# Performance Characterization of Big Data Systems with TPC Express Benchmark HS

Manan Trivedi(✉)

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

**Abstract.** TPC Express Benchmark HS (TPCx-HS) is industry's first standard for benchmarking big data systems. There are many moving parts in a large big data deployment which includes compute, storage, memory and network collectively called the infrastructure, platform and application and in this paper, we characterize in detail how each of these components affect performance.

**Keywords:** Industry standards · Performance · Hadoop · Spark

## 1 Introduction

Performance is key to any application and more so to a big data deployment as it usually involves hundreds and thousands of nodes for both data storage and processing. This paper goes in to the detail on various bottlenecks to performance at each level and to what extent one could improve the performance by overcoming these bottle necks. Even a small percentage gain in performance can be a lot of capex reduction in a large deployment as fewer nodes can take on more tasks. The focus of this paper is on Hadoop performance characterization and goes into details of tuning and consideration of the storage – with Solid State Disks (SSD) and Hard Disk Drive (HDD), Networking - with 10 Gbit and 40 Gbit, application - Spark and MapReduce.

### 1.1 Hadoop Evolution

Apache Hadoop is a software framework used for distributed storage and processing of big data. The two main components of Apache Hadoop are the Hadoop Distributed File System (HDFS), and the MapReduce framework. HDFS implements a fault-tolerant distributed file system. MapReduce is a framework for the parallel processing of data stored in HDFS. The MapReduce architecture divides the task into many smaller jobs. The code for each of these jobs is pushed to each server where the data resides. The framework executes the code on each server in parallel; intermediate results are returned, then combined for the final result.

The first version of the MapReduce framework, MRv1, implemented an architecture that handled both the processing of jobs and the resource management across the cluster. This approach had a number of limitations:

- Single point of failure: If the server coordinating all the tasks, called the NameNode, fails, then all processing fails.

- Scalability: The architecture is limited to approximately 4000 nodes.
- Lock-in: MRv1 requires the use of MapReduce, which is not the optimal choice for many workloads.

To address these limitations the open source community developed another approach to handle job processing and cluster management called MRv2 (also known as YARN). In MRv2, the responsibilities are handled by separate components. This addresses the issues of a single point of failure and scalability while also opening the framework to run other programming models besides MapReduce. This allows multiple applications to be run on the same cluster at the same time. YARN can assign and reassign resources for different concurrent applications to allow better utilization of the cluster's resources (Fig. 1).

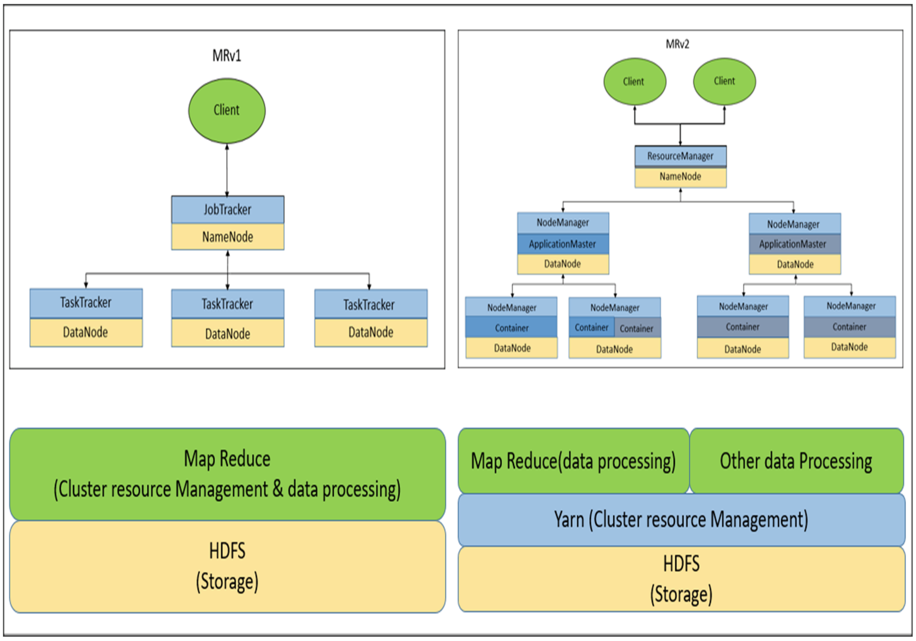

**Fig. 1.** MRv1 and MRv2 architecture

The ability to run new programming models on the cluster opens the door to address the issues with MapReduce. Parallel data processing as implemented by Apache Hadoop executes map and reduce phases that output intermediate data sets that are themselves input to the next map and reduce phase. There can be many such phases and MapReduce's key constraint is that it writes the intermediate data sets out to the disk, and then reads from the disk for the next phase. As such, MapReduce's speed is governed by the I/O bandwidth of the storage system.

To overcome the disk, I/O constraint of MapReduce the open source community developed Apache Spark. Apache Spark addresses the issue by reading the data into memory, storing and transforming it there before producing the final result. As the majority of operations are performed in memory at electronic speeds the system executes much faster. In addition, Apache Spark provides a rich set of functionality for in-memory processing that is both fault-tolerant and easier to program than MapReduce. It's in-memory approach enables applications for real-time processing of streaming data and interactive analysis.

Another solution to the disk I/O constraint issue comes from improvements in storage technology that have made SSDs a viable choice for data storage in big data systems. SSDs enable much faster read and write access as they do not have the same physical limitations of spinning disks and moving heads that hard disk drives have. Finally, advances in network technology have created faster throughput speeds which also benefit big data systems.

## 2 Introduction to TPCx-HS Benchmark

TPCx-HS is the industry's first standard for benchmarking big data systems [1–3]. It is designed to provide verifiable performance, price-to-performance, and availability metrics for hardware and software systems that use big data.

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 [5]. While the 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. 2, and the lower value is used for reporting.

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:

- TG: Data generation phase completion time, with HSGen reported in hh:mm:ss format
- TS: Data sort phase completion time, with HSSort reported in hh:mm:ss format
- TV: Data validation phase completion time, reported in hh:mm:ss format

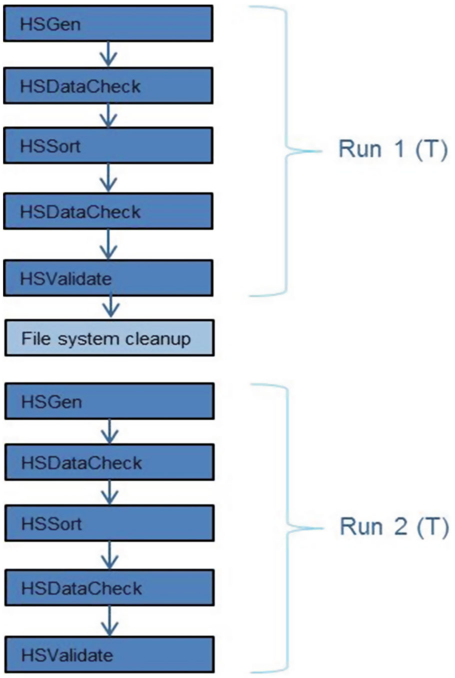

**Fig. 2.** TPCx-HS Benchmark processing

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 Performance Characterization

The tests were conducted a series of TPCx-HS to characterize the performance in various deployment scenarios. The test configuration consisted of Cisco UCS Integrated Infrastructure for Big Data cluster with 17 Cisco UCS C240 M4 Rack Servers. The Cisco UCS Integrated Infrastructure for Big Data is built using the following components:

- Cisco UCS 6300 Series Fabric Interconnect, provide high-bandwidth, low-latency connectivity for servers, with Cisco UCS Manager providing integrated, unified management for all connected devices. The Cisco UCS 6300 Series Fabric Interconnects are a core part of Cisco UCS, providing low-latency, lossless 40 GB Ethernet, Fibre Channel over Ethernet (FCoE), and Fibre Channel functions with management capabilities for systems deployed in redundant pairs. Cisco Fabric Interconnects offer the full active-active redundancy, performance, and exceptional scalability needed to support the large number of nodes that are typical in clusters serving big data applications.
- 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 1.5 TB of memory, and 24 small-form-factor (SFF) disk drives plus two internal SATA boot drives and Cisco UCS Virtual Interface Card (VIC) 1387 adapters.

The Cisco UCS Integrated Infrastructure for Big Data cluster configuration consists of two Cisco UCS 6332 fabric interconnects, 17 Cisco UCS C240 M4 servers with two Intel Xeon processor E5-2680 v4 series CPUs, 256 GB of memory, and 24 SFF disk drives or 8 SFF 1.6 TB SATA SSD plus two internal SATA boot drives and Cisco UCS VIC 1387 adapters, as shown in Fig. 3. Table 1 lists the software versions used.

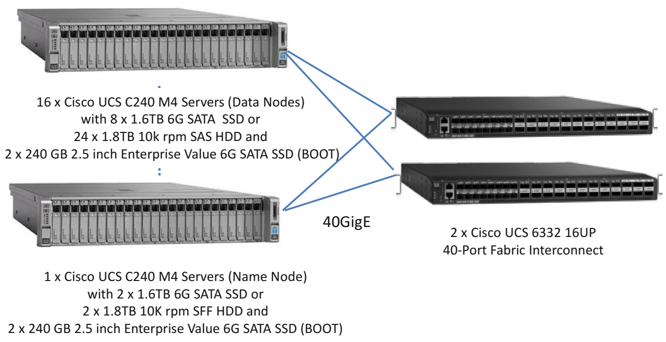
16 x Cisco UCS C240 M4 Servers (Data Nodes)
with 8 x 1.6TB 6G SATA SSD or
24 x 1.8TB 10k rpm SAS HDD and
2 x 240 GB 2.5 inch Enterprise Value 6G SATA SSD (BOOT)
40GigE
2 x Cisco UCS 6332 16UP
40-Port Fabric Interconnect
1 x Cisco UCS C240 M4 Servers (Name Node)
with 2 x 1.6TB 6G SATA SSD or
2 x 1.8TB 10K rpm SFF HDD and
2 x 240 GB 2.5 inch Enterprise Value 6G SATA SSD (BOOT)

**Fig. 3.** Cisco UCS integrated infrastructure for big data cluster configuration

**Table 1.** Software versions

| Layer | Component | Version or Release |
|---|---|---|
| Computing | Cisco UCS C240 M4 server | C240M4.2.0.13d.0.0812161132 |
| Network | Cisco UCS 6332 fabric interconnect | 3.1 (2b) |
| | Cisco UCS VIC 1387 firmware | 4.1 (2d) |
| | Cisco UCS VIC 1387 driver | 2.3.0.31 |
| Software | Red Hat Enterprise Linux (RHEL) server | 7.2 (x86_64) |
| | Cisco UCS Manager | 3.1 (2b) |
| Hadoop | Cloudera Enterprise | Version 5.10.0 |

### 3.1 Cisco UCS Integrated Infrastructure for Big Data Cluster Configuration

- 16 × Cisco UCS C240 M4 Servers (Data Nodes) with:
- 8 × 1.6-TB 6-Gbps SATA SSD or
- 24 × 1.8-TB/1.2 TB 12-Gbps SAS 10 K-rpm SFF HDD
- 2 × 240-GB 2.5–in. Enterprise Value 6-Gbps SATA SSD (Boot)
- 2 × 40 Gigabit Ethernet
- 2 × Cisco UCS 6332 fabric interconnect
- 1 × Cisco Nexus® 9372PX Switch

## 4 Hardware and Software: Performance Characterization

In the following section, we will do an in-depth performance analysis with various permutations of these hardware and software constituents:

- Baseline Performance Tuning parameters (Infrastructure and Operating System)
- Performance tuning parameters of MRv2
- Performance characteristics comparison of MRv1 vs. MRv2
- MRv2 Storage Configuration Comparison (HDD vs. SDD, 2 vs. 4 vs. 8 SSD)
- MRv2 Network Configuration Comparison (10 g vs. 40 g)
- Apache Spark: Comparison of Default Settings to Tuned Parameters
- Apache Spark Storage Comparison: HDD vs. SSD

### 4.1 Baseline Performance Tuning Parameters

Apache Hadoop is based on a new approach to storing and processing complex data, with reduced data movement. It distributes the data across the cluster. Each machine in the cluster stores and also processes the data. Infrastructure and operating system tunings can have a significant performance impact, depending on the applications and their respective workloads. Therefore, it is important to individually tune the compute, network and storage parameters of the system to achieve optimal performance for the cluster.

Hadoop is a complex application designed to address many different types of workloads. Very often, the default settings are not optimized for the best performance, instead being defined to work out of the box on the minimum hardware required. Tuning the Hadoop settings can produce significant performance improvements [4].

The key areas for Hadoop performance tuning are: infrastructure (compute, network and storage), operating systems and Hadoop parameters. These parameters are covered in-depth in a previous paper I wrote titled "Performance Evaluation and Benchmarking" as part of Springer's Lecture Notes in Computer Science Series.

The focus of this paper is to study the MRv2 architecture and compare its performance to MRv1. We will use the MRv1 performance tuning and results from our earlier study published under: "Lessons Learned: Performance Tuning for Hadoop Systems."[1]

## 4.2 Apache Hadoop MRv2 Tuning

The default Apache Hadoop MRv2 settings are not optimized for performance. Instead, they are defined so the system works out of the box with the minimum hardware requirement. HDFS provides storage for all the data and is a core component of Apache 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. Here are the tuning parameters which we used to tune the cluster for MRv2.

The following are the parameters and tuned values for the test cases run in this paper (Tables 2 and 3).

- hdfs-site.xml
- mapred-site.xml

**Table 2.** hdfs-site.xml settings

| Parameter | Value |
|---|---|
| dfs.blocksize | 1 GB |
| dfs.datanode.failed.volumes.tolerated | 4 |
| dfs.datanode.handler.count | 40 |
| dfs.datanode.max.xcievers, dfs.datanode.max.transfer.threads | 32000 |
| dfs.namenode.handler.count | 1400 |
| dfs.namenode.service.handler.count | 55 |
| dfs.namenode.servicerpc-address | 8022 |
| Java Heap Size of NameNode in Bytes | 16 GB |
| Java Heap Size of Secondary NameNode in Bytes | 16 GB |

[1] **Note:** These settings represent a starting point for tuning a big data system. The actual best values will vary based on the workload of the system.

**Table 3.** mapred-site.xml settings

| Parameter | Value |
|---|---|
| Mapreduce.client.submit.file.replication | 3 |
| yarn.app.mapreduce.am.command-opts | -Djava.net.preferIPv4Stack=TRUE -Xmx2800m |
| io.file.buffer.size | 128 KB |
| mapreduce.job.reduce.slowstart.completedmaps | 0.85 |
| mapreduce.job.reduces | 895 |
| mapreduce.task.timeout | 3 min |
| mapreduce.map.java.opts | -Djava.net.preferIPv4Stack=true -XX:+UseParallelGC -XX:ParallelGCThreads=8 -XX:-UseAdaptiveSizePolicy -XX:+DisableExplicitGC |
| mapreduce.reduce.java.opts | –Djava.net.preferIPv4Stack=true -XX:+UseParallelGC -XX:ParallelGCThreads=8 -XX:-UseAdaptiveSizePolicy -XX:+DisableExplicitGC |
| mapreduce.task.io.sort.factor | 100 |
| mapreduce.task.io.sort.mb | 1500 MB |
| mapreduce.reduce.shuffle.parallelcopies | 30 |
| yarn.nodemanager.heartbeat.interval-ms, yarn.resourcemanager.nodemanagers.heartbeat-interval-ms | 160 ms |
| yarn.scheduler.fair.preemption | Resource Manager Default Group |
| zlib.compress.level | BEST_SPEED |
| yarn.app.mapreduce.am.resource.mb | 3 GB |
| ApplicationMaster Java Maximum Heap Size | 1 GB |
| mapreduce.map.memory.mb | 2500 MB |
| mapreduce.reduce.memory.mb | 3 GB |
| mapreduce.map.java.opts.max.heap | 2300 MB |
| mapreduce.reduce.java.opts.max.heap | 2800 MB |
| yarn.nodemanager.resource.memory-mb | 246 GB |
| yarn.nodemanager.resource.cpu-vcores | 56 |
| yarn.scheduler.maximum-allocation-mb | 246 GB |
| yarn.scheduler.maximum-allocation-vcores | 56 |

## 4.3 MRv1 vs MRv2 (YARN)

MRv1, the first version of the Apache Hadoop framework makes use of a job tracker that creates a set of map and reduce tasks which are then managed by the appropriate task trackers on each node.

The next version of the MapReduce framework, MRv2, introduces YARN (Yet Another Resource Negotiator). YARN separates cluster resource management and MapReduce specific logic. The Resource Manager tracks and allocates available resources and an Application Master process is created for each application which is responsible for the entire life cycle of the MapReduce application.

To understand MRv2 performance compared to MRv1 we did an in-depth study of the performance of both frameworks by running a TPCx-HS benchmark. This led to the following observations:

- Overall, in terms of total time taken, MRv1 performed faster than MRv2.
- While MRV2 was faster in HSSort phase, and MRv1 was faster in the HSGen and HSValidate phases.

The test results here show the comparison of MRv1 vs. MRv2 at a 3-TB scale factor.

Result: MRv1 vs MRv2 (YARN).

This observed performance penalty is offset by the numerous benefits of the MRv2 framework in terms of scalability, fault-tolerance and support for simultaneously running multiple applications. The results in Table 4 show that MRv1 is performing 6% better than MRv2.

These results are based on one set of tuning parameters. These parameters will vary from workload to workload.

Cluster Detail: 16 data nodes each containing 8 × 1.6 TB Intel SSDs and 2 × 40 G Network connectivity (Fig. 4).

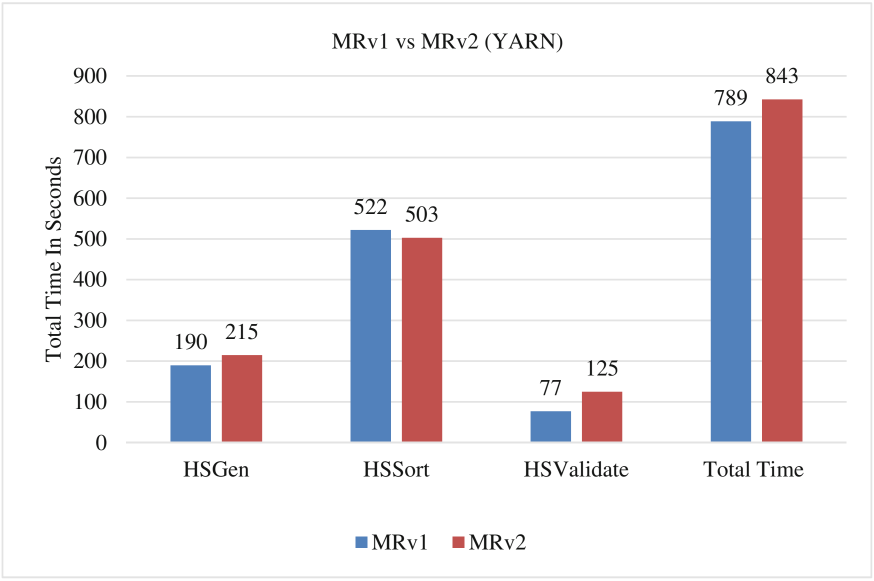

**Fig. 4.** MRv1 vs MRv2 (YARN)

Table 4 lists detailed response times for each benchmark phase.

**Table 4.** MRv1 vs MRv2 (YARN)

| Phase | MRv1 | MRv2 |
|---|---|---|
| HSGen | 190 | 215 |
| HSSort | 522 | 503 |
| HSValidate | 77 | 125 |
| Total time | 789 | 843 |
| HSph@SF at 3-TB scale factor | 13.52 | 12.71 |

## 4.4 MRv2 Storage Configuration Comparison

Apache Hadoop solves the big data problem by breaking the data up into smaller chunks and storing them across many servers. The processors of each of these individual servers are then used to operate on their locally stored data. This use of many smaller servers with direct attached storage is a key reason Apache Hadoop scales in such a linear fashion.

The use of this architecture means that the I/O bandwidth, i.e. how fast we can read and write data, is the key constraint for the system. Recent advances in storage technology have made solid state disks (SSDs) a viable choice for big data systems. However, the performance gains from using SSDs are so dramatic that they exceed the total available bandwidth of the internal throughput of the system. As a result, when comparing SSDs to HDDs, we have to look at both raw performance and price-performance.

## 4.5 MRv2 HDD vs SSD

The choice between hard disk drives and solid-state drives needs to be made based on the expected workload. HDDs will provide more raw storage capacity at the expense of throughput while SSDs provide the best performance and price-performance but with a lower total capacity. There is also an endurance factor with SSDs based on the number of expected write operations.

The test results here show the comparison of 24 HDDs vs. 8 SSDs using Apache Hadoop MapReduce version 2 (MRv2) at a 3-TB scale factor.

Result: MRv2 HDD vs SSD.

Results of the tests using MRv2 with HDDs vs. SSDs are shown below. The results demonstrate that eight SSDs do the work of 24 HDDs with better performance for all tasks. The results also show that SSDs are a better value with a performance improvement of 2%

Cluster Detail: 16 data nodes each containing 24 × 1.8 TB 10 K SAS HDDs vs. 8 × 1.6 TB Intel SSDs with 2 × 40 G Network connectivity (Fig. 5).

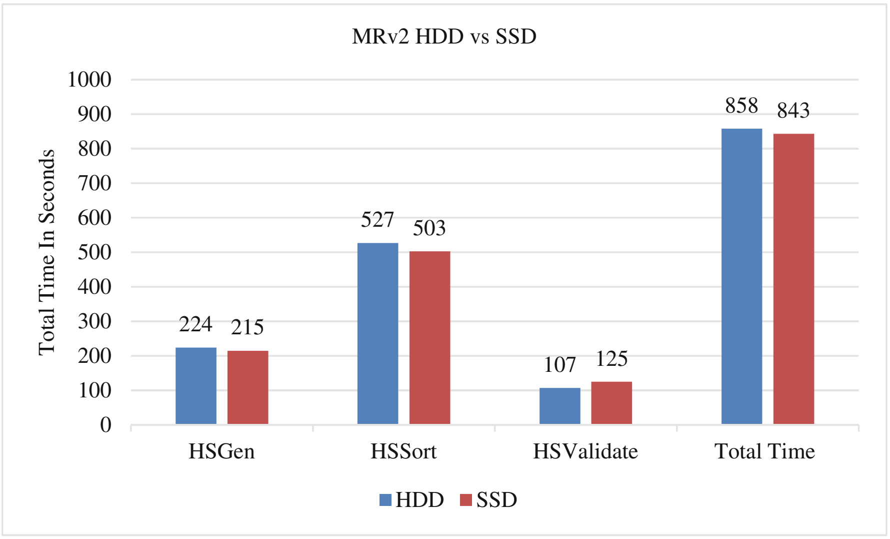

**Fig. 5.** MRv2 HDD vs SSD

Table 5 lists detailed response times for each benchmark phase.

**Table 5.** MRv2 HDD vs SSF

| Phase | HDD (1.8 TB) | SSD (1.6 TB) |
|---|---|---|
| HSGen | 224 | 215 |
| HSSort | 527 | 503 |
| HSValidate | 107 | 125 |
| Total time | 858 | 843 |
| HSph@SF at 3-TB scale factor | 12.47 | 12.71 |

### 4.6 SSD Performance Comparison (2 vs 4 vs 8)

One of the key advantages of Apache Hadoop is that it scales linearly with the more data nodes and more data disk drives.

This test compares the performance of 2 vs 4 vs 8 SSDs in each server, using Apache Hadoop MapReduce version 2 (MRv2) at a 3-TB scale factor.

Result: SSD Performance Comparison (2 vs 4 vs 8).

Results of the tests using MRv2 with 2 vs 4 vs 8 SSDs are shown below. The results demonstrate the linear scaling of performance using 8 SSDs. As shown in Fig. 6, on an individual server basis, Apache Hadoop performance improves with the number of drives per node.

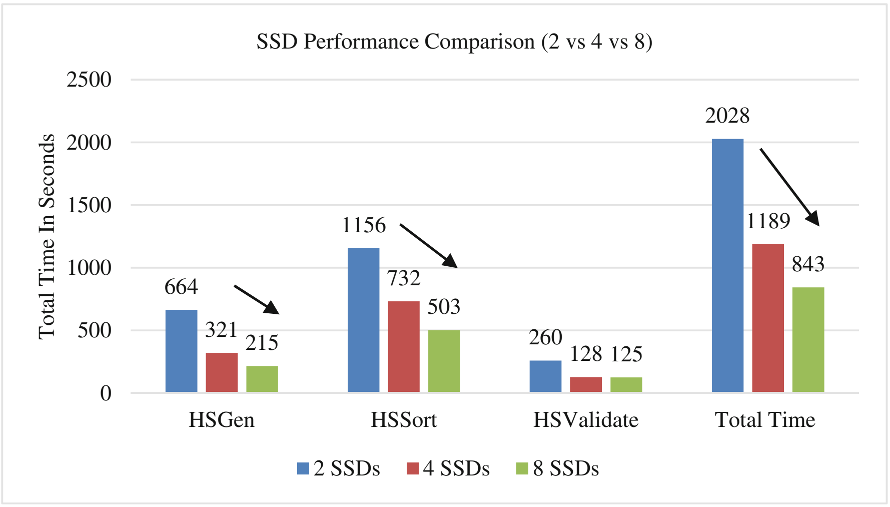

**Fig. 6.** SSD performance comparison (2 vs 4 vs 8)

Cluster Detail: 16 data nodes each containing 2, 4 and 8 × 1.6 TB Intel SSDs and 2 × 40G Network connectivity.

Table 6 lists detailed response times for each benchmark phase.

**Table 6.** SSD performance comparison (2 vs 4 vs 8)

| Phase | 2 SSDs | 4 SSDs | 8 SSDs |
|---|---|---|---|
| HSGen | 664 | 321 | 215 |
| HSSort | 1156 | 732 | 503 |
| HSValidate | 260 | 128 | 125 |
| Total time | 2028 | 1189 | 843 |
| HSph@SF at 3-TB scale factor | 5.32 | 9.08 | 12.71 |

Test Result: End-to-End Write I/O Bandwidth Utilization.

TPCx-HS enables fair comparisons to be made between software and hardware systems. It also exercises various subsystems. Figure 7 shows disk write IO bandwidth utilization for 2, 4 and 8 SSDs using one of the node's end-to-end run. As we are seeing in the chart below 8 drives are performing 3 times faster than 2 drives. We are seeing it scaling linearly as the number of the SSDs increase.

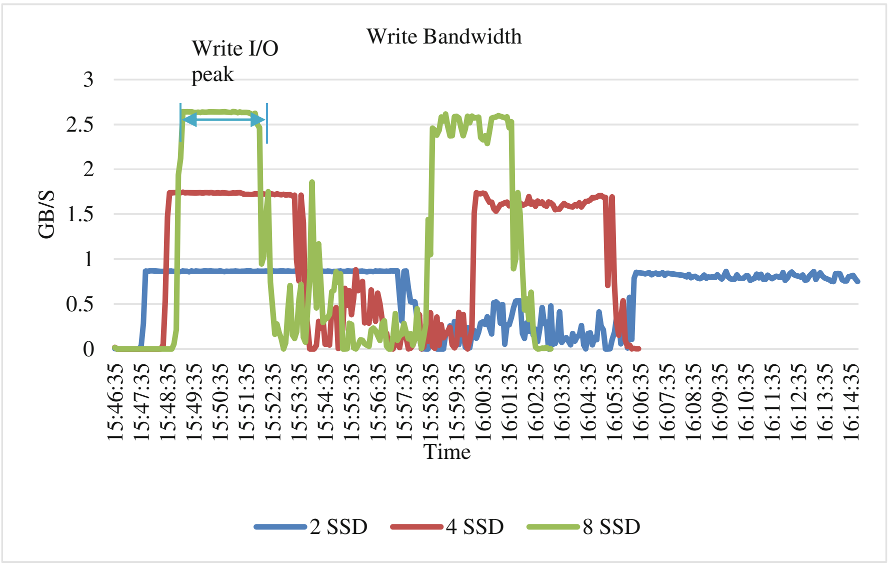

**Fig. 7.** End-to-End Write I/O bandwidth utilization comparison (2 vs 4 vs 8)

## 5 MRv2 Network Configuration Comparison

The impact of the network on big data systems is enormous. An efficient and resilient network is a crucial part of a good Apache Hadoop cluster because the network is what connects all the nodes. The network is used to load the data, read the data, and write the intermediate data sets and final output.

The impact of the failure of a network device is dire. Individual jobs and even entire applications may need to be restarted with the workloads pushed to remaining available nodes. The network must be well designed with fault-tolerance, redundancy and multiple paths between computing nodes. It must also be able to scale with the data.

The network can quickly become the constraining factor, and this is becoming more common as technologies like Apache Spark and SSDs proliferate. In response to this, faster networks have been developed. The current generation of 40G networks are aimed squarely at big data systems with local storage using SSDs or high throughput HDDs. Upgrading to the latest generation of Cisco UCS fabric interconnects, we increased the underlying fabric from 10 Gbps to 40 Gbps.

### 5.1 10G Network vs. 40G Network

Comparing network bandwidth to IO bandwidth can be confusing as networks are commonly measured in bits per second while IO bandwidth is measured in bytes per second. Converting the network measurements to bytes:

- Standard 10 Gbps networks = 1.25 Gbps
- New 40 Gbps networks = 5 Gbps

For servers with 24 1.8 TB drives, the total IO bandwidth is 5.4 Gbps. This is over four times the available bandwidth of a 10 Gbps network. Big data applications are not transferring the maximum IO bandwidth across the network all the time. But they do exceed the bandwidth at times and when they do the performance is directly affected.

One way to characterize this is to execute a performance comparison using both 10 Gbps and 40 Gbps networks. The results are shown below. Note that the performance impact will be greater using SSDs as the total IO bandwidth can exceed 7 Gbps.

Result: 10 Gbps Network vs 40 Gbps Network.

Results of the tests using MRv2 with 10 Gbps vs. 40 Gbps are shown below. The results demonstrate that the 40 Gbps network improves the write bandwidth which helps the applications to write and read the data faster over the network for all tasks. Overall, the results show that 40 Gbps performs 14% faster than 10 Gbps.

Cluster Detail: 16 Data Nodes each containing 24 × 1.8 TB 10 K SAS and 2 × 10G vs 40G Network connectivity (Fig. 8).

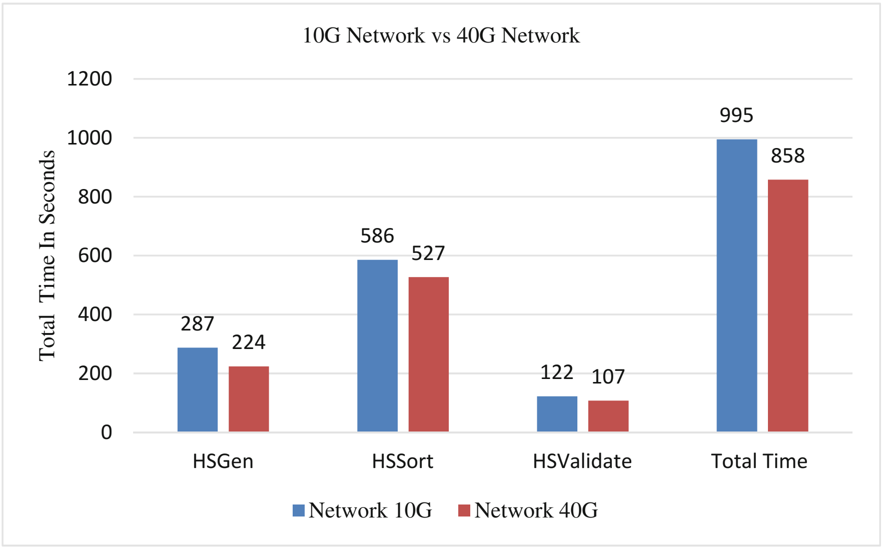

**Fig. 8.** 10G Network vs 40G Network

Table 7 lists detailed response times for each benchmark phase.

**Table 7.** 10G Network vs 40G Network

| Phase | 10G Network | 40G Network |
|---|---|---|
| HSGen | 287 | 224 |
| HSSort | 586 | 527 |
| HSValidate | 122 | 107 |
| Total time | 995 | 858 |
| HSph@SF at 3-TB scale factor | 10.85 | 12.47 |

## 6 Apache Spark

While MapReduce has become a standard for batch processing, Apache Spark is a better choice for real-time data processing and interactive analysis. Apache Spark was developed to overcome the disk I/O constraint of MapReduce. The model for processing distributed data in parallel uses many cycles of first mapping the data, then reducing it. Each of these cycles produces an intermediate output which is input to the next. MapReduce writes these intermediate sets of output to disk, which is then read from disk as input to the next cycle. Thus, the overall performance is gated by the relatively slow speed of disk I/O. Apache Spark addresses this disk I/O bottleneck by reading the data into memory and then performing all data operations in memory, eliminating the disk I/O constraint.

### 6.1 Apache Spark Tuning

Out of the box, Apache Spark is not optimized for performance. Instead, the default parameters are designed to work without modification on the minimum hardware requirements.

Tuning the parameter can yield significant performance improvements. The tuning parameters discussed in this section provide a guideline towards improved performance for real-time data processing workloads.

Table 8 is the list of Spark parameters which are tuned across the different test cases covered in this section.

**Table 8.** List of Spark tuning parameters

| Parameter | Value |
|---|---|
| spark.shuffle.compress | true |
| spark.broadcast.compress | true |
| spark.io.compression.codec | org.apache.spark.io.SnappyCompressionCodec |
| spark.shuffle.spill.compress | true |
| spark.kryo.referenceTracking | false |
| spark.executor.extraJavaOptions | -XX:+PrintFlagsFinal -XX:+PrintReferenceGC -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+PrintAdaptiveSizePolicy -XX:+UnlockDiagnosticVMOptions -XX:+G1SummarizeConcMark |
| spark.shuffle.spillAfterRead | true |
| spark.kryoserializer.buffer | 2000 |
| spark.default.parallelism | 2110 |

### 6.2 Comparison of Default Settings to Tuned Parameters

An interesting starting point is a comparison of out of the box performance vs performance with parameters tuned specifically for the System Under Test (SUT).

Result: Default Settings vs Tuned Parameters.

As a part of this experiment, we tuned the Spark for running the TPCx-HS benchmark on the cluster with below details.

Observation: Spark, with our tuned parameters performed significantly better: 63% performance improvement over default settings. Note that Apache Spark performs better when all the data fits in the memory, so with this cluster configuration, the 1 TB scale factor test performs better than 3 TB scale factor.

Although, in-memory processing provides significant advantages, it also adds an extra layer of consideration when tuning the system. The results below are based on one set of tuning parameters. The tunings will vary from workload to workload.

Cluster Detail: 16 Data Nodes each containing 8 × 1.6 TB Intel SSDs and 2 × 40G Network connectivity in both test scenarios (Fig. 9).

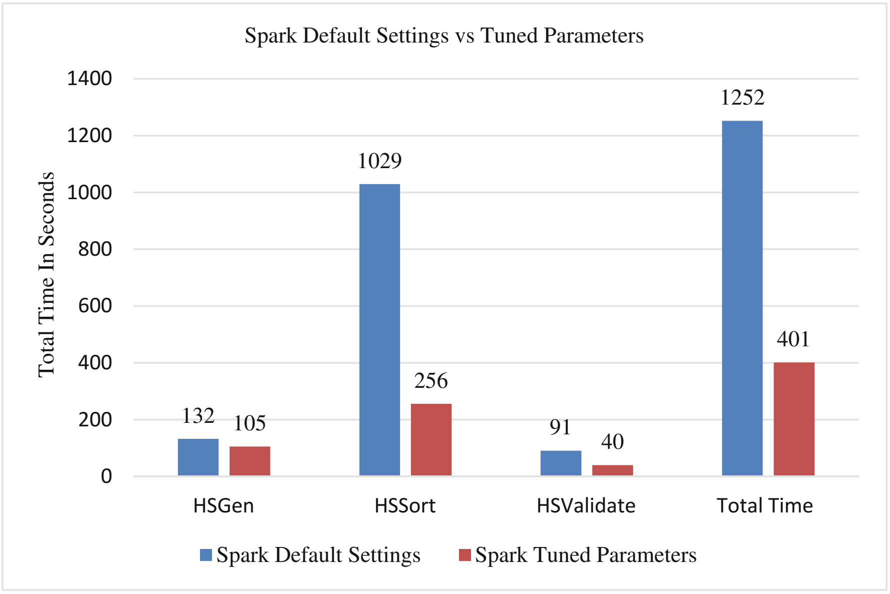

**Fig. 9.** Spark default settings vs Tuned parameters

Table 9 lists detailed response times for each benchmark phase

**Table 9.** Spark default settings vs. Tuned parameters

| Phase | Spark default | Spark tuned |
|---|---|---|
| HSGen | 132 | 105 |
| HSSort | 1029 | 256 |
| HSValidate | 91 | 40 |
| Total time | 1252 | 401 |
| HSph@SF at 1-TB scale factor | 2.85 | 8.97 |

**Note:** Spark test was performed with 1 TB scale factor using TPCx-HS

## 6.3 Apache Spark Storage Comparison: HDD vs SSD

Apache Spark reads the data into memory and processes is it there. This initial read of the data is constrained by disk I/O. However, if instead of HDDs you use SSDs you can further improve performance. But, by how much? Answering this question helps to understand if the additional cost of SSDs is worth it.

Further, if there is more data than will fit in memory, or the intermediate result sets exceed the amount of available memory, Apache Spark will "spill" the data to disk (conceptually equivalent to operating system "swapping"). When this happens, disk I/O as a constraint re-enters the performance equation.

Result: Spark HDD vs SSD

We have done a study of the performance comparison between HDDs and SSDs using Apache Spark by running the TPCx-HS benchmark on the test setup described below. This led to the following observation:

The observed performance shows that SSDs performed better with Spark than HDDs. Spark's processing engine is designed to use both in-memory and on-disk, so it performs operations when data does not fit in memory. This is where the high I/O performance of SSDs overcomes the slower read and write access of HDDs and Spark's performance is improved. As a result, with larger data sizes or scale factors SSDs performance will be better than HDDs.

Cluster Detail: 16 data nodes each containing 24 × 1.8 TB 10 K SAS HDDs vs. 8 × 1.6 TB Intel SSDs with 2 × 40G Network connectivity (Fig. 10).

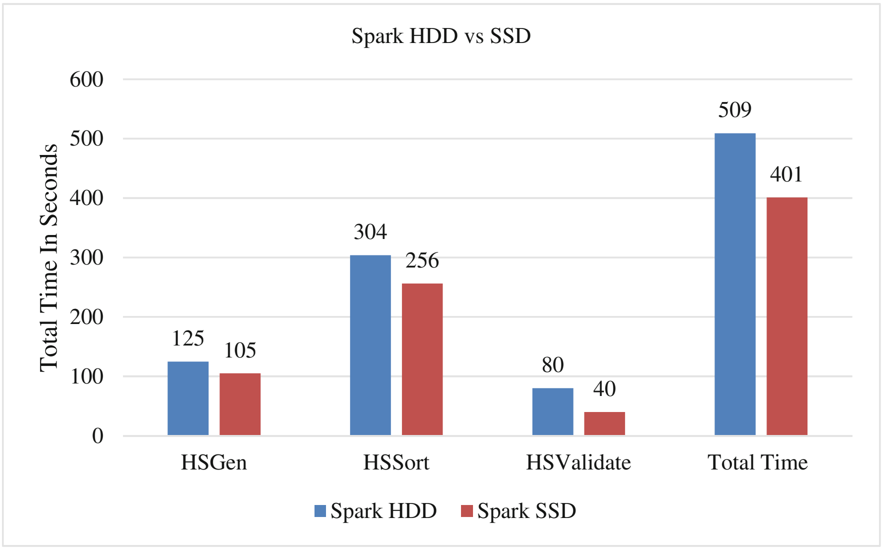

**Fig. 10.** Spark HDD vs SSD

Table 10 lists detailed response times for each benchmark phase.

**Table 10.** Spark HDD vs SSD

| Phase | Spark HDD | Spark SSD |
|---|---|---|
| HSGen | 125 | 105 |
| HSSort | 304 | 256 |
| HSValidate | 80 | 40 |
| Total time | 509 | 401 |
| HSph@SF at 1-TB scale factor | 7.07 | 8.97 |

**Note:** Test was performed with 1 TB scale factor using TPCx-HS.

## 7 Conclusion

This paper 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. In general, we clearly see improvements in performance as the technology advances to address the limitations of the previous generation. This paper quantifies those improvements providing the data needed to make informed decisions.

## References

1. Nambiar, R., Poess, M., Dey, A., Cao, P., Magdon-Ismail, T., Ren, D.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). https://doi.org/10.1007/978-3-319-15350-6_1
2. Nambiar, R.: Benchmarking big data systems: introducing TPC express benchmark HS. In: Rabl, T., Sachs, K., Poess, M., Baru, C., Jacobson, H.-A. (eds.) WBDB 2015. LNCS, vol. 8991, pp. 24–28. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-20233-4_3
3. Nambiar, R.: A standard for benchmarking big data systems. In: BigData Conference 2014, pp. 18–20 (2014)
4. Trivedi, M., Nambiar, R.: Lessons learned: performance tuning for hadoop systems. In: Nambiar, R., Poess, M. (eds.) TPCTC 2016. LNCS, vol. 10080, pp. 121–141. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-54334-5_9
5. TPCx-HS specification. http://www.tpc.org/tpcx-hs/