What Is the Right Balance for Performance and Isolation with Virtualization in HPC?*

Thomas Naughton[1,2,**], Garry Smith[2], Christian Engelmann[1], Geoffroy Vallée[1], Ferrol Aderholdt[3], and Stephen L. Scott[1,3]

[1] Oak Ridge National Laboratory,
Computer Science and Mathematics Division,
Oak Ridge, TN 37831, USA
naughtont@ornl.gov

[2] The University of Reading
Reading, RG6 6AH, UK

[3] Tennessee Tech University
Computer Science
Cookville, TN, 38505, USA

Abstract. The use of virtualization in high-performance computing (HPC) has been suggested as a means to provide tailored services and added functionality that many users expect from full-featured Linux cluster environments. While the use of virtual machines in HPC can offer several benefits, maintaining performance is a crucial factor. In some instances performance criteria are placed above isolation properties and selective relaxation of isolation for performance is an important characteristic when considering resilience for HPC environments employing virtualization.

In this paper we consider some of the factors associated with balancing performance and isolation in configurations that employ virtual machines. In this context, we propose a classification of errors based on the concept of "error zones", as well as a detailed analysis of the trade-offs between resilience and performance based on the level of isolation provided by virtualization solutions. Finally, the results from a set of experiments are presented, that use different virtualization solutions, and in doing so allow further elucidation of the topic.

1 Introduction

As high-performance computing (HPC) systems increase in size and complexity, the associated system software faces new challenges to balance performance, usability and robustness. The use of virtualization in HPC has gained attention in recent years [4,9,13,15,20,21,23,24,6], mainly for enabling isolation, customization and resilience abilities. The benefit of having a user-customized execution environment is one advantage [5,21,23]. Also, the ability to provide increased functionality without having

* The submitted manuscript has been authored by a contractor of the U.S. Government under Contract No. DE-AC05-00OR22725. Accordingly, the U.S. Government retains a non-exclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

** Corresponding author.

to require this in all instances is another use case [4,20]. For example, microkernels have been used on several supercomputers to achieve minimal system-level interference [20], i.e., "*the OS should stay out of the way*". Adding the ability to load virtual machines (VMs) that run more feature rich operating system (OS) environments is one way to balance usability and performance, while maintaining the ability to run natively on the microkernel to achieve the full performance potential [20]. In the context of system resilience, virtualization enables both advanced reactive and pro-active policies by providing capabilities such as VM migration and checkpoint/restart [22]. Furthermore, virtualization has been leveraged for the design and implementation of fault injection techniques in order to study the impact of failures on the execution of scientific simulations [17,16]. These capabilities usually rely on isolation characteristics of virtualization solutions, i.e., isolation decouples the management of resources exposed within the VM from the physical resources, making the VMs independent from the host on which they run.

The HPC community recently introduced the concept of *enclave* as an operating and runtime system design characteristic for addressing current scalability, resilience and performance limitations at extreme scale. For instance, the Hobbes project, which aims at designing operating system/runtime (OS/R) interfaces for extreme-scale systems [4], defines an enclave as *"a partition of the system allocated to a single application or service"* and has proposed a design based on system-level virtualization. Figure 1 shows a diagram of the proposed Hobbes software architecture. The project is focused on techniques to support composition primitives to aid applications and leverages system-level virtualization to provide flexible support for additional OS/R functionality. Resilience is one of the cross-cutting concerns the Hobbes project is seeking to address. This includes work on developing resilience building blocks as well as work to experiment with error management in system software. As new OS/R interfaces are developed, the robustness of the overall system will be probed to identify areas for improvement.

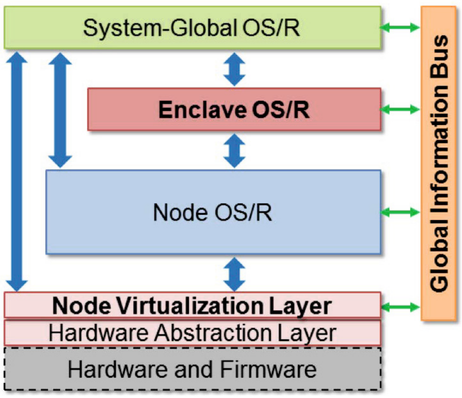


Fig. 1. Hobbes software components for Extreme-Scale OS/R [4]. The component API interactions are reflected by vertical arrows and data exchanges via horizontal arrows; research targets are shown in **bold**.

As HPC software stacks begin to leverage virtualization, questions emerge about what degree of isolation should be maintained to keep applications running efficiently without overly sacrificing fault management primitives (i.e., isolation mechanisms). Since virtualization is an important component of the OS/R research for next generation system software, we consider *the trade-offs and perspectives for balancing performance and isolation* in this context.

The primary contributions of this paper are to study the balance for performance and isolation in the context of HPC resilience. The examination contributes to the more general topic of error models in HPC. The paper discusses: (i) a classification of errors in the context of HPC resilience and HPC virtualization; (ii) an analysis of the trade-offs between performance and isolation for HPC workloads and its impact on system resilience, especially in the context of the Hobbes project; and (iii) experiments that demonstrate variations in the effects of synthetic errors in virtualized environments.

The remainder of the paper is organized as follows, in Section 2 we review related work and provide background information on the topic of virtualization in HPC. In Section 3 we analyze the problem of balancing performance and isolation with VM-based HPC environments, followed by an evaluation of an example error scenario using different virtualization solutions in Section 4. Finally we conclude in Section 5.

2 Background

The *Palacios* virtual machine monitor (VMM) was developed from scratch by Northwestern University (NU) and University of New Mexico (UNM), in cooperation with Sandia National Laboratories (SNL) [18,13]. The Palacios VMM can be run on the *Kitten* microkernel [11,13] and as a loadable kernel module on Linux based systems. The VMM requires the hardware to support virtualization extensions, e.g., AMD-V and Intel VT. Palacios runs on standard x86 commodity clusters and Cray XT 4/5 & XK6/7 supercomputers. Palacios guest VMs can run either 32-bit or 64-bit OS kernels.

The *QEMU* tool is a machine emulator [3], which is distinct from a type-II VMM proper [7] because it may emulate non-native architectures to the guest. For example, the native host architecture might be x86 but the guest virtual machine (VM) could see an ARM or MIPS architecture in the emulated environment. However, the distinction between emulator and virtualization as defined by Goldberg [7] is less important for the the current context. QEMU provides a rich set of features for interfacing with the machine monitor and an embedded debugger. These capabilities make it a common component in OS development environments. QEMU can be combined with the Linux *Kernel-based Virtual Machine (KVM)* to accelerate the virtual machine execution [14]. This removes the emulation capabilities of QEMU and requires the VM and host architectures match. It does however provide the rich frontend and debugger capabilities of QEMU for use with the kernel-implemented KVM backend. KVM runs on x86 machines and requires hardware supported virtualization extensions, e.g., AMD-V and Intel VT. The widespread use of Linux has led to KVM being widely used for type-II virtualization due to its seamless integration with the OS distributions.

The presence of virtualization in HPC be limited and virtual machines may not be supported on all systems. However, virtualization is becoming more widely available and is slowly becoming just another system software feature, especially for environments that use Linux as their base operating system. The Palacios VMM was specifically developed with HPC environments as a target and is under active development for use on Cray supercomputers, e.g., Cray XK7. The KVM VMM is a standard component in modern Linux distributions. Additionally, the QEMU environment can be run entirely in user-space without the need for advanced permissions (i.e., does not require `root` privileges), albeit at a lower performance level if acceleration like KVM or `kqemu` is not used.

3 Analysis

Virtualization. The management of unprivileged *user-space* and privileged *kernel-space* is a standard approach for OS protection. For example, on x86-based systems the hardware offers protection domains, or "rings" (Figure 2), that can be used to enforce this user/kernel separation [10]. System-level virtualization extends the protection layers by adjusting the protection domains to have the virtual machine monitor run at the highest protection level in order to marshal access to physical resources [19]. The result is a guest/host (virtual/native) separation, which places the standard OS user/kernel within a virtual region, i.e., a guest VM. The guest VM runs the OS and the VMM provides a software layer between the guest OS and the physical resources. While overly simplified, this description captures the divisions used to provide stronger isolation between the guest environment (virtual) from the host environment (native).

As mentioned previously, the OS/R marshals access to hardware resources, e.g., CPU, main memory, network interfaces and I/O (storage) devices. The overhead for providing protection is often governed by whether the isolation mechanism is hardware or software based. For example, the cost for unprivileged CPU instructions is equivalent for guest/user and host/user. When managing memory, if there is no hardware-level support (e.g., nested page tables), then shadow pages must be managed in software. The CPU and memory resources are fairly well supported at this stage with hardware level protection mechanisms. In contrast, the multiplexing of network and I/O devices for virtualized environment often requires software based methods. New technology

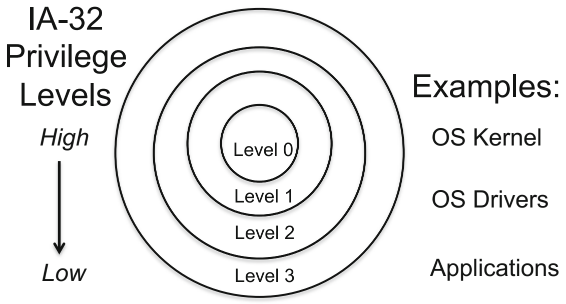
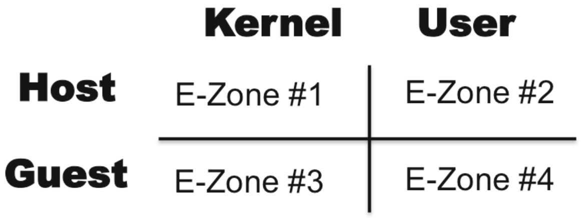


Fig. 2. Intel Architecture (IA-32) hardware supported protection mechanisms provide four privilege levels: high (Level 0) e.g., OS, low (Level 3) e.g., Applications [10]

like Single Root I/O Virtualization (SR-IOV) and IOMMU should help decrease this overhead as they become more widely available.

Resilience. Resilience is an important aspect for HPC systems and is a cross-cutting topic in the Hobbes project. To be clear, we begin by briefly reviewing a bit of resilience terminology from Laprie et al. [1]. A *fault* is a defect that exists and may be "active" or "dormant", and an "active fault" is an *error*. An error that is not contained, e.g., resulting in service interruption, creates a *failure*. There are numerous potential faults, which can originate at different phases: design, implementation, operation, etc. An error model provides an abstraction of the potential faults that may occur and offers structure to help understand and reason about erroneous behavior in a system [8].

Virtualization and Resilience. The kernel/user and guest/host structure described previously provides different *error zones* ("E-Zones") as shown in Figure 3. For example, in a non-virtualized setting, errors in E-Zone#1 (host/kernel) are often considered fatal, and can result in the entire system being compromised or crashing. In contrast, errors in E-Zone#2 (host/user) may be fatal but would (generally) only affect the victim process and should never crash the full system.



	Kernel	User
Host	E-Zone #1	E-Zone #2
Guest	E-Zone #3	E-Zone #4

Fig. 3. Error zones for standard and virtualized systems

Adding virtualization to the system offers additional error zones, where the kernel/user separation is enhanced by incorporating an additional layer for protection (i.e., guest/host). This enables additional isolation to be introduced for what would normally be privileged execution (i.e., guest kernel). In Figure 3, errors in E-Zone#4 (guest/user) are generally assumed to be equivalent to those of E-Zone#2 (host/user). However, the errors in E-Zone#3 (guest/kernel) may have different expected behavior. The effects of errors may have very broad or very limited impact. In the case of "limited effects", the isolation should limit the impact to the victim in the error zone. When errors may have a broader impact, e.g., the entire node, the effects of errors can result in failures for the full system (i.e., "global effects"). If the effects of errors are more disparate then the associated management will be mixed and errors will have "varied effects". Restated, the policy for how to manage the errors will vary.

In summary, the error model for this virtualization-enabled context (Figure 3) is:

E-Zone#1 – errors may crash full system (global effects)
E-Zone#2 – errors may crash individual victim (limited effects)
E-Zone#3 – errors may crash full system (global effects), *or*
errors may be managed in mixed manner (varied effects), *or*
errors may crash individual victim (limited effects)
E-Zone#4 – errors may crash individual victim (limited effects)

The reason for the variation in the error model for E-Zone#3 instances is mainly due to the balance between isolation and performance. Figure 4 illustrates this isolation/performance continuum. In some instances, strong isolation is the most critical criteria and may outweigh performance criteria. For example, if there are many virtual machines on a single host (i.e., web hosting platforms with direct user access) the protection against a user crashing a virtual machine and indirectly crashing the host (and all other users) is more critical than individual user performance. In contrast, more relaxed isolation may be appropriate if performance is important and reasonable protection mechanisms can be used that offer an acceptable level of control.

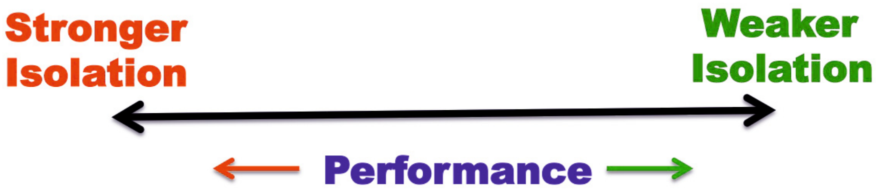


Fig. 4. Performance / Isolation continuum

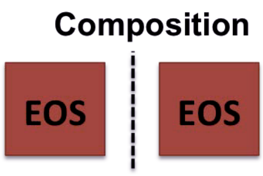
Notice that in the case that strong isolation is the highest criteria, then E-Zone#3 is equivalent to E-Zone#4 in order to limit the effects of errors. Additionally, if performance is the most important criteria (even at the cost of isolation) then E-Zone#3 can devolve to being equivalent to E-Zone#1, where errors have a global effect. As such, a guest/kernel error could crash the full system and have an impact on the entire node (global effects).

The case where E-Zone#3 $\equiv$ E-Zone#1 is still (potentially) a valid case in some HPC contexts. For example, one motivation mentioned previously for virtualization in HPC was to provide additional functionality beyond what is provided by the default microkernel case. In this scenario, virtualization enables users to run full-featured legacy OS instances in VMs on top of the native microkernel OS [20]. Therefore, this error model (E-Zone#3 $\equiv$ E-Zone#1) may be appropriate as the objective is not added protection, but added functionality.

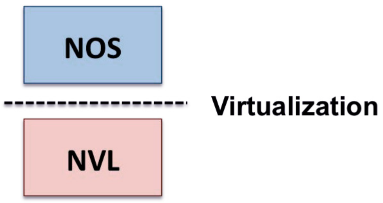
Hobbes Context. The Hobbes project is interested in system software for next-generation systems. This research includes work in designing OS/R interfaces for large-scale HPC systems. There are two key distinguishing elements to the project: (i) enclaves and (ii) composition. An enclave is a partitioned region given to a particular application or service [2,4]. Enclaves house applications and therefore will be composed to form more complex application instances [4]. Virtualization can be used to implement this partitioning and isolation between enclaves. Therefore, enclave composition will require selectively relaxing the isolation to accommodate the required interactions between the OS instances, which are shown in Figure 1 [4].

An important step in the resilience effort for the Hobbes project, and HPC in general, is to begin to refine the error models. As such, we begin by presenting our initial thoughts on an error model that takes into consideration the distinguishing elements of Hobbes, namely: enclaves and composition.

Based on this, an important question for the Hobbes project will be to identify instances where the E-Zone#3 model can employ mixed policies for error management. This is likely to be influenced by the performance cost for implementing the isolation. Additionally, in the Hobbes context the E-Zone#3 policies will be influenced by enclave composition. For example, consider the previously noted case where the objective is to offer increased functionality via virtualization, so E-Zone#3 $\equiv$ E-Zone#1 may be appropriate. When composing different enclaves, this may be less straightforward due to cross-enclave dependencies that should limit indirect effects. Restated, crashing a single enclave (VM) may be acceptable but if there are multiple enclaves (VMs) then the isolation may need to be increased to avoid indirectly affecting another instance. Figure 5 illustrates the two axes that influence the policies for E-Zones in the Hobbes context. In Figure 5(a), two Enclave OS (EOS) instances are shown and reflect the relationship during enclave composition. The other case that will influence the balance of isolation is depicted in Figure 5(b), which shows the vertical relationship between the Node Virtualization Layer (NVL) and Node OS (NOS).



(a) Enclave Composition



(b) Virtualization

Fig. 5. Diagrams showing two key building blocks in the Hobbes design: (a) enclave composition, and (b) virtualization

4 Evaluation

We performed a set of tests to demonstrate the viability of leveraging VMs to improve protection during fault-injection experiments. The experiments presented in this section demonstrate the following: (i) host OS system log (syslog) monitor to notify when violations occur in a Palacios guest VM based on guest OS heartbeats; (ii) demonstration of VM isolation to avoid corruption of host context.

4.1 Setup

The host OS was Linux v3.5.0. KVM was version v1.0+noroms-0ubuntu13, which was installed from the Ubuntu 12.04 LTS packages. QEMU was v1.7.0, which was installed from source. Palacios was Git clone 619573f (11-mar-2014) with minor patches for compilation issues under Linux v3.5.0. The KVM and Palacios VMMs are implemented as loadable kernel modules and were loaded exclusively, i.e., only one module was active during the tests. The QEMU VMM was run without the acceleration support and ran entirely as a user-space application.

The guest OS used a Linux v2.6.33.7 kernel with Busybox v1.20 to create a very small system installation. In the Palacios case, the guest VM is configured to use shadow paging with a Linux virtio-based NIC that is bridged via the VMM to the host network interface. The KVM and QEMU cases did not have networking enabled during the tests.

All tests were performed on a Linux cluster (*SAL9000*) at ORNL. The machine was configured with a dual-bonded 1 Gbps Ethernet interconnect. The 40 compute nodes each have 2 AMD64 CPU (24 cores), a Nvidia Tesla S2050 GPU, 64 GB of memory, and run a Ubuntu Linux 12.04 LTS operating system. The guest OS startup script files were updated to automatically load a custom fault-injection (FI) kernel module (described below). The guest OS configuration was identical in all tests, as was the host OS configuration. The only differences between test runs were related to the virtualization solution being used. The experiments were all run on a single compute node (node40) of the SAL9000 cluster.

4.2 Guest OS Errors

The experiments use different virtualization solutions to demonstrate the isolation provided by VMs. The three virtualization implementations used during these tests were: QEMU [3], KVM [19,14], and Palacios [13,18].

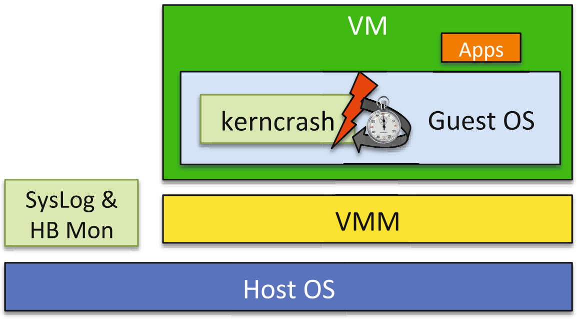


Fig. 6. Diagram showing the VM+FI harness with an OS Kernel crash running in the Guest VM, isolated from the host context managing the experiment

A custom kernel module (`kerncrash`) was written to intentionally cause a failure when a timer expired. The kernel module would perform a divide-by-zero error that forced a fatal exception that would crash the kernel. When the module is loaded into the kernel it sets a timer and after N seconds the error occurs. The tests used manual inspection to confirm that the kernel crashed based on console output, which could also have been obtained from system logs if the VM was forwarding the syslogs over the network or to the host via a virtual serial log. The integrity of the host was determined by whether or not the host was responsive after the crash for running further tests. Figure 6 depicts the self-injected error in the guest OS.

4.3 Testing

The experiments that introduced guest OS errors show the benefits of running with strong isolation. The fatal guest OS experiments were each run 5 times on a single node of the SAL9000 cluster. As expected, the `kerncrash` resulted in a fatal guest kernel error in all instances. The KVM and QEMU test cases offered a clear separation with no change in the host after the guest OS's kernel crash. In the Palacios case the results were mixed. When run on SAL9000, the host became unstable soon after the guest crashed and required a reboot to resolve the issue. At the time of writing, the root cause for this error has not been determined but additional testing on another development machine indicated that the problem might be due to some shared networking used by the Palacios guest. The preliminary investigation indicated that this might be related to the virtual network bridge that Palacios establishes between the host and guest to provide network access in the guest VM. Subsequent experiments on a different machine resulted in similar disruptions of the host environment and showed a Palacios-related kernel thread that was responsible for much of the load (a network related routine running as Linux kernel thread in host OS). This may be a host-level misconfiguration or more simply an implementation bug.

4.4 Discussion and Observations

The use of virtualization potentially offers good separation between the target and control harness. The encapsulation of the VM is helpful for repeating experiments, and can be used to capture output and monitor the guest environment. Also, the reproducibility is useful with benchmarking (both performance and resilience) to ensure consistent machine configurations. The use of virtualization for OS-level targets provides useful support for isolating error studies involving corruption (e.g., soft error fault injection). VMs also enable over-subscription of the native resources, so a single physical machine can be used to run multiple tests concurrently or in series, which can have entirely different configurations.

In the low-level system software use case, the fact that the exact same environment with the exact same OS and synthetic bug could be reproduced on the same hardware platform is a clear benefit of using VMs for resilience investigations. It also allows for repeatable research and low-level debugging capabilities, e.g., VMM embedded debugger in QEMU. The issues raised during testing with Palacios and host/guest sharing highlight a more fundamental point to consider when working with virtualization and

HPC. There are many instances where the isolation properties are intentionally relaxed to gain performance. For example, the host network interface might be directly mapped into the guest OS to provide near native performance from within the guest environment. This might also be due to the fact that the network device is not easily virtualizable, i.e., not able to multiplex between the host and guest OS. These are factors that must be considered when using VMs in a HPC context for performance reasons and they are factors that must be managed when used for resilience investigations.

5 Conclusion

This paper discussed details associated with resilience for virtualization-based HPC systems. In this context, we propose a new error model as well as an initial evaluation. Thus, our contributions are (i) a classification of the various errors, (ii) an analysis of the resilience/isolation trade-offs, and (iii) a set of experiments to elucidate the discussion.

The proposed classification is based on the distinctions between four “error zones” and different scenarios were outlined to illustrate the applicability of the concept to HPC. Experiments were performed that reflected three different data points along the isolation/performance continuum: QEMU, KVM, and Palacios. Finally, we presented an analysis of how the concept of error zones can be used in the context of the Hobbes project, which aims at developing OS/R interfaces for extreme-scale systems. More precisely, we analyzed the impact of failures on the overall Hobbes’ architecture, especially on the composition capability. Ultimately, this study provides input to help respond to questions about errors in HPC environments, and more specifically in cases where virtualization is used.

The current focus has been to refine the error models to provide structure to guide the research into resilience, which can be beneficial for the Hobbes project. In future work, as the OS/R interfaces for the Hobbes software stack (Figure 1) are published, we plan to perform robustness testing [12] on the APIs. The intent is to identify any weakness in the interfaces and offer feedback for improvements in system-level resilience.

References

1. Avižienis, A., Laprie, J.C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing (TDSC) 1(1), 11–33 (2004), `http://dx.doi.org/10.1109/TDSC.2004.2`
2. Beckman, P., Brightwell, R., de Supinski, B.R., Gokhale, M., Hofmeyr, S., Krishnamoorthy, S., Lang, M., Maccabe, B., Shalf, J., Snir, M.: Exascale Operating Systems and Runtime Software Report. Tech. rep., U. S. Department of Energy (December 28, 2012)
3. Bellard, F.: QEMU, a fast and portable dynamic translator. In: USENIX 2005 Annual Technical Conference. Anaheim, CA, USA (April 2005)
4. Brightwell, R., Oldfield, R., Maccabe, A.B., Bernholdt, D.E.: Hobbes: Composition and virtualization as the foundations of an extreme-scale OS/R. In: Proceedings of the 3rd International Workshop on Runtime and Operating Systems for Supercomputers (ROSS 2013), pp. 2:1–2:8. ACM, New York, `http://doi.acm.org/10.1145/2491661.2481427`

5. Engelmann, C., Scott, S.L., Ong, H., Vallée, G., Naughton, T.: Configurable Virtualized System Environments for High Performance Computing. In: Proceedings of the 1st Workshop on System-level Virtualization for High Performance Computing (HPCVirt 2007), Held in Conjunction with the ACM EuroSys 2007, Lisbon, Portugal (March 20, 2007), `http://www.csm.ornl.gov/srt/hpcvirt07`
6. Gallard, J., Lèbre, A., Vallée, G., Morin, C., Gallard, P., Scott, S.L.: Refinement proposal of the goldberg's theory. In: Hua, A., Chang, S.-L. (eds.) ICA3PP 2009. LNCS, vol. 5574, pp. 853–865. Springer, Heidelberg (2009)
7. Goldberg, R.P.: Architecture of Virtual Machines. In: Proceedings of the Workshop on Virtual Computer Systems, pp. 74–112. ACM Press, New York (1973)
8. Goloubeva, O., Rebaudengo, M., Reorda, M.S., Violante, M.: Software-Implemented Hardware Fault Tolerance. Springer (August 2006)
9. Huang, W., Liu, J., Abali, B., Panda, D.K.: A case for high performance computing with virtual machines. In: ICS 2006: Proceedings of the 20th annual international conference on Supercomputing, pp. 125–134. ACM Press, New York (2006)
10. Intel® Corporation: Intel® 64 and IA-32 Architectures Software Developer's Manual – Volume 1: Basic Architecture (February 2014), `http://www.intel.com/products/processor/manuals`, Order Number: 253665-050US
11. Kitten lightweight kernel, `https://software.sandia.gov/trac/kitten` (last visited: August 29, 2009)
12. Koopman, P., DeVale, J.: The exception handling effectiveness of POSIX operating systems. IEEE Transactions on Software Engineering 26(9), 837–848 (2000)
13. Lange, J., Pedretti, K., Hudson, T., Dinda, P., Cui, Z., Xia, L., Bridges, P., Gocke, A., Jaconette, S., Levenhagen, M., Brightwell, R.: Palacios and Kitten: New high performance operating systems for scalable virtualized and native supercomputing. In: IEEE International Symposium on Parallel Distributed Processing (IPDPS), pp. 1–12 (April 2010)
14. Linux Kernel-based Virtual Machine (KVM), `http://www.linux-kvm.org`, `http://www.linux-kvm.org` (last visited: March 30, 2014)
15. Liu, J., Huang, W., Abali, B., Panda, D.K.: High performance VMM-Bypass I/O in virtual machines. In: Proceedings of the Annual USENIX Technical Conference (USENIX 2006), pp. 29–42. USENIX Association (2006), `http://www.usenix.org/events/usenix06/tech/liu.html`
16. Naughton, T., Bland, W., Vallée, G.R., Engelmann, C., Scott, S.L.: Fault Injection Framework for System Resilience Evaluation: Fake Faults for Finding Future Failures. In: Proceedings of the 2nd Workshop on Resiliency in High Performance Computing (Resilience 2009), ACM Press, New York (June 9, 2009); held in conjunction with HPDC 2009, Munich, Germany
17. Naughton, T., Vallée, G., Engelmann, C., Scott, S.L.: A case for virtual machine based fault injection in a high-performance computing environment. In: Alexander, M., et al. (eds.) Euro-Par 2011, Part I. LNCS, vol. 7155, pp. 234–243. Springer, Heidelberg (2012)
18. Palacios: An OS independent embeddable VMM, `http://v3vee.org/palacios`, Project URL: http://v3vee.org/palacios/ (Last visited: April 26, 2014).
19. RedHat: (Whitepaper) KVM - Kernel-based Virtual Machine (September 1, 2008), `http://www.redhat.com/resourcelibrary/whitepapers/doc-kvm` (last visited: April 1, 2014).
20. Riesen, R., Brightwell, R., Bridges, P.G., Hudson, T., Maccabe, A.B., Widener, P.M., Ferreira, K.: Designing and implementing lightweight kernels for capability computing. Concurrency and Computation: Practice and Experience 21(6), 793–817 (2009), `http://dx.doi.org/10.1002/cpe.1361`

21. Scott, S.L., Vallée, G., Naughton, T., Tikotekar, A., Engelmann, C., Ong, H.: Research on System-Level Virtualization at the Oak Ridge National Laboratory. Future Generation Computer Systems (2009)
22. Vallée, G., Naughton, T., Ong, H., Scott, S.L.: Checkpoint/restart of virtual machines based on xen. In: HAPCW 2006: High Availability and Performance Computing Workshop. Held in conjunction with LACSI 2006, Santa Fe, New Mexico, USA (October 2006)
23. Vallée, G.R., Naughton, T., Engelmann, C., Ong, H.H., Scott, S.L.: System-level virtualization for high performance computing. In: Proceedings of the 16th Euromicro International Conference on Parallel, Distributed, and network-based Processing (PDP), February 13-15, pp. 636–643. IEEE Computer Society, Los Alamitos (2008), `http://www.csm.ornl.gov/~engelman/publications/vallee08system.pdf`
24. Youseff, L., Seymour, K., You, H., Dongarra, J., Wolski, R.: The impact of paravirtualized memory hierarchy on linear algebra computational kernels and software. In: Proceedings of the 17th International Symposium on High Performance Distributed Computing (HPDC 2008), pp. 141–152. ACM, New York (2008)