Future Generation Computer Systems 26 (2010) 304–307

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

System-level virtualization research at Oak Ridge National Laboratory☆

Stephen L. Scott*, Geoffroy Vallée, Thomas Naughton, Anand Tikotekar, Christian Engelmann, Hong Ong

Oak Ridge National Laboratory, Oak Ridge, TN 37830, USA

ARTICLE INFO

Article history:
Received 11 January 2008
Received in revised form
23 June 2009
Accepted 6 July 2009
Available online 16 July 2009

Keywords:
Systems
System architectures
Fault tolerance

ABSTRACT

System-level virtualization is today enjoying a rebirth as a technique to effectively share what had been considered large computing resources which subsequently faded from the spotlight as individual workstations gained in popularity with a "one machine–one user" approach. One reason for this resurgence is that the simple workstation has grown in capability to rival anything similar, available in the past. Thus, computing centers are again looking at the price/performance benefit of sharing that single computing box via server consolidation.

However, industry is only concentrating on the benefits of using virtualization for server consolidation (enterprise computing) whereas our interest is in leveraging virtualization to advance high-performance computing (HPC). While these two interests may appear to be orthogonal, one consolidating multiple applications and users on a single machine while the other requires all the power from many machines to be dedicated solely to its purpose, we propose that virtualization does provide attractive capabilities that may be exploited to the benefit of HPC interests. This does raise the two fundamental questions: is the concept of virtualization (a machine "sharing" technology) really suitable for HPC and if so, how does one go about leveraging these virtualization capabilities for the benefit of HPC.

To address these questions, this document presents ongoing studies on the usage of system-level virtualization in a HPC context. These studies include an analysis of the benefits of system-level virtualization for HPC, a presentation of research efforts based on virtualization for system availability, and a presentation of research efforts for the management of virtual systems. The basis for this document was the material presented by Stephen L. Scott at the Collaborative and Grid Computing Technologies meeting held in Cancun, Mexico on April 12–14, 2007.

Published by Elsevier B.V.

1. Introduction to system-level virtualization

System-level virtualization is used for a number of reasons, but the three major justifications are [1–3]: (i) *isolation*, (ii) *consolidation*, and (iii) *migration*. We describe these points in the following sections after a brief description of the terminology used in this article.

Terminology. The execution of a virtual machine (VM) implies that one or more virtual systems are running concurrently on top of the same hardware, each having its own view of available resources. The operating system (OS) of the VMs is referred to as the *guest OS*. This system is a traditional OS, but does not necessarily see all the available physical resources. The guest OS only views resources that have been allocated to the VM. VMs hosted on the same machine are run concurrently with the *hypervisor* managing the concurrent execution. The hypervisor is responsible for mapping the physical resources to those used by the virtual machines. Due to its role as a VM manager, the hypervisor is also called the *Virtual Machine Monitor* (VMM). The hypervisor is typically a small system running beside the VMs. The hypervisor typically does not include drivers or other device specific mechanisms to access the physical hardware, e.g., network cards or hard drives. Therefore, the hypervisor is coupled with a traditional operating system, called the *host OS*. This host OS provides users log-in access for the administration of physical resources and an interface to the hypervisor for VM management.

1.1. Virtualization capabilities

Isolation. Isolation aims at improving the security and reliability of the system by isolating the execution environment for applications in a VM which cannot corrupt the bare hardware [1,3]. For that the virtualization solution exposes to VMs a virtual hardware, *i.e.*, a VM is limited to the execution of unprivileged instructions with the hypervisor overseeing all other operations.

☆ Research sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC for the US Department of Energy under Contract No. DE-AC05-00OR22725.

* Corresponding author.
E-mail addresses: scottsl@ornl.gov (S.L. Scott), valleegr@ornl.gov (G. Vallée), naughtont@ornl.gov (T. Naughton), tikotekaraa@ornl.gov (A. Tikotekar), engelmannc@ornl.gov (C. Engelmann), hongong@ornl.gov (H. Ong).

0167-739X/$ – see front matter. Published by Elsevier B.V.
doi:10.1016/j.future.2009.07.001

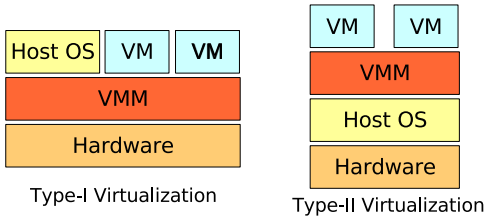

Fig. 1. Classification of virtualization techniques.

Consolidation. Server consolidation is the primary market for virtualization solutions, because it enables the sharing of expensive servers between different customers with the guarantee that each customer will have its own view of the system and be isolated from other users. This allows service providers to consolidate work to fewer servers (cost effective server usage) and also to support incompatible or legacy operating environments without the need for separate hardware.

Virtual machine migration. The capability to migrate entire VMs between servers, makes it possible to improve the quality of service by balancing the global load between several servers without interruption of application execution and by moving VMs (and therefore applications) when a failure is predicted for a specific server. This technology also enables a transparent (for users) programmable downtime of servers by migrating VMs to other servers before server shutdown for maintenance.

1.2. Classification

In the 1970s, Goldberg [4] classified the different system-level virtualization solutions into two categories (see Fig. 1): (i) *type*-I *virtualization* where the VMM runs directly on the bare hardware, and (ii) *type*-II *virtualization* where the VMM runs on top of the host OS. Since the type-I 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 is well suited for development, where some performance may be reduced in exchange for greater diagnostic and development capabilities.

A third hybrid form called *para-virtualization* [1,3] has emerged that fits the criteria of *type*-I virtualization. In para-virtualization, modifications are made to the host OS and guest OS that are essentially a new software-only architecture. This additional work provides the improved performance of *type*-I. For that, the operating system is modified to include an interface with the VMM, enabling more efficient access to resources. For instance, in Xen, the Linux kernel is modified to include *hypercalls*, the equivalent of system calls for communication with the Xen VMM. The modification results in the creation of a new software-only architecture.

2. Why system-level virtualization for high-performance computing?

Today, high-performance computing (HPC) centers need to support multiple execution platforms. For example, at Oak Ridge National Laboratory (ORNL), massively parallel processing (MPP) platforms, such as the Cray XT4, and Beowulf-type clusters, like the ORNL Institutional Cluster (OIC), are available to users. Each of these systems targets a specific OS, requiring users to *port* their application before execution. On the other hand, a user's requirements may also differ. For instance, some users develop their applications on desktops, others on clusters; including different needs in terms of hardware and software requirements. Finally, each application has its own hardware and software constraints.

It is therefore very difficult to provide a single execution environment for all applications that is supported by quite a variety of development environments. To address these issues, the notion of *plug-and-play computing* and *execution environment customization* have been introduced. The idea is to let users specify their needs in terms of system environment and then deploy on demand this environment in virtual machines. The deployment of an application on a new execution platform is therefore direct.

However, even if the raw performance of virtualization solutions is nowadays more suitable for HPC [5–7], their utilization is still very limited: (i) their system footprint is significant and will interfere with application execution and performance; (ii) current solutions support base Intel and AMD architectures but do not fully support their use in some of the more exotic HPC specific architectures; and (iii) the architecture of virtualization solutions is monolithic and does not allow dynamic configuration of the VMM.

In order to have a virtualization solution suitable for HPC, many approaches are possible: (i) development of a new solution from scratch, (ii) development of a new solution based on an existing solution, such as Xen, or (iii) development of a new solution based on a HPC specific operating system, such as Catamount. Each of these approaches have its own advantages and drawbacks. For instance, the development of a new solution from scratch allows the design and implementation of a solution to perfectly meet HPC constraints. However, such a solution is a long term effort because it implies kernel-level development which is challenging. The development of a new solution based on an existing solution (which may be an already existing VMM and a kernel for HPC) gives more short term results but may be difficult to maintain since it was not originally designed for the HPC environment.

3. System-level virtualization and system availability

Modern high-performance computing platforms are composed of thousands or even hundreds of thousands of nodes. Because each node can be subject to a failure, the global availability of the system decreases in proportion to the system scale. Therefore, applications for this environment must be fault tolerant or resilient, able to operate successfully in the face of failure, and the systems should exhibit high-availability traits.

System-level virtualization provides three interesting capabilities that may be exploited for this purpose: (i) VM migration, (ii) VM pause/unpause, and (iii) VM checkpoint/restart. These three mechanisms enable the implementation of three fault tolerance policies: (i) reactive fault tolerance (do something after a failure occurs), (ii) proactive fault tolerance (do something before the failure occurs), and (iii) hybrid policies mixing both reactive and proactive fault tolerance.

The reactive approach fundamentally relies on the concept that there is sufficient contextual information available (checkpoint data), such that it is possible to restart an application after a failure occurs and recover close to the point of failure so that little computation is lost. Because each VM and the applications running on it is a well defined entity, isolated from all other applications and even from the core host operating system, it is possible to checkpoint the entire VM and then subsequently restart it in that original state without any prior knowledge on the part of the applications. This is possible because the context of a VM typically consists of the memory dump and a checkpoint of the file system. While all current virtualization solutions can already dump the memory of the VM (the hypervisor is in charge of the memory management of VMs), the file system is quite a different matter. This is due to the need for successive snapshots of the complete file system. One solution to address this issue is to use stackable