A Benchmarking Case Study of Virtualized Hadoop Performance on VMware vSphere® 5

Performance Study

TECHNICAL WHITE PAPER

Table of Contents

Executive Summary

The performance of three Hadoop applications is reported for several virtual configurations on VMware vSphere 5 and compared to native configurations. A well-balanced seven-node AMAX ClusterMax system was used to show that the average performance difference between native and the simplest virtualized configurations is only 4%. Further, the flexibility enabled by virtualization to create multiple Hadoop nodes per host can be used to achieve performance significantly better than native.

Introduction

In recent years the amount of data stored worldwide has exploded, increasing by a factor of nine in the last five years[1]. Individual companies often have petabytes or more of data and buried in this is business information that is critical to continued growth and success. However, the quantity of data is often far too large to store and analyze in traditional relational database systems, or the data are in unstructured forms unsuitable for structured schemas, or the hardware needed for conventional analysis is just too costly. And even when an RDBMS is suitable for the actual analysis, the sheer volume of raw data can create issues for data preparation tasks like data integration and ETL. As the size and value of the stored data increases, the importance of reliable backups also increases and tolerance of hardware failures decreases. The potential value of insights that can be gained from a particular set of information may be very large, but these insights are effectively inaccessible if the IT costs to reach them are yet greater.

Hadoop evolved as a distributed software platform for managing and transforming large quantities of data, and has grown to be one of the most popular tools to meet many of the above needs in a cost-effective manner. By abstracting away many of the high availability (HA) and distributed programming issues, Hadoop allows developers to focus on higher-level algorithms. Hadoop is designed to run on a large cluster of commodity servers and to scale to hundreds or thousands of nodes. Each disk, server, network link, and even rack within the cluster is assumed to be unreliable. This assumption allows the use of the least expensive cluster components consistent with delivering sufficient performance, including the use of unprotected local storage (JBODs).

Hadoop's design and ability to handle large amounts of data efficiently make it a natural fit as an integration, data transformation, and analytics platform. Hadoop use cases include:

- **Customizing content for users:** Creating a better user experience through targeted and relevant ads, personalized home pages, and good recommendations.
- **Supply chain management:** Examining all available historical data enables better decisions for stocking and managing goods. Among the many sectors in this category are retail, agriculture, hospitals, and energy.
- **Fraud analysis:** Analyzing transaction histories in the financial sector (for example, for credit cards and ATMs) to detect fraud.
- **Bioinformatics:** Applying genome analytics and DNA sequencing algorithms to large datasets.
- **Beyond analytics:** Transforming data from one form to another, including adding structure to unstructured data in log files before combining them with other structured data.
- **Miscellaneous uses:** Aggregating large sets of images from various sources and combining them into one (for example, satellite images), moving large amounts of data from one location to another.

Hadoop comprises two basic components: a distributed file system (inspired by Google File System[2]) and the computational framework (Google MapReduce[3]). In the first component, data is stored in Hadoop Distributed File System (HDFS). The file system namespace in the Apache open-source version of HDFS (and in the Cloudera distribution used here) is managed by a single *NameNode*, while a set of *DataNodes* do the work of storing and retrieving data. HDFS also manages the replication of data blocks. The exception to HA in Hadoop is the NameNode. While Hadoop provides mechanisms to protect the data, the NameNode is a single point of failure, so in production clusters it is advisable to isolate it and take other measures to enhance its availability (this was not done for this paper since the focus was on performance). There is also a *Secondary NameNode* which keeps a copy of the NameNode data to be used to restart the NameNode in case of failure, although this copy may not be current so some data loss is still likely to occur.

The other basic component of Hadoop is MapReduce, which provides a computational framework for data processing. MapReduce programs are inherently parallel and thus very suited to a distributed environment. A single *JobTracker* schedules all the jobs on the cluster, as well as individual tasks. Here, each benchmark test is a job and runs by itself on the cluster. A job is split into a set of tasks that execute on the *worker* nodes. A *TaskTracker* running on each worker node is responsible for starting tasks and reporting progress to the JobTracker. As the name implies, there are two phases in MapReduce processing: map and reduce. Both use key-value pairs defined by the user as input and output. This allows the output of one job to serve directly as input for another. Hadoop is often used in that way, including several of the tests here. Both phases execute through a set of tasks. Note that the number of map and reduce tasks are independent, and that neither set of tasks is required to use all the nodes in the cluster. In Figure 1, a simplified view of a typical dataflow with just one task per node is shown.

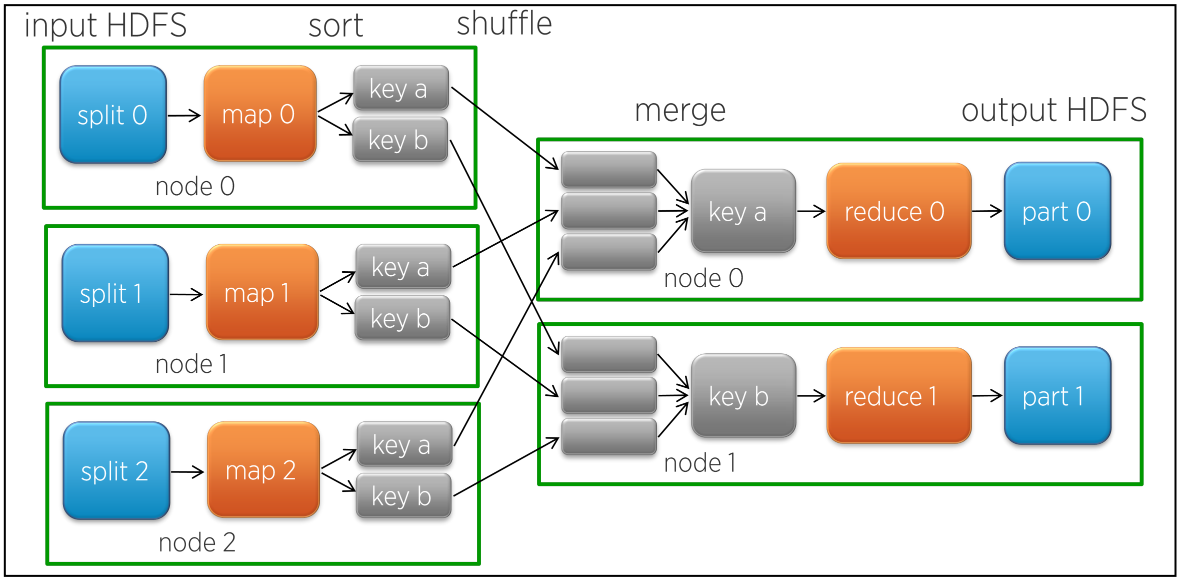

Figure 1. Simplified MapReduce Dataflow with One Task Per Node

Hadoop splits the input data into suitable-sized chunks for processing by the map tasks. These *splits* need to be large enough to minimize the overhead of managing files and tasks, but small enough so there is enough parallel work to do. The map task output is sorted by keys and written not to HDFS, but to local disk (shown in gray). It is then sent to the appropriate reduce task. This is known as the "shuffle" phase and is typically an all-to-all operation that often stresses the bandwidth of the network interconnect. The reduce tasks do the final processing and the output is written to HDFS (shown in blue). If there is more than one reducer, the output is partitioned. A good reference that gives more details on MapReduce, HDFS, and other components is *Hadoop: The Definitive Guide*[4].

Since Hadoop jobs can be very large, with some requiring hours on large clusters to execute, the resource cost can be significant. The cost of a job is inversely related to the throughput Hadoop is able to deliver, and therefore performance is of utmost importance. CPU, memory, network throughput, and storage capacity and throughput all need to be sized appropriately to achieve a balanced system, and the software layer needs to be able to fully utilize all these resources. In Hadoop, this means finding enough parallel work to keep all the computational nodes busy, and scheduling tasks to run such that the data they need are nearly always local (often referred to as moving the computation to the data, rather than moving data to the computational resources). A small overhead or scheduling inefficiency in the virtualization, operating system, or application layers can add up to a large cumulative cost and therefore needs to be understood and quantified.

Why Virtualize?

Hadoop is a modern application with features such as consolidation of jobs and HA that overlap with capabilities enabled by virtualization. This leads some to believe there is no motivation for virtualizing Hadoop; however, there are a variety of reasons for doing so. Some of these are:

- **Scheduling:** Taking advantage of unused capacity in existing virtual infrastructures during periods of low usage (for example, overnight) to run batch jobs.
- **Resource utilization:** Co-locating Hadoop VMs and other kinds of VMs on the same hosts. This often allows better overall utilization by consolidating applications that use different kinds of resources.
- **Storage models:** Although Hadoop was developed with local storage in mind, it can just as easily use shared storage for all data or a hybrid model in which temporary data is kept on local disk and HDFS is hosted on a SAN. With either of these configurations, the unused shared storage capacity and bandwidth within the virtual infrastructure can be given to Hadoop jobs.
- **Datacenter efficiency:** Virtualizing Hadoop can increase datacenter efficiency by increasing the types of workloads that can be run on a virtualized infrastructure.
- **Deployment:** Virtualization tools ranging from simple cloning to sophisticated products like VMware vCloud Director can speed up the deployment of Hadoop nodes.
- **Performance:** Virtualization enables the flexible configuration of hardware resources.

The last item above is the main focus of this paper. Many applications can efficiently use many nodes in a cluster (they scale out well), but lack good SMP scaling (poor scale-up). Many Web servers, mail servers, and Java application servers fall into this category. These can usually be "fixed" in a virtual environment by running several smaller VMs per host.

Whatever the reason for virtualizing Hadoop, it is important to understand the performance implications of such a decision to ensure the cost will be reasonable and that there will be sufficient resources. While some of the relevant issues were investigated for this report, there are many more that were left for future work. Among these are the trade-offs of different kinds of storage, network tuning, larger scale-out, and more applications.

Test Setup

The hardware configuration of the test cluster is shown in Figure 2. Seven host servers of an AMAX ClusterMax system were connected to a single Mellanox 10GbE switch. Each host was equipped with two Intel X5650 2.66GHz 6-core processors, 12 internal 500GB 7,200 RPM SATA disks, and a 10GbE Mellanox adapter. The disks were connected to a PCIe 1.1 X4 storage controller. This controller has a theoretical throughput of 1 GB/s, but in practice can deliver 500-600 MB/s. Power states were disabled in the BIOS to improve consistency of results. Intel TurboMode was enabled but it never engaged (probably because of the high utilization of all the cores). Intel Hyper-Threading (HT) Technology was either enabled or disabled in the BIOS depending on the test performed. Complete hardware details are given in the Appendix.

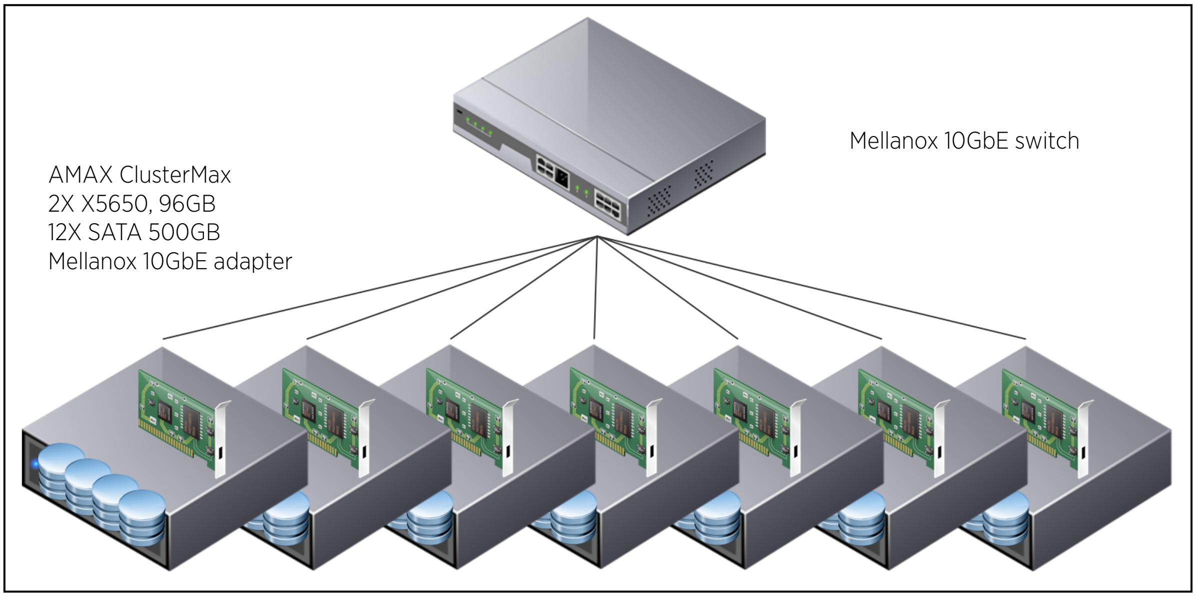

Figure 2. Cluster Hardware Configuration

Two of the 12 internal disks in each host were mirrored, divided into LUNs, and used to install the native operating system and ESXi hypervisor. During the tests, the I/O to these disks consisted of just logs and job statistics. The other ten disks were configured as JBODs and used exclusively for Hadoop data. A single aligned partition was created on each disk and formatted with EXT4. For the virtualized tests, these disks were passed through to the VMs using physical raw device mappings; this ensured both native and virtual tests used the same underlying file systems.

RHEL 6.1 x86_64 was used as the operating system (OS) in all tests. The native and virtual OS configurations were nearly the same, but there were some differences. The native tests used all the CPUs and memory of the machine. In the virtual tests, all the CPUs were split between the VMs (that is, CPUs were exactly-committed) while a few gigabytes of memory were held back for the hypervisor and VM memory overhead (slightly under-committed). The Mellanox MLNX_EN (version 1.5.6) driver was installed in the native OS, with enable_sys_tune turned on (this significantly lowers network latency). In ESXi, a development version of this driver was used. The latter is expected to be released soon. The virtual adapter was vmxnet3. Default networking parameters were used both in ESXi and in the guest OS. Sun Java 6 is recommended for Hadoop; version 1.6.0_25 was used here. Two kernel tunables were increased. These and other OS details are given in the Appendix.

The Cloudera CDH3u0 version of Apache Hadoop was installed in all machines. Replication was set to 2, as is common for small clusters (large clusters should use 3 or more). The HDFS block size was increased to 128MB from the 64MB default. Most of machine memory was used for the Java heap. In the native case, the maximum was set to 2048MB. Slightly less, 1900MB, was used for the virtual tests since the total VM memory allocated was smaller. A few other Hadoop parameters were changed from their defaults and are listed in the Appendix. Probably the most important Hadoop parameters are the numbers of map and reduce tasks. The number of each was different for each benchmark but was the same when comparing native configurations and the various virtual tests. This leaves open the possibility that the particular number of tasks used was more optimal on one platform (native or virtual) than another; this was not investigated here. However, it was observed that the performance of the heavy I/O tests was not very sensitive to the number of tasks.

All of the hosts/VMs were used as Hadoop worker nodes. Each worker node ran a DataNode, a TaskTracker, and a set of map and reduce tasks. The NameNode and JobTracker were run on the first worker node, and the Secondary NameNode was run on the second worker node. It is often advised to run these three management processes on machines separate from the worker nodes for reliability reasons. In large clusters it is necessary to do so for performance reasons. However, in small clusters these processes take very little CPU time, so dedicating a machine to them would reduce overall resource utilization and lower performance.

In the 2-VM/host tests, all the hardware resources of a host were evenly divided between the VMs. In addition, each VM was pinned to a NUMA node (which for these machines is a socket plus the associated memory) for better reproducibility of the results. The native, 1-VM, and 2-VM tests are described as homogeneous because the machines/VMs are all the same size. For the 4-VM/host tests a heterogeneous configuration was chosen where two small and two large VMs were used on each host. The small VMs were assigned two data disks, five vCPUs, and 18400MB each, while the large VMs were assigned three data disks, seven vCPUs, and 27600MB. One of each was pinned to each NUMA node. The number of tasks per Hadoop node was then configured to be either proportional to the number of vCPUs (for computational tests), or to the number of disks (for I/O tests). Clearly, the use of heterogeneous configurations raises many issues, of which only a few will be discussed here.

As mentioned previously, all of the present tests were performed with replication=2. This means each original block is copied to one other worker node. However, when multiple VMs per host are used, there is the undesirable possibility from a single-point of failure perspective that the replica node chosen is another VM 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. Equating the set of VMs running on a particular host to a unique rack ensures that at least the first replica of every block is sent to a VM on another host. From a benchmarking point of view, defining an appropriate topology also ensures that the multiple-VM cases do not make replicas local to their hosts and thereby (inappropriately) gain a performance advantage over the single-VM and native configurations.

Hadoop Benchmarks

Several benchmarks were created from three of the example applications included with the Cloudera distribution: Pi, TestDFSIO, and TeraSort. Table 1 summarizes the maximum number of simultaneous map and reduce tasks across the cluster for the various benchmarks. These were the same for native and all the virtual configurations.

BENCHMARK	MAP		REDUCE	
	HT on	HT off	HT on	HT off
Pi	168	84	1	1
TestDFSIO-write	140	140	1	1
TestDFSIO-read	140	140	1	1
TeraGen	280	280	0	0
TeraSort	280	280	70	70
TeraValidate	70	70	1	1

Table 1. Maximum Total Number of Simultaneous Tasks per Job

Pi

Pi is a purely computational application that employs a Monte Carlo method to estimate the value of pi. It is very nearly “embarrassingly parallel”: the map tasks are all independent and the single reduce task gathers very little data from the map tasks. There is little network traffic or storage I/O. All the results reported here calculate 1.68 trillion samples. These are spread across 168 total map tasks (HT enabled) or 84 map tasks (HT disabled). For both homogeneous and heterogeneous cases, the number of map tasks per Hadoop node is equal to the number of CPUs or vCPUs. All the map tasks start at the same time and the job finishes when the last map task completes (plus a trivial amount of time for the reduce task). It is important that all map tasks run at the same speed and finish at about the same time for good performance. One slow map task can have a significant effect on elapsed time for the job.

TestDFSIO

TestDFSIO is a storage throughput test that is split into two parts here: TestDFSIO-write writes 1000020MB (about 1TB) of data to HDFS, and TestDFSIO-read reads it back in. Because of the replication factor, the write test does twice as much I/O as the read test and generates substantial networking traffic. A total of 140 map tasks were found to be close to optimal for the HT disabled case, while 70 map tasks were not enough and 210 gave close to the same performance as 140. The same number of tasks was used for the HT-enabled tests since additional CPU resources were not expected to help this I/O dominated benchmark. The number of map tasks per worker node in the heterogeneous case was made proportional to the number of available disks (that is, four for the small VMs, six for the large).

TeraSort

TeraSort sorts 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. It is split into three parts: generation, sorting, and validation. TeraGen creates the data and is similar to TestDFSIO-write 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 and writes sorted data to HDFS in a number of partitioned files. The application itself overrides the specified replication factor so only one copy is written. The philosophy is that if a data disk is lost, the user can always rerun the application, but input data needs replication since it may not be as easily recovered. TeraValidate reads all the sorted data to verify that it is in order. The map tasks do this for each file independently, and then the single reduce task checks that the last record of each file comes before the first record of the next file. Results are reported for two sizes that often appear in published Hadoop tests: 10 billion records ("1TB") and 35 billion records ("3.5TB"). A total of 280 map tasks was used for TeraGen, 280 simultaneous (several thousand in all) map tasks and 70 reduce tasks for TeraSort, and 70 map tasks for TeraValidate.

Benchmark Results

Table 2 and Table 3 present elapsed time results with HT disabled and enabled respectively.

BENCHMARK	NATIVE	VIRTUAL	
		1 VM	2 VMs
Pi	792	740	762
TestDFSIO-write	640	706	614
TestDFSIO-read	499	532	453
TeraGen 1TB	664	700	580
TeraSort 1TB	2,995	3,127	3,110
TeraValidate 1TB	569	481	495
TeraGen 3.5TB	2,328	2,504	2,030
TeraSort 3.5TB	13,460	14,863	12,494
TeraValidate 3.5TB	2,783	2,745	2,552

Table 2. Elapsed Time in Seconds with HT Disabled (lower is better; number of VMs shown is per host)

BENCHMARK	NATIVE	VIRTUAL		
		1 VM	2 VMs	4 VMs
Pi	695	628	626	650
TestDFSIO-write	648	715	573	562
TestDFSIO-read	471	443	433	427
TeraGen 1TB	659	718	572	598
TeraSort 1TB	2,629	3,001	2,792	2,450
TeraValidate 1TB	470	601	544	480
TeraGen 3.5TB	2,289	2,517	1,968	2,302
TeraSort 3.5TB	13,712	14,353	12,673	13,054
TeraValidate 3.5TB	2,460	2,563	2,324	3,629

Table 3. Elapsed Time in Seconds with HT Enabled (lower is better; number of VMs shown is per host)

Figure 3 and Figure 4 show the elapsed time results for the virtual cases normalized to the corresponding native cases with HT disabled and enabled respectively.

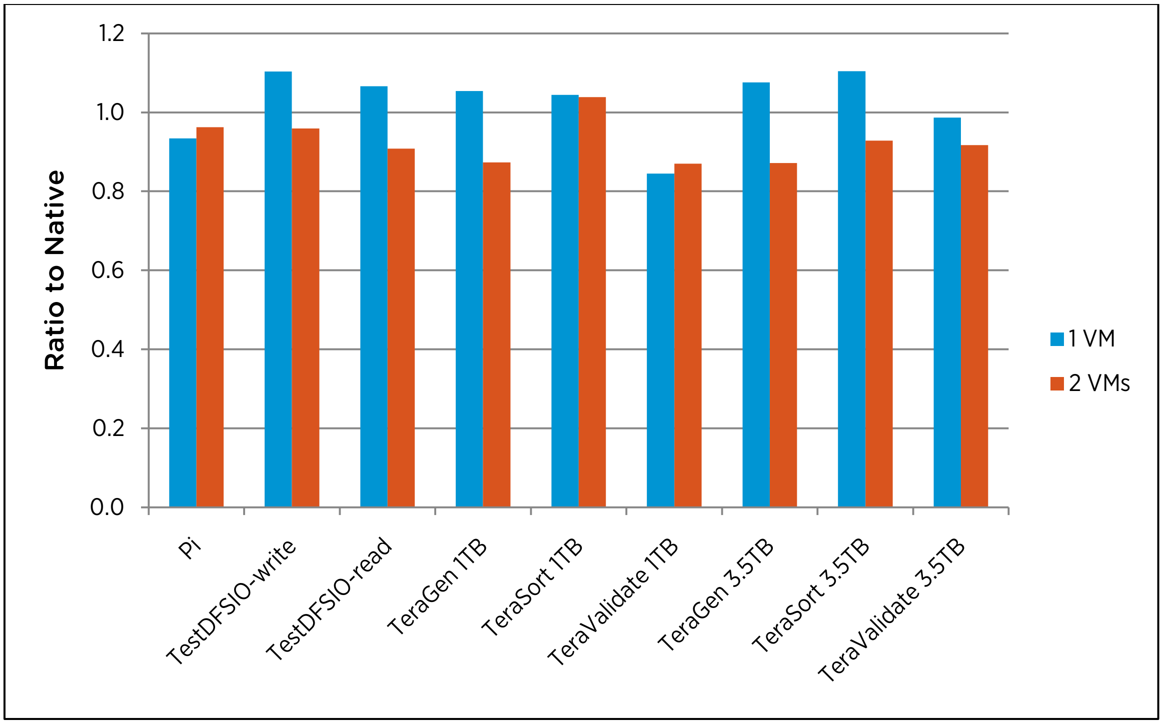

Figure 3. Elapsed Time with HT Disabled Normalized to the Native Case (lower is better)

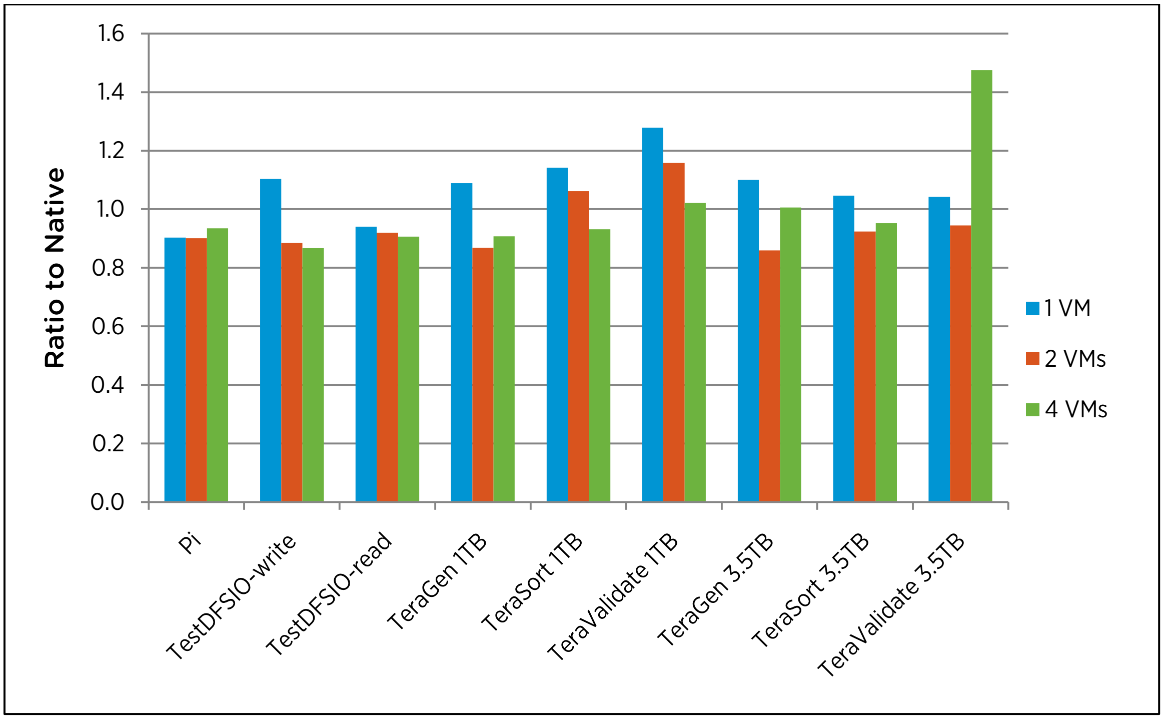

Figure 4. Elapsed Time with HT Enabled Normalized to the Native Case (lower is better)

Analysis

Pi

Pi is 4-10% faster for all of the virtual cases than for the corresponding native cases. This is an unexpected result since normally pure CPU applications show a 1-5% reduction in performance when virtualized. There is a small contribution from map tasks not executing at the same rate (a function of the Linux and ESXi schedulers) which is reflected in how long the reduce task runs (the reduce task starts running when the first map task finishes). Differences in the schedulers may account for 1-2% of the differences in performance. Another Hadoop statistic shows the sum of the elapsed times of all the map tasks. This is up to 9% less for the virtual cases. Together with 100% CPU utilization in all cases, this means that each map task is running faster in a VM. Investigations are on-going to better understand this behavior. Enabling HT reduces the elapsed time by 12% for the native case and 15-18% for the virtual cases.

TestDFSIO

TestDFSIO is a stress test for high storage throughput applications. The results show that in three out of the four 1-VM cases (write/read, HT disabled/enabled) virtual is 7-10% slower than the corresponding native cases. In the fourth 1-VM case and all the multi-VM cases, virtual is 4-13% faster. In this kind of test (sequential throughput, lightly-loaded processors) it is expected that throughput would be strictly limited by hardware (in this case the storage controller), and that the variation among software platforms would be much less. A partial explanation for the observed behavior can be seen from Figure 5, which shows the total write throughput on one of the hosts with different numbers of VMs and HT enabled. The average throughput is lower for the 1-VM case not because the peak throughput is lower, but because the throughput is much noisier in time. Data shown was collected in esxtop at 20 second intervals. With finer intervals, throughput actually drops to zero for a few seconds at a time. These drops are synchronized across all the nodes in the cluster, indicating they are associated with the application and not the underlying platform. With two VMs per host, the throughput is much more consistent and even more so with four VMs. The native case exhibits noise comparable to the single-VM case. It is not understood why more VMs improves the throughput behavior. One possibility is that a single Hadoop node has difficulty scheduling I/O fairly to ten disks and that it can do a better job when fewer disks are presented to it. Although the throughput is substantially higher for four VMs than two VMs for much of the duration of the test, the former finishes only 2% sooner. This is because the heterogeneity of the 4-VM case makes it difficult for all the tasks to finish at about the same time as in the 2-VM case. By adjusting the number of tasks per node, the test writes a number of original files to each node in proportion to the number of disks it has. However, the replicas are spread evenly across all the nodes, destroying this proportionality. Heterogeneous Hadoop nodes have the potential to be useful, but the trade-offs will be difficult to understand. HT does not have a consistent effect on performance, which is to be expected for a workload with low CPU utilization.

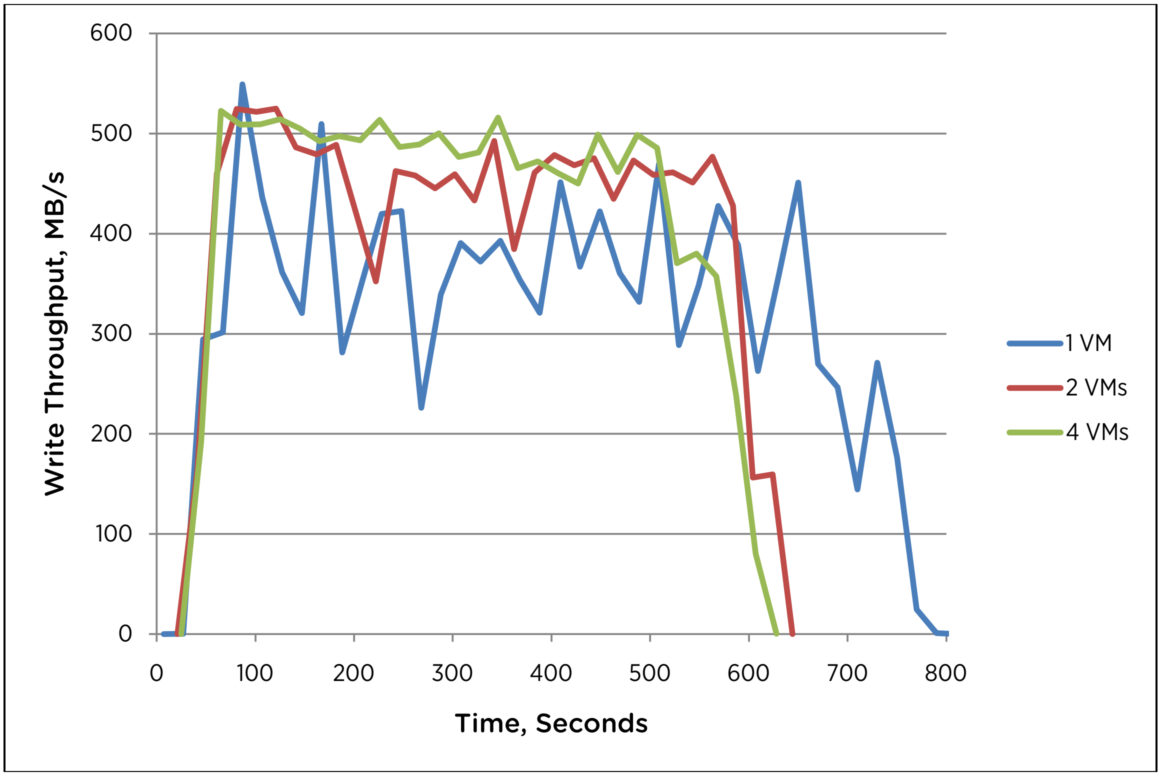

Figure 5. TestDFSIO Write Throughput on One of the Hosts (HT enabled)

TeraSort

The three TeraSort benchmarks also show excellent performance overall when virtualized.

There is a significant amount of noise in the execution time of all three tests on both native and virtual platforms, especially TeraValidate. This is partly due to the greater complexity of the applications, but mostly due to the "long pole" effect where a single long-running task can have an out-sized effect on the total elapsed time. Using a 3.5TB dataset reduces these effects, making the tests more repeatable.

TeraGen writes almost an identical amount of data to disk as TestDFSIO-write and has similar elapsed times for all cases. The CPU utilization is considerably higher (about 50% compared to 30%) but not high enough to affect the elapsed time. As with TestDFSIO-write, the write behavior of TeraGen is sensitive to the number of VMs. The 4-VM cases have the highest sustained throughput for most of the duration of the tests. They take longer to finish than the 2-VM cases because of the heterogeneity of the configuration (see the paragraph at the end of this section).

TeraSort is the most resource-intensive of the tests presented here: it combines high CPU utilization, high storage throughput and moderate networking bandwidth (1-2 Gb/sec per host). Using a single VM per host, the virtual cases are 4-14% slower than the corresponding native cases. The average performance reduction of 8% is reasonable for I/O-heavy applications. There does not seem to be a pattern with respect to HT or size of the dataset. With two VMs per host, the performance improves significantly compared to the native cases: it becomes 4-6% slower for the 1TB dataset and 7-8% faster for the 3.5TB dataset. HT improves performance by 4-12% for 1TB but doesn't have a significant effect for 3.5TB. Partitioning each host into four VMs greatly improves the 1TB case and slightly degrades the 3.5TB case compared to two VMs.

These observations and the lack of strong patterns can be explained by considering CPU utilization, shown in Figure 6 for the 3.5TB TeraSort test with HT enabled. The map tasks and shuffle phase run up to the "cliff" at about 8,000 seconds, and then the reduce tasks run at relatively low utilization. It's the long-pole effect of the relatively few and long-running reduce tasks that causes much of the elapsed time variation. For instance, the 1-VM case finished the map/shuffle phase in the same amount of time as the native case, and most of the reduce tasks earlier, but it ends up finishing the whole job later. Two VMs clearly improves execution speed at approximately the same CPU cost. Four VMs also improves the map/shuffle performance, but with a large decrease in CPU utilization. This is offset by reduced performance in the reduce phase. The reason for the decrease in utilization requires further investigation.

The large differences in performance make it worthwhile to investigate different virtual configurations to find the optimal one. The choice of performance metric can have as much of an impact on finding the optimal configuration as the performance based on any one metric. For example, in a job-consolidation environment (running multiple jobs under Hadoop, or other kinds of applications in other VMs simultaneously), the total integrated CPU utilization is often a better performance metric than job elapsed time, since it is the former that determines how many jobs can run in a given time period. With such a metric, the performance advantage of four VMs over the native configuration would be much greater than the 5% indicated by the job elapsed time. The 1TB case leads to similar conclusions, although the metrics are very different. The map/shuffle phase runs at 96-99% CPU utilization for all the virtual cases and about 75% for the native case. The gains in efficiency with multiple VMs are reflected more in reductions in elapsed time; the 4-VM case finishes the map/shuffle phase (where most of the CPU utilization occurs) 24% faster than the native case (compared to 8% faster in the 3.5TB case).

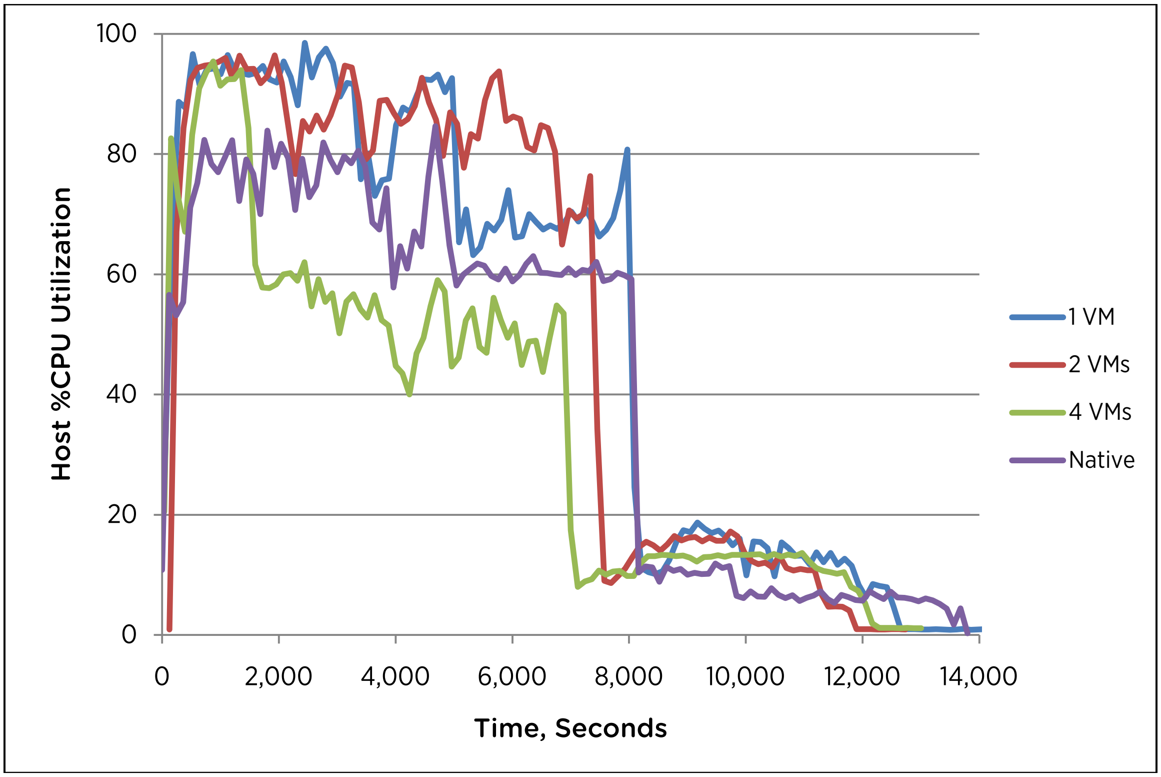

Figure 6. 3.5TB TeraSort %CPU Utilization with HT Enabled on One of the Hosts (lower is better)

TeraValidate is mostly a sequential read test; however, the relatively high run-to-run variability makes it difficult to compare to TestDFSIO-read. The difference between HT enabled and disabled for the 1 TB case does not appear to be significant. The long execution time of the 3.5TB case with four VMs is significant and is discussed below. Otherwise, the native and virtual cases are quite comparable.

One of the features of Hadoop is speculative execution. If one task is running slowly and another node has task slots available, Hadoop may speculatively start the same task on the second node. It will then keep the results from the task that finishes first and kill the laggard. This strategy is very effective if the reason for the slowness is some kind of system problem on a node. The extreme case is node failure where Hadoop will simply restart the tasks on another node. However, if all the nodes are healthy, speculative execution may slow down progress of the job by over-taxing resources. This is what happened for all three of the 3.5TB TeraSort tests with four VMs. The heterogeneity of this configuration leads to differences in task execution, which in turn leads to speculative execution. The effect was largest for TeraValidate: starting a second identical task is usually harmful because there is only one copy of the data that needs to be read. By disabling speculative execution, TeraGen and TeraValidate improved by 11% and 31% respectively to become just 5% and 7% slower than the corresponding 2-VM cases. TeraSort improved only very slightly; this is probably due to the large number of relatively small tasks helping to keep the task slots full.

Conclusion

These results show that virtualizing Hadoop can be compelling for performance reasons alone. With a single VM per host, the average increase in elapsed time across all the benchmarks over the native configuration is 4%. This is a small price to pay for all the other advantages offered by virtualization. Running two or four smaller VMs on each two-socket machine results in average performance better than the corresponding native configuration; some cases were up to 14% faster. Using an integrated CPU utilization metric, which is more relevant in many environments, may yield even greater savings.

The results presented here represent a small part of ongoing work surrounding virtualized Hadoop. Future papers will deal with alternative storage strategies, ease-of-use, larger clusters, higher-level applications, and many other topics.

Appendix: Configuration

Hardware

- Cluster: AMAX ClusterMax with seven 2-socket hosts
- Host CPU and memory:
  - 2X Intel Xeon X5650 processors, 2.66 GHz, 12MB cache
  - 96 GB, 1333 MHz, 12 DIMMs
- Host storage controller:
  - Intel RAID Controller SROMBSASMR, PCIe 1.1 x4 host interface
  - 128 MB cache
  - Write Back enabled
- Host hard disks:
  - 12X Western Digital WD5003ABYX, SATA, 500GB, 7200 RPM
  - 2 mirrored disks divided into LUNs for native OS and ESXi root drives and VM storage
  - 10 disks configured as JBODs for Hadoop data
- Host file system:
  - Single Linux partition aligned on 64KB boundary per hard disk
  - Partition formatted with EXT4:
    - 4KB block size
    - 4MB per inode
    - 2% reserved blocks
- Host network adapter: Mellanox ConnectX VPI (MT26418) - PCIe 2.0 5GT/s, 10GbE
- Host BIOS settings:
  - Intel Hyper-Threading Technology: enabled or disabled as required
  - C-states: disabled
  - Enhanced Intel SpeedStep Technology: disabled
- Network switch: Mellanox Vantage 6048, 48 ports, 10GbE

Linux

- Distribution: RHEL 6.1 x86_64
- Kernel parameters:
  - nofile=16384
  - nproc=4096
- Java: Sun Java 1.6.0_25

Native OS

- CPU, memory, and disks: same as Hardware
- Network driver: Mellanox MLNX_EN (version 1.5.6), enable_sys_tune=1

Hypervisor

- vSphere 5.0 RTM, changeset 1401879
- Development version of Mellanox network driver

Virtual Machines

- VMware Tools: installed
- Virtual network adapter: vmxnet3
- Virtual SCSI controller: LSI Logic Parallel
- Disks: Physical Raw Device Mappings
- 1 VM per host:
  - 92000MB, 24 vCPUs (HT enabled) or 12 vCPUs (HT disabled), 10 disks.
- 2 VMs per host:
  - 46000MB, 12 vCPUs (HT enabled) or 6 vCPUs (HT disabled), 5 disks
  - One VM pinned to each socket.
- 4 VMs per host:
  - HT enabled only
  - Small: 18400MB, 5 vCPUs, 2 disks
  - Large: 27600MB, 7 vCPUs, 3 disks
  - One small and one large VM pinned to each socket

Hadoop

- Distribution: Cloudera CDH3u0 (Hadoop 0.20.2)
- NameNode, JobTracker: node0
- Secondary NameNode: node1
- Workers: All machines/VMs
- Non-default parameters:
  - Number of tasks: Table 1
  - dfs.datanode.max.xcievers=4096
  - dfs.replication=2
  - dfs.block.size=134217728
  - io.file.buffer.size=131072
  - mapred.child.java.opts=”-Xmx2048m –Xmn512m” (native)
  - mapred.child.java.opts=”-Xmx1900m –Xmn512m” (virtual)
- Cluster topology:
  - Native, 1 VM per host: all nodes in the default rack
  - 2, 4 VMs per host: each host is a unique rack

References

1. IDC study. EMC. http://www.emc.com/collateral/analyst-reports/idc-extracting-value-from-chaos-ar.pdf.

2. Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. *The Google File System*. SOSP'03, Oct 19-23. ACM, 2003. http://labs.google.com/papers/gfs-sosp2003.pdf.

3. Dean, Jeffrey and Sanjay Ghemawat. OSDI 2004. http://labs.google.com/papers/mapreduce-osdi04.pdf.

4. White, Tom. *Hadoop: The Definitive Guide*. O'Reilly, 2011.

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 work presented here was made possible through a collaboration with AMAX, who provided a ClusterMax system for this performance testing and who hosted and supported the cluster in their corporate datacenter. The author would like to thank Julia Shih, Justin Quon, Kris Tsao, Reeann Zhang, and Winston Tang for their support and hardware expertise. The author would like to thank Mellanox for supplying the network hardware used for these tests, and in particular Yaron Haviv, Motti Beck, Dhiraj Sehgal, Matthew Finlay, and Todd Wilde for insights and help with networking. Many of the author's colleagues at VMware contributed valuable suggestions as well.