Non-intrusive Virtualization Management using libvirt

Matthias Bolte, Michael Sievers, Georg Birkenheuer, Oliver Niehörster and André Brinkmann
Paderborn Center for Parallel Computing PC^2, University of Paderborn, Fürstenallee 11, 33102 Paderborn, Germany
{photon, msievers, birke, nieh, brinkman}@uni-paderborn.de

***Abstract*—The success of server virtualization has let to the deployment of a huge number of virtual machines in today's data centers, making a manual virtualization management very labor-intensive. The development of appropriate management solutions is hindered by the various management interfaces of different hypervisors. Therefore, a uniform management can be simplified by a layer abstracting from these dedicated hypervisor interfaces.**

The libvirt management library provides such an interface to different hypervisors. Unfortunately, remote hypervisor management using libvirt has not been possible without altering the managed servers. To overcome this limitation, we have integrated remote hypervisor management facilities into the libvirt driver infrastructure for VMware ESX and Microsoft Hyper-V. This paper presents the resulting architecture as well as experiences gained during the implementation process.

I. Introduction

Server virtualization has proven to be a valuable tool to simplify data center management and to increase resource efficiency as well as service reliability. The ability of modern virtual machine managers to run a huge number of virtual machines on a single physical server enables the consolidation of multiple under-utilized physical servers inside a single one, while keeping the property of failure and resource isolation between the virtual machine instances. Virtualization technology further simplifies failure handling, as the virtual machines can be easily restarted on a different physical machine after a hardware failure. Load balancing is supported by features like live-migration, which reduces the downtime required to move a logical service between different physical servers to milliseconds, having no measurable effect on many applications.

Running a few virtual machines has, as discussed, strongly simplified many administration tasks. This simplification has encouraged administrators as well as end users to create hundreds to thousands of virtual machines inside a single infrastructure. This explosion of the number of virtual machines introduces many new management issues. The most obvious are the question about the physical location, where a virtual machine is running and the question about the names and types of virtual machines, which are running on a physical server. Not having a virtual machine management answering these questions would, for example, immediately lead to long, manual recovery phases after physical failures, canceling most of the virtualizations advantages.

Different requirements concerning supported hardware and operating systems have triggered the creation of a number of different virtualization approaches and solutions. Full virtualization uses binary translation to run arbitrary, unmodified operating systems on top of the hypervisor (also called virtual machine monitor), while hardware-assisted virtualization uses CPU traps to handle sensitive instructions. Paravirtualization also modifies the guest operating system to optimize the interplay between virtual machine monitor and the virtual machine itself. In contrast, operating system-level virtualization runs different isolated instances of the user environment on top a single operating system core. These distinctions have let to a number of different virtualization solutions like VMware ESX, Xen, KVM, VirtualBox, OpenVZ, or Solaris containers, which have very different management APIs.

The different management APIs make it very difficult for developers to build virtualization management solutions, which are able to support arbitrary virtualization environments. Commercial providers like VMware and Citrix overcome this problem by providing management solutions, which are tailored and restricted to their specific hypervisors, optimizing the usage of their specific virtual machine monitor. Open source projects like oVirt, Eucalyptus, or OpenNebula try to provide a broader support and to stay independent of a particular hypervisor. This either requires the adaptation of their internal interfaces to the broad set of hypervisor management APIs or the usage of an abstraction layer, which translates from one interface to the different API flavors. Even if this abstraction layer requires the restriction to a common set of core functionalities, which are supported by all hypervisors, it also strongly simplifies the development and maintenance of management environments.

The libvirt project develops such a virtualization abstraction layer, which is able to manage a set of virtual machines across different hypervisors. The goals of libvirt are to provide a library that offers all necessary operations for hypervisor management without implementing functionalities, which are tailored to a specific virtualization solutions and which might not be of general interest. Additionally, the long-term stability of the libvirt API helps these management solutions to be isolated from changes of hypervisor APIs.

Contributions of this paper: Today, there are still no management solutions, which are able to manage arbitrary hypervisors. Administrators are either restricted to use open source management solutions to manage open source hypervisors or they can use proprietary management suites for commercial hypervisors. The libvirt project offers an API that

978-3-9810801-6-2/DATE10 © 2010 EDAA

is able to manage arbitrary hypervisors over a **stable** interface. This paper presents the first implementation that integrates the commercial hypervisors VMware ESX and Microsoft Hyper-V into the libvirt library. Besides presenting the architecture of our integration, we discuss issues discovered during the implementation as well as limitations of API mapping in the context of virtualization management.

II. Related Work

Server virtualization plays a crucial role in todays IT management environments, both as foundation for cloud services as well as in data centers. Popek and Goldberg have analyzed instruction set properties, which determine the virtualizability of a processor [1]. Their most important theorem claims that for an efficiently virtualizable processor, its sensitive instructions have to be a subset of its privileged instructions. Early x86 architectures have not been virtualizable in this classical sense, leading to the creation of very different virtualization implementations [2].

Full virtualization solutions like VirtualBox or VMware replace sensitive instructions with binary translated blocks to run arbitrary operating systems on top of x86 processors [3]. Paravirtualized solutions like Xen require a modification of the guest operating systems, enabling them to become faster, while reducing the universality of the approach [4]. Modern x86 architectures, like Intel VT or AMD's Pacifica introduce new running modes, which make the processor efficiently virtualizable, but restrict it from being recursively virtualizable [5]. Most modern virtualization solutions, like VMware ESX, KVM, and Xen support this hardware-assisted virtualization [6] [7]. Virtualization solutions simplify data center management by their live-migration support and other interesting abilities, like the possibility to dynamically change VM resources during runtime [8] [3] [9]. Nevertheless, the management concepts and the corresponding APIs differ significantly between the different virtualization solutions (see also Section IV).

Virtualization management suites and cloud environments like oVirt, VMware vCenter, XenServer, OpenNebula and Eukalyptus enable system administrators to handle hundreds and thousands of virtual machines [10] [11] [12]. They help to manage virtual machines and images, as well as users and integrate third-party tools like backup support or manage different storage interfaces. The commercial solutions typically only support their own hypervisor, while open source solutions try to be as general as possible. The different APIs make it very difficult to support a broad set of hypervisors without support from a virtualization abstraction layer. Several projects have emerged, which try to provide such a uniform interface to the different hypervisors. The most notables of these projects are the Common Information Model (CIM)-standardization efforts of the Distributed Management Task Force (DMTF) and the libvirt project [13].

III. Hypervisor management using CIM

CIM in the broadest sense is often used as a synonym for Web Based Enterprise Management (WBEM), which is in fact an umbrella term for various technologies specified by the DMTF targeting administration and remote maintenance of computer systems. This includes the specification of a data model, the Common Information Model, which is what CIM literally stands for. Additionally, WBEM enumerates protocols and access patterns in order to work with this model in several DMTF Standard Publications (DSPs), e.g. Representation of CIM in XML, CIM Operations over HTTP or the CIM Query Language Specification. It also includes a specification on how to access CIM in a web service like fashion, called Web Services for Management, or WSMAN for short.

Often mistaken, WBEM is not an API but an infrastructure for accessing administrative system data in an uniform way. Precisely, WBEM states that there are classes, but not how a class representing a specific resource has to be named. There are such specifications called schemas provided by the DMTF, but no vendor is forced to abide. Also, no vendor is forced to use CIM operations over HTTP as the access protocol. For example Microsoft's WBEM implementation WMI uses a proprietary DCOM based protocol.

It should be clear at this point that just the fact that two systems use CIM as the data model does not make them automatically compatible. Both, the schemas and the access methods have to be intermateable. For example, even if WMI implements the schemas proposed by the DMTF, a system not capable of running the proprietary DCOM protocol is unable to access it.

There are DMTF schemas concerning the management of virtual machines, but no vendor actually implements them correctly. Microsoft's Hyper-V API, which uses WMI, is actually based on the DMTF proposal, but there are major differences, e.g. concerning the management of snapshots. For example, the creation of a virtual machine snapshot involves the method CreateSnapshot of the class VirtualSystemSnapshotService as stated by the DMTF schema[1]. However, the Hyper-V API manages the snapshot creation through a method called CreateVirtualSystemSnapshot of the class VirtualSystemManagementService.

This is even worse for other hypervisors claiming CIM support, e.g. VMware ESX. There is simply no way of managing virtual machines of an ESX server using WBEM technologies, because no such functionalities are exposed. VMware ESX server provides information about hardware components or attached storage devices using WBEM, but no virtual machine management functions.

It has to be concluded, that WBEM (including CIM) is not yet a solution for uniformly managing different kinds of hypervisors. This may change in the future, but it has to be questioned, whether vendors are willing to support interoperability or if it just leads to embrace, extend, and extinguish products.

[1]CIM Schema 2.23.0

IV. LIBVIRT, AN ABSTRACTION LAYER

Libvirt is an abstraction layer library for various hypervisor management APIs written in C. It started as a Xen wrapper, but has been extended to several other hypervisors later on. It is conceptual divided into a hypervisor agnostic and several hypervisor specific parts, also called drivers.

Applications can use libvirts public API, which internally maps to appropriate driver functions through an internal driver API. This driver model has proven to be very flexible and easy to extend, so drivers for storage and network interfaces using the same model have been added.

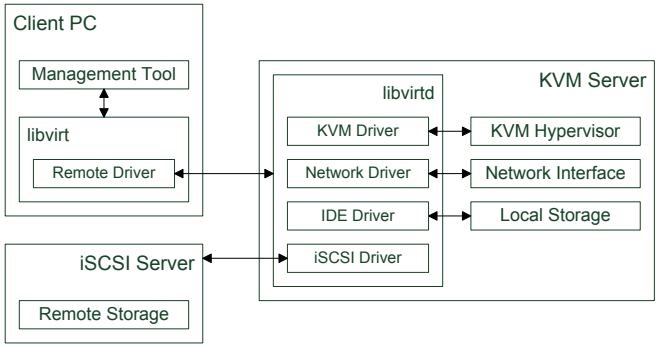

Fig. 1. Driver based libvirt architecture.

Many hypervisors like KVM do not offer remote management facilities. Libvirt tries to circumvent this by implementing a remote driver on the client and an appropriate daemon for handling client requests from the server side, called libvirtd (see also Figure 1). Requests from a client are tunneled through the remote driver to the server, where the specific hypervisor is running. The libvirtd on the server receives the requested commands and locally calls the specific driver (see Figure 2).

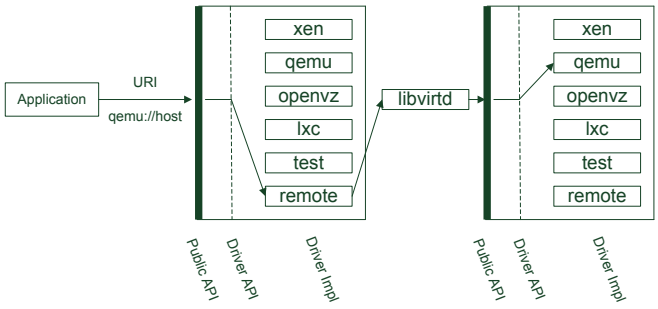

Fig. 2. Remote driver and libvirtd.

Introducing a middleware architecture consisting of the remote driver and the libvirtd daemon offers great flexibility. Communication between client and hypervisor can be, for example, compressed and/or encrypted in a way that is totally transparent to the hypervisor. The major drawback of this architecture is that the physical server, on which the remotely managed hypervisor resides, has to run libvirtd. The problem here is that this requirement cannot always be fulfilled, as for example it is difficult to install software on VMware ESX.

VMware ESX and also ESXi are closed software products and integrating third party components is unsupported. While there is a minimal Linux based operating service console running on each ESX server, altering it is neither supported nor comfortable. Without this, remote management of an ESX server using libvirt and libvirtd is impossible in the conventional way.

Microsoft Hyper-V servers, which are based on the Windows server operating system, allow the installation of third party software, but are also currently unsupported by libvirtd. Our evaluation has shown that porting the libvirtd to Microsoft Hyper-V would not really help us. Firstly, the libvirtd is strongly coupled to Unix-like operating systems, requiring to port the software. Secondly, even having a port to Microsoft Hyper-V would still require us to access its API in nearly the same way as accessing it remotely.

V. VMWARE ESX AND MICROSOFT HYPER-V

As pointed out before, libvirt uses libvirtd to establish remote connections with a hypervisor. This concept basically excludes support for VMware ESX and Microsoft Hyper-V, because libvirtd cannot be efficiently implemented and maintained for these hypervisors. To enable libvirt to manage such hosts remotely, the relevant libvirt drivers have to implement the communication on their own.

Bypassing libvirtd for remote hypervisor management implicates the loss of features provided by the daemon, such as transparent encryption or compression. Because of VMware ESX and Microsoft Hyper-V do not differentiate between local and remote API access, a driver capable of controlling such a hypervisor remotely can possibly be used locally in the future. Therefore integrating remote access into the VMware ESX, respectively Microsoft Hyper-V driver does not break the current libvirt architecture, because switching to a libvirtd approach is possible at least from the architectural point of view.

We will see in the following, which protocols are needed to communicate with an ESX, respectively Hyper-V host and how they can be be implemented to be used within a correspondent libvirt driver. We will also have a brief look at the according APIs and how they influence the concrete driver development.

A. VMware ESX

VMware ESX uses a large, SOAP based API named vSphere API, which was formerly known as Virtual Infrastructure API. Besides single VMware ESX server management, it also covers the management of an entire VMware cluster. A cluster is formed as a combination of several VMware ESX servers in combination with a VMware vCenter server. In this setup, the VMware vCenter is responsible for higher-order cluster management functionality, like load-balancing or virtual machine migration (see Figure 3).

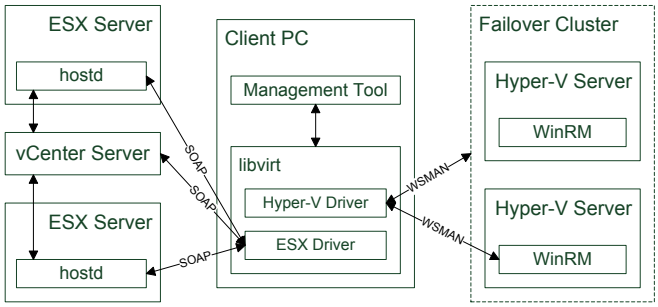

Fig. 3. Communiucation architecture for VMware ESX and Microsoft Hyper-V without libvirtd.

In order to use this SOAP API from within libvirt, it is necessary to write a C-based SOAP client. An obvious and simple approach would be to generate C code from the vSphere API Web Services Description (WSDL) definition, using a WSDL2C generator tool. Unfortunately, this approach does not work very well, as the object types of the vSphere API inherit from each other and generator tools, as the gSOAP[2] toolkit, do not generate working C code, if the degree of inheritance is too big. In addition, the size of the generated code is enormous. Other libvirt developers have tried to use generated C code from the vSphere API before, but gave up this approach for the same reasons.

SOAP does not map easily to C, because C is in contrast to the object-based SOAP protocol not an object oriented language. A gSOAP generated C++ client works without problems, but C++ is not an option, because a mandatory requirement for libvirt drivers is that they have to be written in C.

One solution for this problem is the VMware VIX API, because VMware provides closed source C bindings for it. However, the main libvirt developers voted against it, as the bindings are closed source and because of possible license issues.

Our solution for this problem is a custom SOAP client handwritten in C, which is based on libcurl[3] for the HTTP transport and libxml2[4] for XML (de-)serialization. Writing the client by hand solves the inheritance problem in C. Furthermore, it allows to only implement the necessary part of the vSphere API, so that the SOAP client code is much smaller than the generated C code.

B. Microsoft Hyper-V

Hyper-V is accessed through Microsoft's CIM implementation, called Windows Management Instrumentation (WMI). The native transport when calling WMI remotely is the Distributed Component Object Model (DCOM) binary protocol. Migration is handled by the Microsoft Failover Cluster service and is not part of the main Hyper-V API, but is included in the Failover Cluster API. Fortunately, this API is also WMI-based, so that there is no difference in accessing these functionalities remotely.

The direct use of WMI over DCOM from non-Windows hosts is very difficult, as there is no software implementing this protocol outside the Windows operating systems. There are attempts from within the Samba project, but the results are not yet in an usable state.

Our solution uses WS-Management (WSMAN), which can be used to access WMI using SOAP. WS-Management is a DMTF protocol to use CIM operations over HTTP in a web service like manner. Microsoft implemented this as Windows Remote Management (WinRM) into recent Windows operating systems. Using WinRM enables us to access WMI from non-Windows systems using open source software (see Figure 3).

An example for an open source WS-Management implementation is OpenWSMAN[5]. It provides server and client components for WS-Management environments. The WS-Management client library component for C is of special interest for our approach. It speeds up implementing Hyper-V management from within a driver without the expense of writing an own client side communication stack, as it is necessary for ESX.

VI. Driver development issues

After the fundamentals have been done and remote access to the hypervisor is available, our next step was to implement the libvirt driver API using the hypervisor remote API. Some examples for simple and more complex as well as currently unsolved API mapping problems will be presented in the following.

A. VMware ESX

An example for a simple one-to-one mapping is the vSphere API method `FindByUuid()`, because it allows to obtain a `VirtualMachine` object by its UUID to fulfill the libvirt driver API function `virDomainLookupByUUID()`. Also, this method is often required to implement other libvirt driver API function.

The performance information provided by the vSphere API `PerformanceManager` object is an example for a more complex mapping problem. Each VMware ESX server collects a large set of performance information, but the collected values are only available in relative form. For example, the used CPU time or the number of bytes read from a disk are returned in 5 minute slots. The libvirt driver API, in contrast, expects these values in an absolute form. In the context of the used CPU time or the number of bytes read from disk, these values are expected as the accumulated numbers since the virtual machine has been started. Nevertheless, the information is not available in this form and cannot be calculated in a robust way from the available information.

There is no simple solution to this mapping problem and this problem is still unsolved. A possible solution would be to extend the libvirt API to allow performance information in relative form.

[2]http://www.cs.fsu.edu/~engelen/soap.html
[3]http://curl.haxx.se/
[4]http://xmlsoft.org/
[5]http://www.openwsman.org/

Another unsolved problem is the network management. The VMware ESX networking management is based on virtual switches, virtual port groups, virtual network interface controllers (NIC), and physical NICs. The libvirt API currently has no notion for virtual switches and their assignment to virtual port groups or physical NICs. A possible solution would be to extend the libvirt API with a notion for virtual switches and their relation to other networking elements.

A migration involves two VMware ESX servers. Therefore it is handled as a higher-order cluster management functionality and requires to run VMware vCenter. The migration command is issued on the VMware vCenter server to migrate a virtual machine from the source VMware ESX server to the destination VMware ESX server.

This represents another potential API mapping problem, because libvirt models a migration as a process that involves the source and destination server only. A higher-level management instance like the VMware vCenter is not part of this model.

In order to perform a migration, the libvirt VMware ESX driver has to know which VMware vCenter server is in charge for the involved VMware ESX servers. This information can be retrieved from each VMware ESX server via the vSphere API.

Before the driver is able to successfully issue commands to an VMware ESX or vSphere server, it has to login to the server. The necessary credentials are passed to the driver in its open function. In order to perform a migration later on, the credentials for the connection to the VMware vCenter server have also to be passed to this open function.

B. Microsoft Hyper-V

First of all, there is a fundamental mapping issue between libvirt and the Hyper-V API. While libvirt is a classic C library, the Hyper-V API is realized using Microsoft's CIM implementation named WMI. It is characterized by classes, instances and relations between them. To get information about virtual machines, you have to query the data model using a predefined set of meta operations, also called intrinsic CIM operations, which are shown in Table I.

Extrinsic CIM operations on the other hand are methods of specific classes or instances. For example, changing the power state of a virtual machine is realized as a method of an instance of a class representing virtual machines.

In the following, we examine the libvirt function `virDomainLookupByUUID()`, which has already been discussed for the ESX driver. Having the UUID of a virtual machine, the CIM data model has to be queried to obtain all instances of the `MSVM_ComputerSystem` class. After obtaining all instances, the one with the corresponding UUID has to be selected on client side. This means in detail that the result set has to be checked for an instance with its property `Name` being equal to the given UUID, as Hyper-V stores the UUID in the `Name` property of instances of the `MSVM_ComputerSystem` class.

Issues on network management are similar to ESX, as Hyper-V exposes much more features than can be handled through libvirt.

Functional Group	Dependency	Methods
BasicRead	None	GetClass EnumerateClass EnumerateClassName GetInstance EnumerateInstance EnumerateInstanceName GetProperty
BasicWrite	BasicRead	SetProperty
Instance Manipulation	Basic Write	CreateInstance ModifyInstance DeleteInstance
Schema Manipulation	Instance Manipulation	CreateClass ModifyClass DeleteClass
Association Traversal	Basic Read	Associators AssociatorNames References ReferenceNames
Query Execution	BasicRead	ExecQuery
Qualifier Declaration	SchemaManipulation	GetQualifier SetQualifier DeleteQualifier EnumerateQualifiers

TABLE I
INTRINSIC CIM OPERATIONS

As mentioned before, migration is not accomplished by Hyper-V directly, but by the Failover Cluster service, which shares the Hyper-V access semantic by also exposing its API using WMI. That is why there is no difference in accessing this additional software component.

VII. CONCLUSION

Virtual machine management is still one of today's most important problems in data center management. The development of open source management solutions often suffers from the large number of different hypervisors, which are on the market. Supporting a large fraction of these hypervisors requires the implementation of interfaces for a large number of different hypervisor interfaces. libvirt is the most prominent solution of an abstraction layer, being used by a number of management solutions and offering a stable interface to hypervisor management.

This paper has presented the first approach to include support for closed-source hypervisors into libvirt, enabling the integration of VMware ESX and Microsoft Hyper-V in open source management environments and also strongly simplifying their long-term maintenance via libvirt's stable API. The identified issues in mapping libvirt to a remote API will trigger further interesting architectural extensions to this open source solution.

ACKNOWLEDGMENT

We would like to thank the libvirt development community for helpful discussions as well as Fujitsu Technology Solutions for a generous hardware gift.

REFERENCES

[1] G. Popek and R. Goldberg, "Formal requirements for virtualizable third generation architectures," *Communications of the ACM*, vol. 17, no. 7, pp. 412 – 421, 1974.

[2] J. Smith and R. Nair, *Virtual Machines. Versatile Platforms for Systems and Processes*. Elsevier Ltd, 2005.

[3] C. Waldspurger, "Memory resource management in vmware esx server," in *Proceedings of the 5th Conference on Operating Systems Design and Implementation (OSDI)*, Dec. 2002, pp. 181 – 194.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, Bolton Landing, NY, USA, 2003, pp. 164–177.

[5] G. Neiger, A. Santoni, F. Leung, D. Rodgers, and R. Uhlig, "Intel virtualization technology: Hardware support for efficient processor virtualization," *Intel Technology Journal*, vol. 10, no. 3, pp. 167 –178, 2008.

[6] J. Sugerman, G. Venkitachalam, and B.-H. Lim, "Virtualizing i/o devices on vmware workstation's hosted virtual machine monitor," in *Proceedings of the Usenix Annual Technical Conference 2001*, Boston, Massachusetts, 2001.

[7] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the linux virtual machine monitor," in *Proceedings of the Linux Symposium*, Ottawa, Ontario, Canada, Jun. 2007, pp. 225 –230.

[8] C. Clark, K. Fraser, S. Hand, J. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the* 2^{nd} *Symposium on Networked Systems Design and Implementation*, Boston, Massachusetts, USA, May 2005.

[9] D. Chisnall, *The Definitive Guide to the Xen Hypervisor*. Upper Saddle River, NJ, USA: Prentice Hall, 2007.

[10] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Capacity leasing in cloud systems using the opennebula engine," in *Proceedings of the Workshop on Cloud Computing and its Applications 2008 (CCA)*, Chicago, Illinois, USA, Oct. 2008.

[11] ——, "Virtual infrastructure management in private and hybrid clouds," *IEEE Internet Computing*, vol. 13, no. 5, pp. 14 – 22, Sep. 2009.

[12] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in *Proceedings of the 9th IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, Shanghai, China, May 2009.

[13] M. Johanssen, "Update on system virtualization management," in *Proceedings of the 2nd International Workshop on Systems and Virtualization Management (SVM)*, Munich, Germany, Oct. 2008, pp. 125 – 134.