

System-Level Virtualization for High Performance Computing *

Geoffroy Vallée
Oak Ridge National Laboratory
Oak Ridge, TN 37830, USA
valleegr@ornl.gov

Christian Engelmann
Oak Ridge National Laboratory
Oak Ridge, TN 37830, USA
engelmannc@ornl.gov

Thomas Naughton
Oak Ridge National Laboratory
Oak Ridge, TN 37830, USA
naughtont@ornl.gov

Hong Ong
Oak Ridge National Laboratory
Oak Ridge, TN 37830, USA
hongong@ornl.gov

Stephen L. Scott
Oak Ridge National Laboratory
Oak Ridge, TN 37830, USA
scottsl@ornl.gov

Abstract

System-level virtualization has been a research topic since the 70's but regained popularity during the past few years because of the availability of efficient solution such as Xen and the implementation of hardware support in commodity processors (e.g. Intel-VT, AMD-V).

However, a majority of system-level virtualization projects is guided by the server consolidation market. As a result, current virtualization solutions appear to not be suitable for high performance computing (HPC) which is typically based on large-scale systems. On another hand there is significant interest in exploiting virtual machines (VMs) within HPC for a number of other reasons. By virtualizing the machine, one is able to run a variety of operating systems and environments as needed by the applications. Virtualization allows users to isolate workloads, improving security and reliability. It is also possible to support non-native environments and/or legacy operating environments through virtualization. In addition, it is possible to balance work loads, use migration techniques to relocate applications from failing machines, and isolate fault systems for repair.

This document presents the challenges for the implementation of a system-level virtualization solution for HPC. It also presents a brief survey of the different approaches and

techniques to address these challenges.

1 Introduction

Today several operating systems (OS) are used for high-performance computing (HPC): Linux on clusters, CNK on BlueGene/L [?], and Catamount on Cray [?, ?]. This variety creates a gap between the initial application development and ultimate execution platforms. For example, a developer's workstation may be used with a modest sized cluster for initial development followed by a porting phase to take the code to the HPC platform. That gap introduces an additional cost every time users want to execute an application on a new HPC platform. Also, as HPC systems grow in size and promise greater performance, the rate of failure increases stealing a portion of the increased performance and thus impact an application's time to solution.

One solution to address these issues is to use system-level virtualization. System-level virtualization creates an abstraction of the hardware and executes one or several virtual machines (VMs) on top of this virtualized hardware; in some instances, the virtual machine may also directly access the hardware for performance purposes. Virtualization solutions are today based on the concept of a Virtual Machine Monitor (VMM), also called a hypervisor. The VMM is responsible for the hardware virtualization and execution of VMs on top of the virtualized hardware. The VMM is typically a small operating system that does not include hardware drivers. To access physical resources, the VMM

*ORNL's research sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory (ORNL), managed by UT-Battelle, LLC for the U. S. Department of Energy under Contract No. DE-AC05-00OR22725.

is typically coupled with a standard operating system, such as Linux, which provides device/hardware access.

There are two approaches employed, formalized by Goldberg in the 1970's [?] as: (i) *type-I virtualization* where the VMM and VM run directly on the physical hardware, and (ii) *type-II virtualization* where the VMM and VM run on a host operating system (see Figure ??). Since the type-I

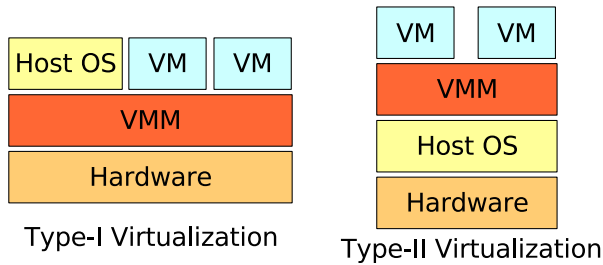


Figure 1. Classification of Virtualization Techniques

virtualization has direct access to resources, performance is comparable to that of native execution. In contrast, type-II virtualization incurs additional overhead due to the layering of the VMM on top of the host OS when servicing resource requests from VMs. The type-II layering makes its approach more suitable for the development phase, where some performance may be reduced in exchange for greater diagnostic and development capabilities.

Today, several system-level virtualization solutions are available, for instance Xen [?] (type-I), QEMU [?] (type-II), or VMWare [?] workstation & server (type-II). However, these solutions are not suitable for HPC because they were not designed to meet the specialized needs of HPC. For instance, Xen has become a rather massive micro-kernel that includes unneeded features for HPC, e.g., a network communication bus; QEMU and VMWare do not support direct access to high-performance network solutions, e.g., InfiniBand.

A system-level virtualization solution for HPC requires only a small set of system services, such as migration, suspend/resume, and checkpoint/restart of VMs. In addition, the solution should afford developers efficient access to resources, a VM adapted scheduling policy, and be lightweight in order to: (a) minimize the system footprint, and (b) guarantee performance prediction and isolation for running VMs. The core of a system-level virtualization solution for HPC is a HPC Hypervisor as shown in Figure ??. Surrounding the core are the five areas of enhancement, which we describe in the remainder of this paper.

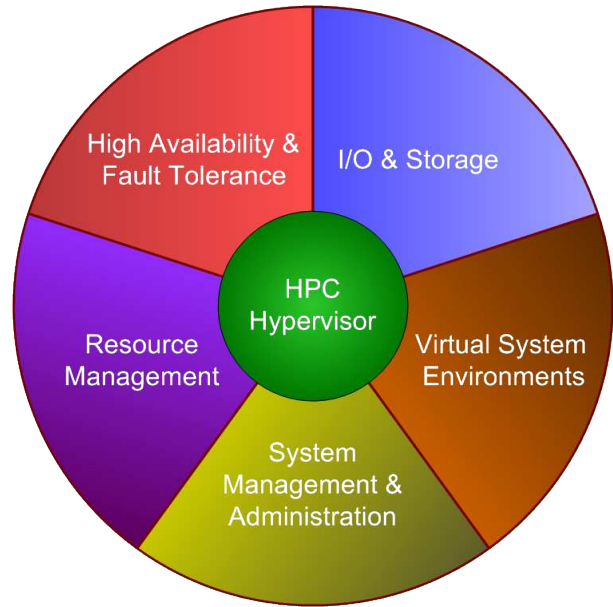


Figure 2. The System-Level Virtualization Components

2 System-Level Virtualization for High Performance Computing

In this section we present the challenges for the implementation of a system-level virtualization solution: a hypervisor for HPC, virtual system environments, high availability and fault tolerance, system management and administration, resource management, and I/O & storage. We also present a brief survey for each of these challenges.

2.1 Hypervisor for High Performance Computing

Current VMM has been initially designed for the server consolidation market. Because of this context and because of the lack of hardware support for virtualization, current VMMs have huge system footprint.

This important system footprint is created by two different factors: (i) the VMM is always coupled to a HostOS which is a full operating system, (ii) for virtualization purpose the VMM store a large dataset (such as the memory map of virtual machines) in memory.

VMM Memory Footprint Typical system-level virtualization solutions typically have a “view” of the virtual machine memory space. This view is used to translate the memory access from the virtual machine to the physical

memory. For instance Xen and KVM [?] implements the concept of *shadow* that shadows a VM's page tables.

Because the VMM keeps a map of the VM's memory space (at least of the page tables, the VMM potentially has a memory footprint of several megabytes. This memory footprint has a direct impact on modern execution platform which are most of the time composed of multicore processors with shared cache. Typically if the core non-shared cache is smaller than the VMM memory footprint, and if the application is memory constrained, cache misses and flushes generated by the VMM execution will interfere directly with the application execution (generating a lot of cache misses and flushes).

In order to address this issue, two approaches are possible: (i) assign the VMM to a specific core and decrease its memory footprint to make it fit in the none-shared memory associated to the core where the VMM is running, (ii) include some hardware support to store part of the VM's "memory map" directly on the hardware. For instance, some AMD processors implemented the concept of *nested page tables* (NPT) [?] in order to eliminate the need for the VMM to implement shadow pages. IO memory management units (IOMMU) are also hardware components that can be used to decrease the memory footprint of the VMM. IOMMUs are typically used to translate memory addresses at the hardware level, in a secure way. Therefore IOMMUs can be used to directly translate memory addresses from the virtual machine memory space to the physical memory space, in a transparent manner for both the VM and the VMM.

System Footprint of the Tuple VMM/HostOS The VMM needs to be coupled to an HostOS which is needed to host the hardware drivers. Everytime an application running inside a virtual machine wants to access the hardware (e.g. for network communications), it has to go be default through the HostOS (which explains why network communications suffer a important overhead, the HostOS being used as a network bridge).

The system footprint of a system-level virtualization solution is therefore not only composed of the system footprint of the VMM itself. The system footprint is actually mostly composed by the HostOS. For instance, using Xen, the HostOS is by default a full desktop Linux distribution (e.g. Fedora Core). Note that this Linux distribution may not even be tuned for HPC, enabling several system services, which at the end create a important operating system noise, critical overhead in large-scale systems [?, ?].

Therefore, in order to achieve high performance computing, the HostOS footprint has to be minimized in order to limit the interference with the application execution. This effort is similar to the standard OS tuning effort for HPC platforms, which includes eviction of unnecessary daemons

and services.

2.2 Virtual System Environments

Current efforts in operating systems (OS) research and development at vendors, such as Cray Inc., and within the DOE Forum to Address Scalable Technology for Runtime and Operating Systems (FAST-OS) concentrate on numerous approaches ranging from custom lightweight solutions, such as Catamount, to scalable Linux variants, like Zep-toOS [?] and Cray's Rainier. In order to address petascale HPC OS issues, such as noise and interference, performance measurement, and profiling, large-scale testbeds for OS research and development are needed. Furthermore, deploying today's HPC systems to production state is time consuming. For example, for an order of ten-teraflop commodity cluster, the delivery-to-production time may range upwards of 6-months; as for more exotic supercomputer class machines, those reaching a few hundred-teraflops, the time to reach stable production status may exceed 12-months or more. Consequently, early evaluation testbeds will help to resolve OS and system software issues before production system deployment.

Another great disappointment is that most scientific applications today are still written to only take advantage of one computer architecture and its specific architectural traits. This means as computer architectures evolve into better/faster machines, we must continually rewrite our scientific applications, further draining funds and personnel that should be put to a better use furthering the science. Similar to OS and system software issues of new systems, early evaluation testbeds can help application scientists port and test their code for the next-generation systems and enable them to assure day-one operation of their codes. Furthermore, it has become clear that there is no one-size-fits-all OS solution. Each OS has its own advantages and targeted scientific applications. Flexibility in OS deployment is needed as these applications need to be able to run on the OS they were designed for.

System-level virtualization technology is capable of providing these testbeds for OS and application development and deployment, and customized system environments for applications by allowing the interchanging of the OS and its runtime environment (RTE) component on demand. One possible solution is to interface system-level virtualization technology with the job and resource management in order to permit specifying the OS and RTE requirements of an application with the actual job submission. At job start, a virtual system environment is installed just for this particular job run. Furthermore, customized OS/RTE combinations may be deployed to fit specific application needs or to enable monitoring and profiling. Performance isolation may be assured by trading off processor cores and by pinning

customized virtual machines to them.

The virtual system environment concept decouples the entire HPC software stack from specific hardware solutions, thus allowing OS and application development and deployment to progress more independently from HPC system deployment. Furthermore, it enables one to fit the OS to an individual scientific application's needs instead of trying to fit scientific applications to specific OS and architectural properties.

HARNESS [?] aims at providing an adaptable virtual machine environment that enables OS/RTE specification in order to fit application needs. However HARNESS is not based on system-level virtualization. Other projects, such as VMPlants [?] allows grid users to define their *workspace*. However, the system administrator cannot enforce any features in the VM, only the user specification is used to create the virtual system environment. It is then very difficult for HPC system administrator to implement local platform usage policies (for instance enforce the use of a checkpoint/restart mechanism for a given class of applications). System administrator should be able to define the constraints on the VMs, based on the usage policies, which have to be "merged" with the virtual system environment definition from the user. Doing so, a single virtual system environment defines both system administrator's and user's constraints.

2.3 High Availability & Fault Tolerance

Large-scale systems are composed of thousands of distributed components, and each of these components is subject to failure. Furthermore, the availability of these large-scale systems is directly dependent on the availability of each of their components.

Two fault tolerance (FT) approaches are traditionally selected to address this issue: *reactive fault handling* and *proactive fault avoidance*. The reactive FT approaches are useful for recovering application state after a failure. On the other hand, a proactive approach to fault tolerance is more suitable to situations where system interruption may be anticipated. Although not all system interruptions are predictable, a certain level of fault prediction is possible by carefully monitoring the workloads and system components. In situations where faults are predictable, substantial performance improvements may be achieved through a proactive approach.

The current approach to implement proactive fault tolerance is at the (i) application, (ii) middleware, or (iii) operating system (OS) layer. The implementation decision usually depends on the degree of transparency. Proactive fault tolerance with system-level virtualization provides the greatest degree of transparency and encompasses all three software layers. Three mechanisms are used to implement proac-

tive fault avoidance mechanisms: virtual machine checkpoint/restart, suspend/resume, and migration. Based on these mechanisms, different policies may be implemented in order to provide proactive fault avoidance in large-scale systems. These policies are also tied to application needs in terms of resource access and execution behavior, and therefore may be different for each application/execution platform.

Current studies focus on proactive fault avoidance rather than reactive fault tolerance. Even if system-level virtualization supports virtual machine checkpoint/restart, the use of this capability for reactive fault tolerance generates important challenges: we already know that process-level checkpoint/restart, solution currently used on large-scale systems, does not scale very well [?], and system-level virtualization because of the important I/O generated by the VM's checkpoint, will aggravate the problem. However, in [?] authors show that system-level virtualization can be used for proactive fault avoidance. In that case, the idea is to migrate VMs from compute nodes where a fault is predicted. However, it is not possible to assume that proactive fault tolerance alone is a suitable solution for system resiliency because all the failures cannot be predicted. Reactive fault tolerance can be coupled with proactive fault tolerance in order to decrease the checkpoint frequency (and therefore decrease the number of I/O operations) but, based on our knowledge, no system current provides this capability. Furthermore, the fault tolerance policy based on system-level virtualization are still very naive, partially because still based on basic fault prediction mechanisms.

2.4 System Management & Administration

In general, a new set of scalable tools is needed for efficient system management and administration for petascale machines. Particularly, scalable tools are needed for configuring and managing virtual system environments, which may consists of a large set of virtual machines comprised of customized OS/RTE. Furthermore, current system policies and procedures need to be adapted to the virtual system environment, including mapping physical machines to virtual machines for accurate resource accounting.

While the introduction of system-level virtualization technology poses a new challenge for scalable system management and administration, existing tools can be revised to handle virtual instead of physical machine instances. For example, the deployment of VMs across a system is similar to a system network boot, and the pre-deployment configuration of VMs is comparable to pre-deployment OS configuration for commodity cluster systems.

In summary, future research and development efforts should focus on adapting existing scalable system manage-

ment and administration tools, and on providing efficient mechanisms for virtualized system environments. Current efforts go in that direction. For instance, OSCAR-V [?] provides an integrated solution for the management of virtual machines but also HostOSes and VMMs. However, the current prototype does not support complex system architecture such as large-system systems which are composed of various node types (e.g. login nodes, I/O nodes and compute nodes).

2.5 Resource Management

With the recent deployment of multi-core processors and the increase of distributed resources in large-scale systems, resource management is a critical topic that must be addressed in order to achieve efficient petascale resource utilization. For example, the deployment of a parallel application within a system composed of thousands of processors, utilizing tens of thousands of cores and each of these associated with banks of memory is quite a complex issue. Batch systems in conjunction with system partitioning are two typical solutions addressing this complex task. Another solution may be via the use of system-level virtualization, which could be used to partition the system and simplify resource exposure to applications.

One approach is to utilize virtual machine (VM) isolation to concurrently execute several VMs on a single processor composed of multiple cores without causing interference between the VMs. In this case, the use of local resources is maximized and the isolation is transparent to applications. This partitions a system into simple computing resources that are exposed to applications through virtual machines, requiring that the VMMs and batch system can effectively map this view of the system to the resource abstractions provided by the VMs.

Another approach directly maps virtual machine resources to the hardware. In this case, to guarantee efficiency, virtual machines should have direct access to resources without interference from the VMM, i.e., *VMM-bypass* [?, ?]. The guest OS running within the VMs is responsible for the management of local resources in order to enable a more efficient application execution. However, it is also possible to take advantage of existing resource management solutions developed for use outside of the virtualization environment.

Resource management is performed at two levels: VM and VMM. One concept to address this issue is to interface virtual environment management with the batch system. Two variants are then possible: (i) deployment of virtual machines on demand, when applications are deployed through the batch system; (ii) deployment of virtual machines before application submission with the application being pushed to virtual machines that fit application needs.

Distributed Virtual Clustering (DVS) [?] is an extension of the MOAB scheduler for automatic deployment of virtual machines at application deployment time. Typically a job is associated to a virtual machine image which is deployed on compute node before the actual application deployment phase. It means that the job submission is decomposed into two phases: (i) the deployment of the virtual machines and (ii) the effective application execution using the standard MOAB capabilities.

2.6 I/O & Storage

Modern computers support a huge collection of I/O devices from numerous vendors with varying programming interfaces. Consequently, the task of providing a system-level virtualization layer that communicates efficiently to all these I/O devices is difficult. Additionally, the I/O devices may contain specialized processors, such as graphics engines and/or signal processors, which aim to move computation away from the main CPU. These I/O devices normally have extremely high performance requirements. This inevitably makes low-overhead I/O devices a critical prerequisite.

I/O device virtualization means that the virtualization layer must be able to communicate with the computer's I/O device using a uniform interface. In the system-level virtualization concept, each I/O request from a *GuestOS* (OS running in a VM) must transfer control to the *HostOS* (OS running in cooperation with the VMM) and then transition through the HostOS's software stack to access the I/O devices. For computing environments with high-performance network and disk subsystems, the resulting overhead can be unacceptably high and do not enable the use of networking solutions for high-performance computing, such as InfiniBand, which uses capabilities such as remote direct memory access (RDMA) [?]. Another problem is that the HostOS (such as a Linux-based VMM) does not provide efficient I/O scheduling and resource management to support performance isolation and quality of service (QoS) to the virtual machines.

Current solutions to these problems focus on advanced techniques, such as *VMM-bypass I/O* and *uniform virtual I/O device interface*. *VMM-bypass I/O* employs an I/O bypass from the VM to communicate directly to the device, which significantly reduces virtualization overhead for I/O devices. The other performance optimization is to export highly specialized virtual devices that do not directly correspond to any existing I/O devices. The VMM in turn intercepts all I/O requests from VMs to virtual devices and maps them to the correct physical I/O devices. This reduces the GuestOS's overhead from communication I/O commands and yields higher performance. It is not clear which solution is more efficient but recent trends in I/O subsystems

indicate hardware support for high performance I/O device virtualization.

The RDMA support in virtualized environment generates challenges slightly different. In order to perform RDMA operations, the sender and the receiver needs first to register the memory. It typically means that some memory will be “shared” between the network card and the process memory space. In a virtualized environment, that implies two points: (i) memory can be “shared” between the virtual machine and the network card (but by default the VMM concept assumes that the virtual machine is completely isolated and that all hardware accesses have to go through the HostOS via the VMM), (ii) everytime the virtual machine access the memory, it is possible to translate memory addresses (physical address space versus virtual machine’s address space).

In short, virtualized I/O devices will need to interface with the VMM to maintain isolation between hardware and software, and ensure VMM support for VM migration and checkpointing. A virtualized I/O device approach must also ensure minimum overhead, allowing the use of VMs for even the most I/O intensive workloads.

In terms of storage, high performance parallel file systems such as Lustre [?], GPFS [?], or PVFS [?] are typically required by the demands of the HPC environment. Along with high performance, these file systems provide an abstraction between logical and physical storage. Because of the diversity of storage and networking technologies, these file systems must in turn constantly deal with interoperability issues associated with the various vendor storage solutions. Thus, there is clearly a need to support system-level storage virtualization providing both a hardware and software level of abstraction. System-level storage virtualization can be loosely defined as creating a uniform view of multiple physical storage solutions networked together and provide a single logical storage view. It enables advanced high-end storage concepts, such as data fail-over between various storage solutions and data staging from permanent storage to temporary intermediate storage to compute nodes. Currently there are still a number of open questions regarding which layer of storage software stack should be virtualized and how to implement it.

3 Impact of System-Level Virtualization for High Performance Computing

If the challenges previously presented can be addressed in a suitable way, system-level virtualization can become a disruptive technology, modifying the way high performance platforms are today used.

System-level virtualization enables capabilities than can modify the way high performance computing platforms are today used. These capabilities include VM live migration,

VM checkpoint/restart, and VM pause/unpause, in a complete transparent manner. It also clear that virtualization and emulation have similar concepts and could be used together in order to extend features we presented in Section ?? to emulated architecture.

Because of that system-level virtualization can be a disruptive technology for HPC programming, HPC application development, HPC system administration and research.

Programming Paradigm High performance computing is facing today critical issues because of the scale of the modern execution platforms. A typical execution platform is composed of thousands of distributed components and the development of applications for such platforms lead to three challenges: How to move data (which is on the storage subsystem) to/from the application? How to parallelize applications to hundreds or even thousand of nodes? How to checkpoint/restart applications in order to guarantee resiliency?

Instead of explicitly expressing communications between processes, like when using MPI, virtual machines can be moved across the execution platform and the application can then assume data is always local.

Furthermore the hybrid programming paradigm (for instance MPI+OpenMP or MPI+PThread), which is a more and more popular solution for the programming of large scale systems, can be extended. Hybrid programming consists of using for the same application both the message passing and the shared memory programming paradigm in order to express different level of parallelism and take a full benefit of local multi-cores or multi-processors capabilities.

Application Development Operating system adaptation instead of application adaptation. Currently the application has to be “ported” everytime it has to be executing on a new execution platform (mostly because of different operating systems and runtime environments). Thanks to system-level virtualization users will be able to define their own execution environments which will be easily deployed on high-performance computing platforms.

Therefore application developers will be able to focus on the science of their application instead of focusing on the execution platform.

System Administration System administrator can isolate the application and its execution environment; they only have to focus on system administration of the virtualization solution (i.e. the VMM and the HostOS).

Furthermore, because virtual machine can be managed independently to the physical platform, software and hardware updates are simpler: (i) virtual machines can be updated offline and effective after a simple redeployment, (ii)

virtual machines can be moved away from nodes for hardware or software update.

Finally because virtual machine is isolated from the physical platform, the HostOS can be used to monitor the virtual machine execution and therefore the application. Thanks to capabilities such as virtual machine migration and checkpoint, it is even possible to have a transparent application execution management. For instance, if the application has an execution time slot of t minutes, the system administrators can setup the system to automatically checkpoint the running virtual machine. Doing so, system administrator can enforce usage policies, without application modifications and without wasting the allocated execution time (the application can be restored later on).

Foster Research and Education System-level virtualization and emulation are two different capabilities: system-level virtualization exposes to VMs the hardware architecture (e.g. x86 or x86.64) whereas emulation exposes to VMs a non-compatible architecture compared to the physical hardware. However, current system-level virtualization solutions have a lot of common concepts and similar implementation. Because of these similarities, it is possible to switch from a specific solution to another, switching from virtualization to emulation. This concept is key for fostering research and education in computer science.

For instance, it has always been difficult to educate student to high-performance computing, even distributed computing or operating systems. With system-level virtualization it is possible to “emulate” and expose a specific view of an hardware platform. For instance, stacking VMs on a single node, it is possible to “simulate” a cluster; emulating a processor it is possible to teach hardware architecture via experimentation even if the real hardware is not available.

Finally, system-level virtualization also fosters research in hardware architecture and operating systems. For hardware architecture research, it is possible to implement an emulator for the next hardware generation, even before having the first sampling; for operating system research, it is possible to deploy research prototypes on execution platform without compromising the hardware.

4 Conclusion

This document presents the challenges for the implementation of a system-level virtualization solution for high performance computing: (i) a hypervisor suitable for HPC (i.e. with a small system footprint), (ii) the support of virtual system environments, (iii) the support of high availability and fault tolerance capabilities, (iv) the support of advanced resource management capabilities, (v) the use of system-level virtualization for resource management, and (vi) the support

of efficient I/O mechanisms and storage solutions for virtualized environment. For that, we identified six domains for which a research effort is or has to be initiated.

It is also clear that no integrated solution is today available to address these challenges. System-level virtualization promises to be a important research topic in operating system for the next few years.

Acknowledgement

Ideas presented in this document are based on discussions with those attending the September 20-21, 2006, Nashville (Tennessee, USA) meeting on the role of virtualization in high performance computing. Attendees included: Stephen L. Scott – meeting chair (Oak Ridge National Laboratory), Barney Maccabe (University of New Mexico), Ron Brightwell (Sandia National Laboratory), Peter A. Dinda (Northwestern University), D.K. Panda (Ohio State University), Christian Engelmann (Oak Ridge National Laboratory), Ada Gavrilovska (Georgia Tech), Geoffroy Vallee (Oak Ridge National Laboratory), Greg Bron-evetsky (Lawrence Livermore National Laboratory), Frank Mueller (North Carolina State University), Dan Stanzione (Arizona State University), Hong Ong (Oak Ridge National Laboratory), Seetharami R. Seelam (University of Texas at El Paso), Chokchai (Box) Leangsuksun (Louisiana Tech University), Sudharshan Vazhkudai (Oak Ridge National Laboratory), David Jackson (Cluster Resources Inc.), and Thomas Naughton (Oak Ridge National Laboratory). We thank them for their time and suggestions for this document.