A SNMP-based Virtual Machines Management Interface

Ricardo Hillbrecht, Luis Carlos E. de Bona
Federal University of Paraná, Dept. Informatics
P.O.Box 19.081 - 81.531-980 - Curitiba, PR Brazil
{ricardoh, bona}@inf.ufpr.br

Abstract—This paper presents Virtual-Machines-MIB, a MIB (Management Information Base) directed to virtual machines management through SNMP (Simple Network Management Protocol). Virtual-Machines-MIB aims to define a standard interface for virtual machines management, allowing the management of several virtual machines monitors, like Xen, KVM and VMware, with a common SNMP management tool. Different from previous virtual machines management MIBs, which allows the manager to perform only monitoring operations, Virtual-Machines-MIB allows to perform control operations, like create, delete, restart, turn on, pause and shut down virtual machines. It is also possible to use the proposed solution to change a virtual machine’s name, amount of RAM, virtual CPU’s and virtual storage drives. Practical results are presented using ordinary SNMP management tools performing KVM and Xen management. To do this, SNMP agents which support Virtual-Machines-MIB were developed and installed on KVM and Xen hosts. These SNMP agents are based on NET-SNMP public domain’s agent, that was extended to support Virtual-Machines-MIB using libvirt API.

I. INTRODUCTION

Virtualization is a technology that allows more than one operating system instance to run on the same machine at the same time. A virtualization layer provides low-level support for creating multiple virtual machines (or VMs), which are independent and isolated from each other. This virtualization layer is called virtual machine monitor (or VMM) [1]. The operating system of the virtual machine is called guest OS, and the operating system of the physical machine is called host OS. The use of virtual machines simplifies data centers physical management, increase resource efficiency and services reliability [2].

Recently there was the rise and popularization of cloud computing, a new paradigm of organization and delivery of services through the Internet. Virtualization is the foundation of cloud computing, since it offers the ability to aggregate computing resources of several clusters of physical machines and dynamically assign virtual resources to applications based on demand [3]. In cloud computing, the processing of customer applications takes place in centralized facilities operated by a service provider such as Amazon EC2, Google App Engine, Microsoft Windows Azure, Salesforce.com and Rackspace, among others. Due to the widespread use of cloud computing and the growing amount of VMs, new challenges risen for the management of computing environment [4], [2], as well as for the management of VMs.

The main VMM’s on the market, such as Xen [5], VMWare [6] and Hyper-V [7], provide their own management applications. As examples of management applications, it is possible to list Citrix Xen Center, VMWare Virtual Center and Microsoft Virtual Machine Manager. These applications are able to manage only the corresponding VMM. There are few management systems able to support more than one VMM, because the manufacturers keep the management of VMs restricted to their own management systems for commercial reasons [4]. Also, there are open source projects directed to management of cloud computing infrastructure which can manage more than one VMM, like oVirt [8], Eucalyptus [9] and OpenNebula [10]. The main obstacle in implementing a system capable to manage more than one VMM is the lack of standardized management interfaces or protocols.

Network management systems include tools that allow effective network monitoring and control [11]. The Simple Network Management Protocol (SNMP) is the most used architecture of networks management systems. An SNMP system is composed of managers and agents which communicate using the management protocol. Managed nodes contain an agent, which is a management entity that has access to management instrumentation. The SNMP architecture defines a Management Information Base (MIB) as a collection of related management objects which are kept in order to allow management applications to monitor and control the managed nodes [12]. The MIB provides an uniform management interface between the manager and the agent, allowing the manager to discover, read and write the management objects available in the MIB of a given node.

The paper in [13] presents an evaluation of SNMP as a virtual networks management interface. In [4] it is stated that the definition of a MIB for the management of VMs would be an advance. In [14], is proposed to use SNMP as a monitoring and control protocol in a virtual machines management system. The work in [15] proposes a MIB to monitor the use of hardware resources of VMs. The VMM’s manufacturers, such as VMWare and Citrix, include interfaces for SNMP monitoring in their products. Recent works also make use of SNMP management framework, like a GRID infrastructure monitoring system [16], a virtual router management system [13] and a distributed architecture of software-based routers [17]. Currently there are a number of management systems that uses SNMP as the management protocol, taking advantage...
of the existing set of standardized MIBs, which are quite widespread, thus allowing management systems to manage a variety of devices.

This paper proposes the use of SNMP for monitoring and controlling VMs. The use of SNMP is an adequate solution for managing virtual machines since it allows a uniform management interface between the manager and the virtualization host, a standardized and well known communication protocol and the integration of VMs management with existing management systems. In order to allow a centralized monitoring and controlling, it presents a MIB called Virtual-Machines-MIB, which defines a standard interface for the management of VMs. Through this standardized interface, it is possible to manage different VMMs using the SNMP protocol.

Practical results are presented using ordinary SNMP management tools performing KVM and Xen management operations. To obtain these results, SNMP agents that support Virtual-Machines-MIB were developed and installed on KVM and Xen hosts. These SNMP agents are based on NET-SNMP public domain’s agent [18] and support Virtual-Machines-MIB using libvirt API [19].

The rest of the paper is organized as follows. Section II presents the existing MIBs and other approaches of virtual machines management. Section III presents Virtual-Machines-MIB. Section IV presents the architecture of a Virtual-Machines-MIB implementation. Practical results follow in section V. Section VI concludes the paper.

II. RELATED WORK

The VMM’s manufacturers, such as VMWare and Citrix, include interfaces for SNMP monitoring in their products. VMWare ESX Server includes a number of MIB definitions forming vmware subtree with OID “.iso.org.dod.internet.private.enterprises.vmware” (1.3.6.1.4.1.6876).

VMWare MIB separates physical machine’s data of VM’s data placing them in separate groups. Physical machine’s data are placed in vmwSystem group, that contains VMWare product name and version, and vmwResources group, that contains data about physical CPUs, memory and storage devices.

Data about virtual machines are placed in the group vmwVirtualMachines, which contains the vmwVmTable that stores the VM’s list. This table stores data about each VM, as its name, configuration file path, guest OS name, RAM size, state, guest OS state and number of CPUs.

Additional tables contains more information about each VM. There are tables with host bus adapters, virtual disks, virtual network, floppy and CD-ROM informations. Each entry of these additional tables refers to the VMs of vmwVmTable by the VM identifier field.

A research group at the University of Braunschweig, Germany, developed a MIB for Xen managing [20]. XenMIB also has separate groups for data about the physical machine and data about the virtual machines. The physical machine’s groups contains the Xen version, physical RAM and the number and frequency of physical CPUs.

The xenDomainTable contains VM’s data, like its name, state, used and total RAM. Two additional tables contain data about VM’s CPUs and networks.

Both VMWare and Xen MIBs separates physical machine’s and VMM’s data of virtual machine’s data placing them on distinct groups. Both MIBs have a table with the list of existing virtual machines and their key attributes, such as name, state and amount of memory. Other data, such as information about CPU’s, HBA’s, virtual disks, virtual network interfaces, floppy and CD-ROM, can be found in separated tables, where each entry contains the identifier of the corresponding VM.

The MIBs of VMWare and Xen have only objects for monitoring the resources of physical and virtual machines. Thus, they not allow to change the values of management objects, what would be required to perform the control of virtual machines. Furthermore, the presented MIBs does not follow a common model for VMs management.

The work in [15] stresses the importance of a standardized interface for monitoring virtual machines on multiple virtualization platforms like VMware, Xen, KVM and VirtualBox, due to the coexistence of these platforms in cloud computing providers. Suggests that such a standardized interface should be SNMP based, using the MIB-II and Host Resources MIB to monitor the physical machine and proposing a new MIB to virtual machines monitoring, named NCNU-VM-MIB.

An implementation of NCNU-VM-MIB is presented, where libvirt is used to get information about the VMs. An experiment is performed by monitoring virtual machines in three different VMM’s: VMWare, KVM and Xen. NCNU-VM-MIB allows only monitoring operations, so it doesn’t allow control operations such as creating and deleting VMs or change its state. Due to the small number of objects present in NCNU-VM-MIB, is not possible to obtain informations such as the guest operating system and virtual storage drives.

Besides the works presented above, there is libvirt-snmpp [21], a subproject of libvirt that provides SNMP functionality for libvirt. With libvirt-snmpp, is possible to monitor virtual domains as well as set domain’s attributes over SNMP. Libvirt-snmpp allows to obtain informations about domain, control domain status and be informed about certain events.

Libvirt-snmpp provides a simple table of domains. Each row contains domain’s name, state, number of CPUs, RAM, RAM limit and CPU time. Libvirt-snmpp defines a MIB with objects that allow to set domain’s state and to delete domains, others are read-only. The libvirt-snmpp project relies only on libvirt functionality, and do not intend to perform operations beyond the ones offered by libvirt.

Other management approach, different from SNMP, is the Web-Based Enterprise Management (WBEM), defined by the Distributed Management Task Force (DMTF) and supported by numerous hardware, software and services vendors. WBEM is a set of management and internet standard technologies developed to unify the management of distributed computing environments, facilitating the exchange of management data across technologies and platforms.

The data that is transported via the encoding and transport
definitions of WBEM are defined in the Common Information Model (CIM). CIM is a conceptual information model for describing the management of a given entity, that is not bound to a particular implementation. This allows the interchange of management information between management systems and applications, either "agent to manager" or "manager to manager". DMTF defines a CIM schema to represent and manage a generic virtual system, emphasizing the demand for a standardized interface of virtual machines management. A comparison of WBEM, SNMP and other management frameworks can be found in [22].

The work in [23] proposes a virtual machines management architecture based on REST (Representational State Transfer), what also is a management approach different from WBEM and SNMP. The work suggests that REST can replace both the communication protocols between the management station and the human manager, as well as between the management station and managed components. According to [23], the adoption of REST as a single interface for management data transfer reduces the difficulties in sharing data between multiple applications, increasing interoperability between different management applications. The Open Cloud Computing Interface (OCCI) [24] also is a RESTful Protocol and API for clouds management. OCCI was originally initiated to create a remote management API for IaaS model-based services, after evolving to serve other models as PaaS and SaaS.

III. Virtual-Machines-MIB

The Virtual-Machines-MIB defines a standardized management interface including the set of functions that are common between current VMMs. Different from existing MIBs, Virtual-Machines-MIB includes not only read-only, but also read-write objects, permitting in addition of monitoring also to control managed entities.

The control objects of Virtual-Machines-MIB includes state control of VMs and functions related to the configuration of virtual hardware (RAM, CPU’s and virtual disks). The basic VM’s state control objects follow what suggests the Distributed Management Task Force (DMTF) in [25], which are as follows: define, start, stop, pause, shutdown and shutoff. Is also possible to change a VM’s Name, create and delete VM’s and create, delete and modify VM’s templates.

In addition to the control objects, Virtual-Machines-MIB also provides monitoring objects. The main monitoring informations about VM’s are: (1) unique identifier (UUID), (2) name and version of the guest operating system, (3) name and version of VMM, (4) frequency and architecture of each virtual CPU (5) milliseconds used by each virtual CPU (6) free RAM (7) RAM in use, (8) total capacity of each storage device (9) used and available capacity of each storage device (10) CD-ROM and floppy drives, (11) kernel used by the guest operating system, (12) additional parameters of the guest kernel, (13) network settings, (14) network traffic and (15) HBA’s information.

A. Virtual-Machines-MIB Organization

The Virtual-Machines-MIB organization defines a subtree contained by the group VirtualMachines. This group contains, in the first level, nine tables and two subgroups, that are shown in Figure 1(a).

The VMs are listed in VirtualMachinesTable (Figure 1(b)), which contains more informations about each VM. VmUUID contains a unique identifier of the VM. The UUID was chosen as the VM’s identifier because it can uniquely identify a given VM even when managing a great number of VMs on several physical machines.

The object VMName allows reading and changing the name of the VM. VmOsIndex points to one entry in SupportOsTable (Figure 2(a)), that contains more data about the guest operating system. VmState allows reading and changing the state of the VM. VmConfFile contains the configuration file name of the VM, which can be a file with domain’s XML description [19] or a text file specifically formatted for each VMM. VmKernelIndex points to one entry in KernelTable (Figure 2(b)), that contains more data about guest OS kernel. VmRowStatus contains the status of each conceptual row of the table. This object is responsible by the creation and exclusion of VMs.

DiskImagesTable contains disk images files that can be used by virtual machines as storage devices. The Figure 1(c) shows the objects of DiskImagesTable.

StorageTable contains the disk images currently allocated to the VMs as storage devices. Each entry in this table builds a relationship between one virtual machine and one disk image, since disk images are the backend of storage devices. This table allows to connect and disconnect disk images of VMs, as well as replace an already connected disk image. The objects of StorageTable can be seen in Figure 1(d).

MemoryTable and CpuTable contains information about RAM and CPU usage at each VM. Figure 2(c) shows MemoryTable objects, and Figure 2(d) shows CpuTable objects. The objects memoryVmUUID and cpuVmUUID relates each entry of MemoryTable and CpuTable with one particular VM by its UUID.

The group VirtualMachineMonitor contains the name (VMMName) and version (VMMVersion) of VMM. HbaTable contains information about the virtualized host bus adapters connected to the VMs. NetworkTable contains information about the network connections of each VM such as IP and MAC addresses, interface throughput and others.

B. Use of Templates

The operations of creating a new VM and connecting a virtual disk on a VM are performed including new lines in certain tables. When adding a new line into a table, the values for each field of the new line should be informed. As it is not possible to inform the values of each field in the same SNMP message responsible for the inclusion of the new line, the fields initially receive the values stored in a template.

The operation of creating a new VM consists in changing the value of vmRowStatus of a virtualMachinesTable entry.
to the integer “5”, which corresponds to the operation createAndWait [26], what creates a new line in the table. The operation of connecting a virtual disk on a VM consists in changing the value of storageTable field storageRowStatus on the line corresponding to the desired VM to the integer “4”, corresponding to the operation createAndGo. These operations include one more line on their tables, however do not inform the values for all fields of the new line.

Thus, once the operation to create a new VM has been triggered, a line is added to the VMs table and fields like name, amount of memory, amount of CPUs and others are obtained from template. Likewise, when the operation of connecting a virtual disk on a VM has been triggered, a line is included in storageTable and the virtual disk name to be connected to the VM is obtained from the template.

Virtual-Machines-MIB stores templates in virtualMachinesMibTemplates group. The table virtualMachineTem플Table stores the templates for creating new VMs, the object virtualMachineTemplate stores which template for creating VM is currently in use, and the object storageTemplate stores the disk template to be connected to a VM.

Many entries can be inserted in virtualMachineTemplateTable, and the object virtualMachineTemplate will indicate what entry of VirtualMachineTemplateTable is currently being used as new VMs template. Also, it is possible to create new and modify existing templates. Once the new VM is created based on the current template, Virtual-Machines-MIB allows the manager to edit the configuration of the newly created VM in further operations, by changing the VM’s name, amount of memory, CPU’s and virtual storage units.

The object storageTemplate stores an integer that corresponds to the diskImageIndex field of diskImagesTable. This
field identifies an entry of diskImagesTable that is the template disk image currently in use, which will be initially connected to the VMs when a new storage device is created. Once the new storage device is created and connected to the template disk image, the manager can edit this entry using Virtual-Machines-MIB, making it point to the desired disk image that will be used by the VM.

IV. ARCHITECTURE TO IMPLEMENT VIRTUAL-MACHINES-MIB

This section describes the architecture to implement Virtual-Machines-MIB. The implementation architecture of is based on an SNMP agent, responsible of obtaining information and perform the operations. The external management station must communicate with the agent through the SNMP protocol. The SNMP agent must obtain the information described in the Virtual-Machines-MIB from various sources, such as the host OS API and the VMM API, the libvirt, and the XML files with the templates for creation of VMs. The architecture described is shown in Figure 3.

The libvirt programming library [19] is an application programming interface to create virtual machines management tools compatible with various VMM’s. Libvirt was used in this work to implement the interactions between Virtual-Machines-MIB objects and the virtual machines monitor.

Libvirt currently supports multiple virtual machine monitors such as KVM, Xen, QEMU, Virtual Box and VMWare. It introduces a middleware that interacts with each supported VMM and provide a standardized interface for virtual machines management applications. Libvirt is divided into two parts: one is independent of the VMM, and other is VMM specific. The VMM specific part is composed by drivers. Thus, for each managed VMM, there must be in libvirt the corresponding driver. The applications use the libvirt public API, which internally maps to the appropriate driver.

The libvirt’s goal is to provide a common layer, stable enough to safely manage virtual machines and, if possible, remotely [19]. Libvirt contains the management functions offered by each supported VMM, such as provisioning (installation of guest operating system), starting, stopping, creating, modifying, monitoring and migrating of virtual machines.

Libvirt API is used to obtain the most part of Virtual-Machines-MIB data, but some tables doesn’t rely on libvirt. The data of SupportedOsTable, that contains information about guest OSs, and KernelTable, which contains kernel data of each guest OS, must be obtained from other sources, such as the VMM API.

The tables NetworkTable and HbaTable also have objects that can not be obtained by libvirt, being necessary to obtain them from other sources that vary according to the VMM in use, such as VMM or host OS API.

Other Virtual-Machines-MIB features also does not have an equivalent libvirt function. For example, the VMName object, of virtualMachinesTable, allows to change the name of a VM, and libvirt doesn’t have a function to change a VM’s name, but it has functions that assists this task. To implement this feature, the VM’s XML description must be obtained using the libvirt function virDomainGetXMLDesc. Libvirt defines the XML format that describes a VM in [19]. After obtaining the XML, it must be parsed and the VM’s name changed, so the VM must be redefined with the new XML description. The redefinition of a VM with a new XML is done using the function virDomainDefineXML with the new XML as parameter. With this method, the name change will only take effect at the next boot of the VM.

Libvirt also does not offer a function to list the existing virtual disks of a VM, a required information to obtain the storageTable entries. The method used to obtain this information is to get the XML description of each VM using the function virDomainGetXMLDesc and use a XML parser to obtain the storage devices from XML data.

The templates for creation of VMs, stored in table virtualMachineTemplateTable, are derived from XML files stored in the host. These XML files describe a virtual domain in the XML format specified by libvirt. Virtual-Machines-MIB parses the XML files and inserts one template in virtualMachineTemplateTable, which can be edited by SNMP “SET” operations.

V. EXPERIMENTAL RESULTS

Virtual-Machines-MIB was experimented by installing the extended SNMP agents in two machines capable of hosting VMs, one with KVM and another with Xen. Both VMM’s were managed from the management station using traditional unmodified SNMP tools of the NET-SNMP package. The architecture used for the experiment is shown in Figure 4.

To demonstrate the management methodology to be adopted when using SNMP and Virtual-Machines-MIB, the following experiments were performed: (1) get the name and version of VMM, (2) list the VMs on a physical machine, (3) create a
new VM, (4) change the name of the newly created VM, (5) change the amount of RAM of the newly created VM, (6) connect a given virtual disk in the newly created VM (7) start the newly created VM, (8) insert another virtual CPU in the newly created VM, (9) delete the newly created VM.

The first experiment is to obtain the name and version of VMM. Table I displays the results in KVM and Xen.

The second experiment is to list the VMs on a physical machine. VirtualMachinesTable lists the existing VMs on a given host, its content is shown on Table II. A new VM can be created by changing the value of vmRowStatus object in any line of the virtualMachinesTable to the integer value “5” (createAndWait). Thus, a new VM is created based on the template settings (virtualMachineTemplateTable).

After the new VMs were created, the next experiment is to change the name of the newly created VM. Initially, they are named as defined in the field domainTemplateName of virtualMachinesTable. Next, this name will be changed to “vm-debian-1000” on KVM and “hvm-xen-1000” on Xen. Table II shows virtualMachinesTable with the highlighted lines corresponding to the newly created VMs after the name change.

The next experiment is to change the amount of RAM of the newly created VM. The memoryTable content is listed on Table III, where the entries corresponding to the newly created VMs are highlighted. To change the amount of RAM, the object memoryTotalMb of the corresponding memoryTable entry must be set to the desired amount of RAM in megabytes.

After changing the amount of RAM, the next experiment is to connect a given virtual disk in the newly created VM. This activity is done through storageTable, that lists all VM’s storage volumes. Initially the newly created VM is connected to the disk image indicated by the template. It must be replaced by the disk image that really should stay connected. The content of storageTable is shown in table IV, where the entries corresponding to the newly created VMs are highlighted. To replace their disks, the value of the column storageDiskImageIndex must be replaced by the value of diskImageIndex of the desired entry of diskImagesTable.

The next experiment is to start the newly created VM. To change the state of a given VM, the value of the object vmState of virtualMachinesTable must be changed to the desired state. The vmState object supports the following values: 1 (defined), 2 (running), 3 (blocked), 4 (paused), 5 (shutdown), 6 (shutoff) and 7 (crashed). To start the newly created VMs, the vmState column values in Table II must be set to 2 (running) in the corresponding entries.

The next experiment is to insert another virtual CPU in the newly created VM. The CPUs of all VMs are listed in cpuTable. The newly created VM contain initially one virtual CPU, as indicated by the template. To insert one more virtual CPU in the newly created VM, the value of cpuRowStatus of the corresponding cpuTable entry must be changed to the value 4 (createAndGo). The table V shows the content of cpuTable with the entries corresponding to the newly created VM’s highlighted after inserting one additional CPU on each VM.

The last activity is to delete the newly created VM. To delete a VM, the corresponding row of virtualMachinesTable (Table II) must be removed.
TABLE II
CONTENT OF VM-MACHINES TABLE ON KVM AND XEN, WITH THE NEWLY CREATED VMs AFTER NAME CHANGE.

<table>
<thead>
<tr>
<th>KVM Output</th>
<th>Xen Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmId</td>
<td>vmUUID</td>
</tr>
<tr>
<td>1</td>
<td>7cae0a2c</td>
</tr>
<tr>
<td>2</td>
<td>a0216fad</td>
</tr>
<tr>
<td>3</td>
<td>05c6bf0e</td>
</tr>
<tr>
<td>4</td>
<td>945c6a0e</td>
</tr>
<tr>
<td>5</td>
<td>0be03b74</td>
</tr>
<tr>
<td>6</td>
<td>f4d32541</td>
</tr>
<tr>
<td>7</td>
<td>ff74ed37</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>KVM Output</th>
<th>Xen Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>cpuVmUUID</td>
</tr>
<tr>
<td>1</td>
<td>ff74ed37</td>
</tr>
<tr>
<td>2</td>
<td>ff74ed37</td>
</tr>
<tr>
<td>3</td>
<td>a0216fad</td>
</tr>
<tr>
<td>4</td>
<td>0be03b74</td>
</tr>
<tr>
<td>5</td>
<td>f4d32541</td>
</tr>
<tr>
<td>6</td>
<td>7cae0a2c</td>
</tr>
<tr>
<td>7</td>
<td>5c6bf0e</td>
</tr>
<tr>
<td>8</td>
<td>945c6a0e</td>
</tr>
</tbody>
</table>

TABLE V
CONTENT OF CPU-MACHINE TABLE ON KVM AND XEN.

II) must be destroyed. This operation is performed by changing the value of the object `vmRowStatus` on the corresponding entry to the integer “6” (destroy), therewith the VM are deleted and its resources are freed.

VI. CONCLUSIONS

This work presented a strategy to centralize the management of multiple virtual machines monitors. Virtualization is a technology that allows more than one operating system instance to run on the same machine at the same time. Recently the cloud computing, a new paradigm of organization and delivery of services, risen in the Internet. Virtualization is the foundation of cloud computing. The adoption of cloud computing increases the interest in integrating virtual infrastructures, that often are built on different virtual machine monitors.

The main VMMs on the market provide their own management applications that are specific tailored manage only their own VMM. Another problem is that the VMMs do not offer a common and standardized management protocol or interface.

The solution presented in this paper uses the SNMP to perform the management of VMs, including monitoring and controlling. In order to allow a centralized monitoring and controlling, it presents a MIB called `Virtual-Machines-MIB`, which defines a standard interface for the management of VMs. The proposed MIB was implemented based on the NET-SNMP agent and on the `libvirt` API. Finally, experiments were
performed which consists in managing virtual machines hosted by KVM and Xen with ordinary SNMP management tools.

Future work includes the addition of new management features to the MIB, for instance disk images creation and management, boot order change and virtual networks management. The support of other VMMs also must be performed.

REFERENCES