

Virtualization for High-Performance Computing

Mark F. Mergen, Volkmar Uhlig, Orran Krieger, Jimi Xenidis
IBM T. J. Watson Research Center
Yorktown Heights, NY 10603

{mergen, vuhlig, okrieg, jimix}@us.ibm.com

ABSTRACT

The specific demands of high-performance computing (HPC) often mismatch the assumptions and algorithms provided by legacy operating systems (OS) for common workload mixes. While feature- and application-rich OSes allow for flexible and low-cost hardware configurations, rapid development, and flexible testing and debugging, the mismatch comes at the cost of — oftentimes significant — performance degradation for HPC applications.

The ubiquitous availability of virtualization support in all relevant hardware architectures enables new programming and execution models for HPC applications without losing the comfort and support of existing OS and application environments. In this paper we discuss the trends, motivations, and issues in hardware virtualization with emphasis on their value in HPC environments.

Categories and Subject Descriptors

D.4.7 [Operating Systems]: Organization and Design

1. INTRODUCTION

The requirements of high-performance computing (HPC) on an operating system (OS) significantly differ from typical server and workstation workloads. HPC applications have historically pushed the limits of CPU performance and memory size to run ever-larger problem sizes. Space and time multiplexing — as provided by hardware virtualization — is of no importance to HPC users. In this paper we argue that virtualization provides other benefits like the specialization of HPC operating systems while preserving legacy compatibility.

With virtualization, multiple operating systems can safely coexist on one physical machine. The machine is multiplexed by a small privileged kernel, commonly referred to as a *hypervisor* or *virtual machine monitor* (VMM), which provides the illusion of one or more real machines. While dating back to the early 1970's [5,14], virtual machines lately have had significant attention for server consolidation in data centers. Consolidation of oftentimes highly underutilized servers greatly reduces hardware and maintenance cost. Virtualized devices and live migration of running operating systems decouple software installations from the physical hardware configuration. Coexistence of different versions of operating systems avoids incompatibilities, reduces testing and upgrade costs, and eliminates issues with conflicting software packages.

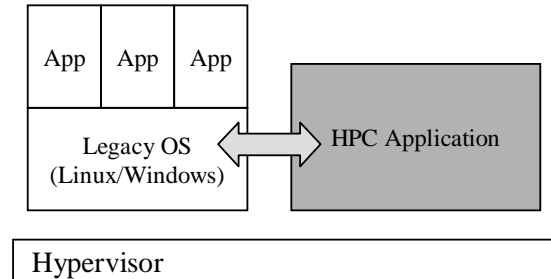


Figure 1: Architectural overview of a coexisting HPC application with a legacy OS

While the consolidation aspect of virtualization is less relevant to HPC workloads, virtualization enables the *specialization* of operating systems with almost full control over hardware resources. The hypervisor safely multiplexes the hardware resources of the physical machine but leaves the specific hardware resource allocation to the operating system in the virtual machine. Multiple, potentially very different operating systems can thereby coexist. These OSes, specialized for particular workloads, are also called library OSes.

This coexistence with minimal interference is the key value of virtualization for HPC. An HPC application can now bypass legacy OS mechanisms and algorithms and can act independently and use hardware-specific optimizations like super-pages. However, the colocation with legacy OSes enables the HPC application to use selected services that are convenient (see Figure 1). The virtual machines can communicate via a low-overhead and low-latency communication mechanism provided by the hypervisor or share parts of the physical memory.

For example, the plethora of hardware devices used in low-cost workstations of cluster environments require a sufficiently diverse number of device drivers. Support on a one-off basis for specialized applications is at least cumbersome if not unfeasible. Here, a colocated legacy OS can serve as a provider for device access with minimal overall resource overhead [13]. HPC applications that require traditional or non-critical OS services, such as access to a file system, debug facilities, or a TCP/IP stack can similarly forward those requests. The PROSE prototype [6] is an example of such a

system structure.

In the remainder of this paper we discuss the benefits, limitations, and usage cases of virtualization for HPC such as security, checkpoint/restart, preemption, introspection and portability.

2. PRODUCTIVITY

Virtualization can enhance productivity in development and testing of HPC applications and systems, in several ways. With authorization, the hypervisor can allow one VM to monitor the state (including memory), interrupts and communications (including calls to the hypervisor) of another VM, for debugging or performance analysis. Because this introspection code runs outside the VM being monitored, it can look at any path in the OS or application, without the limitations that may occur when a debugger is inside, and shares hardware state with, the OS or application being debugged. This may be especially beneficial when developing or using a specialized minimal library OS for a specific HPC application, mentioned above [11].

The hypervisor(s) can provide a virtual cluster of VMs, one for each node in a specific configuration of an HPC application that uses a cluster programming model like MPI. The VMs can be allocated across fewer real nodes, even a single node, to test at realistic scale but with fewer resources. Virtualization allows testing on the machine to be used for production, using only modest resources, simultaneously with production applications that are using most of the resources. Moving an application from testing to production is only a matter of allocating adequate real resources to the virtual nodes (VMs) used to execute the application. Productivity may also be enhanced by using a virtual cluster, running multiple copies of the OS and application, to achieve scaling in an application originally written for a non-scalable OS, avoiding the rewrite for another OS. This was a primary motivation of the Disco system [3].

The decoupling of hardware and operating system can significantly reduce the restart cycle of a system. The “virtual reboot” avoids the latencies of hardware re-initialization by the BIOS [4]. A pre-booted and frozen VM image can be shipped to all nodes in a cluster and significantly reduce the startup time of the system. Similarly, changing a VM image to a different OS and application does not reimpose a hardware initialization latency.

3. PERFORMANCE

Virtualization raises two performance issues: the cost of virtualization itself and the performance benefits it offers. Recent hardware and software developments have been at least partially motivated to reduce the cost of virtualization. The most popular microprocessors, including Intel [9, 10], AMD [1], and IBM Power [7], all have hardware features to support virtualization and reduce its performance cost. Software paravirtualization, used by Xen [2] and IBM hypervisors [8], seeks to present hypervisor interfaces, for example to specify memory address translations, that are functionally equivalent to but less performance-costly to implement than exactly virtualizing the analog hardware, for example page tables. Software pre-virtualization [12] is a technique of semi-automatically annotating OS code so that it

can be adapted for a specific hypervisor at load time but remain compatible to real hardware.

The performance of HPC applications may benefit from virtualization in several ways. Virtualization facilitates specialized OSes that are performance optimized for classes of HPC applications, as discussed above. A hypervisor can guarantee resource allocations to a VM, such as a fixed partition of real memory, a fixed percentage of CPU cycles, or a maximum latency to interrupt handling code for a real time VM. The virtual nodes (VMs) of a virtual cluster can be scheduled to run concurrently, in different real nodes or using different processor cores of one or a few real nodes. This gang scheduling allows a cluster HPC application, while running, to communicate between nodes in real time, as it would without virtualization, which may be crucial to efficient forward progress.

4. RELIABILITY AND AVAILABILITY

Because of VM isolation, hardware or software failures, as in a processor core, memory, OS or application, directly affect only one VM, unless the hypervisor itself suffers a failure. If the affected VM cannot recover itself and truly fails, its non-failing hardware resources can be unambiguously reclaimed by the hypervisor and used in restarting the failed VM or by other VMs. Other VMs are not aware of the failure unless they have a communication dependency on the affected VM, but they may run slower if they shared a failed processor core, for example. This fault isolation enhances system reliability and increases the probability of completion of long-running HPC applications, without any special effort by the OSes that run in the VMs.

When authorized, introspection allows one VM to capture the complete OS and application state of another VM, either on request or periodically. Because this is the state of resources virtualized at a low level, this state can be used to create another VM, with equivalent but different real resources in the same or another real node, and continue processing. This checkpoint/restart capability enables preemption by high-priority work, inter-node migration of work in a cluster for load balancing, and restart from a previous checkpoint after transient hardware or software failures, if the application is idempotent with respect to such a restart. Recovery from hardware failures is particularly important for very long-running computations.

Preemption allows new uses, such as what may be called real-time HPC, where a large number of nodes are preempted for a brief time to compute a result needed immediately. All these scenarios enhance system availability, require little or no effort in the OS or application and are important to HPC applications, because they prevent loss of progress in long-running HPC applications.

5. SECURITY

The VM isolation and introspection properties previously discussed provide a platform for building secure systems. If the hypervisor is small because it implements only a few abstractions and services, then it may be possible to offer strong evidence that it correctly implements a specification. Then the VM isolation and authorized introspection properties can be trusted. An isolated VM can have no unautho-

rized interaction with other non-hypervisor software running on the real machine. If a trusted program is loaded into an isolated VM by the hypervisor, then its defined communications with other software authorized to communicate with it can be trusted. Another VM, if authorized to do so, can use introspection to audit the state of a VM, for example to look for viruses. Introspection can also be used to continuously monitor the communications and state of a VM, to verify independently its correct operation. Secure systems are very important for HPC applications that process classified government data.

6. SOFTWARE COMPLEXITY

Hypervisor-based systems offer the promise of a fundamental restructuring of system software that can reduce the complexity of software in development, testing, distribution and maintenance. This promise, however, rests on the often-unstated assumption that the hypervisor will be much more stable than the usual OS, with very infrequent new versions or interface changes, because it implements such a small set of abstractions and services.

A list of simplifications is possible. Since only the hypervisor bootstraps on real hardware, the related hardware configuration and initialization need only be done once rather than every time a different OS is started. Since most devices are virtualized, only one real driver is needed for each device type and most OSes only need to implement generic virtual drivers that communicate with the real drivers. The real drivers can be supplied by an OS such as Linux or a library OS designed specifically to support drivers, running in a service VM.

Because a hypervisor allows several different OSes and versions of them to concurrently share the resources of a real system, each application or middleware program can be developed for just one OS and version deemed most suitable, rather than for every OS that a user might want for other reasons. This has the potential to dramatically reduce the effort and cost of developing and testing software.

The application or middleware program can be packaged with the OS it was tested with, for distribution as a single unit. The user installs the combined package and doesn't need to worry about version compatibility between the OS and application or middleware. OS fixes apply only to specific combined packages and can't cause unintended side effects to other software that uses the same OS but in a different package. User maintenance costs should decline, software should become more stable and this stability should lead to improved software quality.

7. RESEARCH QUESTIONS

Virtual machine systems have existed for many years and microkernel systems, which are somewhat similar, have also been extensively studied. However, many interesting and important questions remain worthy of serious effort and offer opportunity for hardware and software innovation, as the following small selection may indicate.

- What are the appropriate minimal abstractions and services that a hypervisor should implement in the

most privileged state of the machine to achieve minimal performance cost, secure sharing, VM isolation, policy enforcement only, no resource exhaustion vulnerability and a basis for tractable reasoning about correctness?

- Can a hypervisor serve as a hardware abstraction layer, with platform independent abstractions and services encapsulating all or most platform dependencies, requiring a hypervisor rewrite to move to a different microprocessor but only recompilation of other software, while providing VMs and their OSes with sufficiently low-level control of resources and performance?
- Is it possible for a hypervisor to provide specialized lightweight virtual machines for functions such as device drivers, with lower context switching and inter-VM communication costs than full-function VMs, to reduce significantly the overall cost of virtualization?
- Does virtualization make it possible to implement a hybrid full-function OS as a combination of two pieces: a small library OS that provides an optimized subset of performance-critical services for a class of HPC applications and runs in the VM with the application, plus a full-function OS that runs in a separate VM and uses introspection of the first VM to provide all other non-performance-critical services?
- What are the core primitives and important capabilities a hypervisor has to support for HPC (and is it feasible to upgrade an existing hypervisor like Xen), such as large pages, deterministic scheduling, and direct access to high-performance self-virtualizing devices such as Infiniband, while preserving virtualization properties primarily targeted towards server-class workloads, such as preemption and migration?
- What additional hardware features would improve the performance or functionality of virtualization, especially in support of the goals implied in some of the previous research questions?

8. CONCLUSION

Virtualization, that is increasingly attractive for commercial systems, as implemented by a small hypervisor that runs below the usual OS layer, has the same potential to benefit HPC applications in the dimensions of flexible OS variety, productivity, performance, reliability, availability, security and simplicity. A rich of agenda of research activities remains to enable realization of these potential benefits.

9. REFERENCES

- [1] Advanced Micro Devices. *AMD64 Virtualization Codenamed "Pacifica" Technology, Secure Virtual Machine Architecture Reference Manual*, May 2005.
- [2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, et al. Xen and the art of virtualization. In *Proc. of the 19th ACM Symposium on Operating System Principles*, Bolton Landing, NY, October 2003.
- [3] Edouard Bugnion, Scott Devine, and Mendel Rosenblum. DISCO: Running commodity operating

- systems on scalable multiprocessors. In *Proc. of the 16th ACM Symposium on Operating System Principles*, pages 143–156, 1997.
- [4] Sung-Eun Choi, Erik A. Hendriks, Ronald Minnich, Matthew J. Sottile, and Aaron Marks. Life with ed: A case study of a linuxBIOS/BProc cluster. In *HPCS*, pages 35–41, 2002.
 - [5] Robert P. Goldberg. Survey of virtual machine research. *IEEE Computer Magazine*, 7(6), 1974.
 - [6] Eric Van Hensbergen. P.R.O.S.E. partitioned reliable operating system environment. Submitted to *ACM Operating System Review*, April 2006.
 - [7] IBM. *PowerPC Operating Environment Architecture, Book III*, 2005.
 - [8] IBM Research, <http://www.research.ibm.com/hypervisor/>. *The Research Hypervisor*.
 - [9] Intel Corp. *Intel Vanderpool Technology for IA-32 Processors (VT-x) Preliminary Specification*, 2005.
 - [10] Intel Corp. *Intel Vanderpool Technology for Intel Itanium Architecture (VT-i) Preliminary Specification*, 2005.
 - [11] Samuel T. King, George W. Dunlap, and Peter M. Chen. Debugging operating systems with time-traveling virtual machines. In *Proceedings of the USENIX Annual Technical Conference (USENIX'05)*, April 2005.
 - [12] Joshua LeVasseur, Volkmar Uhlig, Matthew Chapman, Peter Chubb, Ben Leslie, and Gernot Heiser. Pre-virtualization: Slashing the cost of virtualization. Technical Report 2005-30, Fakultät für Informatik, Universität Karlsruhe (TH), November 2005.
 - [13] Joshua LeVasseur, Volkmar Uhlig, Jan Stoess, and Stefan Götz. Unmodified device driver reuse and improved system dependability via virtual machines. In *Proc. of the 6th Symposium on Operating Systems Design and Implementation*, San Francisco, CA, December 2004.
 - [14] Gerald J. Popek and Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. In *Proc. of the Fourth Symposium on Operating System Principles*, Yorktown Heights, New York, October 1973.