HPC-VMs: Virtual Machines in High Performance Computing Systems

Albert Reuther, Peter Michaleas, Andrew Prout, and Jeremy Kepner
Computing and Analytics Group
MIT Lincoln Laboratory
Lexington, MA, USA
{reuther, pmichaleas, aprout, kepner}@ll.mit.edu

Abstract— **The concept of virtual machines dates back to the 1960s. Both IBM and MIT developed operating system features that enabled user and peripheral time sharing, the underpinnings of which were early virtual machines. Modern virtual machines present a translation layer of system devices between a guest operating system and the host operating system executing on a computer system, while isolating each of the guest operating systems from each other.**

In the past several years, enterprise computing has embraced virtual machines to deploy a wide variety of capabilities from business management systems to email server farms. Those who have adopted virtual deployment environments have capitalized on a variety of advantages including server consolidation, service migration, and higher service reliability. But they have also ended up with some challenges including a sacrifice in performance and more complex system management.

Some of these advantages and challenges also apply to HPC in virtualized environments. In this paper, we analyze the effectiveness of using virtual machines in a high performance computing (HPC) environment. We propose adding some virtual machine capability to already robust HPC environments for specific scenarios where the productivity gained outweighs the performance lost for using virtual machines. Finally, we discuss an implementation of augmenting virtual machines into the software stack of a HPC cluster, and we analyze the affect on job launch time of this implementation.

Keywords – virtual machines, high performance computing.

I. INTRODUCTION

Despite the hype and ubiquity in recent years, the concept and technology of virtual machines has been around for over four decades. The first virtual machines were developed to share expensive mainframe computer systems among many users by providing each user with a fully independent image of the operating system. On the research front, MIT, Bell Labs, and General Electric developed the Multics system, a hardware and operating system co-design that featured (among many other things) virtual memory for each user and isolated program execution space [1]. Commercially, the pioneer in this technology was IBM with the release of System 360/67, which presented each user with a full System 360 environment [2]. Virtual machines went out of style in the 1980s as mainframes and minicomputers lost market share and personal computers became more widely accessible at work and at home.

The x86 architecture, on which the PC revolution rode, was not designed to support virtual machines, but in 1997, VMware developed a technique based on binary code substitution (binary translation) that enabled the execution of privileged (OS) instructions from virtual machines on x86 systems. Another notable effort was the Xen project which in 2003 used a jump table for choosing bare metal execution or virtual machine execution of privileged (OS) instructions. Such projects prompted Intel and AMD to add the VT-x and AMD-V virtualization extensions to the x86 and x86-64 instruction sets in 2006, which further pushed the performance and adoption of virtual machines.

Virtual machines have seen use in a variety of applications, but with the move to highly capable multicore CPUs, gigabit Ethernet network cards, and VM-aware x86/x86-64 operating systems, the use of virtual machines has grown vigorously. This has been particularly true in enterprise (business) computing. Historically, enterprise services were deployed to minimize complexity and conflict on each hardware server. The goal was to cost-effectively isolate all of the services that are offered. Practically, this means that one or a few enterprise service components were deployed on each hardware server. So every time a new service was to be deployed or an existing service was to increase in scale, more hardware servers were added to the server closet, server farm, or data center. After years of expansion, this trend cannot be sustained by most organizations. Since most enterprise services are not particularly resource intensive (i.e., they do not consistently require more than a few percentages of CPU time, disk accesses, network calls, etc.), these enterprise services are very good candidates for deploying within virtual machines.

Countless books and articles tout the many advantages of virtualization for enterprise computing. These advantages include:

This work is sponsored by the Department of the Air Force under Air Force contract FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government.

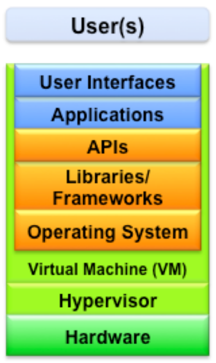

Figure 1: Software Stack including Virtual Machine

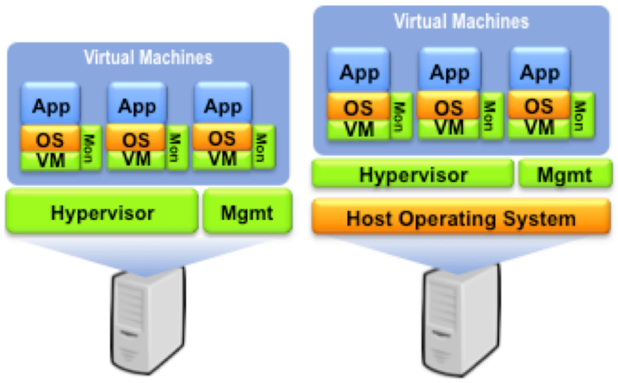

Figure 2a: Type 1 hypervisor, 2b Type 2 hypervisor

- Workload consolidation with service isolation: hosting many enterprise services on each server by executing one or a few enterprise services in each of several virtual machines;
- Rapid provisioning of prototype services: demonstrate services without having to purchase additional hardware;
- Flexible use of available servers: load balancing of enterprise services between servers by migrating virtual machines among them;
- Multiple operating system types: flexibility of hosting multiple OS types on the same hardware server; and
- Possibly more effective security: quickly snapshot servers to known good states, centrally enforce patching, etc.

Along with these advantages come some challenges including management of virtual machine images and load balancing. However, the primary challenge is disk, network, and other I/O performance.

Are these opportunities and challenges also applicable to supercomputing? In what circumstances are virtual machines a good fit for high performance computing (HPC) applications and activities?

Section II explains how virtual machines work. Section III explains why virtual machines are generally not a good fit for high performance computing (HPC) and outlines some scenarios in which virtual machines may be advantageous for HPC. Section IV presents the results on launching customized VMs on LLGrid, while Section V relates this work to other research. Finally, Section VI discusses future research work and concludes this paper.

II. Inside Virtual Machines

The concept of operating systems came about to provide users with an abstraction to the system hardware, providing common services to the user, and managing and controlling shared resources that the hardware provide [3]. To host one or more operating systems within virtual machines on a host server, the virtual machine environment must provide the same capabilities.

Modern virtual machines are software abstractions of hardware architectures. Virtualization is accomplished by installing a hypervisor, which is also known as a virtual machine monitor (VMM). Much like the operating system provides abstracted services from the underlying hardware to the user, the hypervisor abstracts services from the underlying hardware as depicted in Figure 1. The hypervisor abstracts the underlying infrastructure, provides common services to one or more guest operating systems and their applications, and manage and control the shared resources that the underlying infrastructure provide. Hypervisors are deployed in one of two modes: type-1 and type-2.

Type-1 hypervisors are installed directly on the hardware; in this deployment, the hypervisor is essentially a minimal OS for executing virtual machines. Figure 2a depicts the stack of a type-1 hypervisor. Type-1 hypervisors are usually used for server farms to host multiple virtual machines and their enterprise services on servers.

Type-2 hypervisors are installed within a host operating system, and they are executed as a process in the host OS as depicted in Figure 2b. Each guest virtual machine is then a child process to the hypervisor process, and processes within each virtual machine are co-managed by each virtual machine OS and the hypervisor. Type-2 hypervisors are usually used on desktop and laptop computer to enable the use of multiple OSs for users. Because a full host operating system underlies the virtual environment, there is more performance overhead involved in running type-2 hypervisors; however, such a multi-OS environment for desktop and laptop users is a very productive and convenient environment.

To manage and control the shared resources and provide common services to the guest operating systems, the hypervisor has its own kernel-like functionality as well as virtual device drivers that abstract the actual device drivers resident in the hardware. The kernel-like functionality manages and executes processes from the VMs. Much work in the research and commercial worlds has been done in this area, and CPU intensive applications that are executed within VMs now pay a very small performance penalty for being executed within the VM. However, a greater performance penalty is paid by applications that perform much I/O through the hypervisor virtual device drivers such as network accesses, disk accesses,

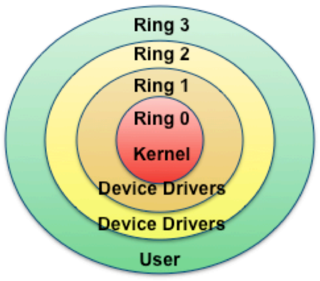

Figure 3: Operating system privilege rings

etc. And the more I/O is required the higher the performance penalty. [4]

The other source of performance penalties for virtual machines has to do with executing privileged instructions like managing memory and I/O operations. Most hardware platforms (including x86 and x86-64) and most operating systems have multiple levels of privilege, often called privilege rings. The x86 and x86-64 architecture has four privilege rings numbered 0 through 3, as depicted in Figure 3. Most x86 and x86-64 operating system including Windows and Linux only use rings 0 and 3 for supervisory and user privileges, respectfully. In order for a ring 3 user application to execute privileged operations like requesting I/O or memory management services, the application must make a system call into the operating system kernel, where carefully vetted ring 0 supervisor level kernel code executes the privileged request on behalf of the application.

With virtual machine environments, the hypervisor, virtual machine(s), and guest operating systems execute with ring 3 user privileges. A virtual machine OS does not have access to ring 0 supervisory privileges in the underlying hardware or host OS. In the x86 and x86-64 environment, this challenge was bridged by binary translation in VMware, jump tables in Xen (as were mentioned in Section I), among others. It is this overhead of privileged operations that is the other performance penalty.

The introduction of virtualization-enabling instruction in Intel VT-x and AMD's AMD-V in 2006 added ring -1 to the privilege ring, which is the supervisory ring for VMs. This allowed hypervisors to use much less "glue" code to enable supervisory operations from virtual machines. Most hypervisors for x86 and x86-64 architectures now take advantage of these instructions, and this has lessened the performance penalty for executing supervisory activities.

III. VMS AND HPC

A. *Performance/Productivity Trade-Offs*

As mentioned before, enterprise services usually do not require a great amount of I/O performance. However, performance is central to HPC. While HPC applications are usually associated with processor-intensive performance, most HPC applications also require high performance I/O. The ratio of computation to I/O performance for any HPC application is just a matter of degrees. In other words, because execution in VMs present a performance penalty (particularly with I/O), HPC application performance will be impacted when executed from within virtual machines, and it is just a matter of how much the performance is impacted.

Often overlooked in this pursuit of HPC performance is the productivity of the scientists and engineers: some trade-offs can be made in performance that could make the scientists and engineers much more productive. The LLGrid system has been a testbed for exploring these performance/productivity trade-offs [5]. In this research we explored in what HPC applications such a performance/productivity trade-off would be reasonable for using virtual machines.

Not all of the advantages for virtualized enterprise computing hold for HPC, though three of the five do: rapid provisioning of prototype services, flexible use of available servers (inherent in HPC scheduling), and multiple operating system types. Several more advantages for HPC are detailed in [6] and [7], including reliability, reduced software complexity, and easier management.

For LLGrid, the two most appealing advantages are rapid prototyping and multiple operating system types. Often a part of starting a new research project involves purchasing one or more servers before really understanding the computational requirements of the application codes. We intend to enable VM instances on LLGrid so that these research projects can explore the computational requirements of their application codes without having to wait for a server order to arrive. With regard to supporting multiple operating system types, we routinely work with research teams that have a scientific code suite that has been validated and verified with a certain OS type, version, etc. that is different than the one installed on LLGrid. Usually this means that the research team must revalidate and re-verify their scientific code suite on the LLGrid software stack. Another scenario is that the scientific code suite is no longer supported on the hardware and OS that comprises LLGrid. If they could execute their code suite within virtual machines with the identical environment in which they were verified, the research teams would sacrifice some performance, while gaining productivity by not having to revalidate and reverify.

B. *VMs on LLGrid*

There are three approaches for using virtual machines on HPC clusters for scientific computing:

1. HPC on an infrastructure-as-a-service (IaaS) cloud,
2. HPC on a virtualized cluster, and
3. Type-2 VMs on traditional HPC cluster.

The majority of virtualized HPC applications and systems research has been done on the first two approaches. With HPC on an IaaS cloud, a user provisions a set of HPC-oriented (higher core count, more memory per core, etc.) virtual machines on a public IaaS cloud provider like Amazon's Elastic Compute Cloud (EC2) or Rackspace Cloud. Once provisioned, the user executes their HPC application on the public cloud instances. This solution usually utilizes type-1 bare-metal hypervisors, which implies better CPU performance. This solution is good for occasional HPC activities and rapid prototyping, but virtual machine costs, virtual storage costs, and transfer bandwidth costs can make this a very expensive option if it is used a lot. Also, some organizations cannot use public cloud resources to due to privacy and security concerns.

With HPC on a virtualized cluster, all of the compute nodes are virtual machines that are managed and deployed using a VM management system like VMware ESX. Generally the VMs are statically allocated onto the available hardware, though it is possible to migrate VMs from server to server when necessary. This approach can be realized with both type-1 and type-2 hypervisors. Usually a HPC scheduler like Grid Engine, LSF, Torque, etc. is used to match and execute jobs on virtual compute nodes. This solution provides flexibility in what and how many OS types are allocated across the cluster, and users just use the resource like they would any other HPC cluster. However, all HPC jobs must be executed within VMs with all of the performance penalties that may be incurred.

The third approach is to provision the cluster in the traditional way with a certain current OS version installed on the bare metal. This OS version is chosen to accommodate as many different HPC jobs as possible, and every effort should be made to execute HPC jobs in the bare-metal, native OS. If another OS type or version is needed, then VMs are used to encapsulate the job(s). The VM-encapsulated jobs are launched through an HPC scheduler just like any other job, and the job sets up the VM, launches the job, monitors the job, and eventually tears down and discards the VM when the job is finished.

Because there is an underlying OS, this solution requires a type-2 hypervisor. To rapidly prototype this capability, we wanted a simple interface to launching and managing VM instantiation and configuration. While all of the major hypervisor solutions have a software application programming interface with which control code can be written, only one of the major type-2 hypervisors has a fully supported command line interface for launching and managing VM instantiation and configuration: Oracle VirtualBox.

The key intent of this rapid prototype was to show fast launch times of the VM and its computational job. Therefore, we stripped down the VM operating system image to have no services and only the libraries that were required for scientific computing. If other libraries are needed they will be symlinked into the users virtual machine environment from a centrally shared library repository. Each VM job submitted to the scheduler clones a VM image, registers the image to the compute node that the scheduler has chosen, and begins the boot sequence of the VM. The job execution in the VM is written into the initialization scripts of the VM-encapsulated OS so that when the VM has fully booted, it immediately begins execution of the job. After execution of the job, the VM enters the shutdown script, and upon shutdown the VM image is discarded. If the VM job is aborted before execution is completed, the VM catches the kill signal sent by the scheduler, and the VM is cleanly powered down and discarded.

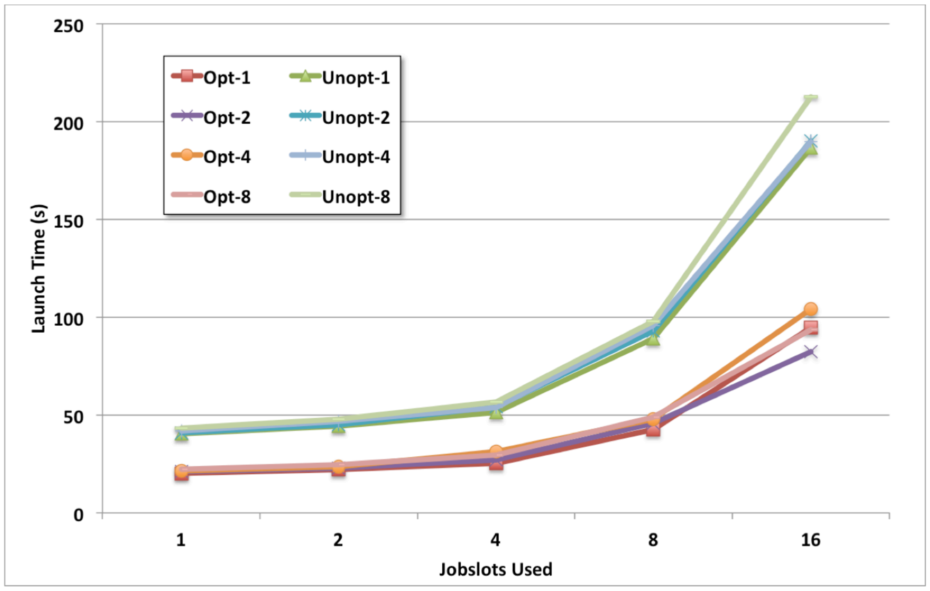

Figure 4: Results of launch timing

While the default is to launch one VM job per job slot, we have added the ability to overload the jobslot with multiple VM instances. That is, we can launch two or more VMs and their related jobs per scheduler jobslot. The number of VMs that can be launched per jobslot is limited by the resources that are available on a compute node. This feature can be useful when testing the scalability of network-centric distributed computing environments where each VM job uses relatively little memory or CPU power.

IV. RESULTS

We demonstrated our VM on HPC capability on the LLGrid TX-2500 cluster at MIT Lincoln Laboratory. In our Grid Engine (version 6.2u5) scheduler, we temporarily dedicated eight nodes for launching VM jobs and measuring their launch and shutdown times. We delivered the VM images from a user account on the central file system, a DDN 10K storage array, connected to the compute nodes via 10GigE core network, and 1GigE connection to each node from the rack switches. Each compute node was a Dell PowerEdge 1955 blades with dual socket, dual core 3.2 GHz CPUs, 8 GB of RAM per blade, and two 36 GB 2.5” SAS hard drives. For this demonstration we used one of these compute nodes to launch VM jobs onto eight other compute nodes. The VM images were Debian Linux 6.0.4 i386 installations.

Timing was accomplished with a socket-based timing logger running on the job-launch server so that there was no network-induced clock skew. The overhead of sending very short socket messages to the timing logger is much smaller than the precision of network time protocol (NTP) synchronized servers.

Figure 4 depicts the launch timing test results. Each of the eight lines in the graph shows the launch times of the VM jobs.

The individual lines are for optimized or unoptimized Debian VM images and the number of VMs that were launched per jobslot. For example, Unopt-4 is the curve for an unoptimized Debian image with 4 VMs per jobslot. The x-axis of the graph shows the number of scheduler jobs that were launched as a job array. Multiplying the number of scheduler jobs times the number of VMs per jobslot is the total number of VMs that were launched. For instance for the Unopt-4 series of launches, a total of 4, 8, 16, 32, 64 VMs were launched to generate the Unopt-4 line. Each of the 40 scenarios depicted in the graph were launched 10 times, and the value in the graph is the average of the 10 launch times. The timing is from when the Grid Engine job was submitted from the launch node command line to when the VM job began executing a simple computational application (adding two numbers).

The results show that the stripped down VM operating system images dramatically reduced the launch time for both single VM job launches and high density VM job launches. The optimization effort brings the launch times down to launch latencies that are fairly reasonable. We are in the process of running more trials various number of total VMs and VMs per compute node to get better granularity of our results.

Furthermore, we see that the launch times of the VM jobs stays relatively flat when launching 1 to 8 jobs. This is because the Grid Engine scheduler is spreading the jobs across each of the eight nodes in the cluster. When launching 8 jobs, there are some bottlenecks in the network when delivering the VM images to the nodes for execution. These bottlenecks in the network are more pronounced when launching 16 jobs because two or more scheduler job slots are loading the VM images from the central storage.

V. Related Work

In the past decade, many papers have been written on various aspects of virtual machines and related subjects such as cloud computing. These papers span topics from the underlying technology; to computational, network, and I/O performance; to best practices for managing virtual machines in an organization. Some of the papers explore the overhead of using virtual machines for HPC users, and it is this topic that is most related to the work in this paper.

A team at Taiwan's National Center for High-Performance Computing, Tainan, developed a hybrid bare-metal/virtualized computing cluster called Formosa3 [8]. They modified the Torque scheduler to interface and manage OpenNebula, which subsequently managed the VMs on the compute nodes (an approach 1 and 2 hybrid with type-2 hypervisor). OpenNebula was responsible for provisioning and deprovisioning VMs on the compute nodes; once a compute node was provisioned, it signaled Torque that the VM was ready to accept a VM job. In their cluster, compute nodes could either run VM jobs or HPC jobs, but they did not allow the two types of jobs to comingle on the same compute node. They included benchmarking of VM job launches and found similar results to our unoptimized results.

The Clemson University School of Computing team used the KVM hypervisor on a cluster for grid computing research [9]. They statically allocated VMs onto compute nodes with the hypervisor running on the compute node operating system (approach 2 with Type 2 hypervisor), and they used the Condor scheduler to launch jobs onto the static VMs. They demonstrated how a virtualized cluster could be a part of a multi-organization grid computing infrastructure and ran a variety of benchmarks. Just like the Formosa3 team, they used stock OS images, and their launch results were similar to our unoptimized results.

The research presented in this paper show that there is merit in optimizing the VM images improves launch times dramatically. Unlike the Formosa3 and Clemson efforts, our hypervisors and VM jobs, are launched directly and on-demand by scheduler scripts, which enables agility in supporting multiple OSs and a wide variety of user needs.

VI. Future Work and Conclusion

This rapidly prototyped effort with LLGrid was just the beginning of how we are considering integrating virtual machines into the software stack of LLGrid and how we could use the capability for specific scenarios where the productivity gained outweighs the I/O performance losses for using virtual machines. We have demonstrated the launch of single VMs as well as multiple independent VMs (also known as job arrays). In the next phase of this work, we will demonstrate the launching of parallel jobs, such as MPI applications, that require interaction between the processes. Also we plan to demonstrate our VM capability for use in prototyping and testing persistent distributed services on LLGrid. This will include integration with a database-backed dynamic domain name service (D-DNS) for dynamic host-IP address lookups. Soon we also intend to exploit this capability to enable the work of other research teams at Lincoln.

While using virtual machine technology in HPC does not have all of the advantages that it does in the enterprise computing arena, there are scenarios in the productivity/performance trade-offs are beneficial. We have demonstrated a very flexible approach to integrating virtual machine jobs amongst traditional batch and interactive HPC jobs, while providing reasonable VM job launch latencies.

References

[1] F.J. Corbató and V.A. Vyssotsky, "Introduction and overview of the Multics system," AFIPS, 1965.

[2] R. Dittner and D. Rule Jr., *Best Damn Server Virtualization Book Period*, Syngress, 2007.

[3] A. Silbershatz, P.B. Galvin and G. Gagne, *Operating System Concepts*, Addison Wesley, 2011.

[4] P. Luszczek, E. Meek, S. Moore, D. Terpstra, V. Weaver, J. Dongarra, "Evaluation of the HPC Challenge benchmarks in virtualized environments," *6th Workshop on Virtualization in High-Performance Cloud Computing*, Bordeaux, France, August 30, 2011.

[5] N. Travinin Bliss, R. Bond, J. Kepner, H. Kim, and A. Reuther, "Interactive grid computing at Lincoln Laboratory," *Lincoln Laboratory Journal*, Vol. 16, Number 1, 2006.

[6] Mark F. Mergen, Volkmar Uhlig, Orran Krieger, and Jimi Xenidis. "Virtualization for high-performance computing." *SIGOPS Oper. Syst. Rev.* 40, 2 (April 2006), 8-11.

[7] Wei Huang, Jiuxing Liu, Bulent Abali, and Dhabaleswar K. Panda. "A case for high performance computing with virtual machines. In *Proceedings of the 20th annual international conference on Supercomputing* (ICS '06). ACM, New York, NY, USA, 125-134.

[8] C.H. Li, T.M. Chen, Y.C. Chen, and S.T. Wang, "Formosa3: A cloud-enabled HPC cluster in NCHC," World Academy of Science, Engineering, and Technology Journal, Vol. 73, No. 38, 2011.

[9] M. Fenn, M.A. Murphy, S. Goasguen, "A study of a KVM-based Cluster for Grid Computing," In *Proceedings of the 47th Annual Southeast Regional Conference* (ACM-SE 47). ACM, New York, NY, USA, Article 34 , 6 pages.