Managing Large Networks of Virtual Machines

Kyrre M Begnum – Oslo University College, Norway

ABSTRACT

As the number of available virtualization tools and their popularity continues to grow, the way in which virtual machines can be managed in a data center is becoming more and more important. A few commercial tools such as VMware ESX and XenEnterprise exist, but they are limited to a certain virtual machine technology and offer no way to expand the tool's capabilities to local needs. This paper discusses an open source management tool, MLN, for large virtual networks transparent of their virtualization platform. The current version supports the two popular open source virtual machine packages: Xen and User Mode Linux. MLN uses an extensible configuration language for the design of the virtual machines and their internal configuration. Large groups of virtual machines can be managed as logical groups. We present a web-server hosting scenario and a on-demand render farm as case studies to show the usefulness of our tool. The text concludes with a short discussion on the difficulties of offering abstraction to virtualization platforms.

Introduction

With the growth in numbers of virtualization platforms, the system administrator will face significant challenges in getting them to work together. Most of the platforms that are popular today have their own areas of application. For example, User-Mode Linux does not require root access to install and can mount folders on the physical host as partitions. Xen has impressive performance and is easy to connect to the LAN. VMware offers graphical front-ends for VM creation and management.

Consider a data center that hosts virtual machines for third parties or a university IT department that provides virtual laboratories for their students.

Ideally, one should be able to choose the virtualization platform that suits the task best. However, the management interface of the platform is currently so tightly connected to the given product that it is impossible to make a decision about the platform without also considering how the virtual machine should be managed.

Although virtualization platforms usually offer a simple management interface, they are often difficult for non-system administrators, e.g., as teachers, to use. Further, these tools offer limited support for specifying attributes inside the virtual machine, such as users or network setup. Once the virtual machine is running, the system needs to be configured additionally by hand, a process that is known to be error prone.

Furthermore, if a user wants to manage many virtual machines, she needs to be able to group virtual machines together in a meaningful way and to manage virtual machines spread out on several servers. The free management tools scale badly because they are aimed at running a few virtual machines on a single host. She must buy a very expensive, more advanced tool in order to still have control over multiple virtual machine hosting servers. The question of how to manage virtual machines on a large scale is therefore a matter of cost and design tool rather than platform and capability.

Managing VMs also inevitably overlaps with specifying and implementing their configuration. For example, a system administrator is given the task to design and configure a cluster of 50 nodes in a render farm running entirely on virtual machines. A minimal version of the same cluster also has to be build from the same specification for testing. Similarly, a web-hosting company wants to consolidate several customers onto the same physical network using groups of virtual machines that belong to each customer. How can we minimize the drain on the system administrators time resulting from VM administration and configuration? How easy is it to migrate virtual machines between different platforms and/or to migrate them between servers? In general, how can they focus entirely on using the virtual machine rather than on its implementation and configuration?

To summarize, these are the current challenges virtual machine administrators face today:

- Virtual machine management software support only a single virtual machine platform.
- Free management tools are often intended for running few virtual machines. No support for grouping nor the design of larger networks.
- Commercial tools offer better support for larger networks but are proprietary and impossible to modify.
- Host configuration is usually not a part of the management software.

From the field of configuration management we know many tools that group unrelated configuration properties into a abstract configuration language thereby enabling the user to address the whole computer system or network from a standard interface [1, 2]. They provide an abstraction layer so that the user can focus on the intended policy without knowing the details of the platform nor the necessary steps required to achieve it. Based on the popularity and effectiveness of this approach, we present an open source tool, MLN (Manage Large Networks), that takes a similar tactic to virtual machine configuration and management.

MLN allows the administrator to design, create and manage whole networks of virtual machines and their internal configuration in a template-based fashion. Two popular open source virtualization platforms, Xen and User-Mode Linux, are currently supported. MLN supports an expandable configuration language using plug-ins that allows the data center administrator to include local configurations easily. A MLN network daemon allows for virtual networks to be spread and managed across several physical servers.

MLN is freely available today and has been used successfully as a tool for virtual student laboratories. In this paper we show how the tool can offer significant improvements to the management of virtual machines, despite their underlying platform, in real-life data centers.

This paper is organized as follows: the next section provides some brief background information. We then outline the main features of the configuration language. The two subsequent sections showcase the tool used in real-world contexts: a web hosting scenario and as a management interface for an on-demand render farm. We then present the interaction and control commands of MLN are presented. Finally, we evaluate the tools current capabilities and present directions for future work.

Background

The diversity of today's virtualization platforms make them suitable for a wide range of tasks. We see a growth in the interest of virtual machines in many areas:

- ***Consolidation and Commercialization***. Virtualized hosting and service encapsulation is perhaps the most attractive use today, as they are commonly associated with cost saving, flexibility, uptime and security.
- ***Research***. A virtual machine is more adaptable in terms of assigned hardware resources and fits well into self-management scenarios [3, 4].
- ***Testing***. Creating test-beds for services and software is also becoming more common.
- ***Education***. Advanced student labs can be implemented with less cost and more flexibility using virtual laboratories [5].

User-Mode Linux [6] is a version of the GNU/Linux kernel that can be run as an application on a running Linux system. The User-Mode Linux kernel is started on the command line, and host parameters such as memory size and filesystem image are supplied as arguments. Folders can be mounted as partitions, and one does not require root access to start virtual machine instances. User-Mode Linux is considered to be lightweight in terms of resources and easy to install. A switch emulator (uml_switch) is supplied as a tool and enables the user to create network topologies entirely in user space. User-Mode Linux does not offer any higher level configuration tools, although several third party software projects exist today (with varying progression) [7].

Xen is a virtual machine monitor that uses the concept of parallelization [8, 9] to enable several concurrent operating system instances to run simultaneously on a thin management layer called a "hypervisor." Xen virtual machines (called *domains*) have low overhead and are considered to be almost as fast as if the operating system were running directly on the hardware. Xen installation and management requires root access. An attractive feature of Xen is the ability to migrate running virtual instances seamlessly across physical servers without down-time.

Connecting Xen virtual machines together in networks is done using bridge devices on the physical server. A bridge device functions the same way as an Ethernet switch and can either provide isolated internal networks on the server or bridge the physical network, making the virtual machines appear to be regular hosts on the LAN. A Xen domain is defined in a configuration file that addresses virtual machine features such as memory, disk image and simple network parameters. A Xen daemon (xend) is responsible for managing the domains. A tool called xm will create a single virtual machine based on the supplied domain configuration file. A commercial tool called XenEnterprise is available for purchase which features increased server, management and resource control [9]. From the available information on the XenSource site at the time of this writing it is difficult to assess the management and design capabilities of XenEnterprise.

VMware [10] is a well-known actor in the virtual machine industry. For brevity, we will consider the freely available tools currently offered by VMware and how they fit into our approach. VMware offers a free product called "VMware Server," which has both a web and graphical application interface for managing virtual machines even on remote servers. The software offers an easy way to create a single new virtual machines but has no way to define groups of virtual machines. Also, since every new virtual machine is created graphically it becomes cumbersome if a user wants to design a large network of say 50 virtual machines spread out over 15 physical servers and make sure they have a consistent configuration. A simple tool, called VMware Player offers a quick way for users to run single pre-configured virtual machines. VMware also has a group of products aimed at hosting scenarios, but since they are not freely available, they are not considered in this text.

MLN: A Management Tool for Virtual Machines

MLN (Manage Large Networks) [11] was first used in 2004 as a tool for providing a virtual firewall lab running User-Mode Linux for students [5]. It has since then been expanded to support Xen as a virtualization platform and to include a plug-in framework. MLN can be downloaded from http://mln.sourceforge.net and has its own installer, which also downloads and installs a version of User-Mode Linux. Xen must be installed separately.

The MLN configuration language contains both system variables and grouping mechanisms. In MLN, a logical group of virtual machines is defined as a *project*. A file in the MLN configuration language will typically define one project.

Defining Projects

The structure of the language is a hierarchical sequence of blocks containing keyword/value pairs. A block is enclosed in curly brackets ({ }). A keyword is generally expressed in the form `keyword value` but is not bound to it. Some keywords are lines with several parameters. It is often natural to place one keyword/value per line, but semicolons can be used to place several pairs on the same line.

Each host and network switch will have one block. All hosts in a project do not have to be connected in the same network.

A project has no restrictions regarding the number of hosts or switches. The only mandatory part of a project description is a block of global definitions with at least the name of the project. Project names must be unique for each user. Definitions of one or more hosts and perhaps network switches constitute the network topology. Hosts can have several network interfaces that can be assigned to switches in arbitrary topologies.

Here is an example of a ring-topology:

```
global {
       project ring
}

host router1 {
    network eth0 {
        switch A
   }
   network eth1 {
        switch B
   }
}

host router2 {
    network eth0 {
         switch B
    }
    network eth1 {
         switch C
    }
}

host router3 {
    network eth0 {
         switch C
    }
    network eth1 {
         switch A
    }
}

switch A { }
switch B { }
switch C { }
```

This is a simple but complete MLN project. In later examples we will show how configurations such as network addresses, users and startup commands are included.

Features for Larger Projects

Language features such as superclasses and variables are helpful when the project is big. We will review these two features next.

Superclasses are a concept from Object Oriented Programming. They describe a class which another class is a subclass of (i.e., a parent). In MLN, a superclass is a description of a virtual machine from which other virtual machines can inherit from. A superclass virtual machine will not be built by MLN. Its most common use is to define a configuration that is to be kept constant and let a group of hosts point to it.

In the example below, the virtual machine `node1` inherits all the keywords from the superclass `common`. It also specifies additional keywords, such as the network interface address. Notice that hosts are free to override keywords from a superclass.

```
superclass common {
       memory 128M
       free_space 1000M
       xen
       network eth0 {
              netmask 255.255.255.0
       }
}

host node1 {
     superclass common
     network eth0 {
             address 10.0.0.1
     }
}
```

Hierarchies of superclasses can be constructed. Hosts only inherit from a single superclass (or superclass hierarchy).

MLN supports string variables in its syntax. This enables the user to keep information consistent across keywords. Consider the following example:

```
global {
       project example1
       $password = 2mf9fmcaioa8w
}
```

```
host node {
      root_password $password
      users {
         jack $password
      }
}
```

The variable $password is defined in the global block and used later on inside a host. Variables have scope, so if a variable is used, MLN will look for its value upwards in the block structure and lastly inside the global block. Variables can be expanded into strings if the variable name is enclosed in brackets ([ ]).

Virtual Appliances

Every virtual machine has its own filesystem. One approach to virtual machine management is to create new machines that boot into an installer the first time and install a new version of an operating system. Another approach is to use a ready-made filesystems which are installed and configured with software already.

MLN supports a repository of these filesystems, called *templates*, from which the user can choose from. Templates vary in size based on the amount of installed software they contain. A virtual machine based on a template of this kind is the basis for what is called *virtual appliances* [12] which can be specialized to perform specific tasks (as the examples later will show). The encapsulation of software components in this way has the benefit that experts can put together and properly configure software, and enables users to hit the ground running with a working virtual machine. For example, in educational contexts, this allows students to focus on using a software tool without having to learn how to install and configure it first.

A variety of templates can be downloaded from the MLN web-site. Users and system administrators can also modify existing virtual appliances as well as create new ones.

Plug-ins

It is not the intent of this tool to re-invent configuration management paradigms in its own language. The plug-in architecture is a way to allow other configuration management tools to be integrated as easily as possible with MLN.

A plug-in that is executed can do two actions: access the entire MLN data tree and change the project before it is built, or configure virtual machine filesystems during the build process. Plug-ins have no need to write their own parsing code.

In the following example, we want to build a project where the virtual machines use the configuration management tool cfengine [1] for internal maintenance. The template used in this project already has the cfengine software installed, but for flexibility, we want to be able to define the cfagent policy inside the MLN project as a block inside a host or superclass. The cfagent policy should be written to a file /cfengine/inputs/cfagent.conf when the project is built. Here is a MLN project with an embedded cfengine policy:

```
superclass common {
    template ubuntu-server-cfengine.ext3
    cfagent {
       control:
         any::
          actionsequence =
            (
            shellcommands
            processes
            )
    }
}

host agent {
    cfagent {
      shellcommands:
          "/usr/bin/updatedb"
      processes:
         "cron" signal=hup
    }
}
```

A plug-in in the Perl programming language that writes the above specified cfagent into a file does not have to be more than the following code:

```
sub cfenginePlugin_configure {
  my $hostname = $_[0];
  if ( getScalar("/host/$hostname/cfagent")){
    my @cfagent_poligy =
          getArray("/host/$hostname/cfagent");
    writeToFile($hostname,
             "/cfengine/inputs/cfagent.conf",
             @cfagent_policy);
  }
}
1;
```

The benefit of this approach is that it is easy to combine MLN with tools that the community is experienced with and that can handle long-term management of the host while it is running. Many system admins have established policies which can be integrated this way using a small amount of code and removes the task of adding the policy manually on each virtual machine. We will see an example later where a plug-in is used to modify the data structure and not the filesystem.

Distributed Projects

Until now, the examples have all been on the same server. MLN also provides a network daemon for distribution of projects among several physical servers. A physical server is in MLN called a service_host because it provides a hosting service to the virtual machine. A physical host must be made aware of it being a service host in its local MLN configuration files.

A virtual machine is assigned a service host the following way:

```
host startfish {
     service_host huldra.iu.hio.no
```

```
        memory 96M
        network eth0 {
                address dhcp
        }
 }
```

Project Organization

All the files belonging to a project are stored in a dedicated project folder. The contents of each project folder is the start and stop scripts for the network switches and each VM together with its filesystem image (unless it is placed in a LVM partition). Starting and stopping a project will result in the corresponding scripts being called. UML does not possess any other way to interact with it then through the command line in the time of writing. Xen, on the other hand, is working on an RPC-based approach for VM management which in time might be possible for MLN to interact with.

In the following example, we show a configuration for a data-center that provides virtualized hosting in the form of *virtual sites*. Customers can deploy a gateway and a set of servers on a back-net for their services. A typical example would be a web service with redundant load balanced servers. For simplicity, we omit another tier of database servers.

The physical layout is set up with a single gateway server and a back-net of hosting nodes. The gateway server will host all the virtualized gateway machines. It is possible to physically mirror the gateway server also. A single customer may span one or several of the back-end nodes. Back-end servers may contain one or more virtualized machines from several customers. The customers can chose the amount of web-servers they want to deploy based on the expected load on their web-sites. See Figure 1 for an example setup.

Every server runs the MLN daemon. Each virtual site is represented as one MLN project. The virtual machines in the project are spread across the servers using the service_host keyword. The project is built across all the servers that host a node from that project.

```
global {
      $cust_name = kafe
      $default_gateway = 10.0.0.141
      project $cust_name
}

# general settings
superclass common {
      xen
      lvm
      root_password *********
      free_space 1000M
      memory 128M
      term screen
      template ubuntu-server.ext3
      network eth0 {
          bridge back-net
          netmask 255.255.255.0
      }
      files {
         /customers/$[cust_name]/www
              /var/www
      }
}

host gw {
     superclass common
     service_host gateway1
     memory 256M
     network eth1 {
           address 128.39.73.101
           netmask 255.255.255.0
           gateway 128.39.73.1
     }
```

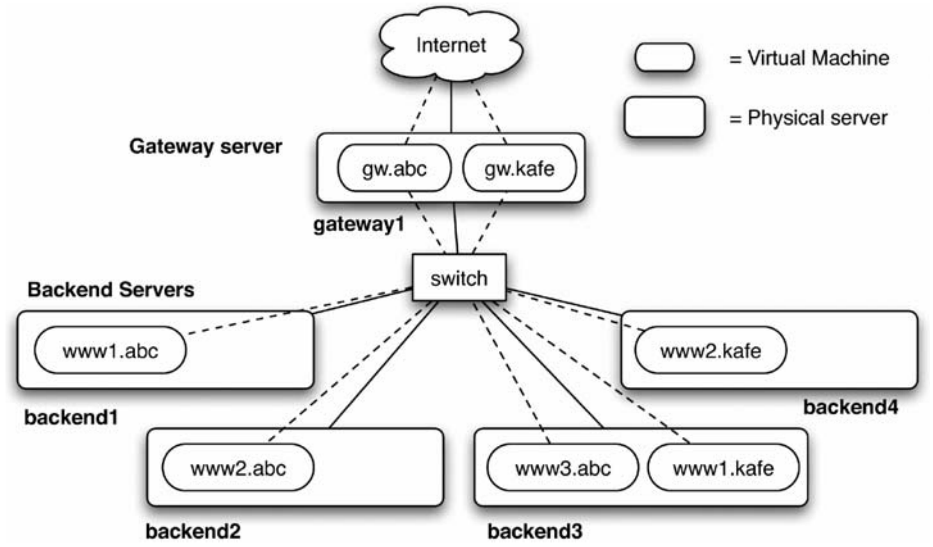

Figure 1: A web-hosting scenario where virtual machines are encapsulations for customer services. Two customers are accommodated in this setup: "abc toys" with three web-servers and a gateway and "kafe on-the-corner" with two web-servers and a gateway. In MLN they are represented as the two projects abc and kafe.

```
      network eth0 {
            address $default_gateway
      }
     startup {
       echo 1 > /proc/sys/net/ipv4/ip_forward
       iptables -t nat -A POSTROUTING
                 -o eth1 -j MASQUERADE
     }
}
host www1 {
      superclass common
      service_host backend3
      network eth0 {
            10.0.0.142
            gateway $default_gateway
      }
}
host www2 {
      superclass common
      service_host backend4
      network eth0 {
            10.0.0.143
            gateway $default_gateway
      }
}
... continues to node N ...
```

The example above shows three virtual machines connected together where the host gw has an extra network interface to the outside. The superclass common defines common keywords for all the virtual machines. In one case, the memory keyword is overridden locally by the gateway host. A variable is used to keep track of the gateway address on the back-net. This way we make sure all the back-end nodes point to the correct address and that the gateway actually has that address. The keywords xen and lvm enable the virtual machines to run on the Xen platform and to put their filesystems in LVM partitions for maximum performance. The files block defines what files should be copied into the filesystems at build time. In this case, it is the source files for the web-servers.

The project is spread over three physical servers. The following command can be used to build the project:

```
mln build -f kafe.mln
```

MLN will attempt to contact the daemons on all the involved servers except where the build command is launched. Once the project is built, it can be started using the following command:

```
mln start -p kafe
```

A remaining task is to configure the virtual site to its intended product. Generally this might be done by the owner of the project. Auto-configuration using promise theory and roles on virtualized sites like this was explored in a separate paper [13]. The hosting company can also offer dynamic configurations, where the number of web-servers is adjusted to the real-time loads on the web-site.

Case Study 2: An On-demand Render Farm

Managing a cluster for rendering can be expensive for small companies as one needs to support a location and hardware for it. In addition, hardware performance increases each year making the local render farm outdated quickly unless one has extra room for expansion.

In this scenario, we consider a small animation company that does not support their own render farm. Instead, they contract with a data center to provide virtual machine hosting for a render farm of virtual machines that the animation company manages themselves. The cost model is pay-per-use, so the animation company only has costs when they are doing actual work, something they consider as an advantage. The data center can rent out their servers to other customers as well and may even run several customer networks at the same time. Moreover, since the data-center is likely to upgrade their servers on a regular basis, they are more likely to attract customers of this kind who will get better performance over time with no extra costs.

The way this is realized with MLN is that the administrator at the animation company has a small, local test-bed on a single machine running a lightweight virtualization platform. There, the template for the render nodes and the master is maintained. Once the animation company has a new contract, a render farm is deployed from these templates. A contract with the data center is made with regard to the number virtual machines to deploy and their resources.

To ease the design of the MLN project for the render farm we introduce a plug-in, *autoenum*, that enumerates the render nodes for us. Here is a project for one master and 50 render nodes:

```
global {
    project renderfarm_customerX
    # the following block contains the
    # configuration for the autoenum
    # plug-in
    autoenum {
      superclass render_node
      addresses enum
      addresses_begin 2
      numhosts 50
      network 10.0.0.0
      service_hosts {
         #include /tmp/servers.txt
      }
    }
    $gateway_address = 10.0.0.1
}
superclass common {
    term screen
   xen
   lvm
}
```

```
superclass render_node {
   superclass common
   template renderNode.ext3
   free_space 1500M
   memory 256M
   network eth0 {
       netmask 255.255.255.0
       gateway $gateway_address
   }
}
host master {
   superclass common
   template renderMaster.ext3
   free_space 5GB
   memory 512M
   network eth0 {
      netmask 255.255.255.0
      address $gateway_address
      bridge cluster-network
   }
   network eth1 {
      netmask 255.255.255.0
      address 128.39.73.102
      gateway 128.39.73.1
   }
}
```

The entire render farm of 51 virtual machines is specified in only 45 lines of code. Actually, the render farm could be increased to 254 without increasing the complexity of the project. We see the use of two superclasses, common and render_node. The platform specific details are all in the first superclass. Xen is chosen as the virtualization platform with LVM partitions for their hard-disks. A simple test bed of only a few nodes using User-Mode Linux that runs on a laptop could be realized with only minor changes to the file above. This way, the administrator from the animation company could make sure the software on the two templates works as he intends before the full-blown cluster is created and charges start accumulating.

The autoenum block in the beginning of the project sets flags for the plug-in. This plug-in has a different intention compared to the one showed in Section 3.4. Upon parsing, the plug-in will fetch the information from the autoenum block and use it to create the rest of the virtual machines that make the cluster. This is done by adding them to the data structure before the project is built. This plug-in is therefore not something that expands the configuration of each virtual machine filesystem, but adds design features and logic to MLN based on local needs.

The list of servers where the nodes are spread out is written in a separate file. The #include statement is used to read in the contents of that file during the MLN parsing process. The autoenum plug-in will assign the render nodes to the servers.

Control Commands and Monitoring

The MLN command builds and starts the virtual machines and networks defined as projects. Here are some examples:

- mln build -f example.mln
  Build the project from the file example.mln.
- mln -P /my/important/projects start -a
  Start all the projects in the folder /my/important/projects.
- mln status -p mysql-servers -u
  List all the switches and virtual machines belonging to the project mysql-servers that are currently running.
- mln stop -p web-services -w 120
  Stop all the virtual machines belonging to the project web-services. Wait 120 seconds for the hosts to shutdown. After the time is elapsed, destroy the remaining ones of the project.

The last example is useful when the physical host is to shut down and has limited time to wait for all of the virtual machines to shut down properly.

Regular users can use MLN without root privileges while using User-Mode Linux as a virtualization platform. Only users with administrator access can start projects that are based on Xen.

The building of a distributed project is started the same way as for other projects. MLN will contact the service hosts for the virtual machines not intended for the local machine and will send them the project for them to do their part.

Upon receiving the build request, the daemons start to build the project in the background and await a subsequent request from the same client asking for the output. Starting and stopping a project will also result in the attempt to contact the other service hosts so that the entire project is managed simultaneously.

MLN will always start the network switches before the virtual machines. The boot order and time to wait between each virtual machine can be specified. Best practice is to avoid a strain on the system by simply letting MLN sleep a few second between each host starts. An example of this, introducing a three second pause, is:

```
mln start -p example -s 3
```

A project is often part of a bigger context on the network or the physical server. Often times one needs to run specific commands on the physical server prior or after the virtual machines have started, such as adding firewall rules or modifying routing information. MLN provides blocks for additions of shell commands so that they are run by MLN at specified points during the starting or stopping of a project.

Modifying Existing Projects

It is not always possible to initially design a project to be optimal for its task. Once the project is running, certain design-time decisions, like memory or disk-space, might be re-evaluated and have to be adjusted. The problem is often that the project already is in use and cannot be rebuilt from scratch. This problem was encountered several times when running virtual student labs over the course of a semester (five months).

MLN's approach to this problem is to provide an upgrade command, which will read in a new and modified version of the project and try to upgrade it accordingly. Typical modifications are to change the amount of memory, increase the disk size or even add/remove virtual machines from the project. System specific changes, such as adding users, can also be performed this way. For networks which can scale from a software point of view, like web-servers and computing clusters, the upgrade feature can be used to manage the amount of nodes that participate in the cluster at any given time. The modification of a project can be done manually by the system administrator, but recent literature suggests a range of applications for this within self-managing and adaptive systems [13, 4].

A more fundamental change to the virtual machine would be to change its virtualization platform, like going from User-Mode Linux to Xen. To change service host will result in a migration of the virtual machine between two service hosts. The result of this is that one can start with a lightweight User-Mode Linux virtual machine on a regular laptop or workstation, and the virtual machine could later be moved to a more powerful server using MLN where it would be running on the Xen platform with perhaps more memory assigned to it too.

Monitoring

The user can use MLN to collect the status information from all the servers that run the MLN daemon. The information displayed includes how many projects are running, the number of virtual machines, the amount of used memory and how much memory is left from the allowed maximum for that server. This information is useful for monitoring and planning of new projects.

Here, we see the result of the `mln daemon_status` command on the network discussed in Case Study 1. Note that the total number of projects can be misguiding, as several servers can have a part of a project and that every part will count as a single project in the summary.

Servers can be put into groups and status can be queried on a per group basis, thereby giving more specialized feedback. One example is to only show the resources on the servers assigned for testing or the ones used in production. Whether or not a project or a certain host is up is also possible through MLN.

Discussion

Successes

Consolidation of several virtual machine technologies into one tool is a new and challenging task. Until now, it seems, the focus for development of virtual machine monitors has been on performance, and carving out a niche. The authors do not see any direct competition between the virtual machine platforms used in this project. In fact, the sum of them is a greater gain. The user should have ability to choose which one to use without affecting the choice of the management interface. Through MLN we have provided one way to design large virtual networks before thinking about the platform it will run on.

MLN creates start and stop scripts for each virtual machine and switch. As a result, any virtual machine technology that is controllable from the command line, would be relatively easy to integrate into MLN. There is no common API to virtualization today. A stronger effort to provide a common API to all the virtual machine technologies would greatly improve the result for projects like this and enable MLN to support even more virtualization platforms by talking to the API directly.

MLN has been tested and used as a commercial hosting tool for over a year, during which it has provided us with much feedback on the needed features for and limitations of current tools. Many features, such as the plug-in framework, have spawned from this exchange. The plug-in framework allows for administrators to add features both in configuration scope as well as logic without re-inventing the wheel.

MLN has become the standard tool for virtual machine management at Oslo University College. It provides the means for massive virtual student laboratories in security classes as well as a virtual appliance tool for student projects. Other institutions, such as University of Linkping in Sweden and Oregon State University in the US have also benefited from it in educational contexts. A Norwegian ISP uses MLN

Server	# Projs	#vms	Mem Used	Mem Ava	Groups
gateway1	2	2	512	768	gateways,xen
backend1	1	1	128	896	backends,xen
backend2	1	1	128	896	backends,xen
backend3	2	2	256	768	backends,xen
backend4	1	1	128	896	backends,xen
Total	7	7	1152	4224	

Table 1: Status information.

today in their R&D department to rapidly create virtual test-beds.

MLN is one of the few freely available tools that offer "cold migration," where the virtual machine is shut down first and the filesystem is compressed and copied to the new service host. Live migration is supported in Xen but requires the two servers to be on the same LAN and to have the same CPU architecture and concurrent access to the virtual machine's disk. This is hard to realize transparently to the user as it is bound to a certain platform. Cold migration works in many scenarios where live migration would fail because the servers are of a different architecture and have no concurrent access to the filesystems. Another benefit of this approach is that virtual machines can change other aspects in the migration process. A User-Mode Linux host can migrate into a Xen host with more memory and a different network setup. This is practical for moving test-beds onto more powerful servers of different architecture and to completely separate locations, changing network parameters in the process. All of this is realized using the `mln upgrade` command.

Current Limitations

MLN's configuration language addresses both hardware attributes of the virtual machine and system configurations. It is therefore not possible to avoid the challenges of host configuration management. Currently, GNU/Debian based templates, such as Ubuntu Linux, are best supported. MLN should ideally be able to support several operating systems let alone support different Linux distributions. However, such concerns are part of the ongoing effort of a large systems configuration management community. The plug-in infrastructure of MLN is one way to invite seasoned configuration management systems and third-parties to handle the lower level tasks. However, some languages might fit better then others into this framework and certain new requirements might surface. This research is in progress and will be discussed in a later publication.

Sufficient monitoring of the virtual machines is a critical feature for data centers. MLN supports status on projects, hosts and globally. Memory usage, the number of virtual machines and projects on each server can be collected as well. This works well for monitoring a project's status and to see the level of remaining resources on a physical server. However, a usage indicator as to how much CPU and network traffic is related to each virtual machine might further help capacity planning. Xen has tools like `xentop` and `xenmon` [14] that can monitor network, CPU usage and IO operations of their virtual machines. One improvement to MLN would be to expand the plug-in framework to also enable monitoring and management. This would allow the local data center to develop specializations that assist in capacity planning or fault detection, such as a plug-in that finds free IP addresses or logs operations such as starting and stopping.

Another question is weather or not MLN should provide better encapsulation of each project in order to protect them from each other. In User-Mode Linux, this is possible as the virtual machines run as processes and are assigned to users. In Xen, all virtual machines exist in the same "pool" and have no direct ownership. Although the Xen domains are considered to be securely encapsulated, they might still have network access to other virtual machines. One solution is to create virtual switches on each physical server and to connect those with virtual tunnels. This implementation is in progress at this time of writing and will be presented as a plug-in.

Future Directions

The MLN daemon uses IP-based access control for management access. Added features, such as user support for the daemon and finer access control would indeed be a benefit. This way, one could separate access for building a project and the ability to start and stop them.

Interaction with MLN is currently in the form of a configuration language and shell commands. Although the language features improve design and control over large virtual networks, one can investigate other approaches such as graphical design and control tools. Also, adding support for well-known document formats such as XML may enable MLN to play the role of a back-end for higher level tools.

Future work will also look at the improvement of the distributed management aspects of MLN. Scenarios such as management of large and distributed virtual hosting platforms and how to introduce closer monitoring and fail-over are of particular interest.

Conclusion

We have presented an approach to virtual machine administration that lets the user describe the wanted configuration in an understandable declarative language and then build the virtual hosts and networks from it. The virtualization platform is secondary to the configuration interface. A concept of logical groups of virtual machines enables the user to issue management commands to all virtual machines that belong together. Language features such as inheritance from machine superclasses and variable expansion make it possible to consistently describe large networks in just a few lines and to avoid redundant information.

A plug-in architecture lets the user transparently expand the configuration domain of the language to solve their specialized needs. Part of the toolkit is a daemon that allows management of virtual networks that span several physical servers. All of these features have been harnessed to provide a flexible and powerful way to define, create and manage scenarios for data-centers. Two case studies show the usefulness of our approach; a web-hosting facility and a on-demand render farm are realized using simple configurations and local additions to the MLN language.

Author Biography

Kyrre earned his M.Sc. in Computer Science from the University in Oslo. Apart from his studies, Kyrre has worked as a course instructor at a Linux company where he has written and held courses in system administration and Linux. Kyrre started as a full time Ph.D. student in 2003 at the University College of Oslo. His main research areas are anomaly detection, formal modelling of distributed systems and configuration management.

Acknowledgments

The author would like to thank Professor Mark Burgess and John Sechrest for helpful discussions and pointers throughout this work.

Bibliography

[1] Burgess, M., ''Cfengine: a site configuration engine,'' *USENIX Computing Systems*, Vol 8, 1995.

[2] Desai, N., A. Lusk, R. Bradshaw, and R. Evard, ''Bcfg: A configuration management tool for heterogeneous environments,'' *IEEE International Conference on Cluster Computing (CLUSTER'03)*, 2003.

[3] Liu, X., J. Heo, L. Sha, and X. Zhu, ''Adaptive control of multi-tiered web application using queueing predictor,'' *10th IEEE/IFIP Network Operations and Management Symposium (NOMS 2006)*, 2006.

[4] Xu, W., X. Zhu, S. Singhal, and Z. Wang, ''Predictive control for dynamic resource allocation in enterprise data centers,'' *10th IEEE/IFIP Network Operations and Management Symposium (NOMS 2006)*, 2006.

[5] Begnum, K., K. Koymans, A. Krap, and J. Sechrest, ''Using virtual machines in system and network administration education,'' *Proceedings of the System Administration and Network Engineering Conference (SANE)*, 2004.

[6] Dike, J., ''A user-mode port of the linux kernel,'' *Proceedings of the 4th Annual Linux Showcase & Conference, Atlanta*, 2000.

[7] *The UMLwiki tools page*, 2006, http://uml.harlowhill.com/index.php/tools .

[8] Barham, P., et al., ''Xen and the art of virtualization,'' *SOSP 03*, 2003.

[9] *The xensource homepage*, 2006, http://www.xensource.com .

[10] *The vmware website*, 2006, http://www.vmware.com .

[11] *The mln project homepage*, 2006, http://mln.sourceforge.net/ .

[12] Sapuntzakis, C., D. Brumley, R. Chandra, N. Zeldovich, J. Chow, M. S. Lam, and M. Rosenblum, ''Virtual appliances for deploying and maintaining software,'' *Proceedings of the 17th Large Installation Systems Administration Conference, (LISA '03)*, October, 2003.

[13] Begnum, K., M. Burgess, and J. Sechrest, ''Adaptive provisioning using virtual machines and autonomous role-based management,'' *SELF – Self-adaptability and self-management of context-aware systems, SELF'06*, 2006.

[14] Gupta, Diwaker, Rob Gardner, and Ludmila Cherkasova, ''Xenmon: Qos monitoring and performance profiling tool,'' 2005.