

Pragmatics of Virtual Machines for High-Performance Computing: A Quantitative Study of Basic Overheads

Cam Macdonell and Paul Lu
Dept. of Computing Science
University of Alberta
Edmonton, Alberta, T6G 2E8, Canada
Email: {cam|paul}lu@cs.ualberta.ca

Abstract— Heterogeneous administrative domains, operating systems (OS), and libraries make it difficult for computational scientists to fully utilize metacomputers and grids. Dealing with the presence or absence of features on, say, different clusters, adds complexity. How can a user or high-performance computing (HPC) application abstract out the heterogeneity?

One possible solution is to use a virtual machine (VM) environment that supports guest operating systems and virtual disks. But, over the decades, VMs have sometimes suffered from performance overheads and limited platforms on which they can run. Through a simple quantitative study, we show that recent improvements in software and hardware support reduces the overheads for HPC applications (e.g., GROMACS, BLAST, HMMer) to under 6% for compute-intensive jobs, but 9.7% or higher for more I/O-intensive jobs, on our x86-based platform. We also argue for qualitative and pragmatic benefits of using VMs for HPC, including ease of deployment, improved functionality, and the ability to run jobs on more systems than would normally be accessible. While not perfect, VMs are emerging as a pragmatic tool in HPC.

Keywords— metacomputing, grid computing, virtual machine (VM), bioinformatics, GROMACS (molecular dynamics simulation), benchmarking, file systems

I. INTRODUCTION

Heterogeneity, in various forms, is often a pragmatic barrier to users taking advantage of different computer systems for a high-performance computing (HPC) workload. For example, many scientific computations in HPC consist of a set of similar jobs (sequential or parallel; we mainly focus on sequential jobs in this discussion) for a parameter sweep, such as exploring the forces between two molecules as the relative position of the molecules change [Su and Xu, 2005]. Ideally, the scientist should be able to aggregate the research group’s workstations, the department’s cluster, and the university’s HPC consortia to run different, independent jobs from the workload. But, if the systems have different security infrastructure, run different operating systems (OS), or have different versions of software libraries, then there is a (potentially) complex process of porting and re-configuring the application and jobs for each system.

A. Background

Grid computing [Foster et al., 2002] attempts to solve some of the heterogeneity problems by mandating a class of software that needs to be installed on all systems. For example, if one installs the Grid Security Infrastructure (GSI) on all the systems, it becomes possible to support a common security model on the grid, regardless of the existing, heterogeneous security mechanisms. In essence, grid computing achieves homogeneity by defining a new, homoge-

neous software platform. Other projects [Lu et al., 2006], [Pinchak et al., 2003], [Anderson, 2004] partly address heterogeneity by exploiting existing software systems that are already (nearly) universally deployed (e.g., Secure Shell for security, or basic TCP/IP for client-server interactions) and minimizing the new software required for HPC workloads.

Among the remaining barriers to the mainstream use of diverse computational resources is the heterogeneity of OSes and libraries. It is not an explicit design goal of (most) grid computing nor metacomputing systems to abstract out the differences between OSes for the applications. Java, Javascript, and Flash can be described as homogeneous platforms for the World Wide Web that make it unnecessary (in theory, impossible) for applications to access the OS, thus making heterogeneous OSes less of an issue. But, existing applications have to be re-written for these platforms. And, for example, the Globus Toolkit deals with the heterogeneity of libraries (and other software or hardware) by providing tools to automate *resource discovery* (i.e., finding the platforms that have the right resources and right versions of those resources). However, resource discovery does not actually increase the number of usable systems; resource discovery locates the subset of resources that can be used.

B. Virtual Machines

How can existing, unmodified applications be supported across different platforms, regardless of what OS and libraries, or version of libraries, are available on the host system? One possible answer is through virtual machines (VM) that virtualize the physical hardware, such as VMware [Adams and Agesen, 2006] (www.vmware.com), Parallels (www.parallels.com), and, historically, IBM’s System/360 VM. (Note that Parallels is the name of the commercial product and is not specifically referring to parallelism in HPC.) Unlike Java VMs, VMware (and similar systems) virtualize the hardware without changing the instruction set of the processor or the standard ways of interfacing to input/output (I/O) devices. There are a number of other approaches to virtualization, including Xen’s paravirtualization [Barham et al., 2003], KVM’s Linux kernel-based approach [Qumranet, 2006], and the forthcoming Windows-based strategy from Microsoft. For this study, we focus on VMware because of its relative maturity and current wide availability. A comparison between different VMs and strategies is the subject of future work.

By virtualizing the hardware, a *host server* and *host OS* can host the VM, which in turn can run a *guest OS*. On top of the guest OS, an unmodified version of the application can execute as if it was running on bare hardware with the appropriate OS. Note that we use the term “bare hardware” (as opposed to “virtualized hardware”) to refer to the *combination* of hardware and a (host) OS throughout this discussion.

For example, consider an instance of the GROMACS application (a molecular dynamics (MD) simulator) [Lindahl et al., 2001] that is normally executed on an x86-based server, running a Linux 2.4-based distribution, with libraries from the year 2005. VMware allows the creation of a VM (i.e., a set of files containing the contents of virtual disks) that contains Linux 2.4, the necessary libraries, and GROMACS itself. The resulting VM can run on a host system that consists of, say, x86-based hardware and running Microsoft Windows XP. Furthermore, VMware can run the *same* VM on a variety of guest OSes, including Linux (whether 2.4-based or 2.6-based), Mac OS X, and other versions of Windows. By packaging a VM with GROMACS, Linux 2.4, and libraries, it should be possible to run multiple instances of the *same* VM on heterogeneous host OSes, with different versions of the same OS, and with different versions of their required libraries.

However, the VM-based approach does have some disadvantages:

1. It is non-trivial to create a VM for a scientific application. Packaging the OS, libraries, as well as the application itself requires more expertise (and effort) than is typical of most computational scientists.
2. Contemporary VM products, such as VMware, are limited to x86-based hardware platforms.
3. Virtualization has overheads [Adams and Agesen, 2006]. Emulating different I/O devices and dealing with the issues of privilege (in the traditional sense for OSes) results in a loss of some performance, compared to running directly on a host OS and hardware.

The goal of this paper is to evaluate virtualization overheads, in the context of HPC applications, to consider some of the pragmatic issues related to VMs, and to draw some appropriate conclusions about the advantages and disadvantages of VMs for HPC. After a set of simple, quantitative experiments, we conclude that VMs are promising tools for compute-intensive applications, and are less well-suited for I/O-intensive applications.

II. THE PRAGMATICS OF VMs

A number of arguments have been made in favour of using VMs on grids and similar environments [Figueiredo et al., 2003]. In this section, we focus on three main pragmatic reasons to consider VMs. In the next section, we consider the performance of full-sized applications.

Pragmatics 1: Virtual Appliances: Although creating a VM from scratch does require expertise, a growing trend with VMs is to create so-called *virtual appliances*. For example, a Linux 2.6-based distribution (be it Gentoo, Debian, Scientific Linux, or any other) can be packaged into a VM and distributed as a unit, analogous to shrink-wrapped consumer software. In particular, VMware now supports an online user community where dozens of pre-

packaged appliances have been created and are available for download (<http://www.vmware.com/vmt.n/>). Examples of appliances include self-contained mail servers, network firewalls, and distributed file systems [Closson and Lu, 2005]. And, the virtual appliances can be used with VMware’s free-to-use versions of their VM, known as VMware Player (analogous to the free-to-use version of Adobe Acrobat, known as Acrobat Reader) and VMware Server (which is different than the non-free VMware ESX Server).

Due to the open-source licence of Linux and related software, users can install their applications on the VM, and the resulting appliance can also be redistributed. One can imagine GROMACS being installed on top of an existing Linux-based virtual appliance to create, say, a GROMACS appliance. Therefore, most computational scientists do not have to become experts in installing and configuring Linux; they install their software on the VM in the same way they install their applications on their Linux cluster. Or, they download a pre-made application-specific appliance.

Whether packaging one’s own VM or using a pre-made appliance, the result is a system that can be used *without changes* on a variety of host OSes, regardless of what version of the OS or libraries are on the host. That is a significant reduction in the amount of heterogeneity that the computational scientist has to be concerned about.

We have built a virtual appliance of our own, the Trellis NAS Bridge Appliance (Trellis NBA or TNBA). It is a virtual appliance that provides “bridged” file access [Closson and Lu, 2005] across administrative domains using Secure Shell as the basic security mechanism [Lu et al., 2006]. In HPC, explicit stage-in/stage-out of data is common, tedious, and error-prone. Using a combination of Samba (www.samba.org) and technology from the Trellis Project, TNBA allows unmodified binary applications to access files (using open, read, write, and close, as per a file system) instead of via copying or file transfer. We believe that the packaging of virtual appliances with a distributed file system will provide a useful platform for HPC, especially if the performance overheads are negligible (Section III-E).

Pragmatics 2: x86 Platform: Although VMware and most contemporary VMs are limited to x86-based hardware platforms, it is such an ubiquitous platform that it is still meaningful to consider the VM-based approach. Furthermore, a virtual appliance can be run on (almost) any x86-based system running Microsoft Windows, Linux, and Mac OS X. Therefore, in some situations, the number of actual servers that are usable with the VM can grow in some ways (i.e., more OSes) while shrinking in other ways (i.e., non-x86 servers).

Pragmatics 3: Overheads: Perhaps the most worrisome aspect of VMs is the potential loss of performance. In HPC, a great deal of money and effort is spent to increase performance by a few percent. How much of an overhead does the VM incur? Why would anyone be willing to use an approach that had any additional overhead?

This paper attempts to quantify the basic overheads of the VM approach. Using the GMX benchmarks distributed with GROMACS, we measure the overhead to be less than

6% (Section III-C, Figure 3) for the more compute-intensive workloads. For more I/O-intensive workloads, we measured overheads as high as 9.7% when comparing the VM against bare hardware. Therefore, the VM-based approach may not be ideal for all scientific HPC applications. But, if one’s application is similar to GROMACS, then for about a 6% reduction in performance, one can gain the benefits of a portable virtual appliance (to deal with heterogeneity) that can potentially run on many different x86-based OSes.

III. EXPERIMENTS

The goal of the experiments is to answer the question: Can the current generation of VMs, as represented by VMware, be competitive enough with bare hardware to warrant consideration for throughput-oriented HPC workloads?

To measure the overheads of running scientific applications, three widely used scientific applications related to bioinformatics and biological simulation are measured on hardware and under VMware on the same platform. The benchmarks are BLAST [Altschul et al., 1990] (sequence matching in bioinformatics), HMMer [Eddy, 1998] (machine-learned pattern matching), and GROMACS [Lindahl et al., 2001] (molecular dynamics simulation). These benchmarks are in wide use and often run on clusters with a batch scheduler. Also, GROMACS and HMMer have both been included in the recently released SPEC CPU 2006 benchmarks.

All real-time data points are the averages of five runs, as measured using `gettimeofday()`, where the observed standard deviation of times is very low. There is some concern about the use of `gettimeofday()` inside VMs, so we also experimented with using the timestamps on network ping packets to measure real time [VMware, Inc., 2006]. We found differences of less than 1% in the timings between `gettimeofday()` and `ping` for our runs.

Our test machines are dual Opteron (model 248, running at 2.2 GHz) Linux servers with 4 GB of RAM and a 250 GB disk. The virtual environment is run under VMware Server version 1.0.1, which is free-to-use after registering. Note that we did *not* use the higher-performance VMware ESX Server, which has an associated licencing fee. Benchmarks using VMware ESX Server would be of interest, but for practical reasons, those experiments are left for future work.

Virtual machines were configured with 2 GB of RAM, which is sufficient for all of the non-I/O-related needs of the benchmarks. For our experiments, the memory allocated to a VM is less than the total physical memory of the host hardware. We used 2 GB of RAM in the VM even though the server had 4 GB of RAM since our applications did not require all of the memory, and to eliminate any issues related to overcommitting memory and paging. In general, the amount of useful memory for the VM will be less than the total physical memory, which is one of the trade-offs in the VM-based approach.

Except where it is noted in the results, the virtual machines were configured to use two processors. The operating system on the servers (“host OS”) is Scientific Linux 4.4 (Linux Kernel 2.6). Our “guest OS” inside the virtual machines is Gentoo Linux 2006.1 (Linux Kernel 2.6). Our

experience is that there are no significant *performance* differences between the two Linux distributions; we chose both Scientific Linux and Gentoo to emphasize how VMs allow for different guest and host OSes. Each virtual machine has two 16 GB virtual disks. Growable disks are used for all experiments, but we also report on performance with pre-located disks in selected situations (Section III-D). One disk is used for system data and the second is used exclusively to run the experiments.

In the first part of our study, we compare applications running under a virtual machine, VMware Server, versus running on bare hardware. In this experiment, both the host and guest OSes are 64-bit versions. In the second part of our study, the GROMACS and BLAST applications run within a virtual machine. The virtual machine in this experiment is the current version of our Trellis NAS Bridge Appliance (TNBA) (with the Trellis File System) (Section II), which is packaged with a 32-bit version of Gentoo as the guest OS. The ability to run a 32-bit guest OS on a 64-bit host OS is another example of how VMs can abstract heterogeneity (Section III-F). We ran benchmarks with the data located outside the virtual machine on a different machine on the local network and brought in “on demand”. The machines are connected with gigabit Ethernet across a switch. We compare two techniques for accessing the remote data: the Trellis File System and stage-in/stage-out.

A. BLAST

BLAST (Basic Local Alignment Search Tool) is a widely used tool for finding similar nucleotide or protein sequences. A typical input is a single sequence which is compared against a database of known sequences. The most statistically similar sequences are computed and returned. The BLAST tests were taken from two sources: (1) the Bioinformatics Benchmark System (BBS) benchmark (version 3) and (2) the Pathway Analyst project here at the University of Alberta. For our experiments, we used BLAST version 2.2.15.

The BBS BLAST benchmarks were used in the first part of our study, along with the FASTA NR and PATNT databases from NCBI dated December 12, 2006. The BBS benchmark tests three different programs in the BLAST suite. Using `vmstat`, we did a rough characterization of the I/O intensity of the different programs and found that they required averages of 8,600 blocks per second (bps), 1,016 bps, and 219 bps (of 4 kilobyte blocks) of I/O for the single CPU versions of `blastn`, `blastx` and `tblastx`, respectively. The algorithmic differences between these programs are beyond the scope of this paper, but each varies in its input, comparison method, and measured I/O intensity.

The Pathway Analyst BLAST test (Table I) was used in the second part of our study. For the Pathway Analyst test, the proteome (all proteins) of *E. coli* was compared against 43 organisms, including itself, from the KEGG dataset downloaded on September 28, 2006. In total, 18,841 sequences were searched for against 289,770 sequences in the database. The program `blastp` is used in the Pathway Analyst benchmark. `blastp` compares an amino acid query sequence against a protein sequence dataset.

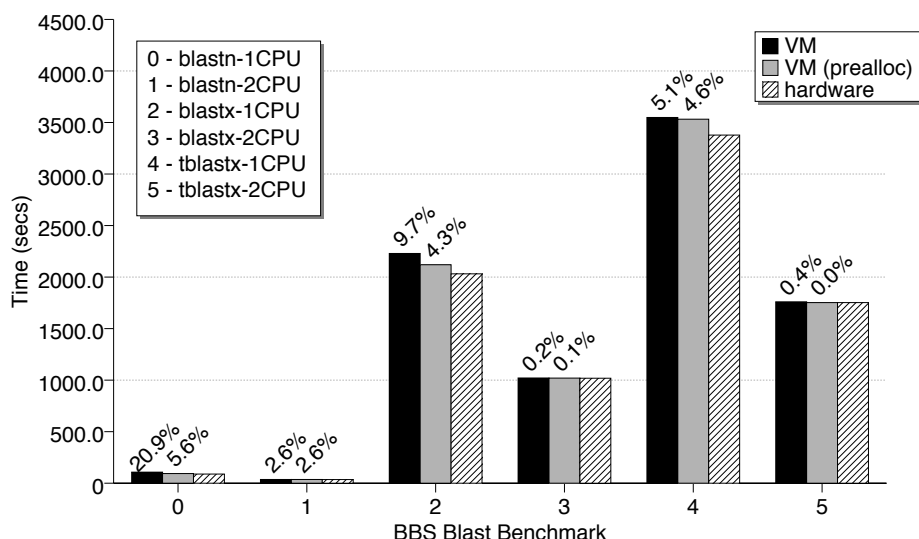


Fig. 1. Performance of the BBS BLAST Benchmark under VMware Server and on hardware. Percentage overhead versus hardware shown above the bar: VM with growable disks and VM with preallocated disks.

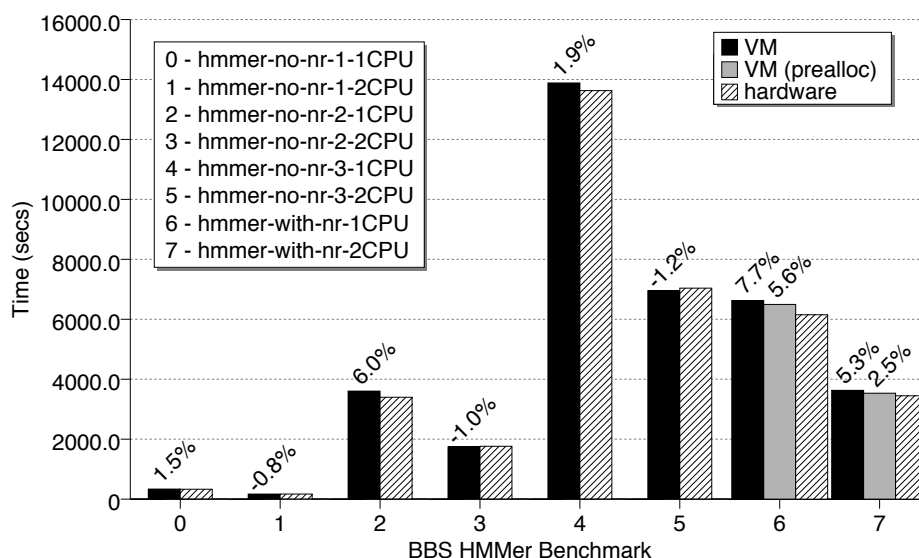


Fig. 2. Performance of the BBS HMMer Benchmark under VMware Server and on hardware. Percentage overhead versus hardware shown above the bar: VM with growable disks and VM with preallocated disks. Datapoints are without (0 to 5) and with (6 and 7) the NR database.

B. HMMer

HMMer (pronounced “Hammer”) is a profile hidden Markov model (HMM) implementation that generates statistical descriptions of sequence families and searches databases for similar sequences. Similar to our BLAST benchmarks, the inputs for HMMer were taken from the BBS. Version 2.3.2 of HMMer was used for the tests. The HMMer benchmark includes HMM training applications and search applications. HMM training (labeled *hmmer-no-nr* in the graphs) are compute-intensive with little I/O. The HMM search applications (labeled *hmmer-with-nr* in the graph) are a database search and so are I/O-intensive, as confirmed by `vmstat`. For the HMM search, the FASTA NR database mentioned above was used as the database to search against.

C. GROMACS

GROMACS is a suite of applications used for molecular dynamics (MD) simulations. It generates the trajectories of the atoms in a molecule, in water, over a period of time, typically on the order of picoseconds or nanoseconds. For this study, we measure the runtime of the computationally intensive `mdrun` program that actually performs the simulation. GROMACS v3.2.1 was used with the FFTW v2.2.15 library. We used GROMACS v3.2.1 as it is the version used by computational biologists that we collaborate with. The newer GROMACS v3.3.1 has had issues with different versions of the GNU C compiler (`gcc`) that has kept our collaborators from using it. As input, we used the GROMACS benchmarking system *gmxbench* that consists of four molecules published by the GROMACS group. The four molecules in

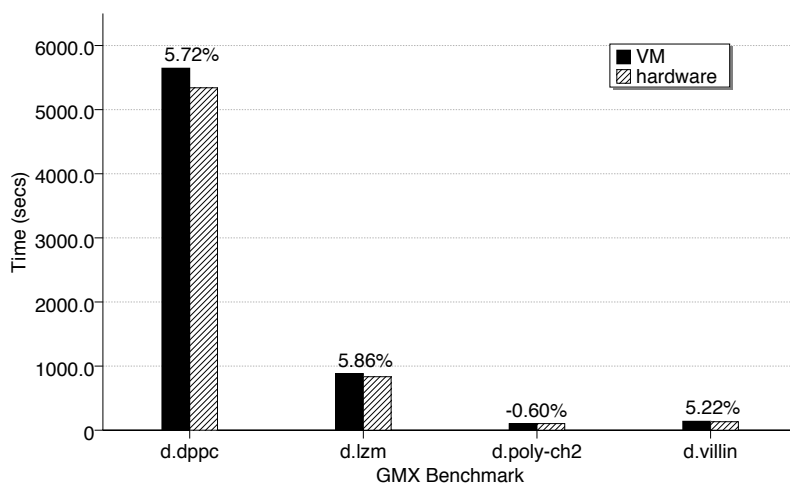


Fig. 3. Performance of GROMACS *gmxbench* under VMware Server and on hardware. Percentage overhead versus hardware shown above the bar: VM with growable disks only.

gmxbench are *d.dppc*, *d.lzm*, *d.poly-ch2*, and *d.villin*. This benchmark is used in both Sections III-D and III-E.

D. Results

Figure 1 shows the results for the BBS BLAST benchmark, with one and two processors via shared-memory parallelism. Each set of bars compare the same benchmark run under VMware versus being run on hardware. Both growable and preallocated disks are evaluated. The percentages above the bars are the percentage overheads in runtime under virtualization.

For growable disks (i.e., virtual disks that allocate storage on demand, instead of at time of creation), the overheads vary from 0.2% to up 9.7% for substantial runs, with a high of 20.9% overhead for a small dataset run. The standard deviation on the average of five runs for each data point is approximately 4%. There are data points (in most of the experiments discussed in this section) in which the VM and hardware times are almost identical, or where the VM times are nominally “faster” than on real hardware by a small amount. However, we focus on the cases where the differences are significant to better understand the worst-case scenarios. For space reasons, all BBS benchmarks are numbered, with the names given in the legend.

Whereas growable disks use less storage for sparsely populated virtual disks, preallocated disks have lower performance overheads [VMware, Inc., 2006]. Our experiments support this tuning guideline as the runtime overheads for BLAST are reduced to between 0% to 5.6% when using preallocated disks. Therefore, if storage efficiency is key, then growable disks can be used, but preallocated disks offer higher performance for BLAST.

Figure 2 shows the results of the BBS HMMer benchmarks run in a similar manner to the BLAST benchmarks. The overheads are shown to be as low as minus 1.2% with growable disks (which is within the standard deviation of the data point). The lower overheads are on the HMMer training benchmarks (i.e., without the NR database, datapoints 0 to 5). The two sets of bars to the right of the graph (labeled 6 and 7) are the HMM searches with overheads as high as

7.7%, with growable disks. These datapoints have higher I/O rates (measured via *vmstat* to be up to 335 bps on average) as they must search the NR database, which is 11 GB in size. With preallocated disks, the overheads drop to 5.6% and 2.5% for the single and dual CPU versions, respectively.

The results of the GROMACS *gmxbench* benchmark are shown in Figure 3. The overheads for *gmxbench* are from minus 0.60% to 5.86%. The *mdrun* program generates four output files that describe the simulated molecular dynamics of the input molecules. These files vary from 200 KB for *d.poly-ch2* to over 2 MB for *d.dppc*, which are small by current standards. As used in *gmxbench*, GROMACS is known to be compute-intensive and is representative of a large class of simulation-based applications.

E. Using Trellis File System for Remote Data Access

As an example of the pragmatic benefits of packaging and virtual appliances (Pragmatics 1, Section II), the Trellis File System can be pre-installed and configured in the VM, along with the application itself. Normally, deploying a distributed file system is either prohibitively complex for most computational scientists or the file system requires privileged access to install. With the VM-based approach, the Trellis File System can be made available as a virtual appliance. The scientist can then co-install their application with the appliance and use it. Running the VM-based appliance and application on a compute node requires no special privilege, since all of the privileged steps are encapsulated inside the VM.

Efficient remote data access is also important in being able to leverage virtualization in metacomputers. Two techniques for accessing remote data were measured in this part of the study. The first is the common practice of using scripts to stage the data in and out of a compute node (in our case the VM). The second is using the Trellis File System to access the data. The Trellis File System accesses the remote nodes and exports a file system interface through a Samba server running inside the virtual machine. For this part of our study, we used the *gmxbench* benchmark and the BLAST search from the Pathway Analyst project (Table I).

Benchmark	Trellis NAS Bridge Appliance Total (% overhead vs. stage-in/out)	Stage-in/Stage-out		
		Total =	computation +	scp
GROMACS d.dppc	4,412.7 (2.6%)	4,307.1	4,300.8	6.3
d.lzm	607.35 (0.6%)	603.7	601.2	2.5
d.poly-ch2	102.7 (1.2%)	101.4	99.4	2.0
d.villin	89.7 (2.5%)	87.5	85.9	1.6
Pathway Analyst BLAST	15,005.5 (5.5%)	14,218.5	14,182.0	36.5

TABLE I: Remote data access from within a virtual machine: Trellis NBA v. Stage-in/Stage-out for GROMACS and BLAST (times are in seconds).

For both remote access methods, the benchmark application is running within the VM and the data is stored on a different node in the same cluster.

There is a row for each of the GROMACS *gmxbench* molecules and the last row is for the Pathway Analyst BLAST run (Table I). The second column is real time in seconds of the benchmark using Trellis NBA to access the remote data and store the results back to the home node. The GROMACS and BLAST applications are unmodified and use the Trellis File System via pre-configured mount points inside the VM. The percentage overhead of Trellis NBA versus stage-in/stage-out is in parentheses. For stage-in/stage-out, the total times are given as well as a breakdown of the time between computation (*mdrun* for GROMACS and *blastp* for BLAST) and explicit data movement via Secure Copy (*scp*).

The additional overheads of TNBA vary from 0.6% to 2.6% for GROMACS and the overhead is 5.5% for the BLAST job from Pathway Analyst. The overheads are due to the use of Samba and Trellis (i.e., extra software layers) within the virtual appliance. For future work, we plan on optimizing the software and improving performance. Currently, for a trade-off of less than 6% in additional overhead, the Trellis NAS Bridge Appliance can provide a distributed file system that does not require any special privilege to install and use, other than installing VMware Server itself.

F. 32-bit vs. 64-bit Assembler Optimizations

When running the GROMACS application in Section III-E (Table I), we observed that the runtimes were lower than both the hardware and VM times from Section III-D (e.g., 4,412.7 seconds in Table I versus 5,340 seconds, which is the left-most *hardware* bar in Figure 3 for *d.dppc*). After some investigation, we realized that running in the 32-bit Trellis NAS Bridge Appliance VM (Table I) was the cause of the difference. The VM hid the fact that the physical processor was an 64-bit Opteron (x86-64) and so the guest OS and GROMACS detected a 32-bit i686 processor. The GROMACS configuration then compiled highly-optimized i686 assembler loops into the GROMACS application. These optimized assembler loops resulted in the much lower runtime. The version of GROMACS (version 3.2.1) we were running did not support the same optimizations under x86-64, so the GROMACS in Section III-D was slower. After tracking down the cause, we also found that the i686 optimizations could not be compiled on the host 64-bit OS. More recent versions of GROMACS (e.g., version 3.3.1) do have the optimized loops for 64-bit OSes.

A full comparison of 32-bit versus 64-bit issues is beyond the scope of this paper. And, this anomaly is likely transitory. But, as discussed above, the ability to run a 32-bit guest OS on a 64-bit host OS is another example of how VMs can abstract heterogeneity.

IV. RELATED WORK AND COMMENTS

Other researchers have studied the overheads of VMs [Adams and Agesen, 2006] and the strategy of using VMs to encapsulate HPC jobs [Figueiredo et al., 2003], [Santhanam et al., 2005]. Our study extends and updates those studies by choosing full-sized applications (e.g., GROMACS, BLAST, HMMer) that are in wide use in our HPC community, and our research group (e.g., the Proteome and Pathway Analyst projects in our department use BLAST and HMMer), instead of relying on industry-standard (but generic) benchmarks such as SPEC [Adams and Agesen, 2006].

The use of virtual machines in distributed environments has been studied before. As part of the In-VIGO system, Figueiredo *et al.* [Figueiredo et al., 2003] examine using virtual machines for distributed computation within a grid framework. They also use VMware. To measure overheads, the authors used two compute-intensive SPEC benchmarks. They report overheads of 1-4% which are lower but consistent with our findings for compute-intensive applications. Their paper did not analyze I/O-intensive applications, but they did measure the impact of other factors such as competing processes. Zhao *et al.* [Zhao et al., 2004], also part of In-VIGO, designed the grid virtual file system (GVFS), a distributed file system for efficiently moving virtual machines across the wide-area network and providing data access within VMs. Network latency for VM images and application data are not separated in their experiments, so the overheads specifically from virtualization are not clear. The conclusions in both papers from In-VIGO are similar to ours: the overheads of virtual machines are acceptable for the benefit they provide in abstracting heterogeneous environments.

Santhanam *et al.* [Santhanam et al., 2005] examine running Condor computations within a virtual environment. The focus is similar to our Pragmatics 1 (Section II) in using VMs as *sandboxes* for the grid. They focus on running computations in four different configurations of Xen virtual machines. The variations involve the location of data and network connectivity of the VM. Their experiments measure data-intensive microbenchmarks involving thousands of concurrent reads and network operations. They characterize the impact that different data access methods, such as

remote I/O and whole-file caching, have on performance. Their baseline for comparison of the VMs is the Condor Vanilla and Standard Universes [Litzkow et al., 1988]. Condor runs batch jobs in distributed, heterogeneous environments. The Condor Vanilla Universe is the closest to our hardware experiments. As expected, their experiments reveal I/O overheads under virtualization. However, due to their exclusive use of microbenchmarks, it is not clear what overheads can be expected for applications.

Our quantitative evaluation uses full-sized HPC applications instead of microbenchmarks. Moreover, our use of the commercial, widely-available VMware, instead of the more research-oriented Condor and Xen combination [Santhanam et al., 2005], is arguably more applicable for our production environments.

V. CONCLUDING REMARKS

Although virtual machines have been around for decades, recent developments in VMs for x86-based platforms have re-opened the VM debate. A variety of VMs are now available commercially (e.g., VMware, Parallels) and as open-source software (with commercial support) (e.g., Linux KVM, Xen). Pragmatically, VMs are a convenient way to package and deploy scientific applications across heterogeneous system. For example, applications can be packaged with their required libraries and support programs, including (perhaps) a distributed file system (e.g., Trellis NAS Bridge Appliance) that would otherwise be difficult or impossible to install without special privilege.

However, an important concern about VMs is whether or not the associated overheads are too onerous. In our simple, quantitative study, we show that the overheads for a compute-intensive application, such as GROMACS, can be under 6%. For more I/O-intensive applications (e.g., BLAST, HMMer with NR database), the overheads can be as high as 9.7%. The overheads will vary depending on the application itself, which is why we chose to do a contemporary evaluation of well-known scientific applications (i.e., BLAST, GROMACS, HMMer) using a modern VM (i.e., VMware). A performance comparison of different VMs and with a broader range of sequential and parallel applications is the subject of future work.

There will always be reasons to not give up any performance at all. But, for the convenience and other benefits of VMs, there may be other situations where the cost-performance trade-off is worth re-visiting.

VI. ACKNOWLEDGMENTS

This research is supported by the Natural Science and Engineering Research Council of Canada (NSERC), the Alberta Prion Research Institute (APRI), the Alberta Ingenuity Centre for Machine Learning (AICML), Alberta Ingenuity, SGI, the Alberta Science and Research Authority (ASRA), and Sun Microsystems. Thank you to the Trellis Project team; Jordan Patterson and the Proteome/Pathway Analyst teams. Thank you to Hemant Gaidhani, Priti Mishra, and VMware, Inc. for permission to report the VMware benchmarks.

REFERENCES

- [Adams and Agesen, 2006] Adams, K. and Agesen, O. (2006). A Comparison of Software and Hardware Techniques for x86 Virtualization. In *12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, San Jose, CA, U.S.A.
- [Altschul et al., 1990] Altschul, S., Gish, W., Miller, W., Myers, E., and Lipman, D. (1990). Basic local alignment search tool. *J Mol Biol*, 215(3):403–10.
- [Anderson, 2004] Anderson, D. (2004). BOINC: A System for Public-Resource Computing and Storage. In *GRID '04: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)*, pages 4–10, Washington, DC, USA.
- [Barham et al., 2003] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. (2003). Xen and the Art of Virtualization. In *Proc. 19th ACM Symposium on Operating Systems Principles (SOSP)*, pages 164–177, New York, NY, USA.
- [Closson and Lu, 2005] Closson, M. and Lu, P. (2005). Bridging Local and Wide Area Networks for Overlay Distributed File Systems. In *Proc. 2nd Workshop on Real, Large Distributed Systems (WORLDS '05)*, pages 49–54, San Francisco, California, U.S.A.
- [Eddy, 1998] Eddy, S. (1998). Profile Hidden Markov Models. *Bioinformatics*, 14:755–763.
- [Figueiredo et al., 2003] Figueiredo, R., Dinda, P., and Fortes, J. (2003). A Case For Grid Computing On Virtual Machines. In *Proc. 23rd International Conference on Distributed Computing Systems (ICDCS)*, page 550, Washington, DC, USA. IEEE Computer Society.
- [Foster et al., 2002] Foster, I., Kesselman, C., Nick, J., and Tuecke, S. (2002). The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG, Global Grid Forum.
- [Lindahl et al., 2001] Lindahl, E., Hess, B., and van der Spoel, D. (2001). GROMACS 3.0: A package for molecular simulation and trajectory analysis. *J. Mol. Mod.*, 7:306–317.
- [Litzkow et al., 1988] Litzkow, M., Livny, M., and Mutka, M. (1988). Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*.
- [Lu et al., 2006] Lu, P., Closson, M., Macdonell, C., Nalos, P., Ngo, D., Kan, M., and Lee, M. (2006). The Trellis Security Infrastructure for Overlay Metacomputers and Bridged Distributed File Systems. *Journal of Parallel and Distributed Computing*, 66(9):1181–1188.
- [Pinchak et al., 2003] Pinchak, C., Lu, P., Schaeffer, J., and Goldenberg, M. (2003). The Canadian Internetworked Scientific Supercomputer. In *17th International Symposium on High Performance Computing Systems and Applications (HPCS)*, pages 193–199, Sherbrooke, Quebec, Canada.
- [Qumranet, 2006] Qumranet (2006). KVM: Kernel-based Virtualization Driver (White Paper). http://www.qumranet.com/wp/kvm_wp.pdf.
- [Santhanam et al., 2005] Santhanam, S., Elango, P., Arpaci-Dusseau, A., and Livny, M. (2005). Deploying Virtual Machines as Sandboxes for the Grid. In *Second Workshop on Real, Large Distributed Systems (WORLDS 2005)*, San Francisco, CA.
- [Su and Xu, 2005] Su, Z. and Xu, Y. (2005). *Ab initio* study of chiral recognition in the propylene imine-hydrogen peroxide complex. *Phys. Chem. Chem. Phys.*, 7:2554–2560.
- [VMware, Inc., 2006] VMware, Inc. (2006). Performance Benchmarking Guidelines for VMware Workstation 5.5 (White Paper). http://www.vmware.com/pdf/WS55_Benchmarking_Guidelines.pdf.
- [Zhao et al., 2004] Zhao, M., Zhang, J., and Figueiredo, R. (2004). Distributed File System Support for Virtual Machines in Grid Computing. In *HPDC '04: Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing (HPDC'04)*, pages 202–211, Washington, DC, USA. IEEE Computer Society.

VII. AUTHOR BIOGRAPHIES

CAM MACDONELL is a Ph.D. student in the Dept. of Computing Science, University of Alberta, Edmonton, Alberta, Canada. His research interests are high-performance computing and operating systems.

PAUL LU is an Associate Professor of Computing Science, University of Alberta, Edmonton, Alberta, Canada. His research interests are high-performance computing, operating systems, and bioinformatics.