Virtualized Hadoop Performance with VMware vSphere® 5.1

Performance Study

TECHNICAL WHITE PAPER

Table of Contents

Executive Summary

A cluster of 32 high-performance hosts was used to run three demanding Hadoop applications. The performance of native and several VMware vSphere® configurations was compared. The apples-to-apples case of a single virtual machine per host shows performance close to that of native. Improvements in elapsed time of up to 13% can be achieved by partitioning each host into two or four virtual machines, resulting in competitive or even better than native performance. The origins of the improvements are examined and recommendations for optimal hardware and software configuration 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 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 and Fault Tolerance. The performance implications of protecting the Hadoop master daemons with FT were examined in a previous paper [2].
- Easier deployment with vSphere tools or Serengeti, leading to easier and faster datacenter management [3].
- Sharing resources with other Hadoop clusters or completely different applications, for better datacenter utilization.

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

- Elasticity enables the ability to quickly grow a cluster as needs warrant, and to shrink it just as quickly in order to release resources to other applications.
- Multi-tenancy allows multiple virtual clusters to share a physical cluster while maintaining the highest levels of isolation between them.
- Greater security within each cluster as well as elasticity (the ability to quickly resize a cluster) require the separation of the computational (TaskTracker) and data (DataNode) parts of Hadoop into separate machines. However, data locality (and thus performance) requires them to be on the same physical host, leading to the use of virtual machines.

A detailed discussion of these points is beyond the scope of this paper, but can be found elsewhere [4]. As great 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 and high-throughput benchmarks (the TeraSort suite). While these applications are not 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 typical production jobs. As such, they are good tools for stressing the virtualization layer. 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 and the system administrator must tune the application to that size. Even when the administrator is willing to do this, some distributed applications were designed to scale-out on small nodes and are unable to scale-up to use all the capabilities of modern resource-dense hosts [5]. In a virtualized environment a smaller size may be chosen if that allows the application to be more efficient, or allows the administrator to create a standard template that enables efficient bin-packing of virtual machines onto host servers of various sizes. For instance, if scaled-up 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 virtual machines on them as proved to work well on the smaller hosts. While this scaling flexibility 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 resources as well as application efficiency to be optimized.

A previous 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.

Configuration

Hardware Overview

The hardware configuration is shown in Figure 1. A cluster of 32 host servers was connected to a single Arista 7050S 10GbE switch. Each host was equipped with two Intel X5687 3.6GHz quad-core processors, 16 internal 146GB 15K RPM SAS disks, and a 10GbE Broadcom adapter. The first 24 hosts have 72GB of 1333MHz memory, while the last eight have 96GB. This slight heterogeneity should have no effect on the virtualized performance since the virtual machines are all the same size, and the hypervisor itself does not cache storage I/O. The native operating system (OS) theoretically has an advantage from a larger buffer cache on the 96GB machines, but in practice the smaller cache appears to be sufficient. The internal disks were connected to a PCIe 2.0 x4 storage controller, the HP P410i. This controller has a theoretical throughput of 2000 MB/sec, but is observed to deliver 1000-1400 MB/sec. Power states, including TurboMode, were disabled in the BIOS to improve consistency of results. Intel Hyper-Threading (HT) Technology was enabled in the BIOS and used by the operating system in all cases.

Complete hardware details are given in Appendix A: Configuration.

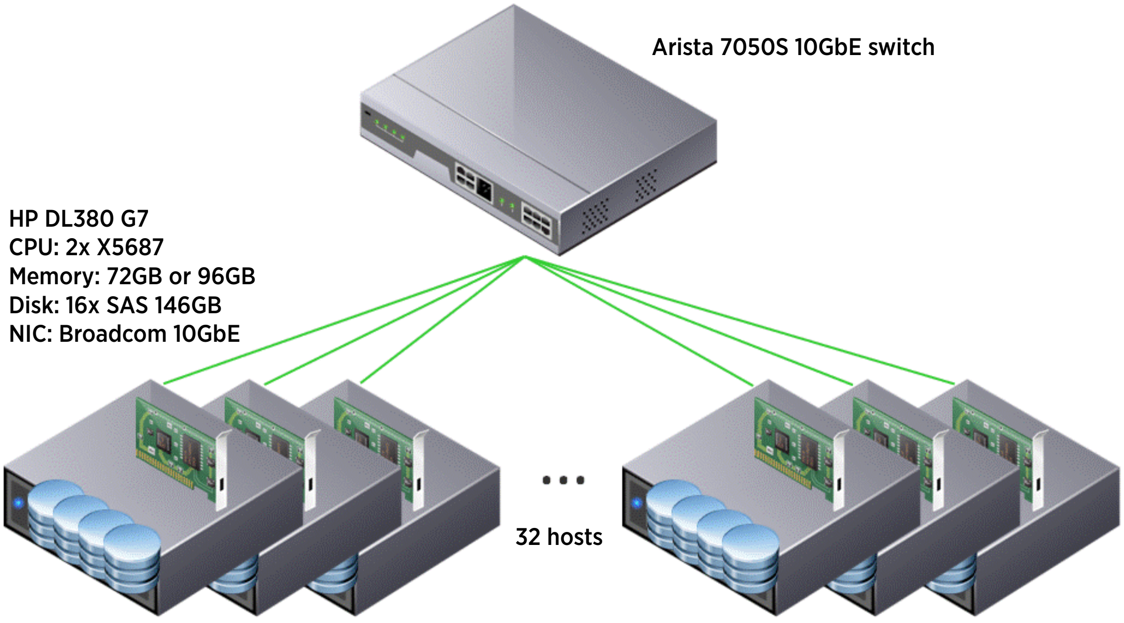

Figure 1. Cluster hardware configuration.

Local Storage

Two of the sixteen internal disks in each host were striped, divided into partitions, 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 so does not create any storage I/O itself. During the tests, the I/O to these two root disks consisted almost exclusively of just Hadoop logs and job statistics. The other fourteen disks were configured as individual disks (that is, there was no striping or RAID). Twelve were used for Hadoop data, while two were kept as spares. The disks were passed through to the virtual machines using physical raw device mappings (known as pRDM). This enabled using the same storage for native and virtual platforms without reformatting, thus eliminating a potential source of differences. A single aligned partition was created on each disk and formatted with EXT4.

System Software

RHEL 6.1 x86_64 was used as the guest operating system in the virtual machines, which were run on VMware vSphere 5.1. The same OS was installed natively and configured as identically as possible to the virtual machines. The virtual network adapter was the default version of vmxnet3 that comes with the OS distribution. Later versions of vmxnet3 that support multiple queues may be used as well; however, it is recommended these be configured with a single queue in most cases for better efficiency. Multiple queues are needed only for much higher packet rates than Hadoop typically drives. At the physical level, the bnx2x network driver was used for both ESXi and the native OS. A single queue (multi_mode=0) was configured in the native OS. Otherwise, default networking parameters were always used. LSI Logic Parallel was chosen for the virtual SCSI controller. PVSCSI is a good alternative, but is usually recommended for workloads with higher IOPs. ESXi and the native OS both use the default hpsa storage driver. Sun Java 6 is recommended for Hadoop; version 1.6.0_26 was used here. A few Linux kernel tunables were increased in order to handle more files and processes.

No hypervisor tuning was done, except each virtual machine was pinned to a NUMA node (which is a processor socket plus the associated memory) for better reproducibility of the results in the two and four virtual machine (VM) per host configurations. For the single-VM platform, virtual NUMA (the default) ensures the guest OS sees the same NUMA topology as the native OS.

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

Virtual Machines

One, two, or four virtual machines were run on each host, for a total of up to 128 virtual machines in the cluster. On each host 16 logical processors, 68GB of memory, and 12 Hadoop data disks were evenly divided among the desired number of virtual machines. That is, the CPU resources were exactly-committed on all hosts and memory was slightly under-committed on the 72GB hosts. A single vSwitch was configured on the 10GbE NIC and shared by all the virtual machines on the host. Virtual machine affinity was applied in the two and four VM per host cases. This was done primarily for repeatability, as memory migrations should be very rare for these configurations but may occasionally occur. Other virtual machine details are given in Appendix A: Configuration.

Hadoop

The Cloudera CDH4.1.1 version of Apache Hadoop was installed on all virtual and physical machines. This distribution supports the second version of the Hadoop File System (HDFS) and both version 1 and 2 of Map-Reduce. MR1 was used here since it is more widely used and generally considered to be more stable. For best performance, the HDFS block size was increased to 256MB from the 64MB default. 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 the Hadoop scheduler. 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 800MB and 1200MB for map and reduce task JVMs, respectively. Best performance was achieved by running one map task plus one reduce task per logical processor. This relatively aggressive configuration is possible because of the high performance hardware used;

on machines with slower SATA disks the common recommendation of one map task and one reduce task per core would likely work better. A few other Hadoop parameters were changed from their defaults in order to increase efficiency further 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" virtualization overhead and therefore this overhead is more fully exposed as increased elapsed time.

As previously noted, all 32 hosts were used to run Hadoop worker nodes. A common recommended practice is to run the NameNode, Secondary NameNode, and JobTracker master daemons on their own machines. For large clusters such resource dedication 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 and so dedicating hosts to them is wasteful of resources. 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. Here, the three master daemons were run in three of the worker nodes to maximize performance of the cluster. No significant performance degradation was found in any of those three nodes, or in a fourth worker node that ran the Hadoop client.

The tests presented here were performed with an HDFS replication factor of two, as is common for small clusters (large clusters are usually configured with three or more). This means each original block is copied to one other worker node. However, when multiple virtual machines per host are used, there is the undesirable possibility from a single-point-of-failure perspective that the replica node chosen is another virtual machine 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. Virtualization adds another layer to the network topology that needs to be taken into account when deploying a Hadoop cluster: inter-VM communication on each host. Development is mostly complete to add virtualization-aware network topology to Apache Hadoop[*]. Among other things, this feature will prevent two copies of the same block from being stored in two VMs on the same host. In the current tests with the replication factor set to two, this behavior can be achieved manually by equating the set of virtual machines running on a particular host to a unique rack. In general for greater levels of replication, this strategy ensures that at least the first replica of every block is sent to a virtual machine on another host.

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 numbers of simultaneous map and reduce tasks across the cluster for these applications are given in Table 1. Also shown in this table are the relative amounts of read and write storage operations.

This set of applications creates, sorts, and validates a large number of 100-Byte records. It does considerable computation, networking, and storage I/O and is often considered to be representative of real Hadoop workloads. Results are reported for eighty billion records, which is referred to as the "8TB" dataset. This is close to the maximum size that can be run on this cluster (TeraSort requires raw disk space about six times larger than the dataset size). Note that the total memory of the cluster is less than 2.5TB, which eliminates the possibility of caching the dataset in memory. Results from tests with a "1TB" dataset size are also reported to enable comparisons with other published tests and to enable experiments with lower disk-per-core ratios.

[*] This is referred to as Hadoop Virtualization Extensions (HVE) and described in [7]. The HVE modifications to Hadoop are tracked in Jira Hadoop-8468 [8].

BENCHMARK	MAP	REDUCE	%READ	%WRITE
TeraGen	512	0	0	100
TeraSort	512	512	40	60
TeraValidate	512	1	100	0

Table 1. Total number of simultaneous tasks and storage characteristics of each benchmark

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 write directly to HDFS so there is no reduce phase.

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 partitioned files. The application itself overrides the specified replication factor (sets it to one) for the output so only one copy is written. The philosophy is that if a data disk is lost then the application can always be rerun, but input data needs replication since it may not be as easily recovered.

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 partitioned file 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 8TB dataset. Each case was repeated several times with machine reboots between and the best run shown. Run-to-run variation is about 1% of the mean for TeraSort and somewhat greater for the other benchmarks. In the next subsections 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, sec			TeraSort phases, elapsed time, sec		
	TeraGen	TeraSort	TeraValidate	map-shuffle	reduce-sort	reduce-merge
Native	478	2084	326	1426	122	536
1 VM	506	2352	342	1662	82	608
2 VMs	496	2095	341	1464	57	574
4 VMs	490	2046	349	1493	36	517

Table 2. Elapsed time data for 8TB dataset size on 32 hosts. Number of VMs is per host.

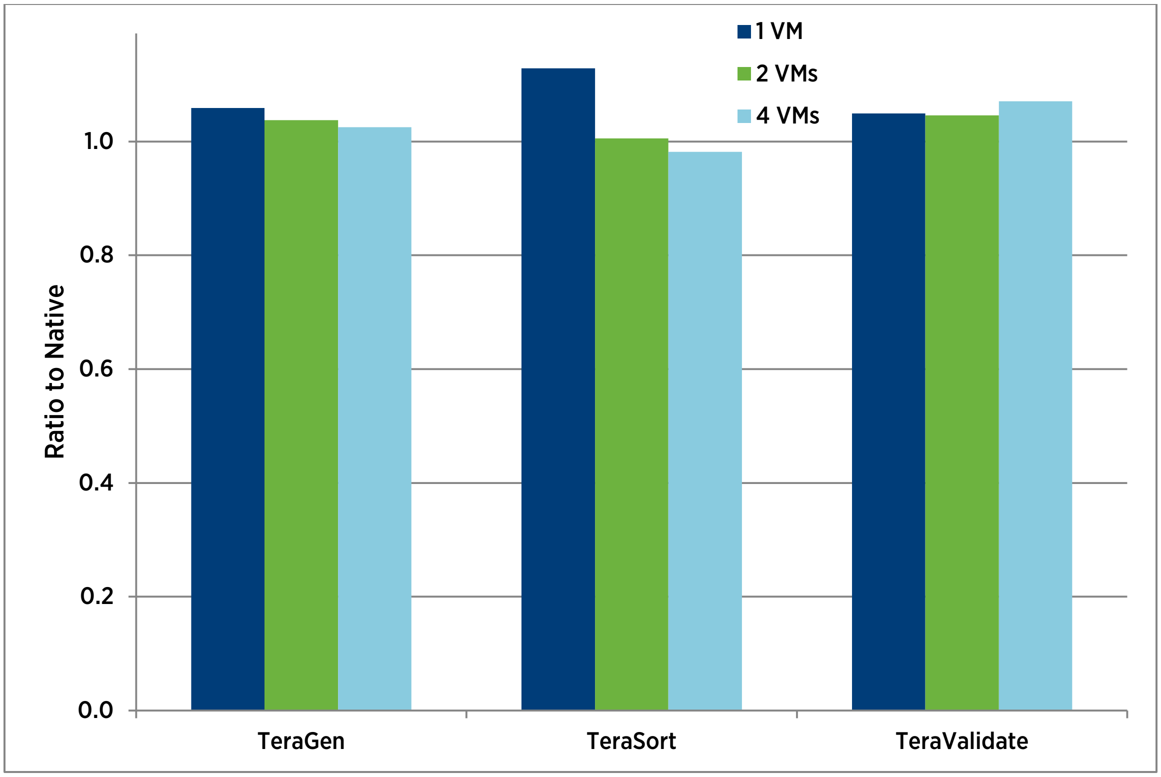

Figure 2. Ratio of elapsed times on virtualized platforms compared to native platform. Number of VMs is per host. Lower is better.

Elapsed Time

The elapsed time results show that the overhead of virtualizing all the TeraSort suite applications is never very large. There is not much performance variation among the platforms for the sequential write-only workload TeraGen. The native platform is slightly more efficient than the virtual platforms and efficiency slightly increases with the number of VMs per host. All of the platforms push between 1000 and 1100 MB/sec disk bandwidth per host, but tests with eight disks per host (33% fewer) on smaller datasets lose only 10% of this bandwidth, which indicates the physical disks are not the bottleneck. Since CPU and the network are not saturated either, this points to a bandwidth limitation of the storage controller for the TeraGen I/O pattern.

TeraValidate is a simple sequential read-only workload with moderate CPU utilization and no network bandwidth. However, it is not as reproducible as the other tests (it tends to get faster with repeated runs). The virtual platforms are 5-7% slower than the native platform for this test.

TeraSort is the most reproducible benchmark in the suite. The large dataset size and relatively long execution time help to balance the work across the cluster. For instance, the dataset comprises 30208 blocks, so each map task slot processes 59 blocks on average. This is large enough to minimize the "long pole" effect of waiting for a single task to finish at the end. TeraSort is also the most important and complex workload, so the remainder of the paper will focus on it.

The "apples-to-apples" comparison of the single-VM platform with the native platform shows that the former has a 13% greater elapsed time. This overhead can be broken down into software and hardware components. Most of the former comes from the costs of virtualizing network and storage I/O. The largest part of the latter is due to managing guest memory pages. This is done in hardware using the Extended Page Tables (EPT) facility on Intel processors [9]. This allows both machine and guest memory pages to be managed in hardware, but requires a two-level page table walk on TLB misses.

The two-VM platform is almost as fast as native, while the four-VM platform is 2% faster. Clearly the speedup gained from configuring multiple virtual machines per host is as large as the virtualization overhead measured on the single-VM platform. TeraSort elapsed time is broken down into three major phases in the last columns of Table 2. The breakdown shows several things about the sources of both the overhead and efficiencies of virtualization. It is necessarily approximate since Hadoop measures overall progress, but individual tasks run independently and are thus not synchronized. For instance, map tasks and the shuffle operation (which is driven by the reduce tasks) are not explicitly tied together. The output of the former is always spilled to disk, but if there are enough hardware resources and the reduce tasks are allowed to start early (see the MapReduce "slowstart" parameter), then TaskTrackers can immediately shuffle map output from the Linux buffer cache rather than reading it from disk. Long-latency read I/O is minimized, and the read-write split for TeraSort in Table 1 is 40-60 instead of 50-50. This form of synchronization is characterized by the entire shuffle finishing soon after the map tasks. The platform does not have as big an effect on this as ensuring there is sufficient memory for the buffer cache. The ability to efficiently chain the map and shuffle operations together is determined primarily by the size of the buffer cache, regardless of test configuration. Sufficient memory for the buffer cache is usually available if the sum of the resident memory (from `top` or similar tools) of the Hadoop JVM processes is not much more than half of total memory.

Another form of synchronization is evident in the reduce-sort elapsed times: each reduce task spends only seven seconds on average in the sort phase, but the tasks run at different times in a period of up to two minutes. This does not necessarily lead to a loss of performance since Hadoop can do other work during this period. Thus the reduce-sort elapsed time is really a measure of how well the reduce tasks are synchronized across the cluster for this phase. Virtualization and multiple VMs are associated with individual reduce tasks getting not as far behind or ahead of the others, although the cause is not understood. As demonstrated for the map-shuffle phase, task synchronization is surely a good property to have; however, it is not clear how much effect the reduce-sort phase has on total performance.

Synchronization is also important for the reduce-merge phase. Here the issue is balancing the amount of work across the reduce tasks in order to minimize the "long pole" effect of laggard tasks. Given a good balance, long poles may indicate slow drives or other hardware issues. In Table 2 the slowest reduce tasks for TeraSort finish one to two minutes behind the bulk of the reduce tasks on all platforms. The slow tasks are not associated with particular hosts or disk drives. This level of imbalance appears to be due to random variations in workload across the cluster and is not related to the dependence on platform of the reduce-merge elapsed times seen in the last column of the table.

For the single-VM platform, map-shuffle and reduce-merge both show about the same overhead as the overall elapsed time. For both phases, configuring multiple virtual machines greatly reduces the elapsed time. However, for map-shuffle all of the reduction is achieved with two VMs, while for reduce-merge most of the reduction comes from four VMs. Both kinds of reductions are due to greater efficiencies enabled by virtualization, as shown in the Hardware Efficiency and Storage Throughput subsections below.

CPU Utilization

Shown in Figure 3 is the CPU utilization for the native and four-VM platforms. The three benchmarks were run with a 30 second sleep between them in order to clearly separate them in the figure. The higher CPU utilization in TeraGen for the four-VM platform is mostly due to virtualizing the very high network bandwidth; HDFS replication results in both transmit and receive bandwidths of about 4 Gb/sec on each host. All platforms saturate the CPU during the map-shuffle phase of TeraSort, where network bandwidths are about 1.3 Gb/sec for both transmit and receive. As noted above, the native platform is about 5% more efficient than the four-VM platform for this phase. After a quick transition to the reduce-merge phase, the virtualized platform has a slightly higher CPU utilization. There is no networking during this phase and the storage I/O can be handled with little overhead due to large I/Os and low IOPs. Some increase in the CPU utilization is expected since the virtual case runs somewhat faster. The processor is both more efficient (due to NUMA) and less efficient (EPT and caches); hardware performance is discussed in the next section.

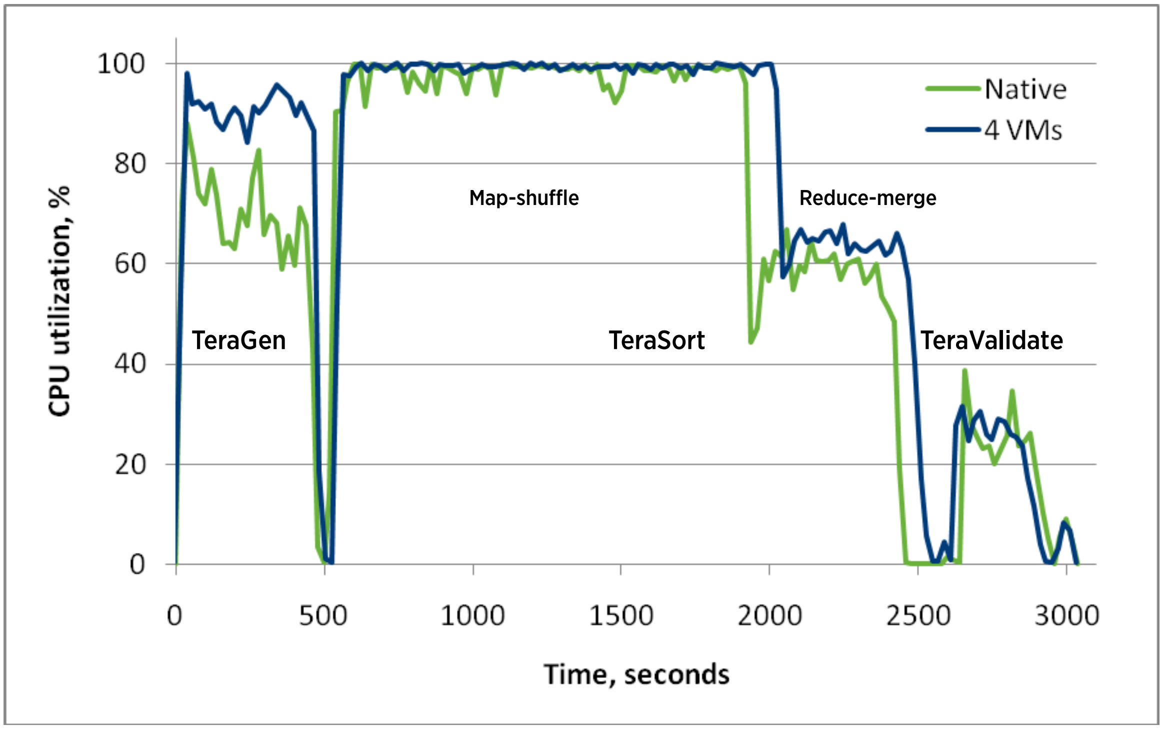

Figure 3. CPU utilization measured on one host for 32-host tests. Number of VMs is per host.

It appears from the chart that the native test finishes TeraSort first, but it actually doesn't (note that TeraValidate starts later). It is only this particular host that finishes its work early; reduce tasks on some of the other hosts are still running. This illustrates the importance of balancing both the workload and the performance of each host across the cluster. Consistent slowness on one host may indicate a hardware problem or software misconfiguration, while slowness over a subset of the cluster often means the hardware is not all identical.[†]

TeraValidate has low CPU utilization, essentially no networking, and reasonable storage IOPs. As a result, there is not much difference in performance among the platforms.

Hardware Efficiency

Intel processors have built-in hardware performance counters that can be used to monitor processor execution [10]. Hundreds of counters are available, but only seven processor core counters may be active at a time on Westmere processors. Internal Intel tools were used to read and process the counters. Hardware counters are most useful when the CPU is the limiting resource; out of all the tests presented here, this happens only during the map-shuffle phase of TeraSort. Shown in Table 3 are a few performance metrics derived from the basic counters. These were taken during the middle (steady-state) of the map-shuffle phase on one host from homogeneous (all 72GB) 24-host tests using a 6TB dataset size. Elapsed time metrics are very similar to the 8TB tests on 32 hosts.

All platforms nearly saturate all processors for the duration of the map-shuffle phase. That is, the processor cores spend little time in the "halted" state. A very small amount of the virtualization overhead is "hidden" by a

[†] In particular, disk drives often exhibit poor performance for a significant period of time before they fail completely. It is standard practice in the computer industry (as well as many others) to source a given component from multiple OEMs, as long as each OEM meets the vendor's criteria. Disk drives are a prime example of this: various models with the same specifications may show different performance, depending on the I/O access pattern. It may be worthwhile, especially for very large installations, to test the various drives using the applications of interest.

reduction in the remaining idle. As shown in Table 2 for this phase, the increase in elapsed time for the single-VM platform compared to the native platform is 16.5%. This virtualization overhead is split almost equally between three major sources. A 5% contribution to the total overhead comes directly from EPT processing. Other counters (not shown) indicate that another 5% is due to other hardware inefficiencies like increased TLB (non-EPT) costs and higher cache miss rates. The latter is reflected in higher memory bandwidths (relative to the amount of data processed). The remaining 6.5% is time spent in the hypervisor. Most of the hypervisor time is consumed processing network I/O. Although storage bandwidth is high (about 550 MB/sec on each host), storage I/O processing in the hypervisor is inexpensive since the IOPs generally stay below 2000. There is also a small amount of hypervisor time used for scheduling and other activities.

Metric	Platform			
	Native	1 VM	2 VMs	4 VMs
Unhalted cycles, %	98.2	99.4	99.6	99.0
EPT cycles, %	0	5.0	5.3	4.4
Memory Bandwidth, GB/sec	13.6	12.7	13.8	14.5
Remote Demand Reads, %	36	38	2	2

Table 3. Selected metrics derived from hardware performance counters during the map-shuffle phase of TeraSort. Number of VMs is per host.

The Linux scheduler tries to keep processes and the memory they allocate on the same NUMA node. In the machines used here (as well as most current machines), a NUMA node is one processor socket plus memory directly attached to it. Memory locality is important since a process accessing memory on the other ("remote") NUMA node will experience much higher memory latency. It becomes even more important on four- and eight-socket machines where remote latency becomes significantly higher and the fraction of requested data that is remote is also higher. For the native platform, 36% of all demand memory reads are remote. This means the Linux scheduler is only partially successful in maintaining memory locality. The single-VM platform is similar to native, indicating the virtual NUMA topology is the same as the physical NUMA topology. Configuring two or four VMs per host drops the remote demand memory reads to 2%. All of that is in the hypervisor since each virtual machine is prevented from doing any remote memory accesses at all. Some of the hypervisor remote memory traffic is inter-VM network traffic within one host, which is handled entirely in memory. An estimate of the effect on application performance based on the difference of remote memory usage in Table 3 is presented in Appendix B: Remote Memory Performance Model. This estimate confirms that the bulk of the elapsed time difference for the map-shuffle phase between the single- and multiple-VM platforms is due to NUMA effects.

The use of large memory pages is important for optimal hardware performance, especially on virtual platforms in order to minimize TLB costs. All versions of vSphere 4 and 5 use host large pages by default. RHEL 6 and other Linux distributions (based on the 2.6.38 kernel or later) implement Transparent Huge Pages, which allocates and manages guest large pages automatically. In earlier versions of Linux, memory must be explicitly pinned for this purpose and the application configured to use it. To avoid the potential loss of host large pages on virtual platforms, a small amount of host memory should remain unallocated to virtual machines. This ensures there will be enough memory for the hypervisor and virtual machine memory overhead and that the host remains in the "high" memory state. Otherwise, if virtual machines are too large and free memory drops below "minFree" (host enters the "soft" memory state), memory will be reclaimed through ballooning and other memory management techniques. While these techniques are essential for keeping all virtual machines in a datacenter running during times of high memory usage, they all require breaking host large pages into small pages. For 72GB hosts, allocating 94% of host memory to virtual machines (as is done for the current tests) works well. The percentage allocated to virtual machines may be higher for larger memory hosts. Memory state, free memory, and minFree can all be monitored in the memory screen of `esxtop`. Guest and native OS memory usage, including large page allocations, is reported in `/proc/meminfo`. Details of vSphere memory management and performance are given in "Understanding Memory Resource Management in VMware vSphere 5.0" [11].

Storage Throughput

During the reduce-merge phase, CPU is not a bottleneck and there is essentially no network traffic. Elapsed time for this phase appears to be entirely dependent on storage performance. Shown in Table 4 are performance metrics from host-level tools for the three virtual platforms (`esxtop`) and for the native platform (`sar` and `mpstat`) during the steady-state portion of the reduce-merge phase. Storage bandwidth is split evenly between read and write operations. Multiplying the bandwidth for either read or write (half the values in the table) by the elapsed time for this phase and by the number of hosts gives a value very close to the dataset size. This means there are no 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.

The bandwidth difference between the native platform and a single VM per host is not entirely understood. It is partially due to the increased latency from virtualizing storage. Part of it may be due to shifting some of the work into the longer reduce-sort phase in the native test. The two-VM platform brings a small increase in bandwidth and a decrease in CPU utilization. The latter indicates some efficiency gain from memory locality, but not as much as in the map-shuffle phase. Also unlike the earlier phase, there is a large gain in performance by configuring four VMs per host.

Metric	Platform			
	Native	1 VM	2 VMs	4 VMs
Total storage bandwidth, MB/sec	1050	860	880	1080
CPU utilization, %	61	61	58	64

Table 4. Performance metrics on one host during the reduce-merge phase of TeraSort. Number of VMs is per host.

To help understand the increase in performance for the four-VM platform during the reduce-merge phase, simple models of storage throughput on one host were created. The underlying simplifying assumptions and results from running simulations based on these models are given in Appendix C: Storage Throughput Models. The "modified" model accounts for most of the throughput differences among the three virtualized platforms. It also shows that the source of these differences is the number of idle disks at any one time. That is, the fraction of all disks that are idle increases with the number of disks under the control of a single DataNode (VM). By splitting the disks among multiple DataNodes, this fraction goes down, allowing greater aggregate throughput.

1TB Dataset

Shown in Table 5 are the elapsed times for the commonly-published 1TB dataset size. The reduce tasks do not need as much memory for this case, so the maximum JVM heap size was reduced to 800MB. Results for eight and four disks per host are also shown. The former follows the common Hadoop recommendation to configure one disk per processor core and the latter shows the effect of under-configuring storage. Using fewer than twelve disks per host was not possible for the 8TB dataset due to insufficient raw disk space.

Tests with this dataset size have much lower elapsed times and are not as repeatable as the larger size, but a few patterns emerge. The one-disk-per-core recommendation above is appropriate for these workloads since more disks improve performance by only a nominal amount. There are several reasons for this. TeraGen is still close to being limited by the controller and not the disks. The map-shuffle phase of TeraSort is still close to being CPU-limited. The reduce-merge phase of TeraSort and TeraValidate do benefit from using twelve disks instead of eight; however, the modified model with four partitions shows only a 27% increase in expected throughput instead of the intuitively-expected increase of 50%. That is, when fewer disks are configured, a smaller percentage of them will be idle at any one time.

Platform	Disks	Elapsed Time, sec		
		TeraGen	TeraSort	TeraValidate
Native	12	74	250	48
1 VM	12	79	285	50
2 VMs	12	75	281	47
4 VMs	12	77	277	52
Native	8	77	267	60
1 VM	8	81	301	59
2 VMs	8	78	280	57
4 VMs	8	80	286	62
Native	4	137	370	94
1 VM	4	121	416	88
2 VMs	4	126	395	99
4VMs	4	140	384	91

Table 5. Elapsed times for 1TB dataset size with 32 hosts. Number of VMs and disks are per host.

Scaling with dataset size is super-linear. The 8TB dataset requires about 8.7 times as much work (due to the order NlogN operations needed for a sort) as the 1TB dataset, but the ratio of elapsed times is significantly less. Part of this is due to the relatively large impacts on the smaller dataset jobs of job start-up and the few tasks that are slow to finish at the end. However, the total amount of CPU consumed is also less than expected for the larger dataset, which indicates real increases in efficiency. This may be related to dynamic JVM optimizations such as the just-in-time compiler which have a greater relative cost for smaller jobs. Scaling is especially different than expected for the two- and four-VM tests, where the performance improvements compared to a single VM per host for the small dataset size are significantly smaller than were observed for the larger dataset. One possible explanation for this is that the greater imbalance of block storage across many VMs when the dataset size is smaller leads to inefficiencies. For the four-VM tests, the ratio of maximum to minimum space used per VM after TeraGen completes is about 1.15 for the larger dataset but increases to at least 1.5 for the smaller dataset.

Under-configuring storage has a big negative impact on job performance. If four disks per host are configured instead of eight, the elapsed time for all benchmarks on all platforms is increased by 34-78%. It is unlikely this can be justified by the hardware cost savings.

Best Practices

Following are a few suggestions for optimal virtualized Hadoop performance. 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 three or fewer). Such partitioning increases 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 on existing machines 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.
- For new deployments, size VMs with the above constraints, and then specify host hardware to accommodate the desired number of these VMs.
- Specify virtual machine memory size to avoid entering host "soft" memory state (shown in the memory screen of `esxtop`) and the likely breaking of host large pages into small pages. For 72GB hosts, leaving 6% of memory for the hypervisor and VM memory overhead is conservative. For smaller hosts, a slightly larger percentage of memory should not be allocated to VMs, For larger hosts, a smaller percentage is needed for memory overhead. See memory management resources for more details ([11] and references therein).
- Ensure large pages are used by the hypervisor, guest OS, and Java. This is the default starting with ESX 3.5, Linux kernel 2.6.38, and RHEL 6.0, as well as recent versions of other Linux 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.
- Avoid multi-queue networking in small 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 (TSO), and support for high memory DMA and multiple scatter-gather elements per Tx frame. These are all supported by vSphere; however, JF requires manual configuration of the virtual switches and adapters.
- If storage is configured manually (not through the vSphere UI), ensure it is block-aligned.
- 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 32-host cluster show that virtualizing Hadoop on vSphere 5.1 works well. Virtualization overhead is never very large, but can be essentially eliminated through careful virtual machine sizing and configuration. This was shown for a cluster of small two-socket hosts, but will become more important for larger hosts with more NUMA nodes and attached disks. Sizing virtual machines so each Hadoop node fits on a NUMA node will greatly increase memory locality. Even smaller virtual machines may be used in order to get the number of disks owned by a DataNode down to three or fewer. A simple model was used to show that this significantly reduces the number of idle disks and increases throughput.

Appendix A: Configuration

Hardware

- Cluster: 32X HP DL380 G7
- Host CPU and memory
  - 2X Intel Xeon X5687 processors, 3.6GHz, 12MB L3 cache
  - 72GB, 6X 8GB and 6X 4GB DIMMs, 1333MHz (24 hosts)
  - 96GB, 12X 8GB DIMMs, 1333MHz (8 hosts)
- Host BIOS settings
  - Intel Hyper-Threading Technology: enabled
  - C-states: disabled
  - Enhanced Intel SpeedStep Technology: disabled
  - Intel TurboMode: disabled
- Host local storage controller
  - HP Smart Array P410i, PCIe 2.0 x4 host interface
  - Hardware revision: Rev C
  - Firmware version: 5.12
  - 1024MB cache
- Host local storage disks
  - 16X Seagate ST9146852SS, SAS 6Gb/s, 146GB, 15,000 RPM, 16MB buffer (HP EH0146FAWJB)
    - Firmware: HPDF
  - 2 striped disks for VM storage and native OS install
  - 12 disks for Hadoop data
- Host network adapter: Broadcom Corporation NetXtreme II BCM57711 10Gb PCIe
- Network switch: Arista 7050S-64, 48 SFP+ ports (10GbE), 4 QSFP+ ports (40GbE), optical

Hypervisor

- vSphere 5.1 GA
- Network driver: bnx2x 1.74.17.v50.1
- Storage driver: hpsa v 5.0.0-21vmw
- Single vSwitch on the 10GbE NIC
  - Virtual Machine Port Group for VM traffic

Virtual Machines

- Virtual hardware: version 9
- VMware Tools: installed
- Disks: Physical Raw Device Mappings
- 1 VM per host
  - 69632MB, 68000MB reserved, 16 vCPUs, 12 data disks
  - No pinning, virtual NUMA, numa.memory.gransize = 1024
- 2 VMs per host
  - 34816MB, 34000MB reserved, 8 vCPUs, 6 data disks
  - 1 VM pinned to each NUMA node
- 4 VMs per host
  - 17408MB, 17000MB reserved, 4 vCPUs, 3 data disks
  - 2 VMs pinned to each NUMA node

Linux

- Distribution: RHEL 6.1 x86_64
- Kernel parameters in /etc/security/limits.conf
  - nofile=16384

  - nproc=4096
- Kernel parameter in /etc/sysctl.conf
  - fs.epoll.max_user_instances=4096
- Kernel parameter in /etc/selinux/config
  - SELINUX=disabled
- Java: Sun Java 1.6.0_26
- File system
  - Single Linux partition aligned on 64KB boundary per hard disk or LUN
  - Partition formatted with EXT4
    - 4KB block size
    - 4MB per inode
    - 2% reserved blocks
- Native OS drivers
  - Network: bnx2x 1.62.00-6, multi_mode=0
  - Storage: hpsa 2.0.2-3
- Virtual Machine guest OS drivers
  - Network: vmxnet3 1.0.14.0-k-NAPI
  - Storage: LSI Logic Parallel 3.04.18

Hadoop

- Distribution: Cloudera CDH4.1.1 with MRv1
- Client, NameNode, Secondary NameNode, JobTracker: each in one of the worker nodes
- Workers: 32 (Native, 1 VM), 64 (2 VMs), 128 (4 VMs)
- Cluster topology: each host is a unique rack
- Non-default parameters
  - Number of tasks: Table 1
  - dfs.datanode.max.transfer.threads=4096
  - dfs.replication=2
  - dfs.blocksize: 256MB
  - 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
  - io.file.buffer.size=131072
  - mapred.map.child.java.opts="-Xmx800m -Xms800m -Xmn256m"
  - mapred.reduce.child.java.opts="-Xmx1200m -Xmn256m"
  - mapred.child.ulimit=4000000
  - mapred.map.tasks.speculative.execution=false
  - mapred.reduce.tasks.speculative.execution=false
  - mapred.job.reuse.jvm.num.tasks=-1
  - mapred.reduce.parallel.copies=10
  - mapred.reduce.slowstart.completed.maps=0.001
  - mapred.job.tracker.handler.count=4
  - mapred.tasktracker.shuffle.fadvise=true
  - mapreduce.ifile.readahead.bytes=16777216
  - io.sort.factor=40
  - io.sort.mb=400
  - io.sort.record.percent=0.15

Appendix B: Remote Memory Performance Model

A simple model of the performance penalty incurred by accessing remote memory (that is, memory on another NUMA node) is constructed by first assuming a constant latency difference between local and remote memory. The actual extra latency depends on factors like memory access patterns and bandwidth. It is generally at least 30 nanoseconds and could be 50 nanoseconds or higher. However, memory latency is partially ameliorated by out-of-order processor execution and multiple threads. A rough estimate of the effective latency penalty for accessing a remote 64-byte cache line is 25 nanoseconds. For a memory bandwidth of 13 GB/sec, the number of cache lines accessed per second is 220 million. The total performance penalty for the single-VM or native platforms over the multiple-VM platforms is estimated by multiplying the number of extra remote cache lines read per second by the latency penalty and then dividing by 16 processors, which gives 0.12 seconds. That is, 12% of the time is spent on extra memory latency, which means an almost 14% increase in elapsed time. This is in-line with other estimates of the NUMA effect on application performance as well as accounting for all of the observed performance difference between the one- and two-VM platforms. The four-VM platform has very similar memory metrics as for two VMs and has nearly the same performance. This confirms that the performance effect on the map-shuffle phase between alternative virtual configurations is directly due to their effects on memory access patterns.

Appendix C: Storage Throughput Models

Models are developed here to simulate the aggregate storage throughput of a set of 16 threads performing I/O to a group of 12 disks on one host. It is based on the following assumptions:

1) Each thread (modeling a reduce task) performs sequential I/O to one randomly selected disk. This is reasonable because Hadoop does not currently consider disk utilization when it makes scheduling decisions.

2) Each thread does a fixed amount of I/O (say one block) to this disk, and then moves on to another randomly selected disk.

3) Storage throughput of a single thread on a single disk is 1: a single block is processed in one unit of time. The maximum (ideal) throughput of the host is therefore 12.

4) All the threads and disks are divided equally into one, two, or four partitions (modeling virtual machines). Each thread in a partition is restricted to accessing only the disks in its partition.

5) Storage throughput of a thread is one of these models:

    a) **Initial:** throughput=1. This gives the initial placement of threads before contention among threads takes effect. This is not a realistic model for throughput because the aggregate throughput is greater than the maximum. This model is equivalent to the Birthday Paradox which has an analytical solution [12].

    b) **Fixed:** throughput=1/T, where T is the number of threads on the same disk. Aggregate throughput of any disk that is not idle is therefore 1. Aggregate host throughput is equal to the number of actively used disks.

    c) **Modified:** throughput = 1/(T+.05*(T-1)*(T-1)). This is an ad hoc model of the degradation in aggregate throughput of a disk due to the randomization of multiple, sequential I/O streams to the same disk. Since the coefficient (.05) can be tuned to get different results, this model should be viewed as indicating the general effect of throughput degradation of multiple threads.

Metric		Active Disks			Throughput		
Partitions		1	2	4	1	2	4
Model	**Initial**	9.02	9.21	9.63	-	-	-
	Fixed	7.11	7.39	8.00	7.11	7.39	8.00
	Modified	6.78	7.14	7.86	6.48	6.86	7.61

Table 6. Average statistics for storage throughput simulations with 16 threads and 12 disks divided between partitions.

Simulations were run for a million units of time for each of these models and each number of partitions. Statistics were gathered on the average number of disks performing I/O ("active disks") at any one time and the average aggregate throughput of the host. Note that perfect disk utilization is achieved for all the models with 12 partitions: each partition has one disk and at least one thread so the disk can never be idle.

Results of the simulations are shown in Table 6. Just like in the Birthday Paradox where the probability of two people in a room sharing a birthday is surprisingly high, the probability of multiple threads writing to one disk while other disks are idle is higher than one might expect. The initial placement of threads shows this effect, but the other two models show it more strongly: 16 threads can be expected to be performing I/O on only seven or eight of the twelve disks. Looking down each column, it is seen that the models that assume lower throughput with more contention have more idle disks. Alternatively, disks that have lower performance for whatever reason effectively "attract" more threads to them; these threads then have long residence times while threads on disks to themselves finish quickly and move on. Looking across the rows, it is evident how to break this cycle: more partitions reduces the number of idle disks and increases the aggregate host throughput. Comparing each row to the next, this effect increases with the sensitivity of thread throughput to disk contention. Finally, the throughput (which is really the more important metric) increases more than the number of active disks as this sensitivity increases. The fixed model predicts a 12.5% increase in throughput going from one to four partitions while the modified model gives a 17.4% increase. The latter model accounts for most of the 25.6% increase in observed storage throughput in Table 4. The modified model predicts a much smaller improvement, 5.9%, for two partitions compared to one. This is larger than, but roughly agrees with, the TeraSort measurements. There is some variation in the measurements: in the earlier 24-host tests, there was a similar small improvement going from one to two VMs, but a 20% increase from one to four VMs. The modified model aligns even better with the reduce-merge elapsed times in Table 2 than with the observed storage throughputs on one host; however, this might be fortuitous.

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] McDougall, Richard. "Towards an Elastic Elephant: Enabling Hadoop for the Cloud." VMware, Inc., 2012. http://cto.vmware.com/towards-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] "Hadoop Virtualization Extensions on VMware vSphere 5." VMware, Inc., 2012. http://serengeti.cloudfoundry.com/pdf/Hadoop%20Virtualization%20Extensions%20on%20VMware%20vSphere%205.pdf.

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

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

[10] "Intel 64 and IA-32 Architectures Software Developer's Manual", Volume3, Chapter 18, http://download.intel.com/products/processor/manual/325462.pdf.

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

[12] "Birthday Problem." Wikipedia, 2013. http://en.wikipedia.org/wiki/Birthday_problem.

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 Chethan Kumar, Reza Taheri, Josh Simons, and Tariq Magdon-Ismail for their careful reviews, as well as many other colleagues at VMware for their contributions to this paper.

vmware®

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