Lessons Learned: Performance Tuning for Hadoop Systems

Manan Trivedi and Raghunath Nambiar(✉)

Cisco Systems, Inc., 275 East Tasman Drive, San Jose, CA 95134, USA
{matrived,rnambiar}@cisco.com

Abstract. Hadoop has become a strategic data platform for by mainstream enterprises, adopted because it offers one of the fastest paths for businesses take to unlock value from big data while building on existing investments. Hadoop is a distributed framework based on Java that is designed to work with applications implemented using MapReduce modeling. This distributed framework enables the platform to pass the load to thousands of nodes across the whole Hadoop cluster. The nature of distributed frameworks also allows node failure without cluster failure. The Hadoop market is predicted to grow at a compound annual growth rate (CAGR) over the next several years. Several tools and guides describe how to deploy Hadoop clusters, but very little documentation tells how to increase performance of Hadoop clusters after they are deployed. This document provides several BIOS, OS, Hadoop, and Java tunings that can increase the performance of Hadoop clusters. These tunings are based on lessons learned from Transaction Processing Performance Council Express (TPCx) Benchmark HS (TPCx-HS) testing on a Cisco UCS® Integrated Infrastructure for Big Data cluster. TPCx-HS is the industry's first standard for benchmarking big data systems. It was developed by TPC to provide verifiable performance, price-to-performance, and availability metrics for hardware and software systems that use big data.

Keywords: Hadoop · Tuning · Industry standard · TPCx-HS

1 Introduction

Big data is expected to fuel the next industrial revolution. An early sign is the wide adoption of big data technologies across major market sectors, including agriculture, education, entertainment, finance, healthcare, manufacturing, transportation, and government. According to IDC, the big data technology and services market experienced six times the growth rate of the overall information and communications technology market in 2015 [1]. This market is expected to be US$34 billion in 2017, and it is expected to be directly and indirectly responsible for US$300 billion in worldwide IT spending. This exponential growth in big data is fueled primarily by several open-source software initiatives and industry-standard infrastructure solutions.

The most prominent software platform by far is Hadoop. In fact, Hadoop and big data are often considered synonymous. Hadoop adaption is predicted to grow at a compound annual growth rate (CAGR) over the next several years across major industry

R. Nambiar and M. Poess (Eds.): TPCTC 2016, LNCS 10080, pp. 121–141, 2017.
DOI: 10.1007/978-3-319-54334-5_9

vertical markets as a mainstream data management platform. Several tools and guides describe how to deploy Hadoop clusters, but very little documentation tells how to increase the performance of Hadoop clusters after they are deployed.

This document explains several BIOS, OS, Hadoop, and Java tunings that can increase the performance of Hadoop clusters. These tunings are based on lessons learned from Transaction Processing Performance Council Express (TPCx) Benchmark HS (TPCx-HS) testing. The tests were conducted on a Cisco UCS® Integrated Infrastructure for Big Data cluster, an industry-leading platform for enterprise Hadoop deployments. However, these tuning parameters are applicable across most Hadoop deployments.

This document also presents the results of tests addressing eight of the most frequently asked questions in tuning Hadoop systems. All test results reported are based on fully compliant TPCx-HS testing based on the specification, but they have not been audited or published.

2 TPC Express Benchmark HS

TPCx-HS is the industry's first standard for benchmarking big data systems. It is designed to provide verifiable performance, price-to-performance, and availability metrics for hardware and software systems that use big data [2, 3].

TPCx-HS can be used to assess a broad range of system topologies and implementation methodologies for Hadoop in a technically rigorous and directly comparable, vendor-neutral manner. And although modeling is based on a simple application, the results are highly relevant to big data hardware and software systems.

TPCx-HS benchmarking has three steps:

- HSGen: Generates data and retains it on a durable medium with three-way replication
- HSSort: Samples the input data, sorts the data, and retains the data on a durable medium with three-way replication
- HSValidate: Verifies the cardinality, size, and replication factor of the generated data

The TPCx-HS specification mandates two consecutive runs to demonstrate repeatability, as depicted in Fig. 1, and the lower value is used for reporting [4].

TPCx-HS uses three main metrics:

- HSph@SF: Composite performance metric, reflecting TPCx-HS throughput, where SF is the scale factor
- $/HSph@SF: Price-to-performance metric
- System availability date

TPCx-HS also reports the following numerical quantities:

- T_G: Data generation phase completion time, with HSGen reported in hh:mm:ss format
- T_S: Data sort phase completion time, with HSSort reported in hh:mm:ss format
- T_V: Data validation phase completion time, reported in hh:mm:ss format

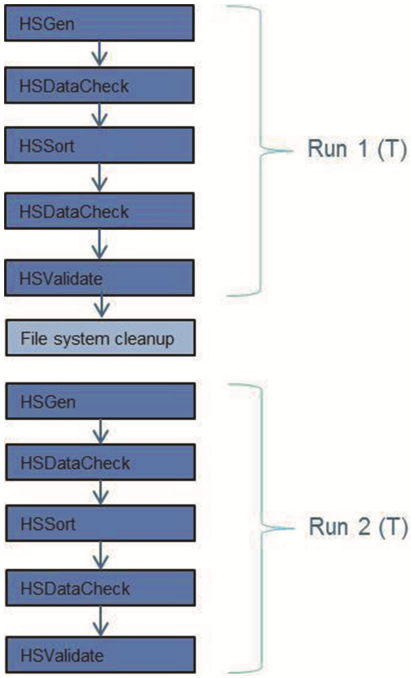

Fig. 1. TPCx-HS benchmark processing

The primary performance metric of the benchmark is HSph@SF, the effective sort throughput of the benchmarked configuration. Here is an example (using the summation method):

$$HSph@SF = \left\lfloor \frac{SF}{(T/3600)} \right\rfloor$$

Here, SF is the scale factor, and T is the total elapsed time for the run in seconds.

The price-to-performance metric for the benchmark is defined as follows:

$$\$/HSph@SF = \frac{P}{HSph@SF}$$

Here, P is the total cost of ownership (TCO) of the system under test (SUT).

The system availability date indicates when the system under test is generally available as defined in the TPC-Pricing specification.

3 System Under Test: Cisco UCS Integrated Infrastructure for Big Data

The tests were conducted on a Cisco UCS Integrated Infrastructure for Big Data cluster with 16 Cisco UCS C240 M4 Rack Servers. The Cisco UCS Integrated Infrastructure for Big Data is built using the following components:

- Cisco UCS 6296UP 96-Port Fabric Interconnect: Fabric interconnects are central to the Cisco Unified Computing System™ (Cisco UCS). They provide low-latency, lossless 10 Gigabit Ethernet, Fibre Channel over Ethernet (FCoE), and Fibre Channel functions with management capabilities for the system. All servers attached to fabric interconnects become part of a single, highly available management domain.
- Cisco UCS C240 M4 Rack Server: Cisco UCS C-Series Rack Servers extend Cisco UCS in standard rack-mount form factors. The Cisco UCS C240 M4 Rack Server is designed to support a wide range of computing, I/O, and storage-capacity demands in a compact design. It supports two Intel® Xeon® processor E5-2600 v4 series CPUs, up to 768 GB of memory, and 24 small-form-factor (SFF) disk drives plus two internal SATA boot drives and Cisco UCS Virtual Interface Card (VIC) 1227 adapters.

The Cisco UCS Integrated Infrastructure for Big Data cluster configuration consists of two Cisco UCS 6296UP fabric interconnects, 16 Cisco UCS C240 M4 servers with two Intel Xeon processor E5-2600 v4 series CPUs, 256 GB of memory, and 24 SFF disk drives plus two internal SATA boot drives and Cisco UCS VIC 1227 adapters, as shown in Fig. 2. Table 1 lists the software versions used.

16 x Cisco UCS C240 M4 Servers (Data Nodes) with:
24 × 1.2-TB 6-Gbps SAS 10,000-rpm SFF HDD
2 × 120-GB 2.5-Inch Enterprise Value 6-Gbps SATA SSD (Boot)
10 Gigabit Ethernet
16 × 10 Gigabit Ethernet
2 x Cisco UCS 6296UP fabric interconnect
1 x Cisco Nexus® 9372PX Switch

Table 1. Software versions

Layer	Component	Version or Release
Computing	Cisco UCS C240 M4 server	Release C240M4.2.0.10c
Network	Cisco UCS 6296UP fabric interconnect	Release UCS 3.1(1 g)A
	Cisco UCS VIC 1227 firmware	Release 4.1(1d)
	Cisco UCS VIC 1227 driver	Release 2.3.0.18
Software	Red Hat Enterprise Linux (RHEL) server	Version 6.5 (x86_64)
	Cisco UCS Manager	Release 3.1(1 g)
Hadoop	Cloudera Enterprise	Version 5.3.2

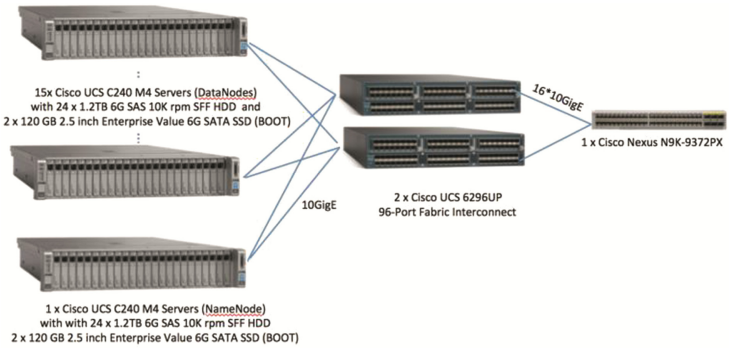

Fig. 2. Cisco UCS integrated infrastructure for big data cluster configuration

4 Performance Tuning

Many factors come into play when tuning a system as complex as big data systems. Performance tuning involves making modifications to hardware, software, and network parameters.

This section lists parameters that can be tuned at the infrastructure, operating system, and Hadoop levels.

Infrastructure

Infrastructure tuning helps achieve optimal utilization of resources. It also helps the application run faster and perform better.

- Server
  - BIOS
    - CPU parameters
    - Intel Turbo Boost Technology
    - Intel Hyper-Threading Technology
    - Prefetcher
    - C-states
    - Power control policy
    - Memory tuning
- Network
  - Network tuning parameters
  - Network interface card (NIC) bonding
  - Jumbo frame (maximum transmission unit [MTU])
  - Quality-of-service (QoS) settings
- Storage
  - RAID 0
    - Write back
    - Read ahead
    - Stripe size

  - JBOD
  - JBOD Versus RAID 0

Operating System
OS performance tuning is used to manage and improve resources that respond to individual requests. OS scalability is managed by monitoring the resource consumption of varying volumes of requests, from low to very high, by changing default OS settings.

- File system
  - XFS
  - Agcount
  - Mount
  - Fstab
- Post-OS tuning
  - sysctl.conf
  - limits.conf
  - CPU frequency and scaling governor
  - Transparent huge pages
  - Linux swappiness
  - I/O scheduler

Hadoop
In addition to tuning the infrastructure and OS, you need to tune Hadoop settings for best performance. Hadoop tuning can have a significant impact on the overall performance of your Hadoop cluster.

- Hadoop
  - Hadoop Distributed File System (HDFS)
    hdfs-site.xml
  - MapReduce
    Java Virtual Machine (JVM) reuse
    Compression
    mapred-site.xml
    core-site.xml

5 Performance Tuning in Detail

This section describes the infrastructure, OS, and Hadoop tuning parameters in detail.

Server Tuning
Hadoop is based on a new approach to storing and processing complex data, with data movement reduced. Hadoop distributes across the cluster the data that each machine in a Hadoop cluster stores, and it also processes the data. Therefore, it is important to tune the processing, or computing, aspect of the system to achieve optimal performance from the cluster.

BIOS settings can have a significant performance impact, depending on the workload and the applications. Table 2 lists the optimal CPU settings for Hadoop based on the tests reported in this document.

Table 2. Optimal CPU settings

Parameter	Setting
Intel Turbo Boost	Enabled
Enhanced Intel SpeedStep	Enabled
Intel Hyper-Threading	Enabled
Core Multiprocessing	All
Executive Disabled Bit	Platform default
Virtualization Technology	Disabled
Hardware Prefetcher	Enabled
Adjacent Cache Line Prefetcher	Enabled
Data Cache Unit (DCU) Streamer Prefetcher	Enabled
DCU IP Prefetcher	Enabled
Direct Cache Access	Enabled
Processor C-State	Disabled
Processor CIE	Disabled
Processor C3 Report	Disabled
Processor C6 Report	Disabled
Processor C7 Report	Disabled
CPU Performance	Enterprise
Maximum Variable Mean Time to Replace or Repair (MTRR) Setting	Platform default
Local x2APIC Advanced Programmable Interrupt Controller	Platform default
Power Technology	Performance
Energy Performance	Performance
Frequency Floor Override	Enabled
P-State Coordination	Hw-all
DRAM Clock Throttling	Performance
Channel Interleaving	Platform default
Rank Interleaving	Platform default
Demand Scrub	Disabled
Patrol Scrub	Disabled
Altitude	Platform default
Package C-State Limit	Platform default

Table 3 lists optimal memory settings for Hadoop based on the tests reported here.

Table 3. Optimal memory settings for Hadoop

Parameter	Setting
Memory RAS Configuration	Maximum performance
NUMA	Enabled
Low-Voltage Double Data Rate (LV DDR) Mode	Performance mode
DRAM Refresh Rate	1 time
DDR3 Voltage Selection	Platform default

Table 4. Optimal network tuning parameters for Hadoop

Parameter	Tuned value	Description
net.core.somaxconn	1024	Changing the net.core.somaxconn Linux kernel settings from the default of 128 to 1024 helps with burst requests from the name node and job tracker. This option sets the size of the listening queue, or the number of connections that the server can set up at one time.
net.ipv4.tcp_retries2	5	This variable helps forward the packets between interfaces. This variable is special; its change resets all configuration parameters to their default state.
net.ipv4.ip_forward	0	IP forwarding is disabled in most Linux distributions because most of them do not set up a Linux router, gateway, VPN server, or dial-in server.
net.ipv4.conf.default.rp_filter	1	This value influences the timeout behavior of a live TCP connection.
net.ipv4.conf.all.rp_filter	1	This value enables route verification on all interfaces.
net.ipv4.conf.default.accept_source_route	0	This setting does not accept source routing.
net.ipv4.tcp_syncookies	1	This setting enables the use of TCP SYN cookies.
net.ipv4.conf.all.arp_filter	1	
net.ipv4.tcp_mtu_probing	1	If there are multiple network interfaces on different IP addresses, this setting will help achieve the desired results.
net.ipv4.icmp_echo_ignore_broadcasts	1	This setting controls TCP packetization layer path MTU discovery. It is disabled by default, and it is enabled when an Internet Control Message Protocol (ICMP) black hole is detected.
net.ipv4.conf.default.promote_secondaries	1	These settings prevent deletion of secondary IP addresses when the primary IP address is deleted.
net.ipv4.conf.all.promote_secondaries	1	
net.core.rmem_max	16777216	These settings increase the TCP maximum buffer size. The four options shown here increase the TCP send and receive buffers, allowing an application to move its data out faster so it can serve other requests. This adjustment also improves the client's ability to send data to the server when it gets busy.
net.core.wmem_max	16777216	
net.ipv4.tcp_rmem	4096 87380 16777216	
net.ipv4.tcp_wmem	4096 65536 16777216	
net.core.netdev_max_backlog	10000	
net.core.netdev_max_backlog	10000	

Network Tuning

The impact of the network on big data is enormous. An efficient and resilient network is a crucial part of a good Hadoop cluster because the network is what connects all the nodes. The network is also crucial for writing data, reading data, and signaling and for HDFS operations and operations of the MapReduce infrastructure. Therefore, the failure of a networking device can have dire affects. A job may need to be restarted, or a workload may be pushed to the remaining nodes, resulting in delay. Therefore, networks must be designed to provide redundancy, with multiple paths between computing nodes, and they must be able to scale.

Table 4 lists some network performance settings that can increase Hadoop performance. These options increase the read and write cache sizes for the network stack. These parameters can be tested with the **systctl –w** command or made permanent by adding the variable to the /etc./sysctl.conf file.

You can tune NIC bonding. A NIC is a computer hardware component that connects a computer to a computer network. Network bonding is a method of combining (joining) two or more network interfaces together into a single interface. This combination increases network throughput and provides redundancy. If one interface is down or unplugged, the remaining interfaces will keep the network traffic up and alive. Network bonding can be used in situations in which you need redundancy, fault tolerance, or load balancing.

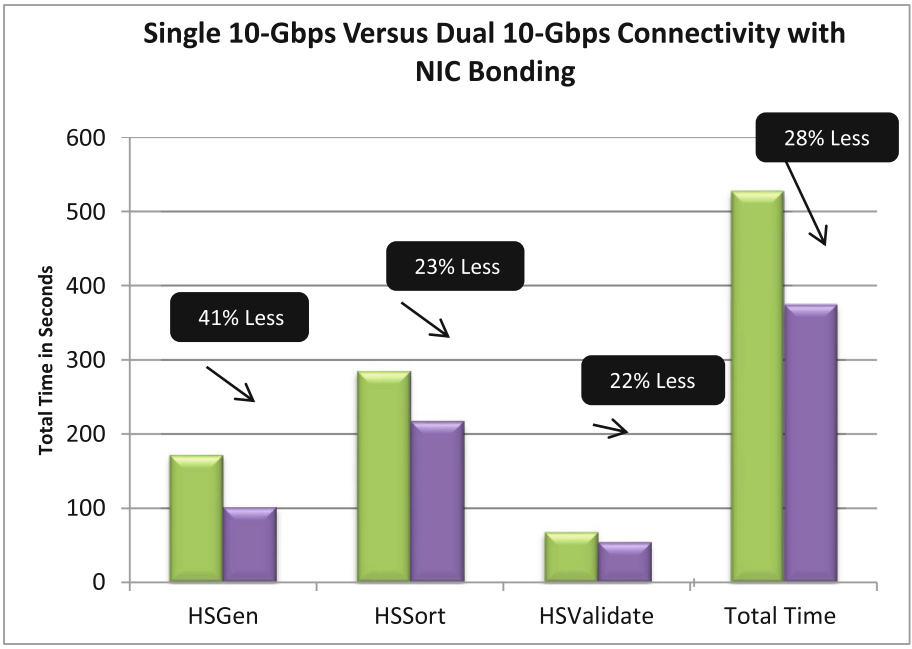

Fig. 3. **Single** 10-Gbps Versus **Dual 10**-Gbps Connectivity with NIC Bonding

Linux allows bonding of multiple network interfaces into a single channel using a special kernel module called a bonding module. The Linux bonding driver provides a method for aggregating multiple network interfaces into a single logical "bonded" interface. The behavior of the bonded interface depends on the mode. In general, the mode provides either hot-standby or load-balancing services. Additionally, link-integrity monitoring can be performed.

Test Result 1: 10-Gbps Versus Dual 10-Gbps Connectivity with NIC Bonding
One frequently asked question relates to the impact of NIC bonding for Hadoop. In older-generation servers, single 10-Gbps connectivity was sufficient. Since the introduction of Cisco UCS C240 M4 servers (based on Intel Xeon processor 2600 v3 CPUs) with 24 SFF disks drives, we have observed significant performance improvements with NIC bonding. In other words, Hadoop nodes can use more than 10-Gbps network bandwidth (Fig. 3).

Table 5. Single 10-Gbps versus **Dual 10**-Gbps **with NIC Bonding**

Phase	No Bonding (Time in Seconds)	2-NIC Bonding (Time in Seconds)	Percentage improvement
HSGen	173	102	41.0%
HSSort	286	218	23.7%
HSValidate	69	55	22.2%
Total Time	528	375	28.9%
HSph@SF at 1-TB Scale Factor	6.72	9.45	

Table 5 lists detailed response times for each benchmark phase.

Test Result 2: 1500 Versus 9000 Maximum Transmission Unit

One the most commonly tuned parameters is the MTU, which defines the largest packet size that an interface can transmit without the need to fragment the packet. IP packets larger than the MTU require IP fragmentation.

The use of jumbo frames (an MTU value of 9000) improves performance because jumbo frames reduce the number of individual frames that must be sent for a given amount of data, and they reduce the need to separate data blocks into multiple Ethernet frames. They also reduce host and storage CPU utilization.

Figure 4 shows the performance improvement with a larger MTU (9000).

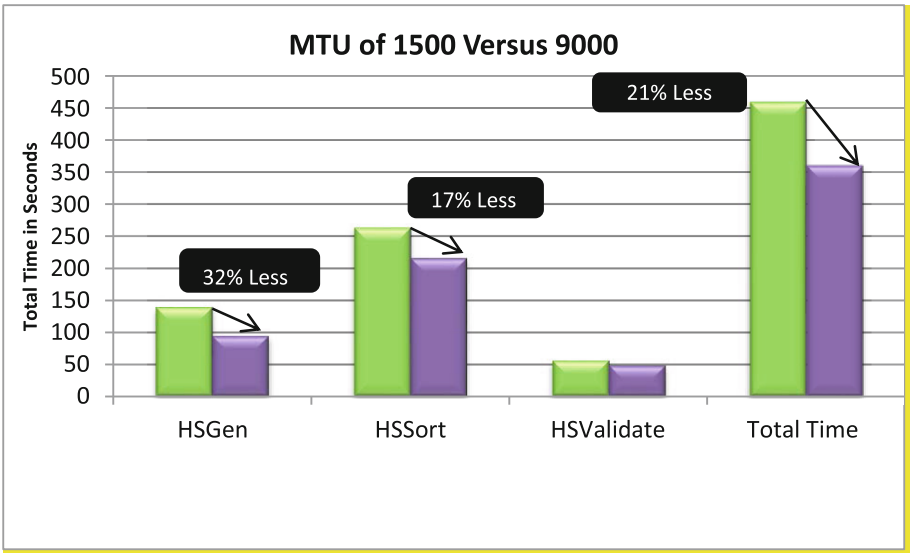

Fig. 4. MTU of 1500 Versus **9000**

Table 6 lists detailed response times for each benchmark phase.

Table 6. MTU of 1500 versus 9000

Phase	Bonding (Multiple NICs at 1500 MTU)	Bonding (Multiple NICs at 9000 MTU)	Percentage improvement
HSGen	140	95	32.1%
HSSort	264	217	17.8%
HSValidate	56	49	12.5%
Total Time	460	361	21.5%
HSph@SF at 1-TB Scale Factor	7.71	9.81	

Test Result 3: Two-vNIC Bonding Versus Three-vNIC Bonding
Cisco UCS VIC technology supports up to 256 virtual NICs (vNICs). Tests with three vNICs provided slight performance improvement, as shown in Fig. 5.

[[PLS CHANGE THE CALLOUTS AS FOLLOWS:]]
Two-vNIC Bonding Versus Three-vNIC Bonding
Time in Seconds
(2 NICs)
(Multiple NICs)

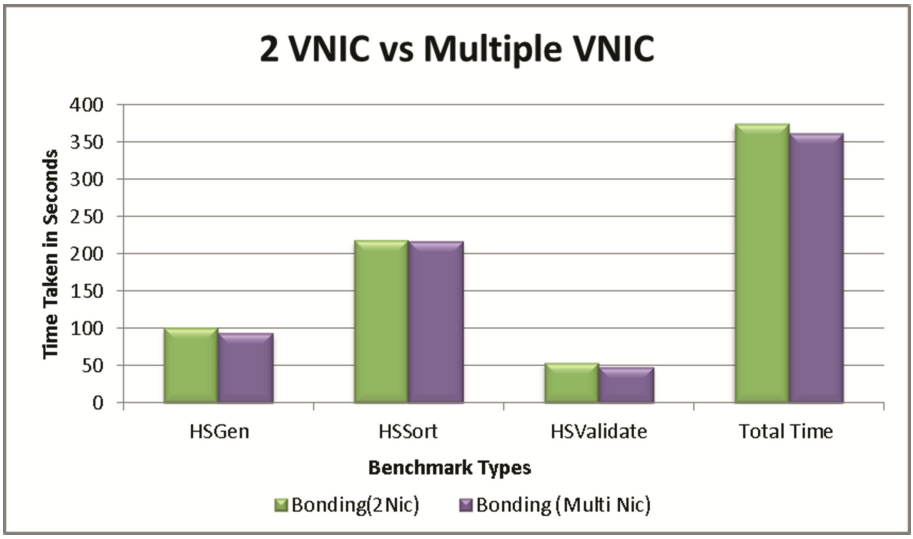

Fig. 5. Two-vNIC Bonding versus Three-vNIC Bonding

Table 7 lists detailed response times for each benchmark phase.

Storage Tuning
Optimal configuration of the storage system is extremely important to achieve the best application performance. In most cases, servers with internal direct-attached storage (DAS) provide the best performance and price-to-performance ratios. Two popular storage controller options are RAID controllers and host bus adapters (HBAs). In

Table 7. Two-vNIC Bonding versus Three-vNIC Bonding

Phase	Bonding (2 NICs)	Bonding (Multiple NICs)	Percentage improvement
HSGen	102	95	6.86%
HSSort	218	217	0.45%
HSValidate	55	49	10.9
Total Time	375	361	3.73%
HSph@SF at 1-TB Scale Factor	9.45	9.81	

addition to RAID functions, RAID controllers offer advanced self-monitoring, analysis, and reporting technology (SMART) features and write-back or flash-based write cache. SMART features detect and report the health of the disk drives beyond the capabilities of JBOD. Caching can improve data load performance in Hadoop deployments. This section describes best practices based on the tests conducted on the Cisco UCS Integrated Infrastructure for Big Data cluster.

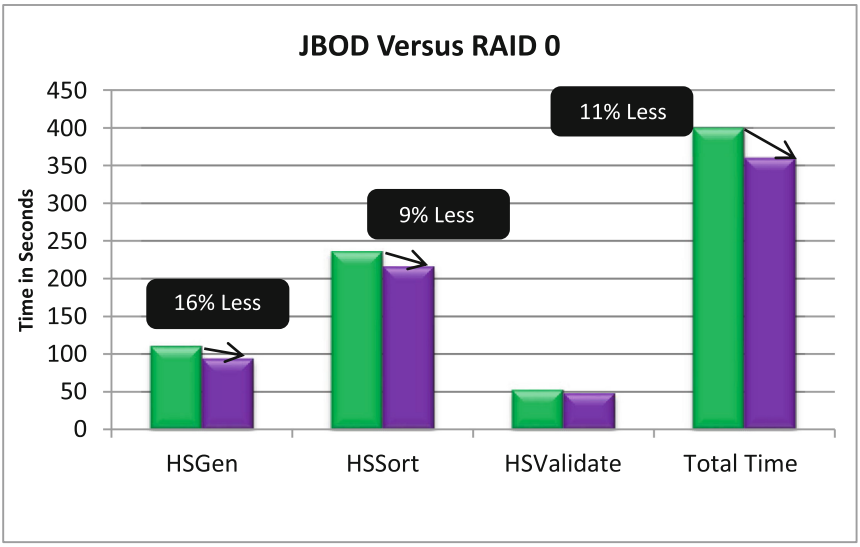

Fig. 6. JBOD Versus RAID 0

Table 8 lists optimal settings for the Cisco 12-Gbps SAS modular RAID controller for Hadoop deployments.

Test Result 4: JBOD Versus RAID

JBOD and RAID 0 work similarly. The main difference pertaining to performance is the effect of controller caching. Figure 6 shows better performance with RAID than with JBOD. The controller cache (a 2-GB module was used in these tests) optimizes writeback operations when the workload is based on large sequential read and write processing.

Table 9 lists detailed response times for each benchmark phase.

Table 8. Optimal RAID controller settings for Hadoop

Parameter	Setting
RAID	RAID 0 for individual disk drives
Controller Cache	Always write back NoCacheBadBBU Read ahead
Stripe Size	1024 KB
Disk Drive Cache	Enabled (read) (Cisco firmware does not allow the write cache to be enabled on disk drives.)

Table 9. JBOD versus RAID 0

Phase	JBOD	RAID 0	Percentage improvement
HSGen	111	95	16.84%
HSSort	237	217	9.22%
HSValidate	53	49	8.16%
Total Time	401	361	11.08%
HSph@SF at 1-TB Scale Factor	8.82	9.81	

Operating System Tuning

Changing some system settings in Linux can increase overall performance. This section discusses these changes and their benefits. Table 10 lists some of the OS performance settings best for Hadoop.

Table 10. Operating system settings

Parameter	Value
vm.dirty_background_ratio	1
vm.swappiness	0
vm.overcommit_memory	0
net.core.rmem_max	16777216
net.core.wmem_max	16777216
net.core.netdev_max_backlog	10000

In addition, the following settings for /etc./security/limits.conf are recommended:

- root soft nofile 64000
- root hard nofile 64000
- hadoop soft nproc 32768
- hadoop hard nproc 32768
- hadoop soft nofile 32768
- hadoop hard nofile 32768

File System Tuning

Different Linux distributions use different default file systems. Testing has shown that XFS seems to be better than Ext3 or Ext4 for Hadoop. XFS is a high-performance

journaling file system that was initially created by Silicon Graphics for the IRIX operating system and later ported to Linux. XFS has a large number of features that make it suitable for deployment in an enterprise-level computing environment that requires implementation of very large file systems.

XFS has very bad performance out of the box. Unlike with Ext4, the file system needs to be formatted with the right parameters to perform well. And if you don't specify the parameters correctly, you need to reformat the file system because you can't change the parameters later. The main parameter that the tests reported here found useful to tune is **agcount**: the number of allocation groups. Allocation groups enable simultaneous I/O processing by multiple application threads. XFS splits the file system into multiple allocation groups to help increase parallelism, because each allocation group has its own set of locks. It is important to create as many allocation groups as you have hardware threads. If the server has a dual CPU configuration with 16 cores and 32 threads with hyperthreading, an **agcount** value of 32 is recommended for best I/O performance.

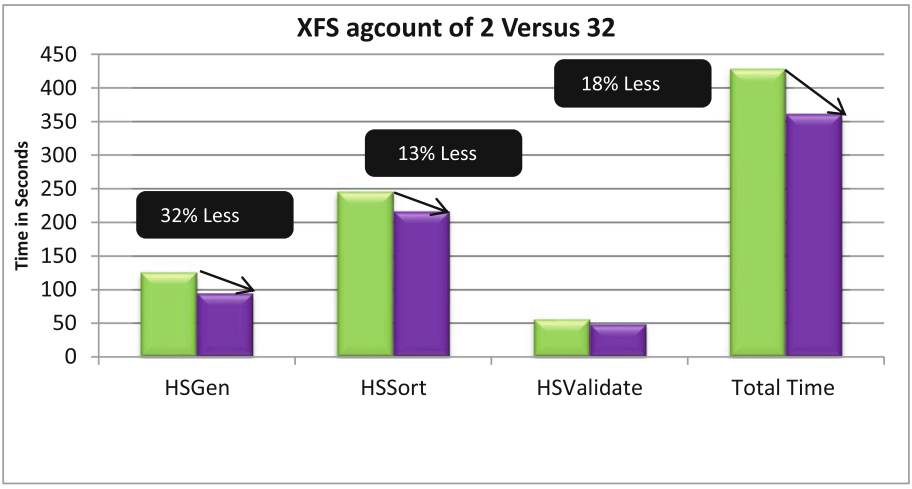

Fig. 7. XFS agcount of 2 Versus 32

XFS supports several mount options that can influence behavior. XFS allocates inodes according to their on-disk locations by default. However, because some 32-bit user-space applications are not compatible with inode numbers greater than 2^{32}, XFS allocates all inodes in disk locations that result in 32-bit inode numbers. This behavior can lead to decreased performance on very large file systems (systems larger than 2 terabytes [TB]), because inodes are skewed toward the beginning of the block device, and data is skewed toward the end. To address this scenario, the **inode64** mount option is recommended.

Linux records information about the time when files were created, last modified, and last accessed. There is a cost associated with recording the last access time. The **noatime** attribute tells the file system not to record the last-accessed time for the file and is recommended for Hadoop deployments.

Test Result 5: XFS with agcount of 2 Versus 32
Tests for conducted with allocation groups of 2 and 32. As shown in Fig. 7, an optimal allocation count is critical for optimizing XFS for Hadoop.

Table 11 lists detailed response times for each benchmark phase.

Table 11. XFS agcount of 2 versus 32

Phase	Agcount = 2	Agcount = 32	Percentage improvement
HSGen	126	95	32.63%
HSSort	246	217	13.36%
HSValidate	56	49	14.29%
Total Time	428	361	18.56%
HSph@SF at 1-TB Scale Factor	8.27	9.81	

Another important OS setting is the CPU frequency and scaling governor (Table 12). The performance mode is recommended for high-performance Hadoop deployments.

Table 12. CPU Governor options in Linux

Governor	Description
ondemand	Dynamically switch between CPUs available if 95% CPU load is reached.
performance	Run the CPU at maximum frequency. This mode is recommended for high-performance Hadoop deployments.
conservative	Dynamically switch between CPUs available if 75% CPU load is reached.
powersave	Run the CPU at the minimum frequency.
userspace	Run the CPU at user-specified frequency.

Transparent huge pages is a commonly used option that works well in most instances, including with Hadoop. However, a problem arises with one feature of transparent huge pages called compaction. This feature defragments memory at the cost of CPU cycles. Testing has shown better performance with compaction disabled. This option can be set with the following command:

```
# echo never > /sys/kernel/mm/redhat_transparent_hugepages/defrag
```

Linux swappiness is a kernel process that finds memory content that has not been used in a while and copies it to the hard drive. The swappiness value can be adjusted from 0 to 100. In most versions of Linux, the default value is 60. The tests reported here show that turning off swappiness (setting swappiness to 0) is optimal for Hadoop deployments. This option can be set with the following command:

```
sysctl -w vm.swappiness=0
```

The I/O scheduler is another important performance tuning option. The recommended I/O scheduler setting for Hadoop is Completely Fair Queuing (CFQ). CFQ is the default setting in some Linux distributions, and it can increase performance by 2 or 3 percent. This option can be set with the following command:

```
echo cfq > /sys/block/sd*/queue/scheduler
```

Hadoop Tuning

Out of the box, many Hadoop settings are not optimized for best performance. HDFS provides storage for all the data and is a core component of Hadoop. Fine-tuning the settings here can produce significant performance improvements. The settings discussed in this section have been tested and will provide improved speed for heavy workloads.

The Hadoop block size defines the number of input splits for a file. Each input split is replicated three times (by default) across the cluster. Map tasks typically operate on these input splits. The number of input splits determines the number of map tasks.

The total read time on hard disk drives consists of seek time (finding the first block of the file) and transfer time (the time needed to read contiguous blocks of data). When dealing with hundreds of terabytes or petabytes of data, these times become significant. Hadoop handles this processing by having lots of map tasks reading and writing data in parallel. However, processing can benefit by limiting the number of tasks running in parallel, because having too many map tasks trying to read and write data is inefficient. The best approach is a balanced number of input splits and map jobs, because having too few map jobs also reduces performance, just as does having too many.

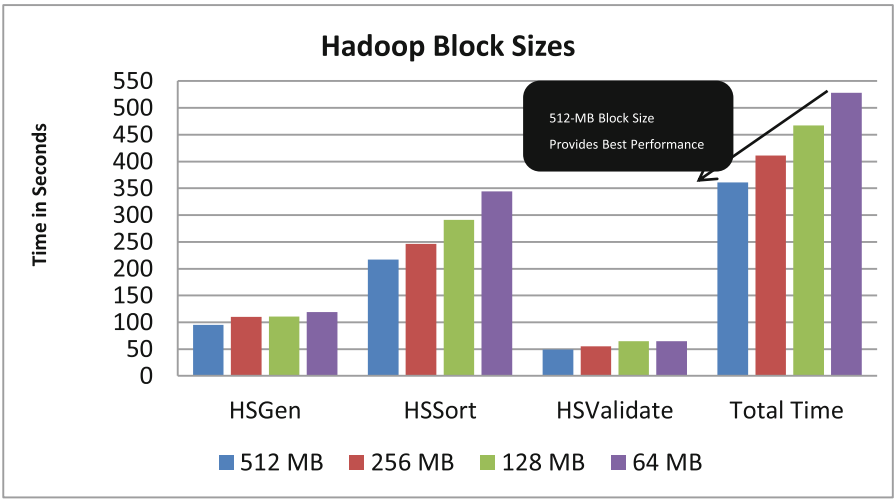

Fig. 8. Impact of Block Sizes

The recommended balance uses this calculation:

Number of launched map tasks = Total size/Input split size (or block size)

Using this formula, for a 1-TB data set with a 64-MB block size, Hadoop would run 15,120 map tasks; with a 512-MB block size, it would run 2160 map tasks.

Test Result 6: HDFS Block Sizes

Tests were conducted with block sizes of 64, 128, 256, and 512 MB. As shown in Fig. 8, 512 MB provided the best performance for the TPCx-HS benchmark. Additional tests conducted with customer workloads reached the same conclusion: that for MapReduce-based applications, larger block sizes provide the best performance.

The configuration is set in hdfs-site.xml as shown in Table 13.

Table 13. hdfs-site.xml Settings

Parameter	Value
dfs.blocksize	512 MB
dfs.datanode.drop.cache.behind.writes	True
dfs.datanode.sync.behind.writes	True
dfs.datanode.drop.cache.behind.reads	True

The general rule for memory tuning is to use as much memory as you can without triggering swapping. The parameter **mapred.*.child.java.opts** can be used to set the task memory. The recommended heap size for both map and reduce tasks is 2 GB, and **ulimit** was set to 4 GB (double the heap size used by all JVM processes) for this workload.

Another important tuning option is to reduce the map disk spill. Mappers generate intermediate data output, which is stored in a memory buffer that is determined by the **io.sort.mb** parameter. This chunk of memory is part of the map JVM heap space. As soon as the threshold is reached (**io.sort.spill.percent**), the content is written to the local disk. This content is called spill. To store the record, the Hadoop framework uses the **io.sort.record.percent** value of the memory allocated by **io.sort.mb**. Performance problems occur when you spill records to disk multiple times. The values of the map output records counter and spilled record counters can be checked for each job, and you can allocate the appropriate memory buffer and the **io.sort.spill.percent** value to use nearly full capacity to enhance Hadoop job performance. These are the recommended settings:

```
io.sort.mb = 1024 MB

io.sort.spill.percent = .98
```

The number of mappers and reducers is critical to get the best performance. This configuration is based on a 16-node cluster, with one server configured as the name node and 15 servers configured as data nodes, and each server with two CPUs with a total of 48 threads. A slight oversubscription of the number of mappers and reducers to the number of cores should be used, because reducers typically don't start at the same time as mappers. Given the 48 threads in the system under test, allocate 36 threads for mappers and 30 threads for reducers for each node. (This number will vary based on the scale factor of the workload and the system configuration.) The number of HDFS blocks in the input files usually determines the number of mappers. The tuning goal of mappers should be to control the number of mappers and the size of the job. When dealing with large files, Hadoop splits the file into smaller chunks so that the mapper can run it in parallel. However, initializing the new mapper job usually takes a few seconds, creating

overhead that should be reduced. To determine the optimal number, several iterations were run.. The configuration for **mapred-site.xml** is shown in Table 14.

Table 14. mapred-site.xml Settings

Parameter	Value
Mapred.map.tasks	540
Mapred.reduce.tasks	450
mapred.tasktracker.map.tasks.maximum	36
mapred.tasktracker.reduce.tasks.maximum	30
mapred.map.child.java.opts	-Xmx800 m -Xms800 m -Xmn256 m
mapred.reduce.child.java.opts	-Xmx1200 m -Xmn256 m
mapred.child.ulimit	4096 MB
io.sort.mb	1024 MB
io.sort.factor	64
io.sort.record.percent	0.15
Io.sort.spill.percent	0.98
mapred.job.reuse.jvm.num.tasks	–1
mapred.reduce.parallel.copies	20
mapred.reduce.slowstart.completed.maps	0
tasktracker.http.threads	120
mapred.job.reduce.input.buffer.percent	0.7
mapreduce.reduce.shuffle.maxfetchfailures	10
mapred.job.shuffle.input.buffer.percent	0.75
mapred.job.shuffle.merge.percent	0.95
mapred.inmem.merge.threshold	0
mapreduce.ifile.readahead.bytes	16777216
mapred.map.tasks.speculative.execution	False

Also, in the hdfs-site.xml file, the **io.sort.factor** parameter controls the number of concurrent streams from the map output that are merged and saved to disk. For heavy workloads with many map tasks, this value should be increased from 10 to 64, to increase the number of streams merged at the same time. This setting has been tested and shown to increase performance, but it should be used with caution on other equipment because it could lead to instability by overworking the system.

Under heavy workloads, Hadoop can launch many thousands of jobs, each of which runs for only a short period of time, and each launching a separate JVM. By default, each JVM must be started and torn down every time. Obviously, this approach is inefficient. It can be improved by changing the parameter **mapred.job.reuse.num.tasks** in the mapred-site.xml file. Change this parameter to –1, and JVMs can be reused for an unlimited number of jobs. This change also helps the platform take full advantage of Java's just-in-time (JIT) compilation, because the JVM does not need to be compiled each time.

Compression can significantly improve Hadoop performance by reducing disk I/O processing and network traffic. It also reduces the amount of disk space used. The TPCx-HS requirements enforce the use of uncompressed job output, but intermediate map output compression is allowed. Table 15 lists the recommended compression parameters.

Table 15. Compression Parameters

Parameter	Value	Description
mapred.output.compress	False	Compression allowed for the MapReduce output
mapred.compress.map.output	True	Compression allowed for intermediate map output
mapred.map.output.compression.codec	org.apache.hadoop.io.compress.SnappyCodec	

Another important tuning parameter is file buffer size, a setting in core-site.xml. The recommended setting for the **io.file.buffer.size** parameter is 131072.

Test Result 7: End-to-End I/O and Network Utilization
Sort workloads are popular in the Hadoop space. TPCx-HS enables fair comparisons to be made between software and hardware systems. It also exercises various subsystems. Figure 9 shows disk read, disk write, network read, and network write utilization from one of the nodes for an end-to-end run.

[[PLS CHANGE CALLOUTS AS FOLLOWS:]]
…Resource Utilization Across Various Phases of Job Processing
Peak Write Throughput Is 2.81 GBps
…Is 1.74 GBps
…Peak Write Throughput Is 2.51 GBps
HSValidate Phase
HSSort Phase
Network I/O Send Peak Throughput Is 1.65 GBps, and Receive Peak Throughput Is 1.92 GBps
Network I/O Send Peak Throughput Is 1.65 GBps, and Receive Peak Throughput Is 1.68 GBps
Network I/O Receive
Network I/O Send

As shown in Fig. 9, in the HSGen phase, peak write throughput is 2.81 GBps, which means that each drive is performing at 117 GBps. This equates to $2.81 \times 15 = 42$ GBps write throughput per cluster. During the shuffle phase, aggregate read bandwidth is 26 GBps, and during the reduce phase, aggregate write bandwidth is 38 GBps. The peak network bandwidth utilization was 1.8 GBps: about 75 percent of dual 10 Gbps connectivity.

Test Result 8: End-to-End CPU Utilization
One frequently asked question relates to CPU utilization. Figure 10 shows the CPU utilization for an end-to-end TPCx-HS run. As noted, CPU utilization was about 97 percent peak at the shuffle and sort phase.

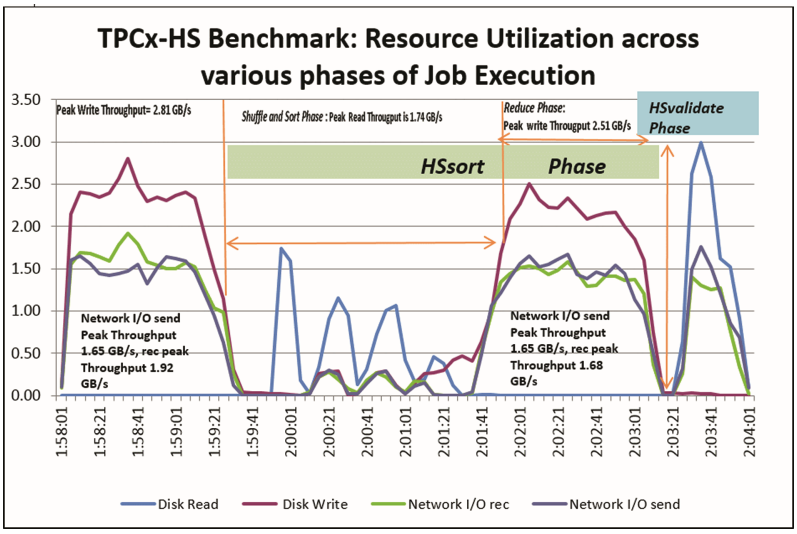

Fig. 9. Resource Utilization across various Job Processing phases

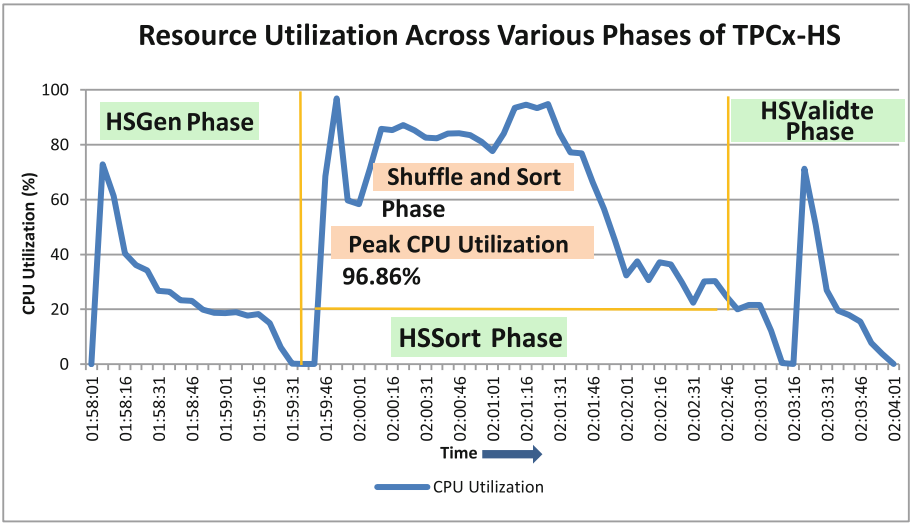

Fig. 10. CPU Utilization Across Various Phases

As observed in the results from tests 8 and 9, the TPCx-HS benchmark exercises the upper boundaries of I/O, network, and CPU processing with Hadoop. This feature makes TPCx-HS a good benchmark standard that enables fair comparison of Hadoop systems, and it also provides a good workload for stress-testing various technologies under development.

6 Conclusion

This document provides a summary of lessons learned from performance tuning for the TPCx-HS benchmark. The tuning parameters and test results have broad applicability across Hadoop-based applications. The test results also address some of the most frequently asked questions about Hadoop system tuning.

References

1. IDC Worldwide Big Data Technology and Services Forecast (2015)
2. Nambiar, R., Poess, M., Dey, A., Cao, P., Magdon-Ismail, T., Da Ren, Q., Bond, A.: Introducing TPCx-HS: the first industry standard for benchmarking big data systems. In: Nambiar, R., Poess, M. (eds.) TPCTC 2014. LNCS, vol. 8904, pp. 1–12. Springer, Cham (2015). doi: 10.1007/978-3-319-15350-6_1
3. Nambiar, R.: A standard for benchmarking big data systems. In: IEEE Big Data Conference, pp. 18–20 (2014)
4. TPCx-HS specification. http://www.tpc.org/tpcx-hs/