Virtualized Hadoop Performance with VMware vSphere® 6 on High-Performance Servers

Performance Study

TECHNICAL WHITE PAPER

vmware®

Table of Contents

Executive Summary

Large advances have been made in hardware and every level of the software stack since the virtualized Hadoop tests published in April 2013. This paper shows how to take advantage of these advances to achieve maximum performance. The cluster size remains at 32 two-processor 2U hosts; however, the processor, memory, network, and storage capabilities are all roughly doubled from those reported in the earlier paper. The performance of native and several VMware vSphere® 6 virtualized configurations were compared using the same TeraSort application suite as before. It was found that the more powerful hosts give a larger advantage to multi-VM per host configurations: virtualized TeraSort is now up to 12% faster than the optimized native configuration. The apples-to-apples case of a single virtual machine per host again shows performance close to that of native Linux. The origins of the improvements are examined and recommendations for optimal hardware and software configurations are given.

Introduction

Apache Hadoop provides a platform for building distributed systems for massive data storage and analysis [1] using a large cluster of standard x86-based host servers. It uses data replication across hosts and racks of hosts to protect against individual disk, host, and even rack failures. A job scheduler can be used to run multiple jobs of different sizes simultaneously, which helps to maintain a high level of resource utilization. Given the built-in reliability and workload consolidation features of Hadoop, it might appear there is little need to virtualize it. However, there are several use-cases that make virtualization of this workload compelling:

- Enhanced availability with capabilities like VMware High Availability (HA) and Fault Tolerance (FT). The performance implications of protecting the Hadoop master daemons with FT were examined in a previous paper [2].
- Easier deployment with vSphere tools or vSphere Big Data Extensions (BDE), leading to easier and faster datacenter management [3].
- Sharing resources with other Hadoop clusters or completely different applications, enabling better datacenter utilization.

In addition, virtualization enables new ways of integrating Hadoop workloads into the datacenter:

- Elasticity: The ability to quickly grow a cluster as needs warrant, and to shrink it just as quickly in order to release resources for other applications.
- Multi-tenancy: Multiple virtual clusters can share a physical cluster while maintaining the highest levels of isolation between them.
- Separating roles: Greater security within each cluster can be achieved by separating the computational (TaskTracker) and data (DataNode) parts of Hadoop into different machines, each with its own access authorization. However, data locality (and thus performance) requires them to be on the same physical host, leading to the use of virtual machines. This also yields more flexible elasticity, in that the computational and data roles can be scaled differently as needed.

A detailed discussion of these points is beyond the scope of this paper, but can be found elsewhere [4]. As compelling as the current and potential future benefits of virtualization are for Hadoop, they are unlikely to be realized if the performance costs are too high. The focus of this paper is to quantify these costs and to try to achieve an understanding of the implications of alternative virtual configurations. This is done through the use of a set of well-understood, high-throughput benchmarks (the TeraSort suite). While these applications may not be generally representative of production clusters running many jobs of different sizes and priorities, they are at the high end of infrastructure resource usage (CPU, memory, network, and storage bandwidth) of production jobs. As such, they are good tools for stressing the OS, virtualization, and resource layers. The ultimate goal is for a Hadoop administrator to be able to create a cluster specification that enables all the above advantages while achieving the best performance possible.

One of the big advantages of virtualizing a distributed workload like Hadoop is the opportunity to manage the scale-up/scale-out trade-offs. In a native environment, the size of each node in the cluster is fixed by the available hardware and the system administrator must tune the application to that size. Even when the administrator is willing to do this, some distributed applications have been designed to scale-out on small nodes and do not scale-up well enough to use all the capabilities of modern resource-dense servers [5]. In a virtualized environment a smaller machine size may be configured if that allows the application to be more efficient. Such sizing flexibility also enables the administrator to create a standard VM template that bin-packs well onto hosts of various sizes. For instance, if four-socket hosts were added to a cluster of two-socket hosts, it would be very reasonable to simply run twice as many of the same VMs on the larger hosts as proved to work well on the smaller hosts. While this ability is extremely important for applications that do not scale up well, it is still important for modern applications such as Hadoop that have been designed for both kinds of scaling since it allows both hardware utilization and application efficiency to be optimized.

An early paper on virtualized Hadoop performance tests on a small cluster [6] included discussions of motivations for deploying Hadoop, its architecture, and reasons for virtualizing a Hadoop cluster. A paper published in April 2013 using 32 hosts tested several configurations and presented models explaining the performance advantage of configuring multiple small virtual machines (VMs) per host [7].

Configuration

Hardware Overview

The hardware configuration is shown in Figure 1. A cluster of 32 servers were connected in a flat network topology. Each host was equipped with two Intel Xeon E5-2680 v2 "Ivy Bridge" 2.8GHz ten-core processors, 256GiB memory, and 23 internal 600GB 10K RPM SAS disks. The servers were each connected to two Extreme Summit 10GbE switches using a two-port Intel adaptor.

Note: Throughout the paper, the "i" in KiB, etc., signifies 1024-based counting, otherwise prefixes are factors of 1000.

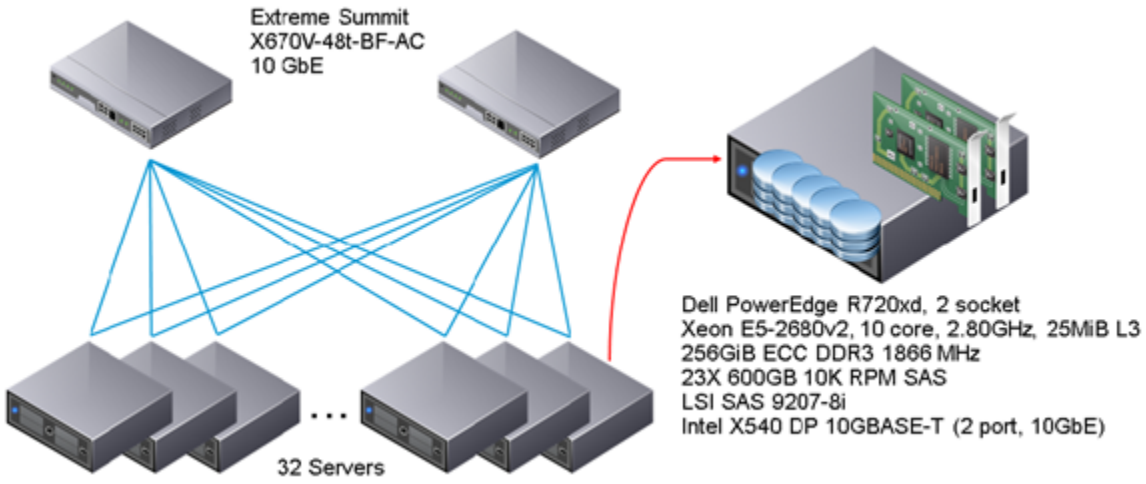

Figure 1. Cluster hardware configuration.

The internal disks were connected to a PCI Express 3.0 x8 storage controller, the LSI SAS 9207-8i, which has a theoretical throughput of 4 GB/s. With earlier-generation controllers, the aggregate throughput using a simple storage microbenchmark can reach the controller limit. While throughput generated by the Hadoop applications described here can be very high, the storage controller was never a bottleneck.

Power states were enabled in the BIOS, which allows power consumption to drop by a factor of two when the cluster is idle. Intel Hyper-Threading (HT) Technology was enabled and used by the hypervisor or native operating system in all cases.

Complete hardware details are given in "Appendix A: Configuration."

System Software

SUSE Linux Enterprise Server (SLES) 11 SP3 x86_64 was used as the guest operating system in the virtual machines, which were run on VMware vSphere 6. The same OS was installed natively and configured as identically as possible as the virtual machines. The VMware Tools package was installed in each VM and its pvscsi and vmxnet3 drivers were used. Sun Java 7 is recommended for Hadoop; version 1.7.0_55 was used here. A few Linux kernel tunables were increased in order to handle more files and processes (these are listed in "Appendix A: Configuration").

Hypervisor tuning was limited to network and storage driver options (see below) and virtual machine migration parameters. One way to avoid VM migration between the two NUMA nodes on a host is to affinitize (or "pin") each VM to a NUMA node. However, this introduces scheduling inflexibility and can be error-prone. Instead, the hypervisor scheduler was allowed to place the VMs automatically and the NUMA scheduling parameters tuned to reduce the migration aggressiveness, while still allowing migration if there is a large resource imbalance. This strategy ensures 100% memory locality for the multi-VM per host platforms. For the single-VM platform, virtual NUMA (the default) ensures the guest OS sees the same NUMA topology as the native OS.

Complete hypervisor and operating system details are given in "Appendix A: Configuration."

Local Storage

Two of the 23 internal disks in each host were partitioned and used to store the virtual machine images and the native OS.

The ESXi hypervisor was PXE-booted from a network repository. ESXi has a memory-based file system and thus does not create any storage I/O itself. During the tests, the I/O to these two root disks consisted almost exclusively of Hadoop logs and job statistics. The other 21 disks were configured as individual disks (that is, there was no striping or RAID), commonly known as a "JBOD" configuration. Twenty were used for Hadoop data (and referred to here as "data disks"), while the remaining one was kept as a spare.

The data disks were passed through to the virtual machines using physical raw device mappings (known as pRDM). This allowed the use of the same storage for native and virtual platforms without reformatting, thus eliminating a potential source of differences (as well as the small CPU cost of using VMFS). pRDM for local disks is not enabled by default in the vSphere UI. This is related to being able to see the UID of the disks; refer to the Knowledge Base articles [8,9] for support and configuration details.

A single aligned partition was created on each data disk and formatted with XFS. XFS supports "allocation groups," which enables greater metadata parallelism and better performance for many use cases. However, for a storage pattern consisting of mostly large I/Os to a partition on a single disk, a single allocation group was found to yield higher throughput and much better predictability (that is, similar performance for all disks across the cluster). The latter is very important for cluster workloads that contain barriers where the application has to wait for the slowest cluster member to finish before being able to proceed to the next phase. A single allocation group in XFS was configured during file system creation with the option "`-d agcount=1`". Results from storage microbenchmark tests showing the effect of `agcount` are given in "Appendix B: XFS Storage Performance."

PVSCSI was chosen for the virtual SCSI controller in the VMs. The mpt2sas storage driver was upgraded to version 19 for the native OS. Version 19 is included with vSphere 6. To increase efficiency on both platforms the number of interrupt vectors was set to 1 using the driver parameter `max_msix_vectors=1`. More vectors are needed only for very high IOPs workloads with small I/Os.

Complete storage details are given in "Appendix A: Configuration."

Networking

By default, vmxnet3 enables multi-queue for both transmit and receive. However, very few workloads require this capability; typically multi-queue is needed when extremely high packet rates of small packets are generated. For others, including Hadoop, it is recommended that vmxnet3 be configured with a single queue for better efficiency. Changing the number of queues requires installing the vmxnet3 version provided by VMware Tools.

For the native OS, a recent ixgbe network driver from Intel was installed. This allowed configuring a single queue (driver option `MQ=0`). This driver has LRO enabled by default. In the ESXi ixgbe driver, LRO was enabled and a single queue was configured (`VMDQ=1`).

The default interrupt throttling rate of 16000 in both the native and ESXi ixgbe drivers was doubled to 32000 for better network latency. This helped throughput for the present tests despite the higher CPU costs, but should only be considered a starting point for other applications and hardware configurations since the latency-CPU trade-off is likely to be different.

The analogous change was not found to be helpful for vmxnet3. Some extra efficiency was gained by pinning network interrupts in the native and guest OSes: the interrupts corresponding to the two NICs were pinned to separate CPUs in the native, 1-VM per host, and 2-VM per host platforms, and to a single CPU for all the platforms with four or more VMs per host.

A vSwitch was configured for each of the two 10GbE NICs on each host. Each NIC was connected to a physical 10GbE switch, as shown in Figure 1. Two vNICs were created for each VM, one on each vSwitch. That is, both physical NICs were fully virtualized for all VMs.

A bonding device was created from the two Ethernet devices in both the guest and native OSes. This device is capable of theoretically delivering up to 20 Gb/s throughput to a single IP address in any one VM, or delivering similar throughput split among all VMs on the host simultaneously. The default bonding mode is designed for a single stream of traffic. When multiple streams are present (as for Hadoop), `mode=2` (`balance-xor`) is much more efficient and delivers higher throughput. In addition, `transmit_mode=layer3+4` was found to work optimally for both native and virtual configurations.

Jumbo frames (`MTU=9000`) was configured for both native and virtual network devices. The physical switches have jumbo frames enabled by default; this needs to be checked in general. End-to-end transmission of jumbo frames was verified with `ping`.

Complete networking details are given in "Appendix A: Configuration."

Virtual Machines

Five different virtual platforms were tested. Each comprised one, two, four, ten, or twenty virtual machines per host, for a total of up to 640 virtual machines in the cluster. On each host all 40 logical processors, 20 Hadoop data disks, and 241GiB (out of 256GiB available) memory were evenly divided among the desired number of virtual machines (Table 1). That is, the CPU and disk resources were exactly-committed and memory was slightly under-committed. All VM memory was reserved and preallocated. These five platforms differ only in how the workload is decomposed into worker machines. Ideally there should be no difference in performance among them. However, hardware and application characteristics (discussed below) lead to substantial differences and an optimal choice.

The ESXi scheduler will place both the CPU and memory of a VM on a single NUMA node (a processor and associated memory for the hosts used here) if the VM is small enough to fit. This ensures that the application running in the VM accesses only relatively fast "local" memory, which in turn is the major source of the good performance found for the multi-VM per host platforms. The native and 1-VM per host platforms cannot avoid some amount of remote memory accesses [7]. However, there is a subtlety for the 2-VMs per host configuration. By default, both VMs are scheduled across both NUMA nodes since the number of vCPUs of each VM is greater than the number of physical cores on one NUMA node. For fully-committed hosts (as here) it is often beneficial to

set `numa.vcpu.preferHT=true` for each VM. This option instructs the scheduler to count logical processors as cores with the result that each of the two VMs is seen to fit on a NUMA node.

Storage was configured as pass-through, and the two physical NICs were fully virtualized as described above.

Other virtual machine details are given in "Appendix A: Configuration."

Hadoop

The Cloudera CDH 5.3.0 distribution of Apache Hadoop was installed in all virtual and physical OSes. This distribution supports the second version of the Hadoop File System (HDFS) and both versions 1 and 2 of MapReduce. Version 1 of MapReduce (MR1) was used here since it is more widely used and generally considered to be more stable.

Parameter Tuning

For best performance, the HDFS block size was increased from the 64MiB default to 256MiB. The larger block size increases application efficiency by creating fewer but longer-running Hadoop tasks. The trade-off is that a larger block size needs more memory and may make balancing the workload across a large cluster more difficult for small datasets.

Less than half of OS memory was used for all the Java heaps in the task JVMs (each task runs in a separate JVM). The maximum heap sizes were set to 800MiB and 1200MiB for map and reduce task JVMs, respectively. Best performance was achieved by running at least one map task plus one reduce task per physical core. However, the optimum depends on the platform (native or number of VMs).

Other Hadoop parameters were changed from their defaults in order to further increase efficiency and are listed in "Appendix A: Configuration." Note that a side effect of configuring for best absolute performance is that there are fewer high-latency operations that can "hide" overhead (for example, virtualization, OS, or device latency) and therefore this overhead is more fully exposed as increased elapsed time.

Master Daemons

A common recommended practice is to run the NameNode, Secondary NameNode, and JobTracker master daemons on their own physical machines. For large clusters, dedicating such resources is necessary for performance reasons. This is also a good idea for enhanced reliability of smaller production clusters. However, for well-tuned clusters of small to moderate size, these daemons take very little CPU or memory, so dedicating entire hosts to them is generally wasteful of resources. On the other hand, overall cluster performance depends on low-latency communication between the worker nodes and the master daemons.

Virtualization increases resource utilization by giving the administrator the flexibility to size each virtual machine according to its needs and to run the appropriate number of them on each host. An example of this was described in the Fault Tolerance paper [2] where the NameNode and JobTracker were run in dedicated virtual machines which were placed on hosts with fewer worker nodes. In the previous performance paper [7], the three master daemons were run in three of the worker nodes to maximize resource utilization. Here, in order to optimize master daemon latency, the NameNode and JobTracker were run in dedicated VMs on separate hosts for all the multi-VM per host configurations (for 20 VMs, an additional VM on each of these two hosts was not used as a worker node and kept idle). These two daemons were run in a single dedicated machine for the native and single-VM per host configurations.

Dedicating resources to the NameNode allows it to run single-threaded (`dfs.namenode.handler.count=1`), which is also a work-around for the multi-threaded random number generation algorithm used for selecting DataNodes for replication (HDFS-7122 [10]). Added benefits are that the worker nodes execute more uniformly across the cluster and run-to-run variation is much smaller than in the April 2013 paper [7]. In all cases, the Secondary NameNode was run in a worker node since it requires negligible

CPU and is not latency-sensitive. The Hadoop client has similar properties and was run on the NameNode machine.

Replication

The tests presented here were performed with a replication factor of three for all HDFS storage, as is commonly recommended. This means each original block is copied to two other worker nodes. However, when multiple virtual machines per host are used, there is the undesirable possibility from an availability perspective that two or even three of the replicas of a given block are in different virtual machines on the same host. Hadoop manages replica placement based on the network topology of a cluster. Since Hadoop does not discover the topology itself, the user needs to describe it in a hierarchical fashion as part of the input (the default is a flat topology). Hadoop uses this information both to minimize long-distance network transfers and to maximize availability, including tolerating rack failures.

Hadoop Virtualization Extensions

Virtualization adds another layer to the network topology that needs to be taken into account when deploying a Hadoop cluster. With multiple DataNodes per host, it is important to ensure that data availability is not compromised. This could happen if two replicas of a block are placed on separate DataNodes on one host. Virtualization-aware network topology has been recently added to Apache Hadoop. Hadoop Virtualization Extensions (HVE) [11,12] enables Hadoop to be fully aware of the virtual topology by introducing the "node group" layer. Typically, a node group comprises the DataNodes running on one host. HDFS will then place all replicas of one block in separate node groups. In CDH 5.3.0, this feature became fully supported.

There is only one physical rack (in terms of network topology) for the cluster used here, so there is no concern about rack availability. This allows the specification of a virtual rack topology. The worker nodes were organized into 16 racks with 2 node groups each. The node groups with the master daemons (JobTracker and NameNode) were placed in the same rack. The advantage of this topology is that each rack has a homogeneous set of node groups (that is, all node groups in a rack have the same number of worker nodes). The second and third replicas of a block are placed in different node groups of a single rack and this homogeneity ensures a uniform distribution of replicas across worker nodes.

Compression

Compression with the Snappy codec was enabled for the intermediate map output data (but not for HDFS data); the amount of storage for such data and the network traffic for the shuffle phase was reduced by almost a factor of five. Some CPU is needed for the compression algorithm but overall performance was noticeably increased.

Hadoop Benchmarks

Several example applications are included with the Cloudera distribution. Three of these are often run as standard benchmarks: TeraGen, TeraSort, and TeraValidate. These are collectively referred to as the TeraSort suite. The maximum number of simultaneous map and reduce tasks (slots) for these applications are given in Table 1. The number of slots (as well as the other Hadoop parameters) was tuned for each platform to achieve minimum elapsed time for the sum of the three applications. Configuring one slot per data disk gives close to optimal performance; only very slightly better performance was achieved on some platforms with more slots. TeraSort dominated the tuning since it has by far the longest elapsed time. The map and reduce slots are managed separately in MR1, so the number of slots shown applies to both kinds of tasks. This is opposed to version 2 of MapReduce, where the slots are combined into a single pool and managed by YARN [17]. The latter approach is often easier to manage, especially for clusters that run many small jobs. However, the former approach allows map tasks and the shuffle part of the reduce tasks to run simultaneously in a way that enables the output of maps to "pipeline" into the shuffle by way of the Linux buffer cache.

PLATFORM	WORKER NODES	RESOURCES PER WORKER NODE			MAP/REDUCE SLOTS		TERASORT MAP TASKS, 30TB DATASET
		CPUs/ vCPUs	Memory, GiB	Data disks	Cluster	Worker	
Native	31	40	256	20	682	22	111848
1 VM	31	40	241	20	620	20	112220
2 VMs	62	20	120.5	10	682	11	111848
4 VMs	126	10	60.25	5	756	6	111888
10 VMs	318	4	24.1	2	954	3	112572
20 VMs	636	2	12.05	1	636	1	111936

Table 1. Worker node resources, cluster and per-worker number of map and reduce slots, total number of TeraSort map tasks. Number of VMs is per host.

This set of applications creates, sorts, and validates a large number of 100-byte records. Each record comprises a 10-byte key (which the sort is based on) and 90 bytes of data. The applications do considerable computation, networking, and storage I/O and are often considered to be representative of large batch Hadoop workloads. Results are reported for 300 billion records, which is referred to as the "30TB" dataset. This is over half the maximum size that can be run on this cluster based on disk capacity. For the replication and compression configuration used here, TeraSort requires raw disk space about 6.5 times larger than the dataset size (three copies each of the input and output, plus compressed intermediate data). Note that the total memory of the cluster is only 8TiB, which eliminates the possibility of caching the dataset in memory. Results from tests with smaller dataset sizes are also reported for "weak" scaling analyses and to enable comparisons with other published tests.

TeraGen creates the data and is similar to TestDFSIO-write (a commonly-used storage test application) except that significant computation is involved in creating the random data. The map tasks each write a "partition" (which is just an HDFS file, not to be confused with a disk partition) directly to HDFS so there is no reduce phase. All the partitions are exactly the same size. Writing to HDFS requires a large amount of network bandwidth for replication: the number of bytes transmitted and received are both twice as great as the dataset size.

TeraSort does the actual sorting, reading the data generated by TeraGen from HDFS and writing the sorted data back to HDFS in a number of partitions. The default replication factor for the output of this application is one, but this is overridden so that there are three copies of all input and output blocks (which is more typical of production practice). The first of two computational phases is referred to as "map-shuffle." For the largest dataset, over 100 "waves" of map tasks are performed in each map slot, where each task sorts one block. The map task output is always spilled to disk, but if there is enough memory available and the reduce tasks are configured to start early (see the MapReduce "slowstart" parameter), then the TaskTrackers can shuffle map output from the Linux buffer cache rather than reading it from disk. The overall shuffle can then complete within 30 seconds after the last map task in most cases. For timing purposes the end of this phase is when the last map task finishes. In the "reduce-merge" phase the shuffle is finished, the reduce-sort operation is performed (a few seconds), and then each reduce task reads the shuffled data, merges it, and writes the result to an output partition. The boundaries of each output partition (minimum and maximum key values) are determined at the start of the application by sampling the data. Increasing the number of samples from the default of 100,000 to 2,000,000 yields far more uniformly-sized output partitions (and consequently a balanced load across the reduce tasks) while increasing the cost of the sampling to only about five seconds.

TeraValidate reads all the sorted data to verify it is correct (that is, it is in order). The map tasks perform this verification for each output partition independently, and then the single reduce task checks that the last record of each partition comes before the first record of the next one. All tests were validated successfully.

Benchmark Results

Shown in Table 2 and Figure 2 are the measured performance data for the three benchmarks on the native and three virtual configurations for the 30TB dataset. Each case was repeated several times with machine reboots between, with the best run shown. Run-to-run variation is about 1% of the mean for TeraGen and TeraSort but greater for TeraValidate. In the next sections, the elapsed times and metrics from various performance tools are examined with the goal of understanding some of the performance effects of virtualization.

PLATFORM	ELAPSED TIME, S			TERASORT PHASES, ELAPSED TIME, S	
	TeraGen	TeraSort	TeraValidate	map-shuffle	reduce-merge
Native	1072	3825	697	2616	1209
1 VM	1134	4292	728	2979	1313
2 VMs	1077	3568	723	2343	1225
4 VMs	1072	3355	695	2151	1204
10 VMs	1098	3460	649	2195	1265
20 VMs	1289	3562	633	2083	1479

Table 2. Elapsed times for the 30TB dataset. Number of VMs is per host.

Elapsed Time

The elapsed time results show that while a few of the virtualized tests are significantly slower than the corresponding native tests, most of the tests run on the multi-VM per host platforms are faster. Understanding the reasons for these differences, together with having knowledge of application resource needs, will lead to an optimal design for a Hadoop platform.

TeraGen

In the April 2013 paper [7], the throughput limitation of the storage controller led to all the platforms having similar performance. With a current generation controller, this is no longer a problem (at least not for hard disks, flash drives are another matter). The result is a much higher sustained throughput (over 2500 MiB/s per host), and a greater sensitivity to which platform is used and to configuration parameters.

A single task slot for the 20-VM per host platform is the only choice since running two map tasks simultaneously per worker node would be too many. This lack of flexibility in tuning the load per worker is a notable drawback of configuring a large number of very small VMs. However, the bulk of the extra elapsed time for this platform is due to how replication works within HDFS. The destination nodes for the second and third replicas are chosen randomly (under certain constraints). With a larger number of DataNodes, it becomes more likely that at a particular point in time some DataNodes will be receiving several replicas, while others will be receiving none. That is, while the storage throughput due to writing the first replica is evenly distributed across the cluster for all platforms, successive replicas lead to an increasingly uneven distribution with more DataNodes. The trend is similar for total capacity utilization. This issue is noted in HDFS-7122 [10], and discussed further in the "Single Replica" section below.

These effects are just noticeable in the 10-VM per host platform. With a more appropriate number of map task slots and a smaller replication effect, the differences from the native platform for the 2-, 4-, and 10-VM per host platforms are no more than 2.4%. There is sufficient CPU available and storage is essentially pass-through from the guest OS to the physical disks, so this remaining small cost is mostly due to the extra network latency from the virtualization layer. Tuning the host and guest network drivers as described above helps to minimize this latency.

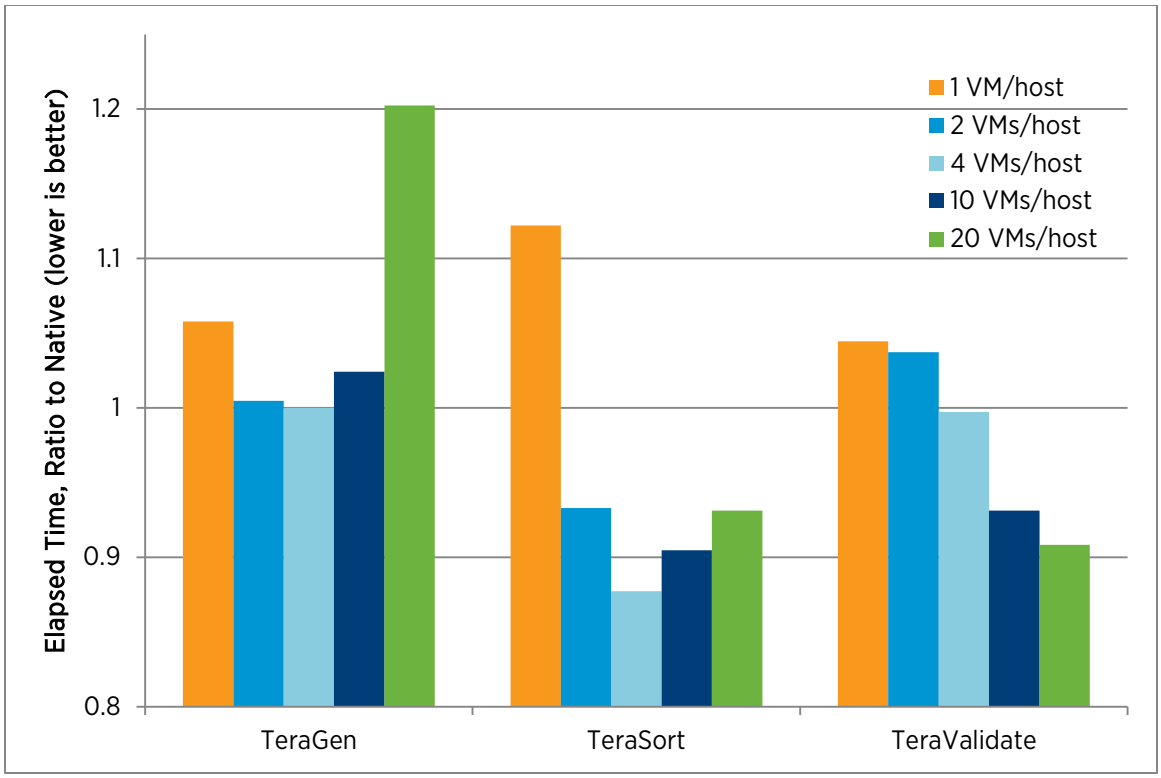

Figure 2. Ratio of elapsed times on virtualized platforms to the native platform. Number of VMs is per host. Lower is better.

TeraSort

The large dataset size and relatively long execution time help to balance the work across the cluster and make this the most reproducible benchmark in the suite. For instance, as noted above, the 30TB dataset comprises about 112,000 blocks, so each map task slot processes at least 118 blocks on average. This is more than large enough to minimize the "long pole" effect of waiting for a single task to finish at the end. The long pole effect is noticeable for the 1TB and 3TB cases discussed in the "Dataset Size" section.

The apples-to-apples comparison of the single-VM per host platform with the native platform shows that the former has a 12% greater elapsed time, which is the same as found previously [7]. This difference can be broken down into approximately equal software and hardware components. Most of the former comes from the costs of virtualizing network and storage I/O, with the rest due to scheduling and other responsibilities of the hypervisor. Most of the latter is due to managing memory pages. This is done in hardware using the Extended Page Tables (EPT) facility on Intel processors [13].

EPT enables the management in hardware of both host and guest memory pages, but requires a two-level page table walk on TLB misses. The use of large memory pages (2MiB) at both levels is essential to minimize EPT costs. Large pages are used automatically by ESXi, Linux (Transparent Huge Pages), and Java, so normally no configuration is necessary. However, if memory is over-committed in the hypervisor (after taking into account the memory requirements of the hypervisor itself), then ESXi will leave the "high" memory state (shown in the memory screen of `esxtop`) and start to break host large pages into small pages (4KiB) in order to apply memory management techniques that allow the VMs to continue to run. While this is a critical capability in many scenarios, memory pressure is avoided and best performance is achieved by allocating only up to 94% of host memory to VMs.

All of the multi-VM per host platforms are significantly faster than native. The optimum is four VMs, which is 12% faster than native and 22% faster than the single-VM per host platform. This is despite the fact that all of these

cases still bear similar I/O and EPT costs as the single VM case. The reasons for this speedup were described in the April 2013 paper [7] and are extended here. The optimal number of VMs was also found to be four in that study, but this resulted in a much smaller 2% elapsed time reduction compared to native. In Appendix C of the April 2013 paper [7], it was shown that significant storage efficiency is gained by having more DataNodes with fewer disks per DataNode. However, this effect applies to HDFS reads or (as noted above) single-replica writes. So, for the 4-VM per host platform, the reduce-merge phase was faster than native in the previous paper [7] with a single replica for the output of TeraSort, while it is somewhat slower than native here with three replicas.

Most of the speedup seen in Figure 2 is due to NUMA effects. For all the multi-VM per host platforms, the ESXi NUMA scheduler puts all of the CPU and memory of a given VM on a single NUMA node. With the ESXi NUMA parameters used here, the VMs never migrate between NUMA nodes. The result is that no VMs on a host ever access memory on the "remote" NUMA node (that is, VM memory accesses are always 100% local). All of this was also true for the previous tests [7], but the speedup here over the 1-VM per host platform is much greater. The 10-core processors have two to three times the processing power and almost double the local memory bandwidth compared to the older quad-core processors, but only 25% more interconnect (QPI) bandwidth between the processors. Thus a larger fraction of interconnect bandwidth is consumed, which drives up remote memory latency. For both the native and 1-VM per host platforms, this fraction was measured previously to be 19%, and now it is estimated to be 27%.

It is possible that remote memory accesses will effectively become increasingly expensive in the future. For example, the next-generation Intel "Haswell" processors have up to 18 cores, but the interconnect bandwidth increases by just 20%.

Memory latency has the largest impact on application performance when the CPU is fully utilized, as in the map-shuffle phase. Table 2 shows that the multi-VM per host platforms are up to 28% faster than the single-VM platform for this phase, compared to 12% faster previously. The increase in elapsed time for the map-shuffle phase of the single-VM per host platform compared to native drops to 14% from 17% previously, most likely because the I/O load (and the associated virtualization costs) during this phase is much lower since compression was not used before.

The reduce-merge phase as a workload is very similar to TeraGen. Both are dominated by writing three replicas of every block to HDFS and the associated network traffic. The reduce-merge phase has the extra work of reading compressed intermediate data from disk. For all platforms, the reduce-merge phase requires a consistent 12-16% greater elapsed time than TeraGen.

TeraValidate

This is a simple sequential, read-only workload with moderate CPU utilization. In theory, it should not require any network bandwidth, as was the case previously [7]. Here it drives significant networking and the storage performance appears to be not as good as it could be. The reason for both of these observations is that each map task reads a whole HDFS file (TeraSort output partition) instead of just a single block. The first copy of a partition resides on the DataNode where it was created. The blocks of the second and third replicas are scattered over the rest of the cluster. When a map task reads a file, there is no preference for which replica is chosen, except that the first block is local. With a single replica, if the first block is local the rest of the file will be too. With more replicas, the rest of the file may be on various remote DataNodes. In particular, for three replicas, up to 2/3 of the total data read may be from remote DataNodes. Various fixes have been proposed (for example, block affinity groups, optimal selection of replica) to ensure that if an entire file is read from HDFS by one map task, all of its blocks will be (normally) local. For now, creating a map task that reads an entire HDFS file should be considered an inefficient (albeit functionally correct) programming practice. Since the amount and pattern of remote data reads is not very repeatable for the current tests, the run-to-run variation is larger than the other applications, about 5%. The strong trend of reduced elapsed time with increasing number of smaller VMs per host is real, but not related to remote data. Instead, it is almost certainly due to the partitioning effect discussed above and in the April 2013 paper [7]: it is more likely that all of the disks managed by a DataNode can be kept doing useful work when there are fewer disks per DataNode. The trend is also seen for the map-shuffle phase of TeraSort, for apparently the same reason.

Single Replica

The origin of some of the above performance differences between the 4-VM and 20-VM per host platforms is confirmed by specifying a single replica for the output of both TeraGen and TeraSort. The elapsed time results for these cases and the native platform are shown in Table 3. Replication is, of course, important for data availability, but it does not in this case help performance, as is often claimed. This is intuitive for the write workloads (TeraGen, reduce-merge), but it is also true for read workloads for several reasons.

- First, in a well-configured cluster, upwards of 99.8% of TeraSort map tasks are data-local, so there is little opportunity for increased locality with multiple replicas.
- Second, for applications like TeraValidate that read a whole HDFS file, the file will be on a single DataNode, which makes it possible for the reads to be 100% local.
- Third, when less data has been written to hard disks, the reads will tend to come from the outer and faster part of the disks.
- Finally, it is possible to achieve an "embarrassingly parallel" configuration for the special case where each map task reads a whole HDFS file, the TaskTrackers each run a single map task, and the DataNodes each manage just one physical disk.

The last point results in each task reading from a dedicated disk completely independent of all other tasks. This happens for TeraValidate with a single replica on the 20-VM per host platform. Each disk sustains 160 MiB/s and the cluster achieves 100,000 MiB/s for nearly the entire duration of the test. For comparison, the maximum read throughput using a microbenchmark to the fastest (outer) part of one disk is 190 MiB/s ("Appendix B: XFS Storage Performance" shows write performance; read performance is about 1% greater). The elapsed time is 39% less than the 4-VM per host platform since, in that case, multiple map tasks on a given worker node are often reading data from the same disks while other disks are idle. In general, a DataNode could choose the optimal disk for writing a new data block, but this is rarely possible for reads.

PLATFORM	ELAPSED TIME, S			TERASORT PHASES, ELAPSED TIME, S	
	TeraGen	TeraSort	TeraValidate	map-shuffle	reduce-merge
Native	883	3216	600	2318	898
4 VMs	906	2887	515	1962	925
20 VMs	919	2933	312	1998	935

Table 3. Elapsed time for the 30TB dataset with a single HDFS replica. Number of VMs is per host.

With three replicas, Table 2 shows the elapsed time for TeraGen is 20% greater for the 20-VM per host platform compared to the 4-VM per host platform. This drops to about 1% with a single replica (Table 3). The reduce-merge phase of TeraSort has almost an identical pattern. Therefore the replication algorithm appears to lose some uniformity when scaling to a large number of DataNodes (as opposed, for example, to the platform being unable to efficiently support more worker nodes). This is supported by examining the distribution of the total number of blocks written by each DataNode for TeraGen with three replicas, which is approximately proportional to the total write load for that DataNode. For the 4-VM per host platform, the maximum number of such blocks is 2774; this is only 4% greater than the average of 2664. For 20-VM per host, the maximum is 576, which is 9% greater than the average of 528. The non-uniformity in instantaneous write throughput would be considerably greater, increasing the relative difference of the two platforms. With a single replica, the amount of data written is always the same on all DataNodes. Several improvements to the HDFS block placement algorithm have been proposed in HDFS-7122 [10] to improve uniformity.

The small difference between the two platforms for the write workloads with a single replica may be due to their "bursty" nature: each task alternates between CPU and storage work. For the 4-VM per host platform, some tasks will be doing one kind of work, while the rest will be doing the other, thus smoothing out resource needs. For the

20-VM per host platform only one task is running in a small worker node, so sometimes there will be a CPU bottleneck. It is possible to alleviate this by configuring three vCPUs per VM, which enables the hypervisor to smooth out resource needs. However, CPU over-commitment should be considered a very advanced tune: it was found to be not helpful for the three-replica configuration or for larger VMs. In any CPU over-commitment scenario, it is highly recommended that the "ready" time of each VM be monitored carefully. A low value of this metric (below 5%) indicates that the VM is still able to acquire all the CPU resources it requests.

CPU Utilization

The CPU utilization for the native, 1-VM per host, and 4-VM per host platforms is shown in Figure 3. This was measured with `mpstat` on native (calculated as 100-`%idle`), and with `esxtop` on the virtual platforms (`%pcpu util`). The three benchmarks were run with a 90 second sleep between them in order to separate them in the figure.

The somewhat higher CPU utilization for TeraGen on the virtualized platforms is mostly due to network virtualization costs due to the very high network bandwidth associated with HDFS replication (see "Network Throughput").

All of the platforms come close to saturating the CPU during the map-shuffle phase of TeraSort. The lowest utilization for this phase is the 1-VM per host platform at 95%; the utilization could have been increased with a larger number of tasks, however, doing this does not reduce the overall elapsed time. During this phase, the CPU cost of virtualization must manifest itself as increased elapsed time. As discussed above, this is more than offset for the multi-VM per host platforms by reduced memory latency. After a quick transition (due to uniform completion of all the shuffle tasks) to the reduce-merge phase, the virtualized platforms have the same or a slightly higher CPU utilization. As for TeraGen, virtualization costs also lead to small increases in elapsed time even though there is spare CPU available. This indicates that increased network latency has a larger effect on elapsed time for these two workloads than increased CPU utilization.

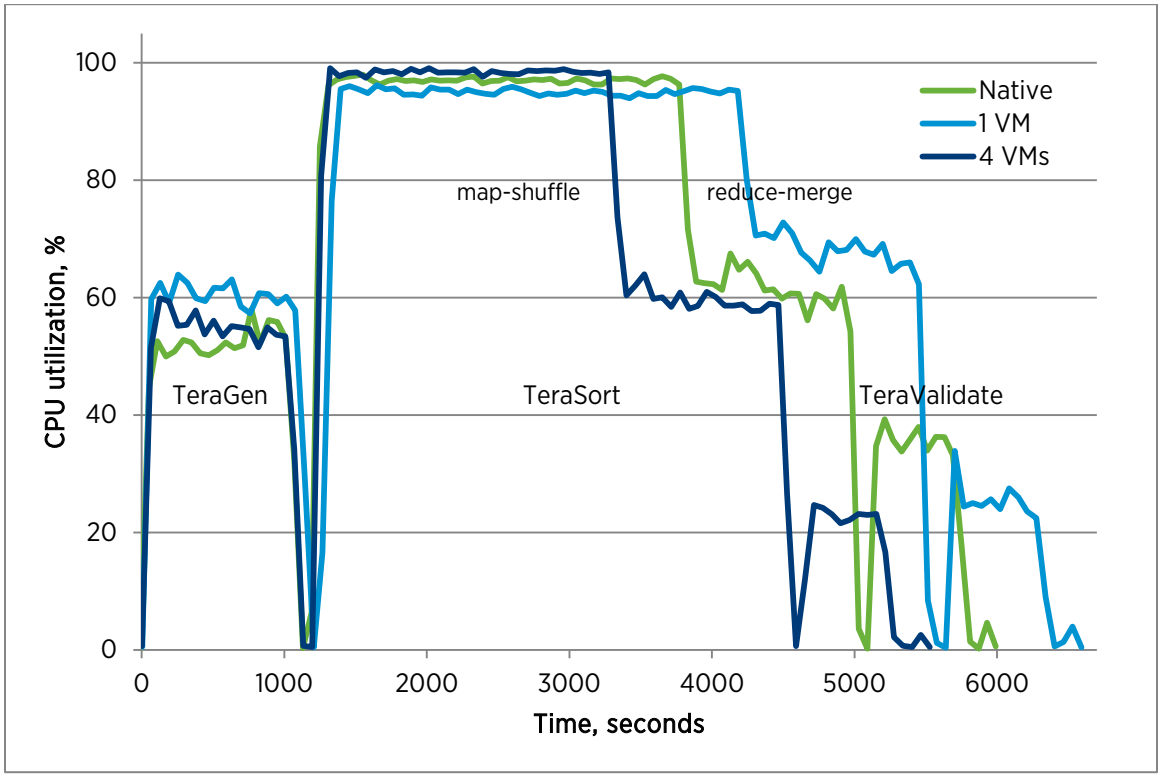

Figure 3. CPU utilization measured on one host for a 30TB dataset. Number of VMs is per host.

TeraValidate CPU utilization for the native platform is 40-50% higher than the virtual platforms. Some increase is expected since the 4-VM per host platform has better memory efficiency and the 1-VM per host platform has a slightly longer elapsed time, however, the magnitude of the difference requires more investigation.

At the end of each benchmark test or phase, there is a sudden drop in CPU utilization. This shows that the load in all cases is well-balanced across the cluster and the performance of the tasks is uniform. An important part of achieving this is ensuring the master daemons, especially the NameNode, do not have to compete for CPU resources with worker nodes and are able to deliver low-latency responses to their clients. Therefore these daemons should be run in dedicated VMs, even though they typically use no more than 20% of a CPU core for a cluster of this size. However, this doesn't mean that the hosts these VMs run on cannot also run some worker nodes, especially for small- to moderate-sized clusters. For the 4-VM per host platform, this means that there are effectively 31.5 hosts running worker tasks, or 1.6% more than the 31 hosts configured as workers for the native and 1-VM per host platforms. For smaller clusters, being able to use the excess capacity of the master daemon hosts for worker nodes becomes much more significant.

Storage Throughput

The total read and write storage throughput for a representative host is shown in Figure 4. Multiplying the throughput for either read or write by the elapsed time for a particular phase and by the number of hosts gives the total amount of data transferred for that phase. For both TeraGen and reduce-merge, the result is a value very close to three times the dataset size. This is exactly what is expected, which means there are no unexpected write operations or extra "spills" to disk. Such spills are often unavoidable (especially for very large datasets) but may also indicate that better tuning of Hadoop parameters is possible. Similarly, the total amount of data read during the map-shuffle phase and during TeraValidate adds up to the database size.

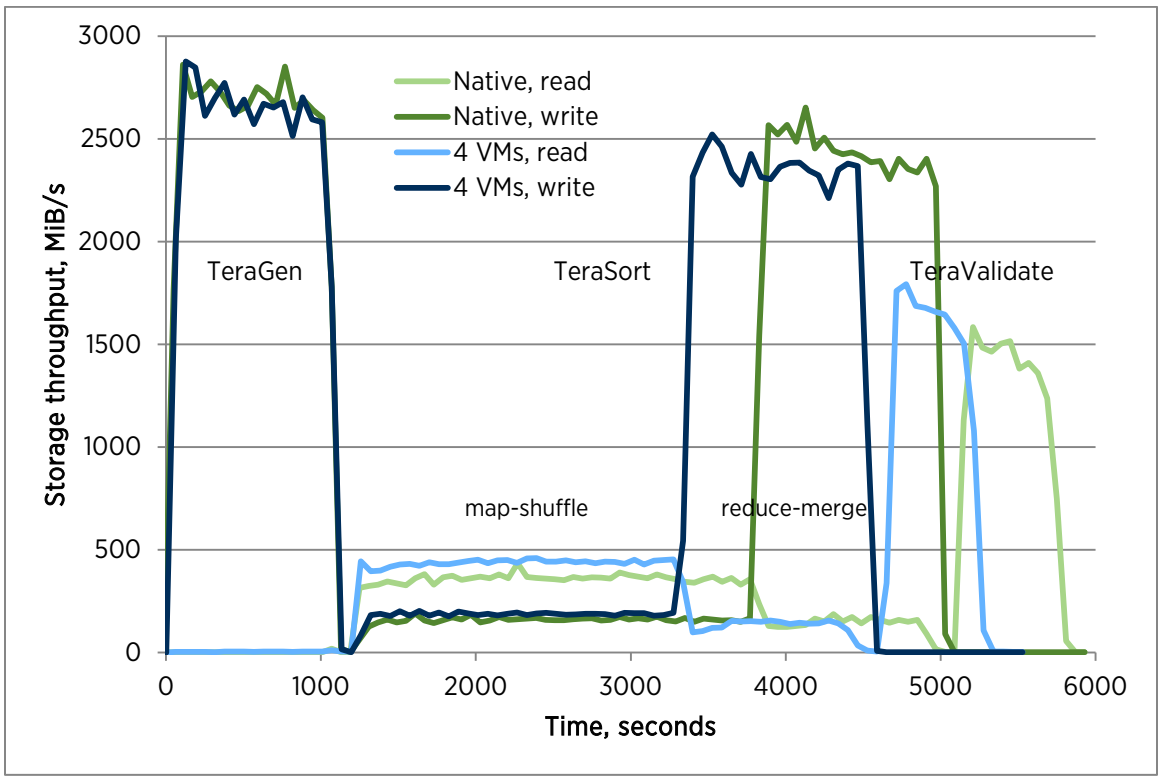

Figure 4. Storage throughput measured on one host for a 30TB dataset. Number of VMs is per host.

Storage throughput associated with the map output of TeraSort is more subtle since it is compressed. This data is written twice: once by the map tasks, and once by the reduce tasks after the data is shuffled. It is then read once

by the reduce tasks during the merge. The figure shows that twice as much data are written than read, as expected. By dividing the total observed amount of data transferred by the expected amount if there were no compression gives an average compression ratio of 4.9. Map output compression should always be considered for such highly compressible data.

The write workloads for the 1-VM per host platform (not shown in the figure) have 6-9% lower storage throughput than the native platform. Since there is sufficient CPU available to cover the virtualization costs, this appears to be at least partially due to the increased latency from virtualizing storage and networking. Thus, tuning for network latency as discussed above helps storage and application throughput for write workloads with replication. The multi-VM per host platforms bring a small increase in bandwidth and a decrease in CPU utilization, which indicates some efficiency gain from memory locality.

Network Throughput

The network receive and transmit throughputs are shown in Figure 5 for the 4-VM per host platform. The traffic is well-balanced across the two NICs, indicating the configuration of the bonding driver is appropriate for these workloads. Receive (Rx) and transmit (Tx) traffic are also nearly the same, which is evidence of uniformity across the cluster. The throughput approaches 8000 Mib/s both ways for each NIC for TeraGen. While 9440 Mib/s is easily achieved on each NIC simultaneously for one-way traffic using the standard networking microbenchmark netperf [18], the limit drops to about 8950 Mib/s each way for two-way traffic. Thus network capacity is close to becoming the performance bottleneck. It appears not to be, since a relatively small amount of read storage I/O reduces the network throughput for the reduce-merge phase.

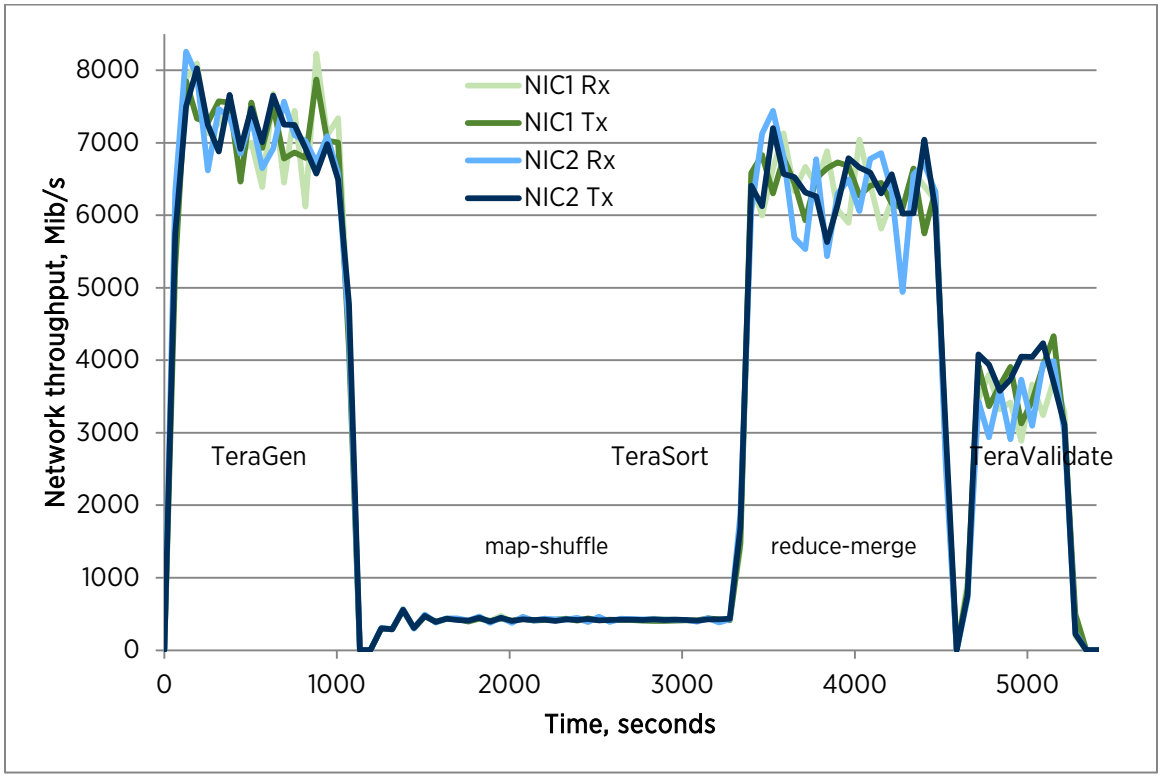

Figure 5. Network receive and transmit throughput for both NICs measured on one host for the 4-VM per host platform.

If newer processors and higher-performing storage are being considered for a new Hadoop cluster, then higher-capacity networking should be part of the specification in order to maintain balance. Upgrading to 40GbE networking should be more than adequate and would require fewer wires. However, compared to adding one or two more 10GbE NIC ports to the bonding driver, using a single 40GbE port loses the advantage of increased

availability in the event of a NIC or switch failure. The low map-shuffle network throughput reflects the effective compression of map output data.

Similar to the storage throughput analysis, each phase can be integrated over time and summed over the cluster to find the total amount of data received and transmitted. Any anomalies may indicate a configuration problem or a lack of understanding of what the application is doing. The result for TeraGen and reduce-merge is twice the cluster size, which is the amount needed to create the second and third replicas on other hosts. For TeraValidate, the result is about 55% of the dataset size. As explained in the "Elapsed Time" section, this is a characteristic of the application interacting with HDFS. The network throughput could be essentially eliminated for TeraValidate by modifying the application (reading blocks instead of files) or by enhancing HDFS as discussed above. Either solution would greatly increase TeraValidate performance (possibly close to the single replica case on the 20-VM per host platform). In general, data visualization such as the above figures are crucial in discovering anomalies and ensuring applications are running as intended.

Dataset Size

Table 4 shows the performance metrics on the 4-VM per host platform for several dataset sizes from the commonly-published 1TB size up to the 30TB size discussed above. Also shown is a throughput metric: the dataset size in GB divided by the TeraSort elapsed time. For good scaling this metric should be constant or slightly decreasing with size, taking into account the `log(N)` work component of a sort (where `N` is the number of rows in the dataset).

Scaling with dataset size is super-linear up to 10TB, due to better amortization of initialization, JVM start-up, Java compilation, and long tail effects. The decrease in throughput for 30TB is due to the `log(N)` effect plus the effect of using more of the slower, inner part of all the disk drives. The former effect should theoretically account for a 4% reduction in throughput compared to the 6% reduction observed.

DATASET SIZE, TB	ELAPSED TIME, S			TERASORT PHASES, ELAPSED TIME, S		TERASORT THROUGHPUT, GB/S
	TeraGen	TeraSort	TeraValidate	map-shuffle	reduce-merge	
1	69	137	33	82	55	7.30
3	124	380	104	248	132	7.89
10	376	1054	214	674	380	9.49
30	1072	3355	695	2151	1204	8.94

Table 4. Elapsed time and throughput for several dataset sizes on the 4-VM per host platform.

Comparison with Earlier Tests

In the previous tests [7], an 8TB dataset was used with an HDFS replication of 2 for the output of TeraGen and 1 for TeraSort. The results of those tests plus the corresponding results from the current setup (except replication was changed to match the earlier tests) are shown in Table 5. The performance of the current setup for the various tests is between 1.9 and 2.6 times faster than previously. This is mostly due to improvements in hardware (each host has 10-core instead of quad-core processors, much more memory, two NICs instead of one, 20 data disks instead of 12), however there were important improvements in the software stack (Hadoop distribution, guest OS, hypervisor) as well. Hadoop tuning also played a significant role, especially isolating the master daemons and using compression for the intermediate map output data.

BENCHMARK DATE	ELAPSED TIME, S			TERASORT PHASES, ELAPSED TIME, S	
	TeraGen	TeraSort	TeraValidate	map-shuffle	reduce-merge
April 2013 [7]	490	2046	349	1491	555
Current	259	824	176	570	254

Table 5. Elapsed time and throughput for previous and current tests for the 8TB dataset size on 4-VM per host platforms. HDFS replication is 2 for TeraGen and 1 for the output of TeraSort.

Best Practices

Some suggestions to optimize virtualized Hadoop performance follow. In the future, TaskTrackers and DataNodes will likely be managed in separate virtual machines, hence the sizing of them should be considered independent.

- Size TaskTracker virtual machines to fit on one NUMA node. Each VM will access only local memory with much lower latency.
- Size DataNode virtual machines so that each owns a small number of disks (preferably five or fewer). Such partitioning increases read storage throughput by flattening the I/O load across the disks.
- If Hadoop virtual machines run both a TaskTracker and a DataNode (as is the standard practice today), size the VMs to be the smaller of the above two.
- Ensure there is sufficient memory left over for the Linux buffer cache in TaskTracker VMs after memory is allocated for all the Hadoop processes. This enables TaskTrackers to obtain map task output directly from the cache instead of reading it from disk.
- After sizing VMs with the above constraints, allocate host hardware to accommodate the desired number of these VMs.
- Ensure large pages are used by the hypervisor, guest OS, and Java. This is the default starting with ESX 3.5 and recent Linux and Java distributions. Manual configuration of guest memory and Hadoop Java heap will be needed if Transparent Huge Pages is not available or not enabled in the guest OS.
- Specify the virtual machine memory size to avoid leaving host "high" memory state with the associated breaking of host large pages into small pages. For 256GiB hosts, allocating no more than 94% of memory to VMs is recommended. For larger hosts, a slightly larger percentage may be allocated, but for smaller hosts less memory should be allocated to VMs. See memory management resources for more details ([14] and references therein).
- Avoid multi-queue networking in all but possibly the largest VMs: Hadoop drives a high packet rate, but not high enough to justify the overhead of multi-queue. Check `/proc/interrupts` for the number of queues and the virtual network adapter options to change it. At the host level, multi-queue networking is usually not needed either, but performance should be tested before disabling this feature in production.
- Evaluate network adapters and drivers with respect to hardware features such as checksum offload, TCP segmentation offload (TSO), jumbo frames (JF), large receive offload (LRO), and support for high memory DMA and multiple scatter-gather elements per Tx frame. These are all supported by vSphere, however, jumbo frames require manual configuration of the virtual switches and adapters.
- If storage is configured manually (not through the vSphere UI), ensure it is block-aligned (for example, as described in the following blog [16]).
- Ensure predictable and uniform storage performance with XFS by setting `agcount=1` during file system creation on Hadoop data disks.

- Check individual hosts and VMs regularly for amount of work done (for example, the number of tasks finished per unit time). Inconsistencies may indicate hardware problems (especially disk drives) or variations in machine configurations.

Conclusion

The results presented here for a cluster of 32 high-performance hosts show that virtualizing Hadoop on vSphere 6 works very well. A few configurations show measurably greater elapsed time, but the reasons for this are explained so that in practice these configurations may be easily avoided. Platforms with two to ten VMs per host are recommended for two-socket hosts, with four VMs being the optimum. In that case, performance is the same as the native platform for the two I/O-dominated tests and 12% better for TeraSort. These results are due in large part to the NUMA properties of current servers, and some evidence is presented that indicates the ability to manage memory through the use of virtual machines will become increasingly important as processor capabilities outrun remote memory bandwidth.

Appendix A: Configuration

Units

- K=1000, M=K*K, G=K*K*K, T=K*K*K*K
- Ki=1024, Mi=Ki*Ki, Gi=Ki*Ki*Ki, Ti=Ki*Ki*Ki*Ki
- b=bit, B=byte, s=second

Hardware

- Cluster: 32X Dell PowerEdge R720xd
- Host CPU and memory
  - Processors: 2X Intel Xeon E5-2680v2, 2800MHz, 25MiB L3 cache
  - Memory: 256GiB: 16X 16GiB ECC DDR3 DIMMs, 1866 MHz
- Host BIOS settings
  - Intel Hyper-Threading Technology: enabled (default)
  - Intel TurboMode: enabled (default)
  - Performance Per Watt (OS) (default)
- Host storage controller
  - Dell H220: LSI SAS9207-8i, Internal Passthrough Host Bus Adapter, x8 PCI Express 3.0
  - Firmware version: 18
- Host storage disks
  - 23X 600GB 10K RPM SAS 6Gbps (WD6001BKHG)
  - 2 disks for VM storage and native OS install
  - 20 disks for Hadoop data

    One partition per disk, aligned at 8MiB boundary:

```
partedUtil mklabel <disk> gpt
last=`partedUtil getUsableSectors <disk> | awk '{print $2}'`
partedUtil setptbl "<disk>" gpt "1 16384 $last \
      EBD0A0A2B9E5443387C068B6B72699C7 0"
```

- Host network adapter: Intel Ethernet X540 DP 10GBASE-T (2 ports)
- Network switch: 2X Extreme Summit X670V-48t-BF-AC
  - Each switch connected to each host

Hypervisor

- vSphere 6.0 (RC build 2412978)
- Network driver: ixgbe 3.7.13.7.14iov-NAPI

  - options: `LRO=1,1 VMDQ=1,1 FdirMode=0,0 InterruptThrottleRate=32000,32000`
- Storage driver: mpt2sas 18.00.00.00.1vmw
  - options: `max_msix_vectors=1`
- Create physical Raw Device Mappings (pRDM) disks:

```
vmkfstools -z <disk> <path>/<file>.vmdk
```

- One vSwitch per 10GbE NIC
  - Virtual Machine Port Group for VM traffic
  - Jumbo frames enabled:

```
esxcfg-vswitch -m 9000 <vswitch>
```

Virtual Machines

- Virtual hardware: version 10
- VMware Tools: installed
- Hadoop data disks: pRDM
- Memory, vCPUS, number of disks: Table 1
- Memory preallocated and reserved
- VM parameters

```
numa.memory.gransize = 1024
numa.vcpu.preferHT = "true"
```

Linux

- Distribution: SLES 11 SP3 x86_64
- Kernel: 3.0.76-0.11-default
- Kernel parameters in `/etc/security/limits.conf`:

```
nofile = 32768
nproc = 32768
```

- Kernel parameters in `/etc/sysctl.conf`:

```
net.ipv4.conf.all.arp_filter = 1
vm.dirty_background_ratio = 1
vm.swappiness = 0
vm.overcommit_memory = 0
net.core.rmem_max = 16777216
net.core.wmem_max = 16777216
net.ipv4.tcp_rmem = 4096 87380 16777216
net.ipv4.tcp_wmem = 4096 65536 16777216
net.ipv4.tcp_mtu_probing = 1
```

- Java: Java HotSpot™ 64-Bit Server VM 1.7.0_55
- Hadoop data disk partition formatted with XFS File system

```
mkfs -t xfs -d agcount=1 –L <label> <partition>
```

- Native OS drivers
  - Network: ixgbe 3.21.2
    - options: `MQ=0,0 InterruptThrottleRate=32000,32000`
  - Storage: mpt2sas 19.00.00.00
    - options: `max_msix_vectors=1`
- Virtual Machine guest OS drivers
  - Network: vmxnet3 1.2.39.0-NAPI
    - options: `num_tqs=1,1 num_rqs=1,1`
  - Storage: pvscsi 1.1.3.0
- Network bonding driver
  - Two slave NICs
  - `BONDING_MODULE_OPTS='mode=2 xmit_hash_policy=layer3+4'`
  - `MTU='9000'`

Hadoop

- Distribution: Cloudera CDH 5.3.0
- `HADOOP_MAPRED_HOME = ${HADOOP_HOME}/share/hadoop/mapreduce1`
- NameNode and JobTracker
  - Multi-VM per host platforms: each in a separate dedicated VM
    20-VM per host platform: extra idle VM in same hosts
  - Native and 1-VM per host platforms: both in one dedicated machine
- Secondary NameNode placed in one of the worker machines
- Number of worker machines: Table 1
- Cluster topology (topology.sh)
  - Multi-VM per host platforms: two hosts (node groups) per Hadoop rack:

```
for i in $*; do
  if [ `expr $i : "cirrus"` -gt 0 ]; then
    echo $i | awk -F "cirrus" '{h=1+($2-1)%32; r=1+(h-1)/2;
                                printf "/r%d/n%d ", r, h}'
  elif [ `expr $i : "192.168."` -gt 0 ]; then
    echo $i | awk -F "." '{h=$3; r=1+(h-1)/2;
                           printf "/r%d/n%d ", r, h}'
  else
    echo -n "/def "
  fi
done
```

  - Native and 1-VM per host platforms: flat (default) topology:

```
for i in $*; do echo -n "/d "; done
```

- Non-default parameters
  - hadoop-env.sh

```
export HADOOP_HEAPSIZE = 1300
```

  - core-site.xml
    - io.file.buffer.size = 131072
    - terasort.partitions.sample = 2000000
    - topology.script.file.name = <path>/topology.sh
  - hdfs-site.xml
    - dfs.datanode.max.transfer.threads = 4096
    - dfs.replication = 3 (final = true)
    - dfs.blocksize = 268435456
    - dfs.datanode.max.transfer.threads = 4096
    - dfs.datanode.readahead.bytes = 4194304
    - dfs.datanode.drop.cache.behind.writes = true
    - dfs.datanode.sync.behind.writes = true
    - dfs.datanode.drop.cache.behind.reads = true
    - dfs.permissions.enabled = false
    - dfs.namenode.handler.count = 1
    - dfs.heartbeat.interval = 1
    - dfs.namenode.replication.interval = 1
    - dfs.blockreport.intervalMsec = 21600000
  - mapred-site.xml
    - mapreduce.framework.name = classic
    - mapred.local.dir = <list of directories>
    - mapred.system.dir = /system/mapred
    - mapred.{map,reduce}.tasks: Table 1
    - mapred.tasktracker.{map,reduce}.tasks.maximum: Table 1
    - mapred.map.child.java.opts = ”-Xmx800m -Xms800m -Xmn256m”
    - mapred.reduce.child.java.opts = ”-Xmx1200m -Xmn256m”
    - mapred.child.ulimit = 4000000
    - mapred.jobtracker.taskScheduler = org.apache.hadoop.mapred.FairScheduler
    - mapred.fairscheduler.locality.delay = 100

mapred.{map,reduce}.tasks.speculative.execution = false
mapred.job.reuse.jvm.num.tasks = -1
mapred.reduce.parallel.copies = 20 (20-VMs per host: 5)
mapred.reduce.slowstart.completed.maps = 0
mapred.job.tracker.handler.count = 10
io.sort.factor = 64
io.sort.mb = 400
io.sort.record.percent = 0.15
mapred.task.timeout = 300000
mapred.tasktracker.shuffle.fadvise = true
mapreduce.ifile.readahead.bytes = 16777216
tasktracker.http.threads = 120
mapred.inmem.merge.threshold = 0
mapred.job.shuffle.merge.percent = 0.95
mapred.job.shuffle.input.buffer.percent = 0.75
mapreduce.tasktracker.outofband.heartbeat = true
mapred.task.cache.levels = 3 (native and 1-VM per host platforms: 2)
mapred.job.reduce.input.buffer.percent = 0.7
mapred.compress.map.output = true
mapred.map.output.compression.codec = org.apache.hadoop.io.compress.SnappyCodec
mapred.jobtracker.completeuserjobs.maximum = 1

Appendix B: XFS Storage Performance

Some simple throughput tests were run to determine the maximum sequential throughput to an XFS filesystem and to determine the effect of the XFS parameter `agcount`. This parameter, which is specified at the time of file system creation, determines the number of "aggregation groups" that XFS will manage. The default depends on disk/LUN size, but will normally be 4 for typical disks. Certain file system locks are applied to the smaller scope of an aggregation group rather than to the whole file system. Very high IOPs workloads then benefit from parallelizing certain metadata operations. However, with I/O sizes typically much greater than 100KiB, Hadoop is a low-to-moderate IOPs workload for which XFS defaults may not be optimal.

The storage benchmark `aio-stress` [15] was used to perform a sequential write test with a 512KiB record size, 20000MiB file size, and a single thread. The test was repeated 28 times sequentially to different files (each in a different directory) on a single 600GB disk that was formatted with XFS and various values of `agcount`. Total data written was 587GB, nearly filling the disk. The results for throughput are shown in Figure 6. The pattern for `agcount=1` is expected: throughput starts off close to 190 MiB/s since the outer (fastest) part of the disk is used first, and then drops to just over 110 MiB/s when the disk becomes full and the inner part must be used. This 41% drop in throughput can easily cause non-uniform storage performance across the cluster, or even within a single host. Greater values of agcount show that XFS partitions the disk into that many regions, and each file is written to one of them. For instance, with `agcount=2` files are written to either the first half of the disk or the second half. The average throughput is similar for all three cases, but only `agcount=1` will give predictable and uniform performance across all disks. Also, larger values of `agcount` will result in significantly lower average throughput if only a portion of a disk is filled.

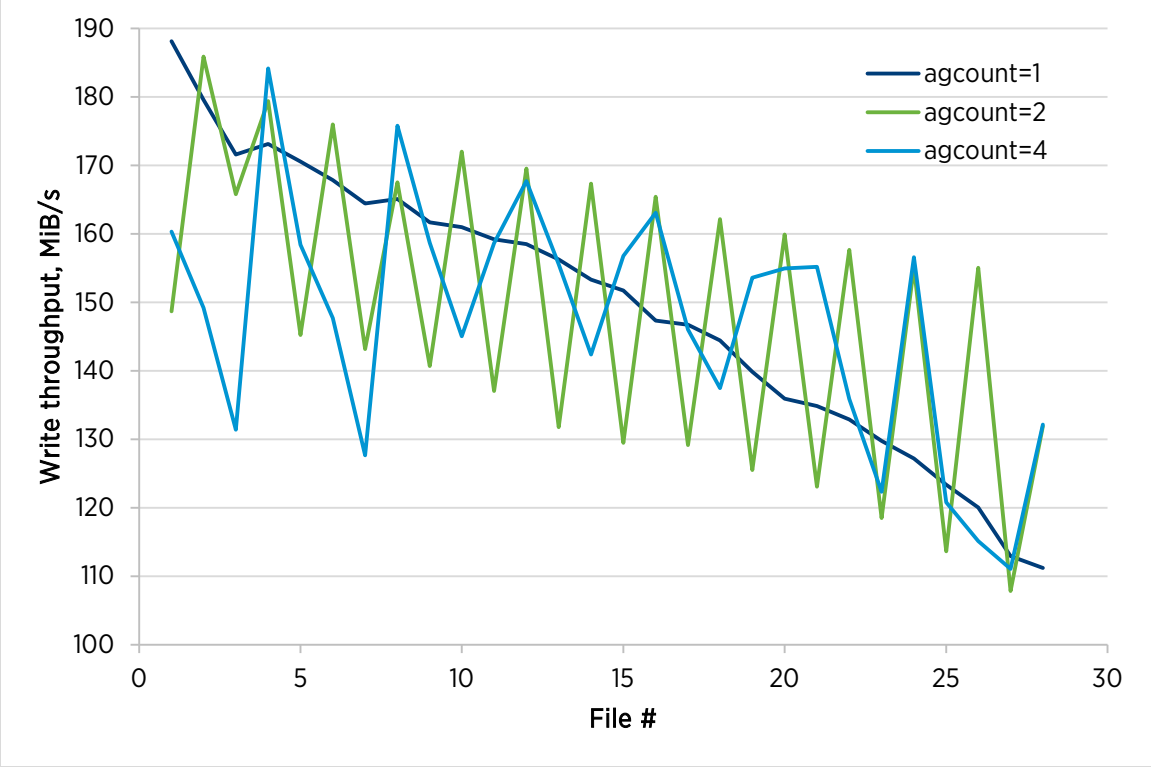

Figure 6. Storage throughput to an XFS filesystem on a single disk for different values of `agcount`. Files are 20000MiB each and written sequentially.

References

[1] White, Tom. *Hadoop: The Definitive Guide*. O'Reilly, 2012.

[2] Buell, Jeff. "Protecting Hadoop with VMware vSphere 5 Fault Tolerance." VMware, Inc., 2012. http://www.vmware.com/resources/techresources/10301.

[3] "Enabling Highly Available and Scalable Hadoop." VMware, Inc. http://www.vmware.com/hadoop/serengeti.html.

[4] Magdon-Ismail, Tariq, et al. "Towards an Elastic Elephant: Enabling Hadoop for the Cloud." VMware Technical Journal, Winter 2013. https://labs.vmware.com/vmtj/toward-an-elastic-elephant-enabling-hadoop-for-the-cloud.

[5] Drummonds, Scott. "Building Block Architecture for Superior Performance." VMware, Inc., 2009. http://communities.vmware.com/blogs/drummonds/2009/02/17/building-block-architecture-for-superior-performance.

[6] Buell, Jeff. "A Benchmarking Case Study of Virtualized Hadoop Performance on VMware vSphere 5." VMware, Inc., 2011. http://www.vmware.com/resources/techresources/10222.

[7] Buell, Jeff. "Virtualized Hadoop Performance with VMware vSphere 5.1." VMware, Inc., 2013. http://www.vmware.com/resources/techresources/10360.

[8] "Raw Device Mapping for local storage." http://kb.vmware.com/kb/1017530. VMware, Inc., 2014.

[9] "Understanding SCSI Check Conditions in VMkernel logs during rescan operations." VMware, Inc., 2014. http://kb.vmware.com/kb/1010244.

[10] "Use of ThreadLocal<Random> results in poor block placement." The Apache Software Foundation, 2014. https://issues.apache.org/jira/browse/HDFS-7122.

[11] "Hadoop Virtualization Extensions on VMware vSphere 5." VMware, Inc., 2012. http://serengeti.cloudfoundry.com/pdf/Hadoop%20Virtualization%20Extensions%20on%20VMware%20vSphere%205.pdf.

[12] "Umbrella of enhancements to support different failure and locality topologies." The Apache Software Foundation, 2013. https://issues.apache.org/jira/browse/HADOOP-8468.

[13] Bhatia, Nikhil. "Performance Evaluation of Intel EPT Hardware Assist." VMware, Inc., 2009. http://www.vmware.com/resources/techresources/10006.

[14] Guo, Fei. "Understanding Memory Resource Management in VMware vSphere 5.0." VMware, Inc., 2011. http://www.vmware.com/resources/techresources/10206.

[15] "AIO-STRESS."OpenBenchmarking.org, 2013. http://openbenchmarking.org/test/pts/aio-stress.

[16] Hogan, Cormac. "Guest OS Partition Alignment." VMware, Inc. 2011. http://blogs.vmware.com/vsphere/2011/08/guest-os-partition-alignment.html.

[17] "Apache Hadoop NextGen MapReduce (YARN)." The Apache Software Foundation, 2014. http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html.

[18] "Netperf." http://www.netperf.org/netperf/.

About the Author

Jeff Buell is a Staff Engineer in the Performance Engineering group at VMware. His work focuses on the performance of various virtualized applications, most recently in the HPC and Big Data areas. His findings can be found in white papers, blog articles, and VMworld presentations.

Acknowledgements

The author thanks Lenin Singaravelu, Tariq Magdon-Ismail, Reza Taheri, Josh Simons, David Morse, and Todd Muirhead for their valuable technical advice and careful reviews of the manuscript.

vmware

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