize -Dmapreduce.map.memory.mb=$map_mem \`
`-Dmapreduce.reduce.cpu.vcores=$red_vcores -write -nrFiles 1000 -size 1GB/3GB/10GB`

`where $test_jar=/opt/cloudera/parcels/CDH/lib/hadoop-mapreduce/hadoop-mapreduce-client-jobclient-tests.jar`

Table 12. TestDFSIO command for 1, 3, and 10 TB datasets – see below for parameters

PARAMETER	ALL	1 VM PER HOST	2 VMS PER HOST	4 VMS PER HOST	BARE METAL
blocksize	1342177280				
map_mem (MiB)		6912	6400	6656	6400
red_vcores	2				

Table 13. TestDFSIO parameters per platform

The results are shown in Table 14 and Figure 8. Like TeraGen, TestDFSIO was faster on the 4 VMs per host platform due to the best match of number of disks (4) per DataNodes than on the other three virtualized platforms (8 or 16) or bare metal (16). For all three sizes, the 4 VMs per host platform was faster than 2 VMs per host, which was faster than bare metal. 1 VM per host was always slowest for the reasons cited above.

The 4 VMs per host platform maxed out at 47.5 GB/s total cluster disk I/O compared to 28.3 GB/s for the bare metal platform.

PLATFORM	TESTDFSIO 1 TB ELAPSED TIME (SEC)	TESTDFSIO 3 TB ELAPSED TIME (SEC)	TESTDFSIO 10 TB ELAPSED TIME (SEC)
1 VM/host	234.4	623.6	2016.7
2 VMs/host	135.0	318.9	1008.3
4 VMs/host	112.7	237.2	749.5
Bare metal	145.3	400.4	1300.4
Performance Advantage, 4 VMs/host over bare metal	28.9%	68.8%	73.5%

Table 14. TestDFSIO performance results showing vSphere vs. bare metal (smaller is better)

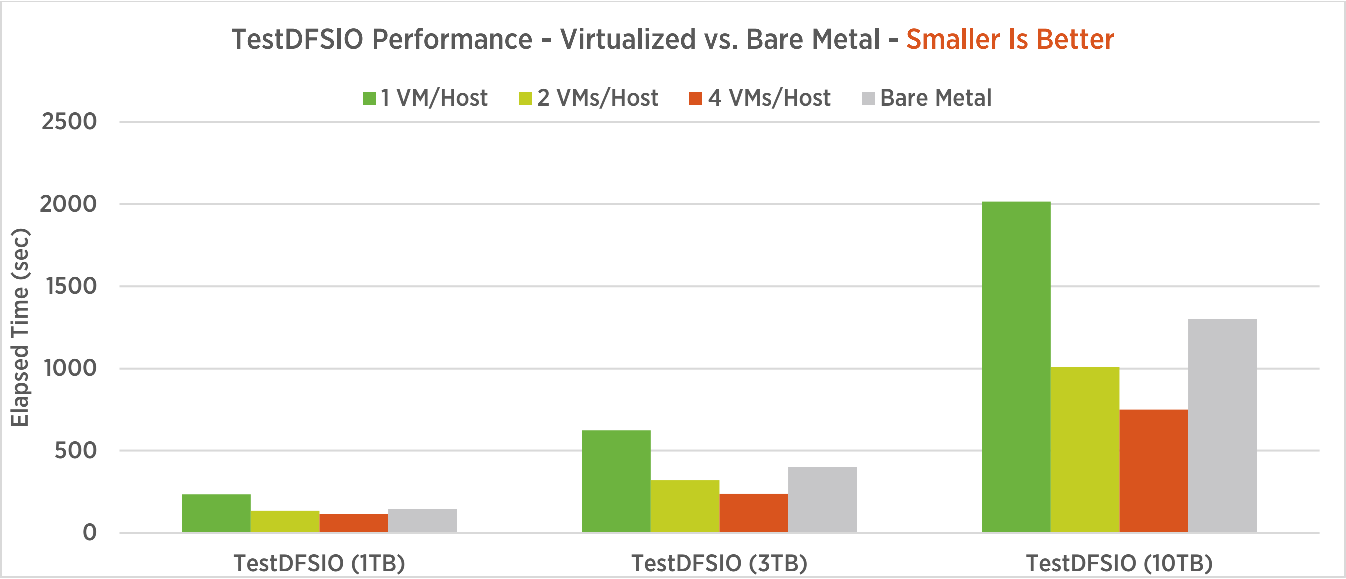


Figure 8. TestDFSIO performance showing vSphere vs. bare metal

Spark Results

The three Spark MLLib benchmarks were controlled by configuration files exposing many Spark and algorithm parameters. The parameters that were modified from their default values are documented for each application following.

For Hadoop, the best performance was seen with large numbers of map or reduce tasks, each with 1 or 2 vcores and 6 to 14 GiB per task, whereas the best performance with the Spark benchmarks was seen with a smaller number of larger Spark executors. As shown in the tables below, the Spark applications ran best with 2 or 4 vcores (`spark.executor.cores`) and up to 34 GiB (`spark.executor.memory` plus `spark.yarn.executor.memoryOverhead`) per Spark executor. For K-means and Logistic Regression, it was necessary to vary slightly the memory settings between platforms to achieve the best performance. The number of resilient distributed dataset (RDD) partitions was initially set to the number of executors times the number of cores per executor so there would be one partition per core. However, for some applications the partition size for the 2 and 3 TB datasets exceeded the 2 GiB Spark partition limit, so additional partitions were specified. Spark was run in yarn-client mode; this means the Spark primary process ran on the Spark gateway on the gateway VM. 20 GiB was assigned to this process throughout (`spark.driver.memory`).

All three MLLib applications were tested with training dataset sizes of 1, 2, and 3 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 six runs were recorded, with the first one discarded and the average of the remaining five values reported here.

K-means Clustering

K-means was tested with 5 million examples and 11,500, 23,000, and 34,500 features. Through testing it was determined that 4 cores per executor was optimal, but rather than specify a total of 160 executors on the 640-core clusters, 120 executors were found to be faster. This translates to 12 executors on each of the 10 bare metal servers or 3 executors on each of the 40 VMs of the 4 VMs per host platform. As before, the number, 120, was reduced by 1 to leave room for the primary application node.

The number of partitions specified was 476 (119x4) for the smaller datasets and was raised to 952 for the 3 TB dataset.

Each executor in the 1 and 2 VMs per host case was assigned 34 GiB of memory (including 10 GiB of overhead memory). This was slightly reduced to 32 GiB for the 4 VMs per host platform and bare metal. So for the bare metal case 384 GiB (12 executors at 32 GiB each) of the 448 GiB container memory was found optimal.

The K-means parameters are shown in Table 15.

PARAMETER	ALL	1 VM PER HOST	2 VMS PER HOST	4 VMS PER HOST	BARE METAL
# examples	5,000,000				
1 TB # features	11,500				
2 TB # features	23,000				
3 TB # features	34,500				
# executors	119				
Cores per executor	4				
1 TB # partitions	476				
2 TB # partitions	476				
3 TB # partitions	952				
Spark driver memory	20 GiB				
Executor memory		24 GiB	24 GiB	22 GiB	22 GiB
Executor overhead mem		10 GiB	10 GiB	10 GiB	10 GiB

Table 15. Spark K-means parameters per platform

The K-means performance results are shown in Table 16 and Figure 9. The spark-perf applications were coded as NUMA-aware, so there were no NUMA misses on any of the platforms. Since the datasets remain in memory, the number of disks per DataNode does not matter. Thus the performance across the four platforms is very similar. The slight advantage seen on the 4 VMs per host platform is due to the large amount of data that has to be transmitted from executors on one node to executors on other nodes in a Spark distributed application. For the VMs on the 4 VMs per host platform, 3 of the 39 nodes (8%) that an executor can write to are, in fact, on the same physical server so the communication occurs through the server's memory bus, much faster than having to go through the network. The same factor is 1 of 19 nodes (5%) for the 2 VMs per host platform and 0% for the 1 VM per host and bare metal platforms.

PLATFORM	K-MEANS 1 TB ELAPSED TIME (SEC)	K-MEANS 2 TB ELAPSED TIME (SEC)	K-MEANS 3 TB ELAPSED TIME (SEC)
1 VM/host	130.9	272.4	549.7
2 VMs/host	130.3	276.9	567.6
4 VMs/host	120.2	258.8	526.5
Bare metal	129.4	277.8	560.3
Performance advantage, 4 VMs/host over bare metal	7.6%	7.3%	6.4%

Table 16. Spark K-means performance results showing vSphere vs. bare metal – smaller is better

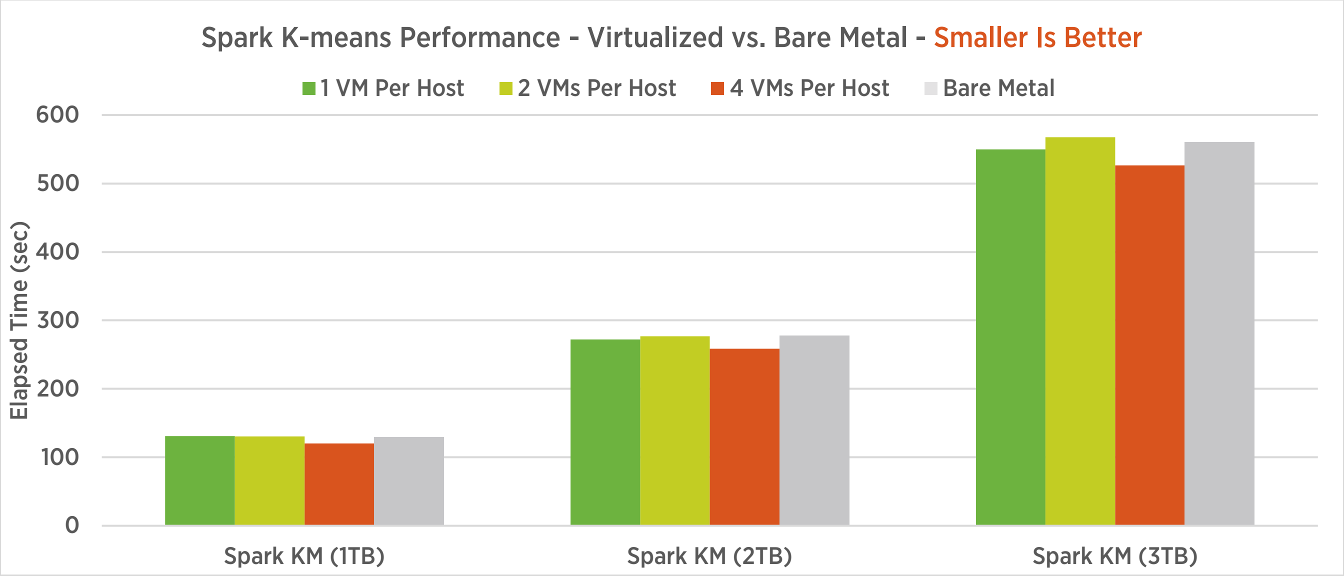


Figure 9. Spark K-means performance

Logistic Regression

Logistic Regression was also tested with 5 million examples and 11,500, 23,000, and 34,500 features. It was found that 2-core executors with 279 executors per cluster (28 per bare metal cluster node) gave the best performance. Executor sizes of 14 or 15 GiB were used (see Table 17). So, using the bare metal cluster as an example, 56 of the 64 container vcores and 420 GiB of the 448 GiB container memory was found to be optimal. The number of partitions specified was set to 558 (279x2) in all cases but one.

PARAMETER	ALL	1 VM PER HOST	2 VMS PER HOST	4 VMS PER HOST	BARE METAL
# examples	5,000,000				
1 TB # features	11,500				
2 TB # features	23,000				
3 TB # features	34,500				
# executors	279				
Cores per executor	2				
1 TB # partitions		558	558	558	558
2 TB # partitions		558	558	558	558
3 TB # partitions		558	1116	558	558
Spark driver memory	20 GiB				
Executor memory		12 GiB	11 GiB	11 GiB	12 GiB
Executor overhead mem		3 GiB	3 GiB	3 GiB	3 GiB

Table 17. Spark Logistic Regression parameters per platform

Logistic Regression performance results are shown in Table 18 and Figure 10. Like K-means, they are close with the 4 VMs per host platform showing a slight lead over the others due to the higher percentage of in-memory transfers.

PLATFORM	LOGISTIC REGRESSION 1TB ELAPSED TIME (SEC)	LOGISTIC REGRESSION 2TB ELAPSED TIME (SEC)	LOGISTIC REGRESSION 3TB ELAPSED TIME (SEC)
1 VM/host	36.0	61.3	99.9
2 VMs/host	35.1	58.9	85.0
4 VMs/host	25.8	38.4	52.7
Bare metal	34.9	54.1	83.8
Performance advantage, 4 VMs/host over bare metal	35.4%	40.9%	59.2%

Table 18. Spark Logistic Regression performance results showing vSphere vs. bare metal – smaller is better

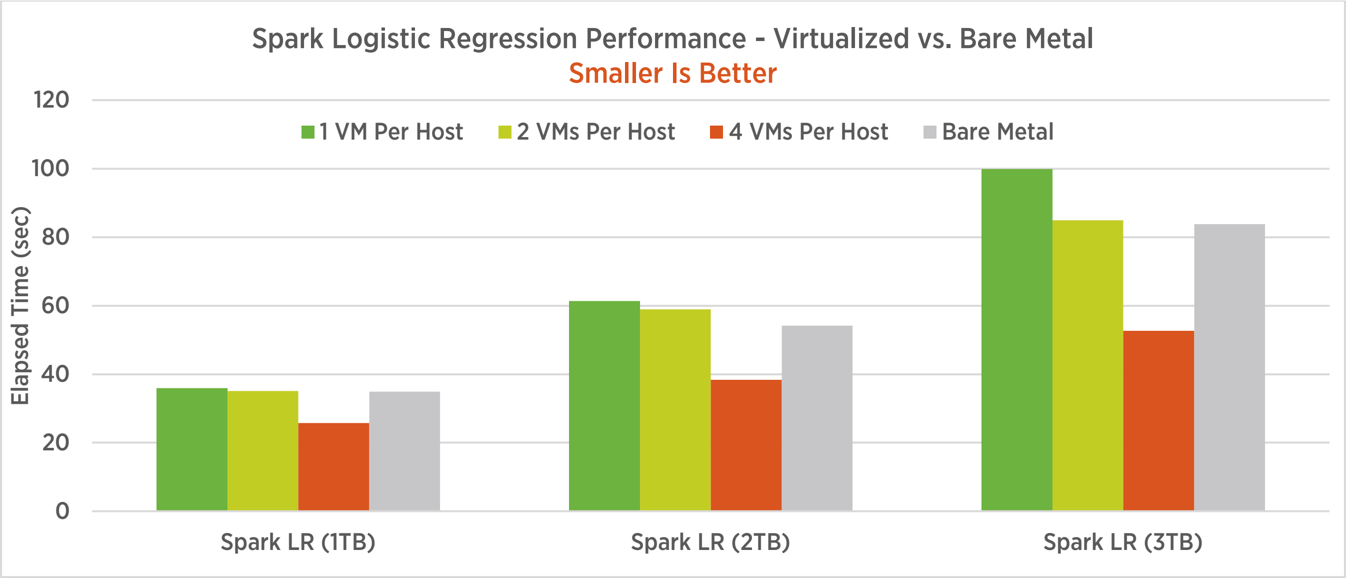


Figure 10. Spark Logistic Regression performance

Random Forest

The Random Forest workload was tested with 5 million examples and 15,000, 30,000, and 45,000 features. Similar to K-means, it ran best with 119 4-core executors. For each platform, 32 GiB of executor memory (including 10 GiB of overhead) was optimal. The number of partitions required scaled with the dataset size, from 476 to 952 to 1666 (see Table 19).

PARAMETER	ALL
# examples	5,000,000
1 TB # features	15,000
2 TB # features	30,000
3 TB # features	45,000
# executors	119
Cores per executor	4
1 TB # partitions	476
2 TB # partitions	952
3 TB # partitions	1666
Spark driver memory	20 GiB
Executor memory	22 GiB
Executor overhead memory	10 GiB

Table 19. Spark Random Forest parameters for all platforms

Random Forest performance results (shown in Table 20 and Figure 11) are once again close, with 4 VMs per host being the best.

PLATFORM	RANDOM FOREST 1 TB ELAPSED TIME (SEC)	RANDOM FOREST 2 TB ELAPSED TIME (SEC)	RANDOM FOREST 3 TB ELAPSED TIME (SEC)
1 VM/host	157.3	340.5	652.7
2 VMs/host	140.7	305.4	594.9
4 VMs/host	133.5	291.7	534.8
Bare metal	149.0	324.1	609.0
Performance advantage, 4 VMs/host over bare metal	11.6%	11.1%	13.9%

Table 20. Spark Random Forest performance results showing vSphere vs. bare metal – smaller is better

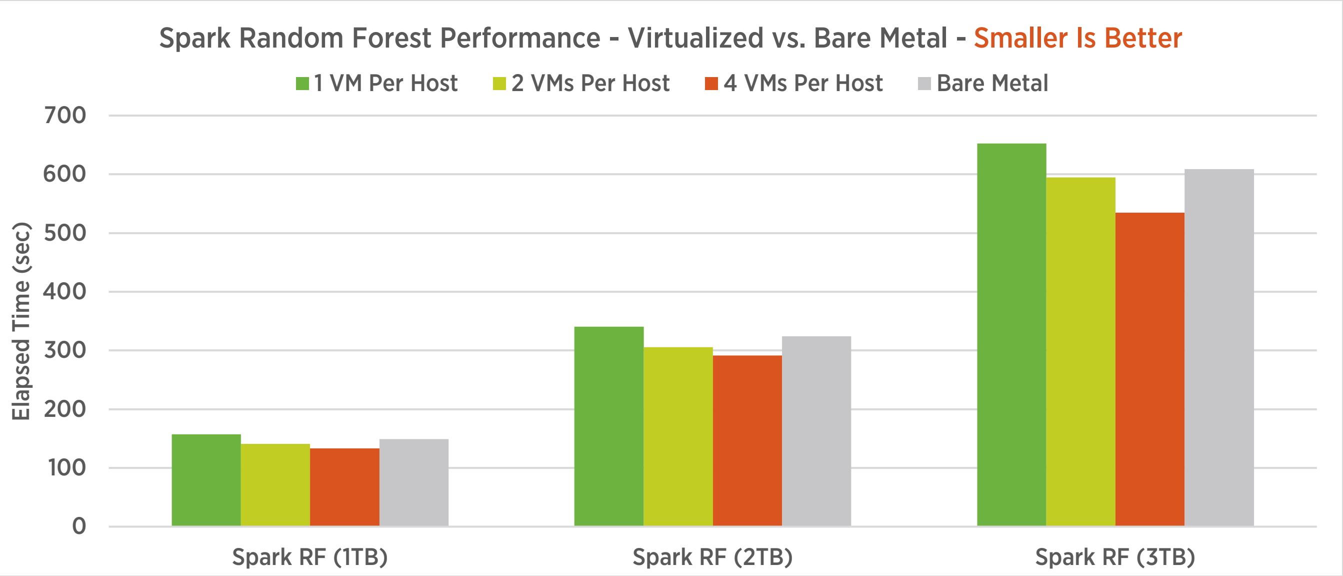


Figure 11. Spark Random Forest performance

Spark Scaling as Dataset Exceeds Memory Size

Spark caches its RDDs in memory while processing, and then writes to disk when necessary. To gauge the performance of the Spark applications when dataset size exceeded cluster memory, the Logistic Regression workload was run on the 4 VMs per host platform with training datasets from 1 TB to 6 TB. The first three fit in memory, the last three require that Spark RDDs be cached to disk. The data is in Table 21 and Figure 12.

DATASET SIZE	LOGISTIC REGRESSION ELAPSED TIME (SEC)
1 TB	24.5
2 TB	37.6
3 TB	52.2
4 TB	157.4
5 TB	203.9
6 TB	274.4

Table 21. Spark Logistic Regression performance for large datasets

As seen clearly in Figure 12, the performance scales linearly with dataset size as the dataset remains in memory, then scales linearly again, but at a slower rate as the dataset gets cached to disk.

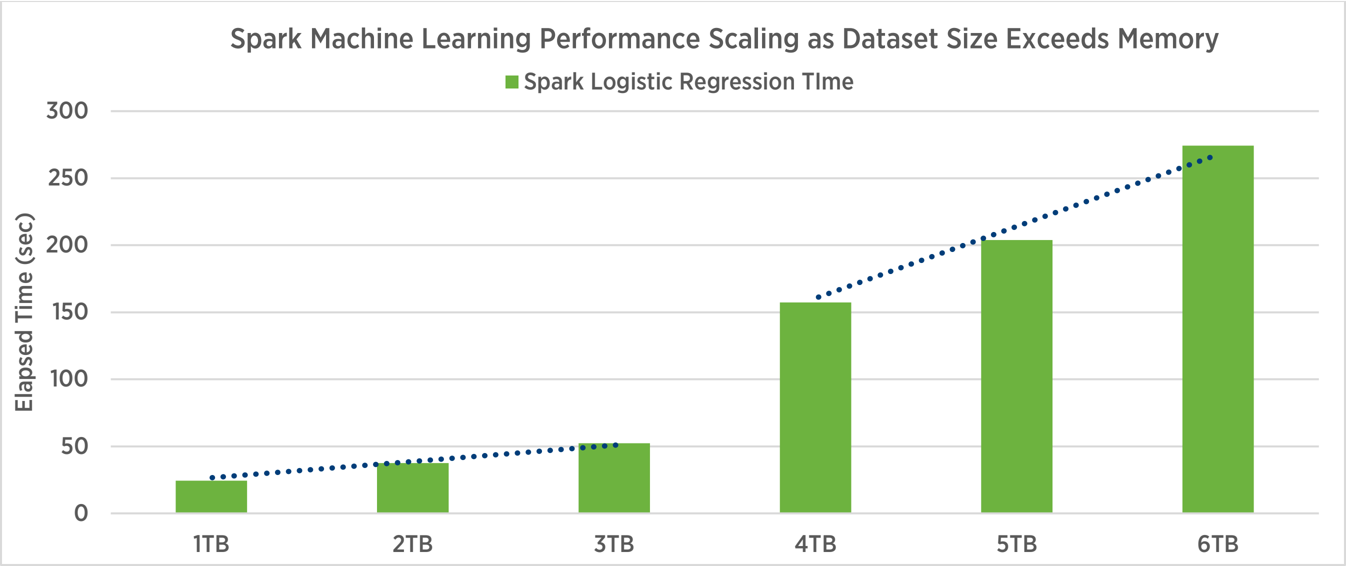


Figure 12. Spark Logistic Regression performance as dataset size exceeds cluster memory

Summary of Best Practices

To recap, here are the best practices cited in this paper:

- Reserve about 5-6% of total server memory for ESXi; use the remainder for the virtual machines.
- Do not overcommit physical memory on any host server that is hosting Big Data workloads.
- Create one or more virtual machines per NUMA node.
- Limit the number of disks per DataNode to maximize the utilization of each disk: 4 to 6 is a good starting point.
- Use eager-zeroed thick VMDKs along with the ext4 or xfs filesystem inside the guest.
- Use the VMware Paravirtual SCSI (pvscsi) adapter for disk controllers; use all 4 virtual SCSI controllers available in vSphere 6.5.
- Use the vmxnet3 network driver; configure virtual switches with MTU=9000 for jumbo frames.
- Configure the guest operating system for Hadoop performance including enabling jumbo IP frames, reducing swappiness, and disabling transparent hugepage compaction.
- Place Hadoop primary roles, ZooKeeper, and journal nodes on three virtual machines for optimum performance and to enable high availability.
- Dedicate the worker nodes to run only the HDFS DataNode, YARN NodeManager, and Spark Executor roles.
- Run the Hive Metastore in a separate MySQL database.
- Set the Yarn cluster container memory and vcores to slightly overcommit both resources.
- Adjust the task memory and vcore requirement to optimize the number of maps and reduces for each application.

Conclusion

Virtualization of Big Data clusters with vSphere brings advantages that bare metal clusters don't have including better utilization of cluster resources, enhanced operational flexibility, and ease of deployment and management. From a performance point of view, enhanced NUMA locality and a better match of disk drives per HDFS DataNode more than outweigh the extra memory required for virtualization. Through application of best practices throughout the hardware and software stack, it is possible to utilize all the benefits of virtualization without sacrificing performance in a production cluster.

These best practices were tested in a highly available 13 server cluster running Hadoop MapReduce version 2 and Spark on YARN with the latest cluster management tools. Similarly configured virtualized platforms including 1, 2, and 4 VMs per host as well as a bare metal platform were installed on the 10 cluster worker servers.

The optimal virtualization point was seen to be 4 VMs on each 2-processor, 16 data disk host. This configuration ensured that the VMs would fit within NUMA node boundaries for fastest memory access, and the number of data disks per VM (4) was in the optimal range of 4–6 per HDFS DataNode for utilization. Next fastest was 2 VMs per host, which also fit within NUMA node boundaries, but at 8 data disks per VM left the disks somewhat underutilized. The 1 VM per host platform was the slowest since it neither exhibits NUMA locality nor good match of disks per DataNode.

Input/output-dominated applications such as TeraGen and TestDFSIO ran significantly faster on the 4 VMs per host platform than on bare metal. TeraSort ran faster virtualized on smaller datasets but at 10 TB the bare metal advantage of larger memory overcame the disadvantage of NUMA misses, so bare metal became faster. Spark Machine Learning Library applications ran up to 60% faster virtualized than on bare metal due to the virtualized advantage of having multiple nodes per physical server.

The large server memory (512 GiB) was put to use by running very large blocksize Hadoop applications for the best performance. The flash disks provided both very fast random reads and writes for YARN NodeManager traffic (on NVMe storage) and fast large sequential reads and writes for HDFS DataNode traffic (on SSDs).

References

[1] Jeff Buell. (2015, February) Virtualized Hadoop Performance with VMware vSphere 6.
http://www.vmware.com/files/pdf/techpaper/Virtualized-Hadoop-Performance-with-VMware-vSphere6.pdf

[2] Dave Jaffe. (2016, August) Big Data Performance on VMware vSphere 6.
https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/bigdata-perf-vsphere6.pdf

[3] The Apache Software Foundation. (2015, June) Apache Hadoop NextGen MapReduce (YARN).
https://hadoop.apache.org/docs/r2.7.1/hadoop-yarn/hadoop-yarn-site/YARN.html

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

[5] Cloudera. (2017) Performance Management.
http://www.cloudera.com/documentation/enterprise/latest/topics/admin_performance.html

[6] Cloudera. (2017) Cloudera Enterprise 5.10.x Documentation.
https://www.cloudera.com/documentation/enterprise/5-10-x.html

[7] VMware, Inc. (2013) VMware Storage Policy API 6.0.
https://code.vmware.com/apis/32/storage-policy

[8] GitHub, Inc. (2016, June) spark-bench.
https://github.com/SparkTC/spark-bench

About the Author

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

Acknowledgements

The author would like to thank Justin Murray, Technical Marketing Manager for Big Data at VMware for many valuable discussions, and Paul Cao and Hai-Fang Yun of Hewlett Packard Enterprise for much technical advice on HPE servers.

VMware, Inc. 3401 Hillview Avenue Palo Alto CA 94304 USA Tel 877-486-9273 Fax 650-427-5001 www.vmware.com

Comments on this document: https://communities.vmware.com/docs/DOC-32739