The Performance Cost of Virtual Machines on Big Data Problems in Compute Clusters

Neal Barcelo
Denison University
Granville, OH 4023
barcel_n@denison.edu

Nick Legg
Denison University
Granville, OH 4023
legg_n@denison.edu

Advisor: Dr. Thomas Bressoud
Denison University
Granville, OH 4023
bressoud@denison.edu

ABSTRACT

To facilitate better management of large data-intensive compute clusters, many cluster owners and providers of cloud computing environments are looking at virtualization technology as a potential solution. However, virtual machines exhibit performance degradation when compared with physical machines since a virtual machine is unable to execute privileged instructions without first going through the virtualization software.

The purpose of our research is to construct a cloud computing environment and assess the performance cost of large-scale cluster computing on large data sets using virtual machines as opposed to physical machines.

We developed and tested three different workloads: a synthetic disk-dominant application, a CPU-intensive real-world scientific modeling application, and a real-world ground density modelling application that exhibits a hybrid of CPU and disk operations. These workloads were then evaluated on Denison's own compute cluster running both physical and virtual machines to assess the performance penalties that may be incurred by using a cluster of virtual machines.

1. INTRODUCTION

The motivation behind compute clusters is driven largely by an ever-expanding set of problems which have two outstanding requirements: these problems require larger and larger storage capacities as the amount of data being processed grows. This massive amount of data must somehow be partitioned and delivered to the set of compute nodes in the cluser. Further, these problems need plenty of processing power in order to be solved in a reasonable amount of time. These emerging "Big Data" computing problems are becoming much more common in today's world.

Big Data computing is a recent phenomenon. Some of the examples of these Big Data sets shown in Figure 1 include: one day's worth of Instant Messaging text in 2002 amounts to 750 GB, 200 traffic cameras in London collect 8TB of data daily, the World Wide Web consists of approximately 1 PB of data, the Human Genomics project has collected about 7000 PB of data, and annual email traffic is greater than 300 PB not counting spam. Certainly, a large amount of storage capacity is required to even save this amount of data - on the order of terabytes and petabytes - but what scientists and users want to do with the data is even more important. Storing massive amounts of data can be solved by large distributed file system servers, but scientists need that data to be easily and quickly accessible in order to process it all in some reasonable time frame. Processing this amount of data requires new models of data access and programming. Parallel databases and closely synchronized networks of parallel computers simply cannot scale to these "Big Data" sizes.

Compute clusters can offer an excellent solution to Big Data computing problems. A closely linked cluster of computer systems connected by a very high speed network can provide cluster owners with a cost-effective and time-effective Big Data computing machine. A cluster is not a super computer; rather it is a closely networked system of individual computers similar to the average user's desktop machine. As such, the massive amounts of data involved in Big Data computing must be partitioned and distributed among all nodes in the cluster. To solve this issue, cluster owners make use of a Distributed File System (DFS). A variety of distributed file systems are available and each

Submitted to MCURCSM 2008 Wooster, OH

have their own sets of pros and cons. Once data distribution is adequately resolved, there must also be some method by which programmers can partition and distribute processing tasks among the cluster's nodes.

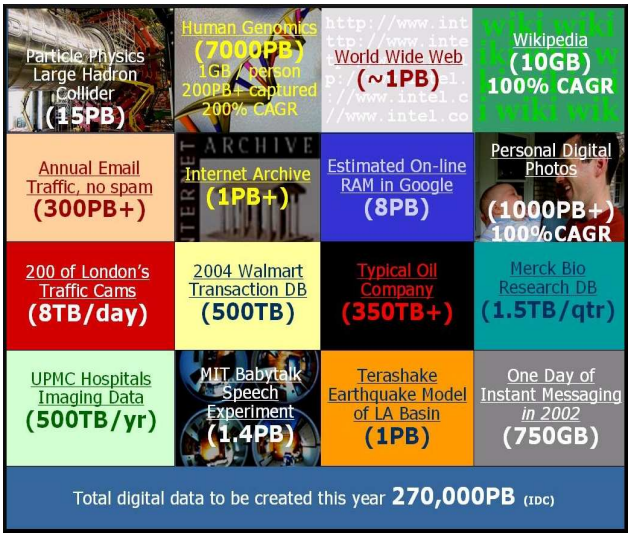

Figure 1: "Big Data" means very large data sets

However, clusters are not without their share of problems. Clusters are often partitioned and distributed to various users as smaller "sub-clusters". This paradigm is known as cloud computing; the cluster as a whole is seen as a "compute cloud" from which partitions of machines can be pulled out and distributed to users.

Clusters generally range from a few dozen up to thousands of networked individual machines with the number of cores reaching into the tens of thousands, so we need to deal with installing and managing software packages on each of those nodes. This is very time consuming, particularly since different users of the cluster will have varying software requirements: one user may need all of his nodes to run Ubuntu GNU/Linux with GCC 4.1 while another user may need all of her nodes to run FreeBSD with GCC 3.5. Unfortunately, since software installation is so time-intensive, users who receive an allocation of machines with their software requirements will be more inclined to simply "keep" these machines to themselves in case they need them again in the future. This phenomenon has been called "cluster squatting" and as a result the compute cloud is often divided into smaller and smaller pieces while used nodes are never recycled back into the pool of "free" cluster nodes.

Current practice is to prescribe a specific system for the entire cloud and users must take it or leave it; the operating system, runtime scheduler, and software stack are all set in stone.

A possible solution to both the time-consuming software setups and the cluster squatting issue is to create a cluster of *virtual machines*. Using virtual machines, the cluster could be partitioned out to multiple users much more easily - software installation setups would be provided by the user, as users themselves would be able to create their own virtual machine image containing their desired operating system and software packages. Furthermore, the virtual cluster idea would solve the cluster squatting issue by providing the ability to redistribute and repartition load across all existing physical cluster nodes; virtual machines which are not actively processing data can be put to sleep or brought offline until they are needed again and as such users cannot "hog" physical cluster nodes.

To manage a virtual cluster, special software is needed. In our research we used an open source project started by the team at Intel Research Pittsburgh. The software project, Tashi, is a cluster management system based on the very idea of controlling a cluster of virtual machines to define virtual clusters that can be apportioned to individual users. Tashi is designed to be virtual machine manager agnostic, that is the system supports the use of multiple virtual machine managers such as KVM, Xen, VMware, and so on. Tashi also does not rely upon any particular distributed file system, so the cluster is easily customizable.

In the following section, we will discuss in detail the background information on distributed computing and virtual machines required to complete our research. In section 3 we discuss our research methods and describe more thoroughly our three distinct workloads. In section 4 we discuss the results garnered from our research. In section 5 we summarize our conclusions and in section 6 we look to some possible future research directions.

2. BACKGROUND

To get to the point of performance assessment we needed to first become familiar with the required underlying technologies. These included the programming paradigm of choice for big data (Map-Reduce by Google) and virtual machine technology.

2.1 Hadoop Programming Model

In our research we used an open source implementation of Google's Map-Reduce distributed computing paradigm called Hadoop. The Map-Reduce programming model executes programs in two distinct phases: map and reduce.

In the map phase of a Map-Reduce application, input is intelligently divided up among the available compute nodes as a series of input shares consisting of key/value pairs. Each input share is then assigned to a map task. A map task processes its share of the input and generates its results in the form of new key/value pairs.

The reduce phase gathers these output key/value pairs from multiple map tasks together and merges where possible to form a single aggregate output.

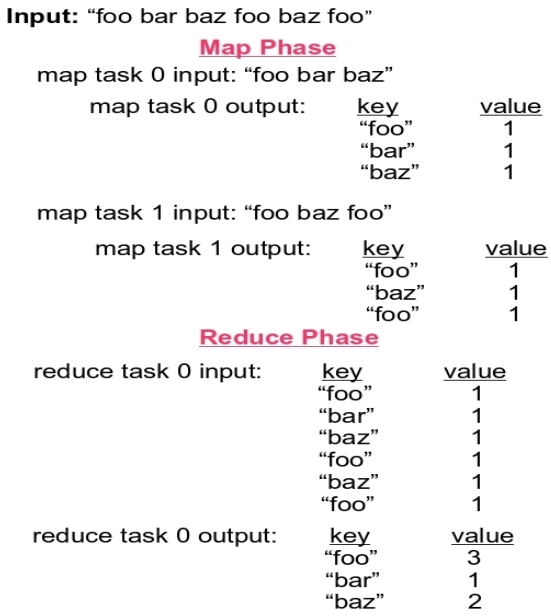

Figure 2: A simple wordcount example

A simple example of Map-Reduce is a word count application. Suppose we have twelve books ranging from Darwin's *Origin of the Species* to Dickens' *Great Expectations* stored as plain text and we want to create a list of all words appearing in these texts as well as how frequently they occur. This is quite a bit of data to handle that clearly doesn't need to be done iteratively; this work can be done in parallel. A Map-Reduce version of the word count application would have its two phases: map and reduce.

The map phase would split the text files up in some intelligent manner, n lines of text per each map task. Then we will see m map tasks distributed across our compute nodes. The map tasks will then have to do the actual work of stepping through its input text and creating key/value pairs x/y for each word it finds. In this case, the key x will be the actual word and the value will always be $y = 1$ since a new pair will be created every time a word is found in the input text (see Figure 2). A possible optimization here would be to optimize this step by combining the values of like keys into a single key/value pair to decrease the work load for the reduce phase.

As map tasks are completed, the reduce phase can begin. Some static number of reduce tasks will be created to handle the m map task outputs which will need to be reduced. These reduce tasks will step through their input and merge key/value pairs with like keys. When all reduce tasks have completed, the key/value pairs resultant from the map phase will have been merged together according to like keys and the reduce phase will output a list of finalized key/value pairs containing every word appearing in the original input text along with that word's frequency y of occurence within the input text.

The Map-Reduce model is popular: Google claims it has implemented hundreds of programs using this model and that it sees over one thousand Map-Reduce jobs executed on its clusters daily. Additionally, Microsoft Research has recently created a more generalized form of Map-Reduce parallel computing called Dryad.

Hadoop itself merely provides us with the framework for running Map-Reduce applications; it only needs to know the network identities of each compute node it may distribute work to and which application to run. As such, Hadoop does not interfere with our plans of running our virtual cluster - it simply will not know the difference between virtual and physical machines. Furthermore, while Hadoop offers its own distributed file system for large-scale data storage (HaDoop File System, or HDFS) it is not necessary to make use of it; we can run Map-Reduce applications with any distributed file system we require.

2.2 Virtual Machines

A virtual machine (VM) is a software implementation of a computer.

There are a variety of Virtual Machine Managers (VMMs) available; these software packages allow real, physical computers to launch virtual machines. Some examples of VMMs include the open source projects KVM and Xen as well as the proprietary VMware.

A VMM must be installed on a physical machine's host operating system; in order to launch virtual machines we first need to start with a host operating system executing the instruction set architecture appropriate to the host hardware platform. The VMM installs an extra layer over the host op-

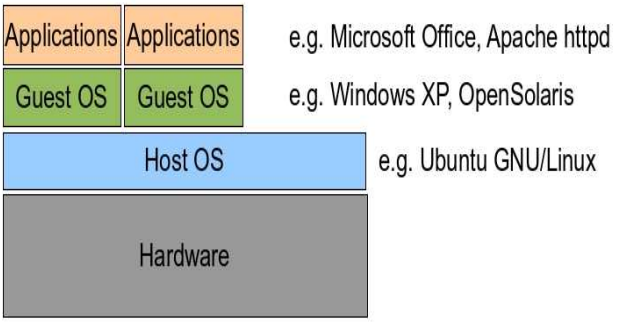

Figure 3: Virtual machine architecture

erating system to allow virtual machines (also called guest operating systems) to communicate with the host operating system which can, in turn, communicate directly with the hardware (see Figure 3).

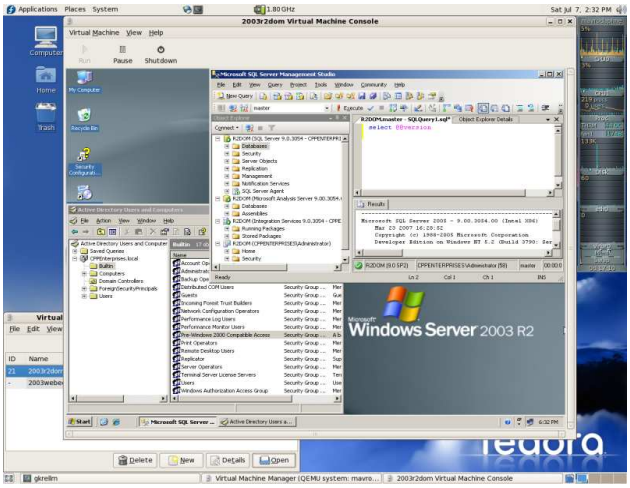

Figure 4: Screenshot of host OS Fedora GNU/Linux running Windows Server 2003 as a KVM guest

A VM is created as a virtual machine image which can be stored on disk. Once launched, a VM resides primarily in memory; depending on its parameters it may or may not write back to its image as its permanent "hard drive". Thanks to this, we can launch any number of VMs from a single image and we can also easily "pause" virtual machines and migrate them to other physical machines within a cluster to facilitate load balancing. Furthermore, users can easily create their own images containing the operating system (GNU/Linux, Windows, etc) and software packages (GCC, Java, etc) they require.

Users can run multiple virtual machines per physical machine. For example, one VM per physical processor core provides a decent balance. There is however a clear performance degredation within virtual machines. Within a VM, non-privileged instructions are executed directly by the hardware. For privileged instructions, however, the VM must go through the VMM which must communicate with the host operating system which then finally must communicate with the hardware itself in order to execute these privileged instructions. It is important to note here the difference between a virtual machine and an emulator: a virtual machine can execute non-privileged user mode instructions as if it were a physical machine but an emulator is a pure software layer which must duplicate a target machine's instruction set in order to "trick" emulated software into thinking it is being run on real hardware. Because of this, virtual machines are much faster than emulators especially when it comes to user mode instructions.

3. METHODS

Virtual machine performance varies greatly depending on the type of instructions being executed: user mode instructions are typically close to physical machine performance while privileged instructions such as I/O perform more slowly. In order to evaluate both instruction types, we used a CPU-intensive as well as a disk-intensive benchmark. Finally we ran a real-world seismic ground modeling application consisting of both CPU and disk operations. Each test over all three applications was conducted five times on both physical and virtual machines at each scaling factor of working cores using the KVM virtual machine manager; our results reflect the average timing data over these five samples.

The amount of execution time required for each of these three workloads limited us to only five samples for each experiement set. For example: two of our disk-dominant experiments required, depending on the amount of working cores, between 30 minutes to five hours of execution time per sample; we ran five samples of these two experiments on a physical cluster which required well over 40 hours for each experiment - then the same experiments had to be re-executed using a virtualized cluster which takes at least as much time as the physical cluster. The result is that just these two of our experiments took over 80 hours of cluster time each. Because of this, we decided to limit ourselves to five samples of each experiment to give a feel for the statistical variation across these experiments while still allowing this phase of our research to be feasible during our time allotment.

In our research we set up two separate clusters to evaluate and distribute the work load. The main cluster, Denison cluster 1, consists of 12 dual-core Intel Pentium D processors running at 3.0 GHz with 1 GB RAM and 80 GB hard drives. These

machines were connected via a 1000 Mbit network. This cluster also contains four quad-core Intel Core 2 Quad machines with 250 GB hard drives and 2 GB RAM; these machines served as our NFS servers for our disk-dominant application.

Denison cluster 2 was built using six machines containing dual-core Intel Core 2 Duo processors running at 2.13 GHz with 2 GB RAM and 80 GB hard drives. These machines were connected via a 100 Mbit network.

3.1 CPU-intensive

With 40 lines of text as input parameter sets and minimal output, a model of extra-galactic jets by Dr. Homan, Physics, of Denison University presented itself to us as an excellent CPU-intensive application. The model is a radiative transfer model of extra-galactic radio jets. These are jets of plasma, electrons and protons in a magnetic field, which flow from the centers of galaxies. In an effort to understand the properties of the magnetic fields and particles within the jets, this simulation runs a variety of models with numerous permutations of the model variables. The model then compares the results of each permutation to the actual radio emission coming from the jets. In the end the model is looking for a possible range of variables which explain the properties of these jets.

For our purposes, we used 40 lines of sight, or permutations of model variables and looked at the relative compute times when this parallel computation was executed on both virtual and physical clusters. The nature of the model is that every line of sight is independent of every other line of sight which allowed for the parallel distribution of the computation across the cluster without losing any precision or requiring communication between computational pieces. Our cluster performed the computations independently and wrote the results of each run to a single text file. The output required for this application was also minimal and the majority of the run time lied solely in the computation.

We ran this test on 24, 16, 8, 4, 2, and 1-core clusters using Denison cluster 1.

3.2 Disk-dominant

In an effort to observe the effects of significant input and output with the distributed file system, we created a synthetic application that would encompass both of these attributes. The high level view of the application consisted of two phases. The first phase was to input the data and the second phase to perform some work over the data. For the storage input set, two differing sized files of randomly generated binary integers were used. The sizes were created differently at 10 GB and 100 GB to allow observation of the effect of volume. We also appropriately sized the data with an effort to avoid unintended caching effects. As each run of the simulation was performed, the data was read in via standard input and a check sum was performed. This application was run with and without moderate user mode processing tasks (check sums) to examine the impact of an I/O-only job versus heavy I/O with a relatively small computation. After this step there was a small output via standard output. The disk intensive side of the application relied primarily upon the large input as little output was written. The goal of this application was to measure the performance of DFS I/O under virtualization on the running platform. The two distributed file systems used in this experiment were NFS and HDFS. NFS data was distributed across four servers while HDFS replicates data, which is distributed across the cluster.

Map-Reduce applications are characterized by the set of map tasks receiving an allocation of the input data set (their input split) and performing their map task on the data; therefore Map-Reduce applications display a basic cycle of reading from the DFS followed by computation over the data. This is exactly the cycle performed by our synthetic disk task, so it represents the structure of typical Big Data applications implemented within the Map-Reduce programming paradigm.

We ran this test on 24, 12, 6, and 2-core clusters using Denison cluster 1.

3.3 Hybrid

This application presented a real world problem with balanced Input/Output and computation. Developed by the team at Intel Research Pittsburgh, the application reads a small (< 1MB) input file and runs a complex, CPU-intensive ground modeling algorithm on the input parameters. Its output is fairly disk-intensive (> 9GB map phase output, > 500MB reduce phase output). This application requires heavy-duty computing power, so we had to scale it down to run well on our cluster. This was done by modifying the input parameters in order to diminish the depth that the application would model. We ran this test on 12, 6, 3, and 1-core clusters using Denison cluster 2.

4. RESULTS

In the following section we present elapsed real time as our fundamental metric for each experimental variable as we scale the number of cores

applied to the parallel program. It is important to note that when scaling beyond four cores, the resultant times showed less variability and conformed to expected trends. However when using very few cores, variability increased dramatically and the observed results were therefore less consistent. Under normal circumstances, the concept of a cluster with fewer than four cores is trivial and most likely not be used.

4.1 Plasma Jets

As expected, the KVM virtual machines performed admirably compared to physical machines in this test since nearly all of the computation involves user mode instructions; the virtual machine cluster was on average 3.03% slower than physical machines (see Figure 5).

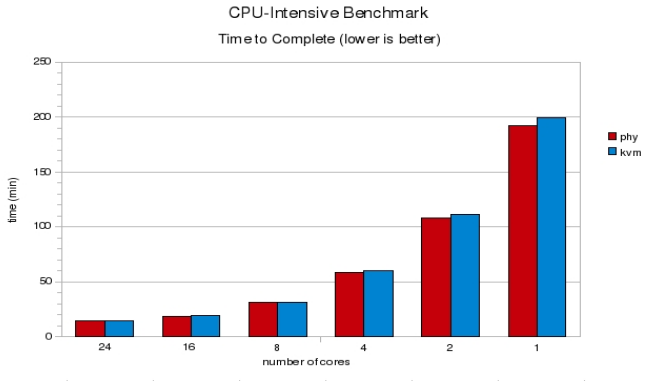

Figure 5: CPU-intensive workload results

The most significant trend observed was that as the number of cores increased, the overhead diminished increasingly. It is important to note that there is no contention on accessing data as the size of the input is trivial. Therefore, there is no overhead associated with increasing the number of cores. This is encouraging as most jobs involving heavy computational problems will want to use a cluster with excessive cores without the fear of losing efficiency to overhead.

4.2 Synthetic Disk

The disk-intensive benchmark started to show some important differences between physical and virtual machines. The primary difference between this application and the CPU-intensive application is that the ratio of privileged instructions to user mode instructions was much greater than that of the CPU-intensive application. Therefore, there was significant overhead as each privileged instruction issued will take longer to execute. The virtual machine cluster was on average 22% slower than physical machines. Considering the two input sizes and the two Distributed File Systems we have four variations. These variations are 10 GB under HDFS, 100 GB under HDFS, 10 GB under NFS, and 100 GB under NFS.

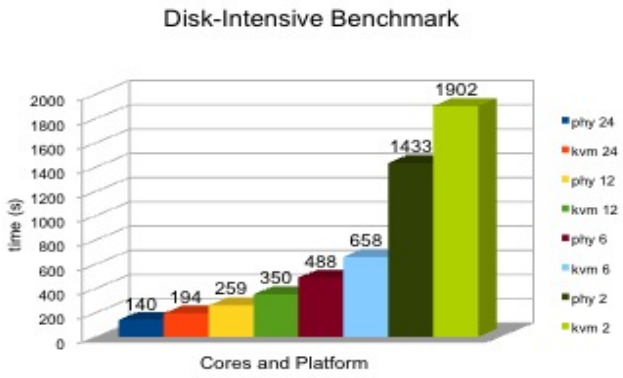

Figure 6: 10GB HDFS Non-Normalized

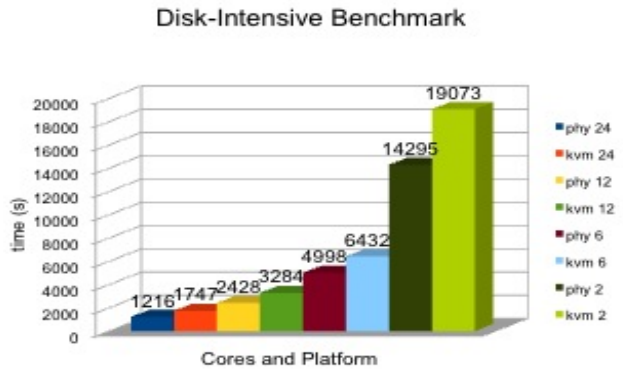

Figure 7: 100GB HDFS Non-Normalized

Looking at the results individually by the four sections we observe some interesting trends. First, we examine the HDFS implementation using a 10 GB input set (Figure 6). We see a range of 33% to 38% overhead. The most important trend to note in this instance is that as the number of cores increases, the overhead also increases. This is the opposite result as observed in the CPU-intensive application. Next looking at the same implementation, however using the 100 GB input set, a range of 29% to 44% overhead was observed (see Figure 7). In general, we see the same positive correlation between number of cores and overhead.

Looking now at the implementation using NFS we observe some similar trends. For the 10 GB input set our range of overhead ranges from 20% to 24%. Comparing this to the HDFS overhead for 10 GB input we see a drastic decrease. Looking lastly at the 100 GB input, our range of overhead is

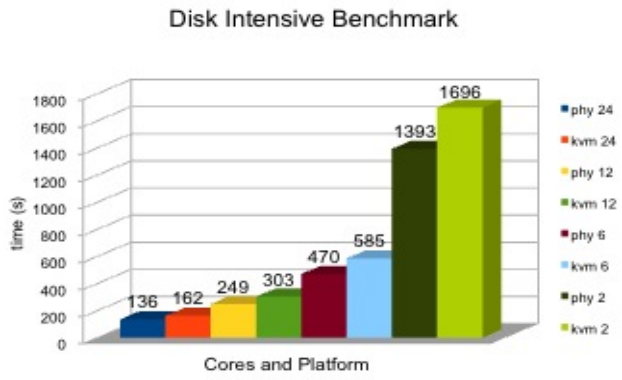

Figure 8: 10GB NFS Non-Normalized

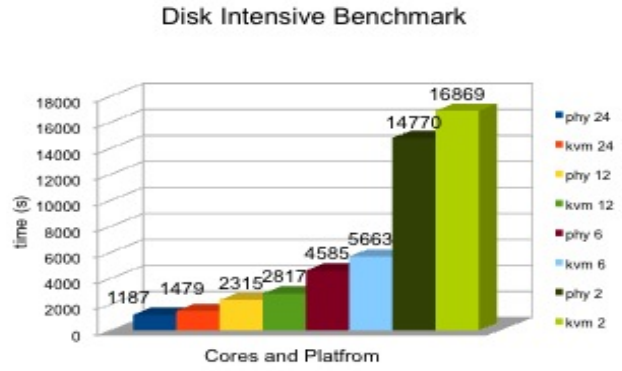

Figure 9: 100GB NFS Non-Normalized

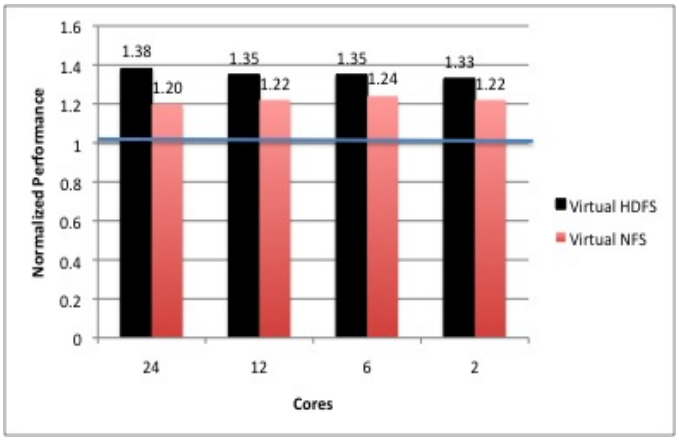

Figure 10: 10GB NFS/HDFS Normalized

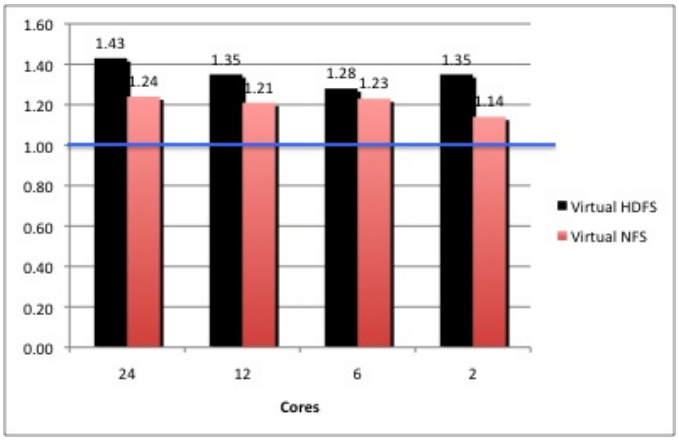

Figure 11: 100GB NFS/HDFS Normalized

from 14% to 25%. This data set also demonstrates a slight correlation between number of cores and overhead. Analyzing the four data sets as a whole the most noteworthy observation is the difference in overhead between HDFS and NFS. NFS performed significantly greater than HDFS across the board (Figures 8 and 9). The other observation of the data as a whole is that most sets demonstrated a trend of increasing cores implying increasing overhead.

In order to evaluate the data across the different number of cores used, we normalized the data according to the physical performance average on the respective number of cores. This demonstrated the average overhead cost due to virtualization without scaling. This was done on all four variations (see Figures 10 and 11).

Examining the cause of this trend deals with the fact that as the number of cores increases, there is an increased contention on the data being accessed. This was not a problem in the CPU-intensive application as there was little input to be read. With this large input file being served up from a limited number of sources, we observe potential contention and increased communication.

4.3 Seismic Ground Modeling

With a mix of CPU and disk operations, this test provides a real-world big data application. Virtual overhead ranged from 21% for the 12 core test up to 73% for single core, with large sampling variance under both physical and virtual implementation (see Figure 12). Realistic use of this application would always involve significant parallelism. There were certain unexpected results observed in this application even when run on physical machines. As the number of cores increases it is expected that the overall execution time would decrease as more cores are working on the same amount of data. For all applications this was true with the exception of this one. We observed that the fastest time was found when using a single core. The reason for this is still unknown and pos-

sibly lies within the implementation of the application on our cluster. Ignoring this outlier, the application scaled as expected when increasing the number of cores.

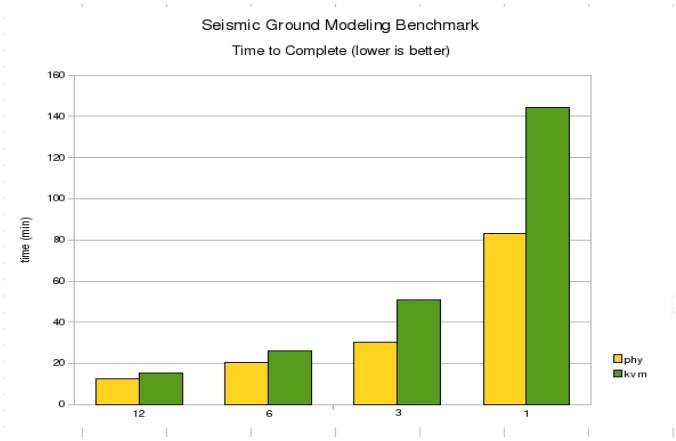

Figure 12: Seismic ground modelling results

5. CONCLUSIONS

Looking at the results of all three applications as a whole the following trends and averages were observed.

For CPU-intensive tasks with little system interaction, virtualization incurred a mere 3% overhead. For Disk-dominant tasks with heavy system interaction, virtualization overhead averaged 22%. For our real-world big data application, virtualization overhead for reasonable levels of parallelization was a very acceptable 21%, but showed the potential dangers of virtualization under low levels of parallelization. Understanding the range of cost and overhead that will be associated with applications run on virtual clusters gives cluster managers and users the knowledge to make the decision on whether virtualization is the right choice for their uses. In general, while virtualization incurs nontrivial costs, cluster management requires solutions and the benefits often justify this cost.

6. FUTURE DIRECTIONS

Possible future directions include extending this research to cover different virtual machine platforms and file systems as well as differing application bases. The only virtual machine platform used in this study was Kernel Virtual Machines, however others such as Xen or VMware are equally competitive and the knowledge of the relative drawbacks compared with KVM would be largely advantageous when considering a virtual cluster implementation. The two main file systems used in our study were NFS and HDFS however, this is a very limited selection. There are many other distributed file systems that are of interest to the high performance computing community.

7. ACKNOWLEDGMENTS

This research was supported by funds from the Anderson Endowment of Denison University and the Bowen Endowment of Denison University.

Thanks to Dr. Homan of Denison University for his radiative transfer model of extra-galactic jets.

Thanks to Intel Research Pittsburgh and Dr. Dave O'Halloran for the seismic ground modeling application.

Tashi is developed by Michael Ryan (michael.p.ryan@intel.com) at Intel Research Pittsburgh. Special thanks to Michael Ryan at Intel Research Pittsburgh for helping us get the seismic ground modeling application up and running on our cluster.

8. SOURCES

1. Noll, Michael. "Running Hadoop On Ubuntu Linux (Single-Node Cluster)." 31 Aug 2008. 12 July 2008 <http://www.michael-noll.com/wiki/Running_Hadoop_On_Ubuntu_Linux_(Single-Node_Cluster)>.

2. Noll, Michael. "Running Hadoop On Ubuntu Linux (Multi-Node Cluster)." 5 Sep 2008. 12 July 2008 <http://www.michael-noll.com/wiki/Running_Hadoop_On_Ubuntu_Linux_(Multi-Node_Cluster)>.

3. Noll, Michael. "Writing An Hadoop MapReduce Program In Python." 10 Oct 2008. 12 July 2008 <http://www.michael-noll.com/wiki/Writing_An_Hadoop_MapReduce_Program_In_Python>.

4. Oxer, Jonathan, Kyle Rankin, and Bill Childers. Ubuntu Hacks. O'Reilly Media, 2006.

5. Pfister, Gregory. In Search of Clusters. 2nd Ed. Prentice Hall PTR, 1997.

6. von Hagen, William. Professional Xen Virtualization. Indianapolis, IN: Wrox Press, 2008.