Running KVM Virtual Machines in Batch Systems

Iustina Melinte, Alexandru Bujor
Automatic Control and Computers Faculty
University POLITEHNICA of Bucharest
Splaiul Independenței nr. 313, Bucharest, Romania
{*iustina.melinte,alexandru.bujor*}*@cti.pub.ro*

February 8, 2013

Abstract

This paper describes a framework for deploying virtual machines as jobs in a batch system. It also adds management features and the concept of template machines that can used for efficiently dispatching new instances. The technology used for virtualization is the open-source Kernel-based Virtual Machines supported by Red Hat, since the Qcow2 disk format provides important support for templates. The document describes also how the infrastructure controlled by this framework can be used for testing and evaluation, as special security policies are required for those purposes.
Keywords: ebtables, iptables, KVM, python, Oracle Grid Engine

1 Introduction

Virtualization became an interesting topic in the last few years as prices on hardware resources started to be more acceptable and specialized software solutions provided better performance, diminishing the virtualization penalty. The new hardware technologies like Intel VT-x and AMD-v further improved the performances of virtual machines, making them an affordable and cost-effective choice for many architectures.

The purpose of the current project is to develop an architecture and a framework for supporting labs, practical exams, testing (possibly Q&A) and other activities that can benefit from a virtualized infrastructure. This paper shows that a thin client architecture with virtual machines running in a computer cluster is more flexible and cost effective than using many dedicated physical machines. Tests performed showed that students use only a very small percentage of the computing power available in their VMs during normal labs and exams, hence overcommitting cpu can be consider to improve cost effectiveness.

When discussing virtualization, there are two main components: the host (the phycsical machine) and the guest (the virtual machine). The software layer that separates them is the hypervisor. There are two major hypervisor types:

The Type-1 Hypervisor (also known as Bare Metal) is a thin software layer that is installed over the hardware components and provides services for the running virtual machines. Basically, it is responsible for security, resource administration and management services. This type of hypervizor is usually a specialized kernel (microkernel) without

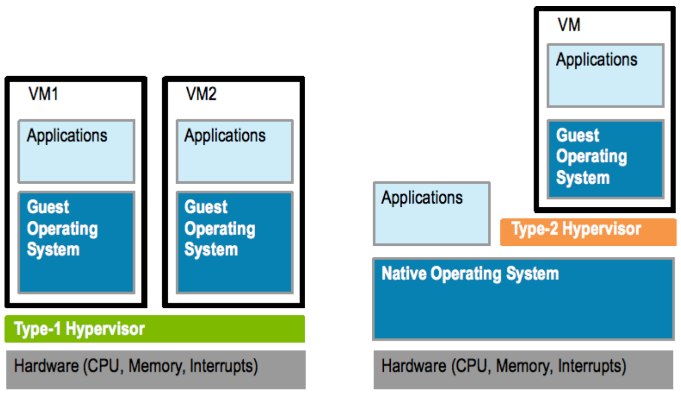

Figure 1: Types of hypervisors [3]

too many additional functionalities. There are few examples of well-known technologies that use this approach: Windows Hyper-V, Oracle XEN and VMware ESX. Some of the advantages of this hypervisor type are presented here [5], one of the most important being the higher performance when compared to the second type. This is the reason why bare metal hypervisors are used mostly in production environments.

The second type is mostly used for paravirtualization or software virtualization. The type 2 hypervisor runs virtual machines as processes over a full-featured operating system. This provides flexibility and allows the administrator to user process-based administrative policies for VMs. Tools performing type 2 virtualization are general purpose tools for testing and labs environments and usually rely on the software to emulate hardware, networking, etc. Examples of applications are: VirtualBox, VMware player and Qemu-KVM (or simply KVM).

Kernel-based Virtual Machine (KVM) is the default virtualization technology starting with version 6 of Red Hat Enterprise Linux. Supported by Red Hat, a proeminent north-american vendor, it was released in february 2007, along with the 2.6.20 Linux Kernel version. Although it is not a completely stable tehnology, KVM is the virtualization core of many cloud solutions, providing many of the required functionalities. The main reason is that KVM is an open source project, hence it is free to integrate it in any other software environment. However, the fact that it is free doesn't mean that performance is an issue, since recent tests show that it is at least as good as the more experienced XEN supported by Oracle [1].

Even if KVM is considered a Type-2 hypervisor, it has an interesting trait: it works only over hardware with support for virtualization, thus delivering performance similar to type-1 hypervisors. Hence not all physical machines can run KVM. Before considering this virtualization technology, one must check the output of the /proc/cpuinfo virtual file. Look for vmx (Intel) or svm (AMD) flags in the output to certify that the current physical machine is capable of hardware virtualization. Once the hardware support is activated, the host Linux OS must load two kernel modules to make use it: kvm and kvm_intel/kvm_amd. The first module is required for activating general KVM support in kernel, while kvm_intel or kvm_amd represent the actual driver. Communication between user-mode emulator (qemu-kvm) and the kernel is performed through the /dev/kvm device.

One of the advantages of KVM and typical for type-2 hypervisors is that virtual machines run as processes. Hence it is possible to dispatch many VMs (processes) over a batch system, thus providing better hardware usage and improved reliability. This approach is similar to the idea of running a cloud over a grid.

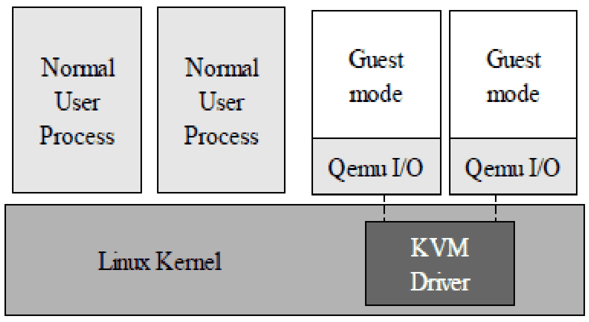

Figure 2: KVM hypervisor [7]

KVM is the backend for many infrastructure as a service solutions. As it part of the software family provided by Red Hat, it can easily be integrated with Clustering and High Availability tools provided in RHEL operating systems and RHEL-based clones like CentOS and Scientific Linux. OpenStack project integrates those technologies in an open-source cloud solution. Enterprise level application also use KVM, one good example being the Red Hat Enterprise Virtualization suite. IBM is another important vendor watching this technology as it already integrated it in ”IBM Smart Cloud and provisioning” product. [2]

The qemu-kvm package deploys the hypervisor and very basic components on the host system, just enough for running VMs. The ”Virtualization Tools” and ”Virtualization Client” groups offer additional tools for virtual machine management. The backbone for all these tools is libvirt.

Developed as a standardized management interface for virtual machines running over Xen, VirtualBox or KVM, libvirt is considered the recommended option for KVM administration. Libvirt represents two major components: the libvirtd daemon and the libvirt API. There is also an internal database containing information about defined virtual machines and more information is available by consulting files in /etc/libvirt folder. The daemon is started/stopped using the service command and can listen to connections from clients on unix sockets and standard TCP/IP sockets, depending on configurations. Clients using the API simply connect to the daemon and issue the commands available (examples: start/stop vm, migrate vm, redefine resources, etc.).

Note that with libvirtd configured to listen on TCP/IP sockets, remote virtual machine management is possible. This option is important for our project, as it allows clients to control VMs running on specific nodes in the batch system.

Two important tools for virtual machine administration are virsh and virt-manager. Virsh is a text-based tool and is very usefull when there are not X Server components installed on host system. It can perform a lot of tasks once a connection with the libvirtd is opened and had context sensitive help included. On the other hand, the Virtual Machine Manager (virt-manager) is performing administrative tasks using a graphical user interface designed with GTK and GLADE.

Virt-manager requires X11 Window System, as it is a graphical tool. However, it is possible to install only a minimal set of X11-related libraries on a remote server, if it is tunneled through a Secure Shell connection. It also requires a connection to a libvirt daemon, either a local or remote. As mentioned, it is possible to use TCP/IP networking to connect to another host and control VMs running on that machine.

In fact, this tool is simply integrating some features provided by libvirt and kvm in a graphical user interface. The application connects to a daemon (default to the daemon running on the localhost) and performs administratice tasks using the libvirt api.

The management tools presented work well for simple systems with virtual machines that are pinned on physical nodes. However, the dynamic scheduling performed inside a cluster does require storing additional information regarding the way that processes (virtual machines) are dispatched. Next chapters will show why the data stored by libvirt is just not enough for full control.

1.1 Oracle Grid Engine

OGE is a distributed resource management (DRM) system typically used high-performance computing clusters. It uses the concept of "jobs" to define a segment of work, which can be a system process, for example. Other terms that describe the operation of OGE are: queues, host groups, qmaster and execution hosts[4]. The central component of the OGE architecture is the qmaster: it runs on a single host, accepting incoming jobs and assigning resources. The execution hosts are the machines that actually run the jobs. They offer a certain number of job slots, usually determined by the number of CPU cores.

Users and administrators interact with the system through command line utilities such as qsub, qdel, qstat and so on. When a job is submitted, it enters a queue until it is dispatched to an available execution host. The process standard out and standard error are redirected into files in the current working directory, by default.

1.2 Virtual machine security

Before thinking about security in a virtualized environment, we must first define its components.

A virtual network is made up of bridges and virtual interfaces (*tap*s). A Linux bridge is a layer 2 virtual device that forwards frames between local interfaces. It can be connected to a physical interface, or to *tap* devices. The latter ones can actually connect a virtual machine to a bridge, in our scenario.

A virtualized environment can be secured in many ways: enforcing Mandatory Access Control through the sVirt service, restricting access to resources (through control groups) to prevent DoS, protecting data with disk-image encrytion and isolating guests network communication.

In this section we will focus on guest isolation, which means restricting access to and from virtual machines, from the network point of view. The goal is to filter traffic between guests and also separate it from the host.

The main difference between virtual networks and physical ones relates to traffic processing: when many guests reside one the same machine, the physical port is shared. This means that all traffic exists through the same "door". Moreover, all communication that happens internally between guests on the same host is invisible to the outside world.

1.2.1 Using 802.1q VLANs

The first approach would be to isolate guests by making them members of different VLANs.

Applying this concept involves creating subinterfaces on the physical port and then bounding them to different bridges, one for each VLAN. The guests are then connected to the bridge through tap interfaces, as shown in Figure 1.2.1.

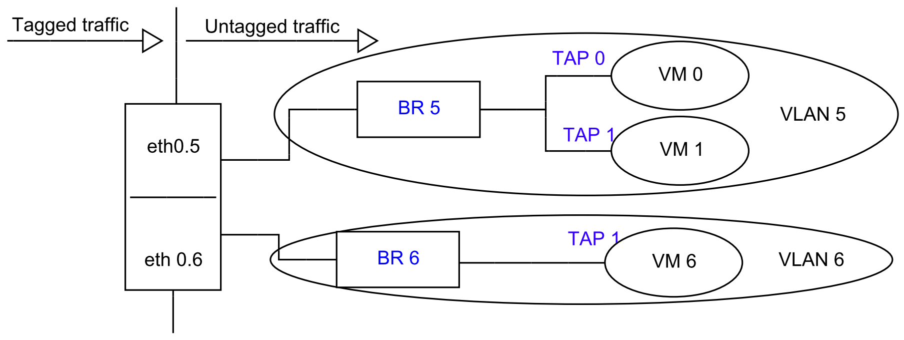

Figure 3: Isolating with VLANs

The main advantage of this method is that it allows the use of preexisting network access controls, the traditional way: the traffic between segments can now be filtered externally. But there are also disadvantages : limited scalability and administration overhead. The degree of flexibility is also limited, since the policies apply only to groups of guests.

1.2.2 Ebtables

Unlike iptables, this package can do layer 2 filtering. It can be used to prevent MAC address spoofing (by allowing only certain packets to exit a virtual interface), to prevent IP spoofing or to restrict any type of traffic between guests. Ebtables is the most flexibile choice for securing virtualized networks.

All the filtering can be done with a topology composed of only one bridge, tied to a physical interface and with all virtual machines connected to it.

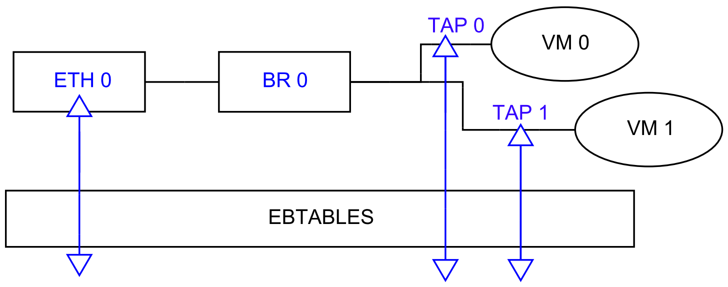

Figure 4: Isolating with Ebtables

As Figure 4 depicts, ebtables is more like a "bridge firewall": it can control the traffic that flows through tap devices too (the identification components used in filtering rules are colored in blue in the picture). For example, the following rules would deny MAC spoofing and IP spoofing for VM0 and VM1 (in this order):

```
# VM0
ebtables -A FORWARD -i tap0 -s ! de:af:de:af:01:01 -j DROP
ebtables -A FORWARD -p IPv4 --ip-src ! 192.168.1.1 -s  tap0 -j DROP
# VM1
ebtables -A FORWARD -i tap1 -s ! de:af:de:af:01:02 -j DROP
ebtables -A FORWARD -p IPv4 --ip-src ! 192.168.1.2 -s  tap1 -j DROP
```

If we wanted to deny all communication between virtual machines (allowing them to access only the Internet), we would use the following for each guest:

```
ebtables -A FORWARD -i tap0  -d ff:ff:ff:ff:ff:ff -j ACCEPT
ebtables -A FORWARD -i tap0  -d 00:15:5d:07:07:8b -j ACCEPT  # gateway MAC
ebtables -A FORWARD -i tap0  -d 00:04:23:c1:ec:52 -j ACCEPT  # DHCP server
ebtables -A FORWARD -i tap0 -j DROP
```

These rules will only allow the necessary traffic for obtaining a DHCP lease and communicating with the outside.

1.2.3 Other virtual machine isolation options

One of the first options we considered was OpenVPN. Each user should be assigned a certificate and an .ovpn configuration file for connecting to his own virtual machine. The procedure also required to allow data traffic from and to vm only through the TUN interface of OpenVPN. The main advantage of this solution was the secure link and the better audit possibilities, as each user was identified by his certificate. However, there were some major drawbacks. First of all was that the student would be able to start services on all interfaces after he connected to the virtual machine's shell, therefore additional complex iptables policies were required. Another problem was represented by the computational overhead for communication, as each packet transported by OpenVPN is encrypted and also processed in user-space. The conclusion was that this solution is not scalable and cannot meet our security requirements.

There is another isolation software suite that restricts access from within a virtual machine called sVirt. Based on SeLinux architecture, it can be used to restrict the access of the qemu-kvm process to specific resources. However, this is not very usefull for this project, as most of the policies are focused on network traffic, rather than on host's resources.

2 Architecture

The application components are depicted in Figure 2.

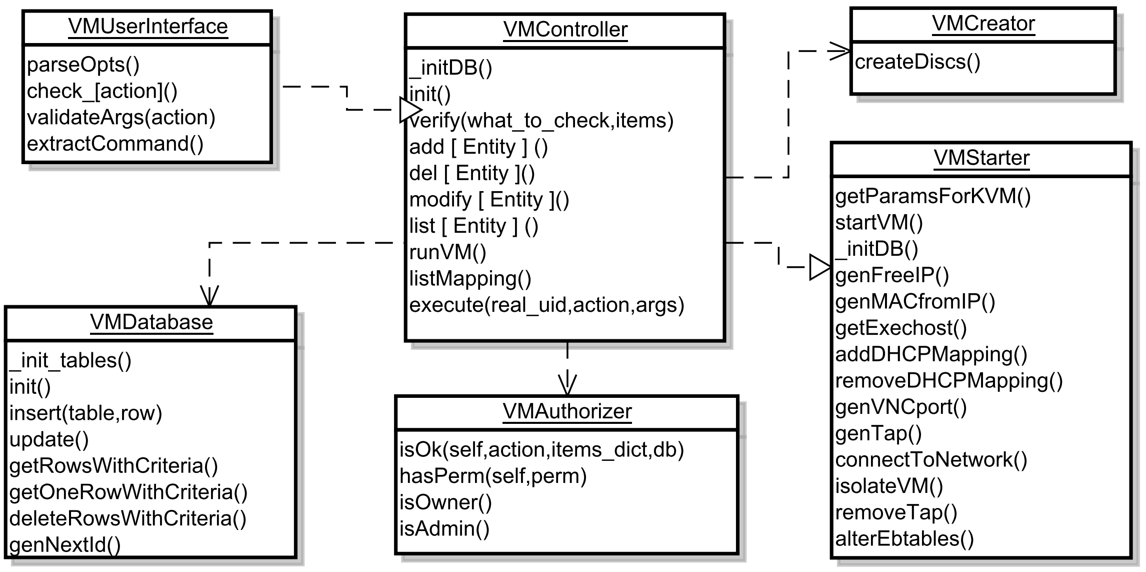

Figure 5: Modules

The VMUserInterface provides a CLI that allows users to interact with the framework. It identifies requested actions, validates input and then passes the information to the controller.

The VMController module is the main component. It uses the VMAuthorizer in order to determine if a user has the needed permissions to execute an action and it also manipulates the database for storing and retreiving information. The parameters of a new virtual machine are defined with using the VMCreator.

When the user wants to start a guest, a job is submitted through the Oracle Grid Engine. This process actually executes the code from the VMStarter component, which starts the virtual machine with all the defined parameters and connects it to the network.

The VMDatabase provides methods for executing CRUD operations with the data. The tables are depicted in Figure 2.

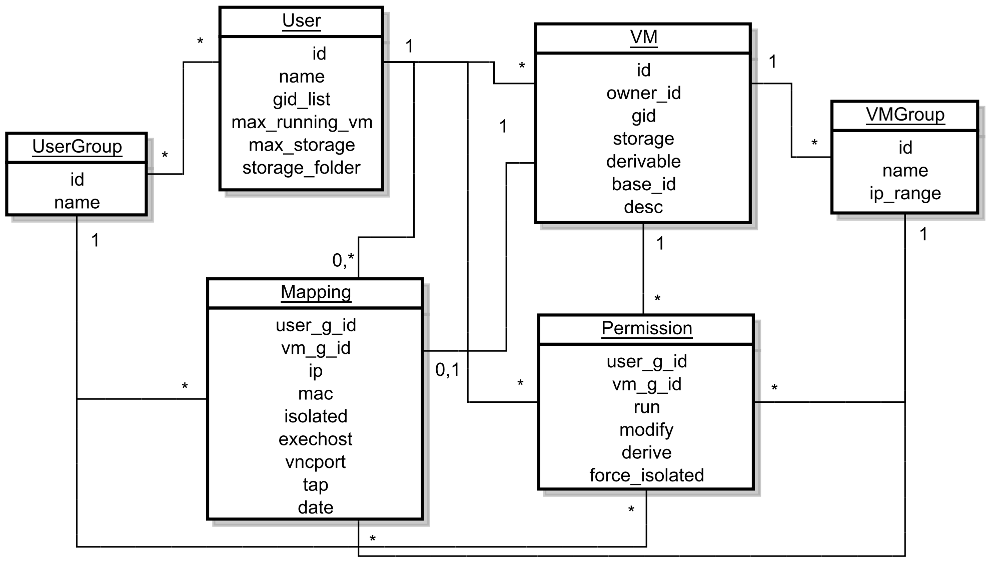

Figure 6: Database

Each user can belong to one or more user groups. A virtual machine, by contrast, can belong to only one VMGroup, since the subnets assigned cannot overlap. This allows for easier management of multiple guests and users, with different access policies.

Permissions represent authorized operations. They can be assigned to a single user or a group, for a single guest or a virtual machine group.

The Mapping table contains information about all active guests on any execution host, the resources they use and how they can be accessed.

3 Implementation

The application code is written in Python. For interacting with the database we used the sqlite module, since it provides the simplest and most flexible API to a SQLite relational database.

As we said earlier, the VMUserInterface is the only module that interacts directly with the user. It identifies the requested action and valides input by using regular expressions, verifying paths, file formats and option combinations.

The available actions include doing basic operations on the entities that are modeled in the database: creating, deleting, modifying and listing users, groups, virtual machines, permissions and mappings. There is a predefined group called ”admin” and only its members have full control over the application (managing users, permissions and groups).

When a new virtual machine is defined, new discs are created with the specified sizes. The user can also define it as a template, which means that whenever someone uses it as a base, differential discs will be created. This is a very useful feature called ”copy-on-write”: the derived disc file stores only the changes that are made to the base image, which helps to preserve space.

Starting a guest implies submitting a job to OGE and going through some steps.

When the job is assigned to an execution host and enters the running state, the VMStarter first allocates the resources needed. It looks for a free IP address in the

VMGroup's ip range and determines the MAC address using a simple convention: the last 16 bits are converted to hexadecimal. After this, a fixed-address lease is inserted in the DHCP server configuration, through the OMAPI protocol. The next step involves creating the (virtual) network connection: a tap interface which is connected to the bridge. At this point, some ebtables rules for filtering traffic are also created. These help to prevent MAC and IP address spoofing, and also deny any type of layer 3 communication between 2 different hosts (for example, during a lab exam the students shouldn't be able to collaborate). After doing the isolation steps, the VMStarter spawns 2 processes: one that actually executes the 'qemu-kvm' command and one that monitors the process until it exits.

While the guest is running, an entry exists in the Mapping table which identifies the the actual resources that the process uses, the network parameters, the execution host where it resides and the VNC display to access it. When the vm stops, the monitoring process detects this, frees up the resources used and also deletes the mapping.

The normal user can always asign permissions to other users, but only for the guests that he owns. For example, he can allow other users to start, modify or clone its machines. But he cannot see any information related to the another user's activity. Only users from the "admin" group can see and control everything that happens in the system. A guest can also be "forced" to run only isolated, without giving the user this choice.

4 Experimental Setup

The application can be tested on a system that has support for virtualization (this can be checked by looking for "vmx" in the /proc/cpuinfo file). The following packages are needed (we used local DHCP and NAT to test connectivity, but it is not necessary to have these two on our host):

```
dhcp
bridge-utils
qemu-kvm
qemu-kvm-tools
ebtables
```

The DHCP server configuration contains the definition of our subnet and the omapi key:

```
### /etc/dhcp/dhcpd.conf
omapi-port 9991;
key omapi_key {
        algorithm HMAC-MD5;
        secret "<encrypted_key_here>";
};
omapi-key omapi_key;

subnet 192.168.100.0 netmask 255.255.255.0 {
        option routers                192.168.100.1;
        option subnet-mask            255.255.255.0;
        option domain-search            "grid.pub.ro";
        option domain-name-servers     141.85.224.15;
        range 192.168.100.2 192.168.100.100;
}
```

The OMAPI shared key is used for inserting leases programatically and remotely.

The virtual machines will be connected to a bridge that can be created with the following command:

```
brctl addbr vbr0
```

5 Scenarios and Results

This chapter focuses on two main topics: using the framework and measuring solution's scalability. First of all, there will be a walkthrough describing how to use the framework. The paper presents what system settings should be performed and the actual commands that are required for some scenarios.

5.1 Scenarios

In the following paragraphs, we will use the short name vm for "virtual machine" (since this term will be used very often) First, we need to add a few system users and a group.

```
groupadd vmusers
for i in {badmin,doraz,alex}; do
groupadd \$i;
useradd -m -d /home/\$i -g  \$i -s /bin/bash  \$i;
usermod -aG vmusers \$i; done
```

In order to allow the users to use this tool, we created a "wrapper" in C, with *execute* and *setuid* permissions:

```
chmod 4755 vmapp
```

The first thing to do is to initialize the database. This can only be done by a preconfigured user. (user 'badmin' with id 503,in our case)

```
su badmin
sudo /root/ui.py --init --dbpath /root/vm.db
```

At this moment, two user groups and a vm group are created: the "admin" group (id 0), the "all_users" group (id 1) and "all_vms" (the default group for virtual machines, having the ip range "192.168.0.1-192.168.0.254"). The user that was configured as a default admin is also inserted in the database, and added to the "admin" group.

Now let's imagine that the admin wants to define a new virtual machine template and allow the users in a certain group to derive it (by making differential copies of the discs). He also wants to isolate all virtual machines that will have this template as a base (in an exam, for example). The following commands create a user group and adds the system users we created earlier.

```
vmapp --ugadd --name studs --ugid 10
vmapp --uglist
-------------------------------------------------------------------------------
id      name
-------------------------------------------------------------------------------
0       admin
1       all_users
10      studs
```

```
vmapp --uadd --name doraz --uid 502 --ug studs --stordir /home/doraz
vmapp --uadd --name alex --uid 504 --ug studs --stordir /home/alex
vmapp --ulist
-------------------------------------------------------------------------------
uid   name     ip_range  gid_list   max_running_vm max_storage storage_folder
-------------------------------------------------------------------------------
503   admin1              (0,)       1000            133G     /home/me
502   doraz               (1,10,)    1000            133G     /home/doraz
504   alex                (1,10,)    1000            133G     /home/alex
```

These 2 users will have some predefined restrictions: a maximum storage value of 13G and the ability to run 2 vms at most simultaneously. They have also been added to the "studs" group.

Now we define the vm group with the desired IP range for this lab:

```
vmapp --vmgadd --name lab1 --vmgid 2 \
--iprange 192.168.100.2-192.168.100.254
vmapp --vmglist
-------------------------------------------------------------------------------
id     name                           ip_range
-------------------------------------------------------------------------------
1      all_vms                        192.168.0.1-192.168.0.254
2      lab1                           192.168.100.2-192.168.100.254
```

A new vm can be defined in multiple ways: by deriving a template vm, by cloning some existing discs (by copying the files) or by creating new (empty) discs. The commands are the following:

```
vmapp --vmadd --name gsr_vm --vmg lab --stor 5G, --mem 2G  --derivable
vmapp --vmadd --name gsr_vm --vmg lab \
--usediscs /path/to/disc1.qcow2,/path/to/disc2.qcow2 --derivable
vmapp --vmadd --name gsr_vm --vmg lab --base another_defined_vm
```

We actually need to use the first or the second one, since we cannot make a template from a differential disc (made with the –base option). If we created new discs, we must install an operating system on it:

```
vmapp --vmrun --vm gsr_vm --install \
--cdrom /tmp/SL-63-x86_64-2012-08-24-LiveCD.iso --isolate --mem 2G
```

What actually happens in this step will be explained in the next paragraphs. For now, let's assume that the template's discs are already bootable. This is how it looks in the database:

```
vmapp --vmlist
-------------------------------------------------------------------------------
vmid  name   owner_id  vm_group  derivable  base_id   desc
storage
-------------------------------------------------------------------------------
1001  gsr_vm    503      2       1           0
{'/tmp/504_gsr_vm_0.qcow2': 5000}
```

The next step is to allow the users from the "studs" group to derive this vm, and also force its isolation.

```
vmapp --permset +d,+i --ug studs --vm gsr_vm
vmapp --permlist
------------------------------------------------------------------------------
uid   vmid run  modify derive force_isolated
------------------------------------------------------------------------------
503   1001 1    1      1      0
10    1001 0    0      1      1
```

Adding a vm also inserts permissions in the database for that specific user and vm, as seen in the output (the owner has "full" permissions over his vms). Other permissions can also be added for individual users or for vm groups, for more flexibility, but it's not necessary in this case. For example, the admin could allow a certain user/group to modify or run a vm/vm group. One important thing to note here is that only an user from the "admin" group can modify permissions. Users can only see their permissions (or those of the groups that they belong to).

After this step, any user from the "studs" group can now derive the vm template:

```
su doraz
vmapp --vmadd --name myvm --base gsr_vm
vmapp --vmlist
------------------------------------------------------------------------------
vmid  name       owner_id   vm_group  derivable   base_id   desc
storage
------------------------------------------------------------------------------
1001     gsr_vm  503        2         1           0
{'/tmp/504_gsr_vm_0.qcow2': 5000}
1002     gsr_vm  502        2         0
{'/home/doraz/502_myvm_0.qcow2': 5000}

vmapp --permlist
------------------------------------------------------------------------------
uid   vmid run  modify derive force_isolated
------------------------------------------------------------------------------
10    1001 0    0      1      1
502   1002 1    1      0      1
```

If the user would have tried to derive the vm or run it without having permissions to do so, we would have received an error messaje, such as: "You don't have permission to derive this vm" (or "run this vm"). As seen in the output, deriving a vm also does some additional steps behind. It creates a differential disc from the Qcow2 base image and inserts permissions for that specific user and vm in the database. The vm also becomes part of the same vm group as the base. Now the user can run the guest:

```
vmapp --vmrun --vm myvm --mem 2G
You don't have permission to run this vm unisolated.
vmapp --vmrun --vm myvm --mem 2G --isolate
Your vm is accessible via host quad-wn29.grid.pub.ro vnc port 5901, display 1
vmapp --maplist
------------------------------------------------------------------------------
uid    vmid  ip               mac                 isolated  exechost
```

```
vnc   tap   date
-------------------------------------------------------------------------------
502    1002  192.168.100.2   de:af:de:af:64:02  1        quad-wn29.grid.pub.ro
5901  tap1  2013-01-22 16:44:51
```

There are many steps that happen behind the scenes with this command. First, the controller checks if the user is authorized to run the vm. If the ”–isolate” option is not provided, an additional check is made to see if any combination of user/user group and vm/vm group with the ”force_isolated” permission exists. If it does, then the user is not allowed to run the machine unisolated. After making these verifications, the controller proceeds with defining the vm instance parameters: it finds the next available IP from the vm group subnet, maps it to its MAC address and inserts a new entry in the DHCP leases file, through the OMAPI protocol. It then submits a new job through the ”qsub” command, which calls the VMStarter. At the same time it creates a new process that monitors this job (by reading its output file). When the job starts to run on an exechost, it defines the resources it needs (the vncport and the tap interface) and then starts the actual kvm process, which is the vm instance. All this information is inserted in the database mappings, to let the users know how to access their machines. The monitoring process will detect when the vm is stopped and will delete its mapping entry. If the ”–isolate” option was also provided, then the vm will be both protected and restricted from MAC/IP address spoofing and any traffic between virtual machines on that host will also be dropped.

5.2 Resource Usage

Now we will focus on measuring scalability. The solution designed can be used for homework testing purposes. During the 1st December 2012 and 23rd December 2012 approximately 300 students were required to submit their homework for ”Local Area Networks” course for grading. A group of 14 dual six-core Opteron servers were reserved for this task and performance data was gathered using cacti and SNMP.

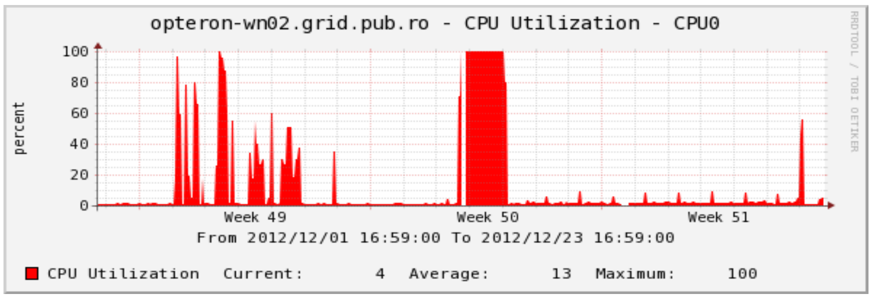

Figure 7: Cpu0 usage - homework uploading and grading

The previous graphics show that there were few periods when CPU was heavily loaded, but the average percentage was 13%.The rest of the cores had similar footprints. The memory graphic describes more accurate the load caused by running virtual machines and show why this is the real bottleneck. The final week of the assesment added significant load on system’s memory with few 100% peaks. This was somehow expected, as most of the students submit their homework only few days before the deadline.

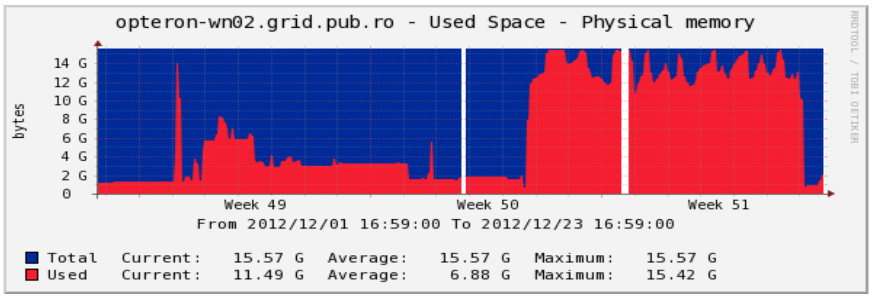

Figure 8: Physical memory usage - homework uploading and grading

On 12th January 2013 the same infrastructure hosted a practical exam for 300 students. The following graphics show resource usage for opteron-wn02.grid.pub.ro host:

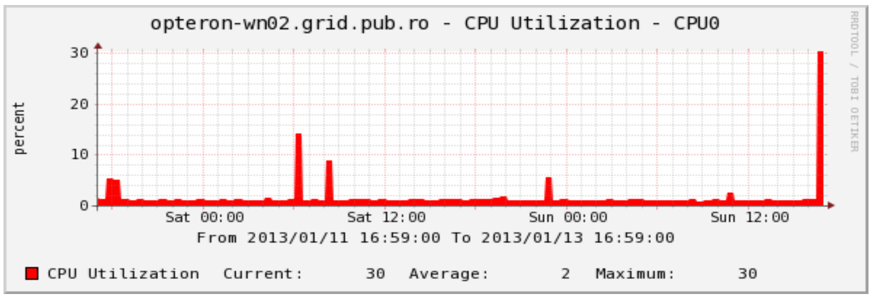

Figure 9: Cpu0 usage - practical exam

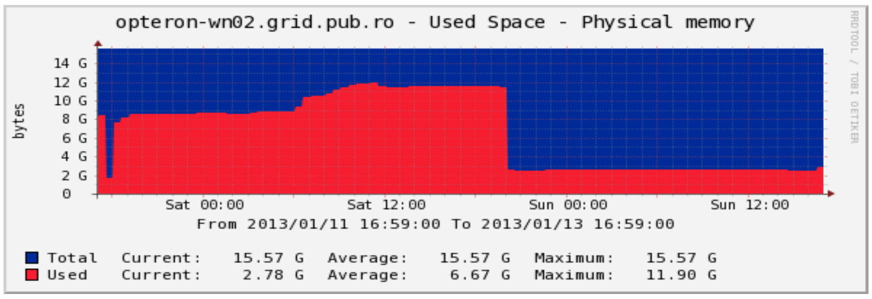

Figure 10: Physical memory usage - practical exam

From the graphics above it is obvius that 12 cores can easily handle test virtual machines that use more than 12 GB of RAM. These experiments, along with this survey [6] confirm that KVM is a good technology at least for testing purposes.

6 Conclusion and Further Work

The framework provides a simple way to configure and run virtual machines in a cluster environment, by using batch jobs with Oracle Grid Engine. This allows the decoupling of logical network resources from the underlying hardware, which provides scalability.

By introducing the concept of *template virtual machines*, the framework provides a fast way to provision preconfigured guests. This feature is suported by the Qcow2 disk image format which allows the creation of differential discs (which are linked to a read-only base image).

This tool can be used for testing in lab environments and for teaching purposes (networking and operating systems courses, for example).

The security features allow the isolation of virtual machines at OSI layer 2 and above, which means complete control over the traffic flow and protection against network attacks (IP/MAC spoofing). The rules, created with *ebtables*, can deny communication between guests on the same node and also between different nodes (execution hosts in OGE terminology).

By providing a security-aware solution, this framework can also be used in practical exams or for certifications.

The application in its current stage provides only the basic architecture, allowing the addition of new functionality. One option for improvement is to use *libvirt* features: defining virtual machine domains and providing more advanced features such as suspending guests, making snapshots and even allowing the user to define his own network.

Acknowledgment

The authors would like to thank Razvan Dobre and Razvan Rughinis for their support. We also mention that resources provided by Automatic Control and Computers Faculty (servers, networking equipments) were very useful for the project.

References

[1] A. Cherci and R. Veraldi. A quantitative comparison between xen and kvm, 2010.

[2] I. IBM. IBM Virtualization Products Guide. http://www-03.ibm.com/systems/virtualization/infrastructure/open/offerings.html.

[3] microkerneldude.wordpress.com. Much Ado About Type-2 Hypervisor, 2010.

[4] I. Oracle. Begginer's Guide to Oracle Grid Engine 6.2, 2010.

[5] techtarget.com. Virtualization Hypervisor Comparasion Type1 vs Type2.

[6] tonian.com. Storage and Virtualization among the Red Hatters. http://www.tonian.com/wordpress/?p=44, 2012.

[7] virtualization station.blogspot.ro. KVM Hypervisor integrated in Linux, 2008.