Integrating Xen with the Quattor Fabric Management System

Stephen Childs and Brian Coghlan

Department of Computer Science, Trinity College Dublin
childss@cs.tcd.ie

Abstract. While the deployment of virtual machines (VMs) within the high-performance computing (HPC) community is proceeding at a great pace, tools for system management of VMs are still lagging behind those available for physical machines. In order to make further progress, VM management must be fully integrated with existing fabric management infrastructure. We present the results of work done to integrate Xen [4] with the Quattor [15] fabric management suite. The principal contributions are the development of a network bootloader for para-virtualised Xen VMs and a Quattor management component for setting up hosted VMs. The combination of these tools allows for full unattended installation of Xen VMs and the automatic configuration of services, all from a single configuration database.

1 Introduction

Deployment of virtual machines (VMs) in the high-performance computing community is taking off rapidly. The past few years have seen the use of virtualisation for Grid service nodes [8], training infrastructures [5,6], dynamic clusters [11], and middleware certification. Initially most projects took an ad hoc approach to the creation and management of virtual machines, developing small tools as necessary. However, as virtualisation becomes mainstream and starts to be deployed in production environments, the need to integrate VMs into existing management infrastructure has become apparent: it should be possible to use existiing installation and configuration mechanisms to install virtual machines.

The term fabric management system is used to describe an integrated suite of tools used to install, configure and monitor the various service and compute nodes that make up a Grid or HPC infrastructure. Quattor [15] is a fabric management system originally developed as part of the European DataGrid [17] project, and now developed as a community effort with contributions from various sites, mostly within the high-energy physics community. It is currently deployed at over forty sites managing over 10,000 nodes. It is a modular system designed to manage all stages of a machine's lifecycle from initial installation on the bare metal to configuration (and reconfiguration) of complex software such as Grid middleware.

Quattor configurations are written in Pan [9], a powerful declarative language allowing complex compositions of configuration templates, and are compiled to

L. Bougé et al. (Eds.): Euro-Par 2007 Workshops, LNCS 4854, pp. 214–223, 2008.

XML profiles that contain a complete configuration tree for the machine. Configuration profiles for a site are stored in a configuration database, and the client machines pull profiles from this database at regular intervals. Programs running on the client (known as components) read the relevant sections of the configuration profile and generate configurations for the services they manage. For example, the `ncm-accounts` component manages user accounts, and `ncm-ssh` manages secure shell (ssh) services.

Quattor also includes a subsystem for configuring a network boot infrastructure. The Automated Installation Infrastructure (AII) uses the information from machine profiles to generate configuration entries for DHCP and pxelinux [3] running on an installation server. AII allows machines to be installed from scratch using Kickstart[1]: once the server and client are configured correctly for network booting, installation of a new node is as simple as switching it on. The aim of this work is to fully integrate Xen virtual machines with AII and Quattor so that they can be automatically installed in the same way as physical machines.

2 Design

The aim of this work is to integrate VMs fully into the fabric management system so that the complete lifecycle can be managed automatically via the configuration database and installation servers. Once correct profiles have been configured for a VM and its host, it should be as easy to install the VM as it is to install a physical machine.

Integrating Xen with Quattor required three main areas of work: i) design of appropriate Quattor data structures to represent Xen VMs and global configuration options, ii) development of a Quattor management component for Xen (`ncm-xen`), and iii) development of a bootloader for para-virtualised Xen domains that was compatible with automatic network installation. While the first two items are specific to Quattor, the bootloader we have developed is useful for other systems that use network booting (especially those using pxelinux).

It should be noted that there was no need to modify core Quattor services. The aim was to make VMs behave as much like physical machines as possible, rather than adapting the core services to treat VMs specially. The combination of a configuration component running on the hosting machine and enhancements to the VM boot process made this possible. This approach results in a simpler system: the installation procedure for VMs is very similar to that of physical machines, and so the normal debugging procedures used by administrators apply. Also, once a VM has been successfully booted and configured, day-to-day management via Quattor is exactly the same as for a physical machine.

2.1 VM Configuration

In Quattor, the configuration state of a physical machine can be entirely encapsulated in its configuration profile. In contrast, there are two aspects to the configuration of a virtual machine: firstly, the configuration of the VM itself, and

secondly the information needed by the host machine to set up the VM correctly. The first category includes the same information as for a physical machine (e.g. user accounts, service parameters, etc.) The second category includes all the information needed by the host machine to set up the virtual machine: location of storage, network configuration (including MAC address to be assigned), amount of memory to be assigned, method of booting, etc. None of this information is visible within the VM, and so must be included in the configuration of the host machine.

```
"/software/components/xen/domains" = push(nlist(
"memory", value("//cagnode50/hardware/ram/0/size"),
"name", "cagnode50.cs.tcd.ie",
"disk",list(nlist
            ("type",'file',
             "path",'/var/xen-grid/cagnode50/fs/disk',
             "device",'sda',
             "size",value("//cagnode50/hardware/harddisks/sda/capacity"),
             "rw",'w'),

            nlist("type",'lvm',
             "hostdevice",'xenvg',
             "hostvol", 'cagnode50-swap',
             "size", 6*GB,
             "device",'hda3',"rw",'w')),
"vif",list('mac='+value(//cagnode50/hardware/cards/nic/eth0/hwaddr)),
"bootloader","/usr/bin/pypxeboot",
"bootargs","vif[0]",
"auto", true
));
```

Fig. 1. Example domain configuration

In Quattor, the configuration for a particular component is normally located under the location `/software/components`. So the configuration for `ncm-xen` is located under `/software/components/xen`. The profile of a host machine includes a list (called `domains`) of data structures for the VMs that it hosts. Figure 1 shows an example of this data structure. Most of the entries are simple strings that will be directly translated to entries in the Xen configuration file. The exception is the `disk` entry, which is a more complex data structure incorporating information needed to create the physical filesystem (i.e. the size) as well as that needed for VM configuration. Much of the information needed to create the VM can be extracted from the VM's own profile (e.g. RAM size, disk capacity, MAC address). The `auto` flag determines whether `ncm-xen` will set up the links necessary to make the VM run automatically on host system startup.

In addition to the data structures representing domains, there are other variables that control the global configuration of Xen on the host, and the operation of `ncm-xen`. For example, `ncm-xen` can optionally create storage for guest VMs:

this is controlled by the `create_filesystems` variable. The `create_domains` variable determines whether VMs are automatically started when `ncm-xen` detects that they are not running.

When `ncm-xen` is run on the host machine, it extracts this list of data structures representing VMs from the machine's profile. `ncm-xen` uses this information to generate Xen configuration files for VMs and to set up storage. Once the VM has been installed successfully, it will retrieve its own profile and invoke the Quattor configuration components to configure services.

3 Implementation

3.1 Network Bootloader

Preboot Execution Environment (PXE) is a standard for automatically retrieving operating system kernels and configuration information over the network at boot time. It is widely used for installation of nodes within the high-performance computing community. When a machine configured for PXE boots up, its network card broadcasts a request for an IP address. A DHCP server will respond with an IP address and the address of the server holding the configuration and OS kernel for that machine. The PXE client server then downloads the OS kernel and uses it to boot the machine.

PXE booting is often used in conjunction with an automatic system installation program such as Kickstart [1]. Kickstart reads a configuration file, sets up filesystems and installs a basic set of packages to get the OS up and running.

PXE clients are usually located either in a boot ROM on the network interface or on the motherboard itself. Boot images loaded from a floppy disk or other bootable media are also sometimes used. This option would be possible for fully virtualised VMs running on machines with hardware support, but could not be used with the para-virtualised (PV) VMs that are currently widely deployed. As a PV Xen virtual machine does not have physical hardware, some other solution is required. Various approaches have been suggested in discussions on the Xen developer list: we have taken the simple approach of writing a program that runs on the host VM and performs a simulated PXE boot process on behalf of its guest VMs.

Xen supports the use of a "bootloader" for guest VMs. This is simply a program that is run on the host VM (as part of the VM boot process) to retrieve a kernel for a guest VM. This scheme has been used by others to implement `pygrub`, a program that looks inside the guest's filesystem, works out which kernel to use, and copies it back out to the host's filesystem to boot the VM. We have implemented a network bootloader, `pypxeboot`, that takes a similar approach but which retrieves the configuration and kernel over the network rather than from the guest's filesystem. The Trivial File Transfer (TFTP) [18] protocol is used for downloads.

`pypxeboot` is invoked as part of Xen's VM creation procedure, and performs the following steps (illustrated in Figure 2) to retrieve a boot kernel:

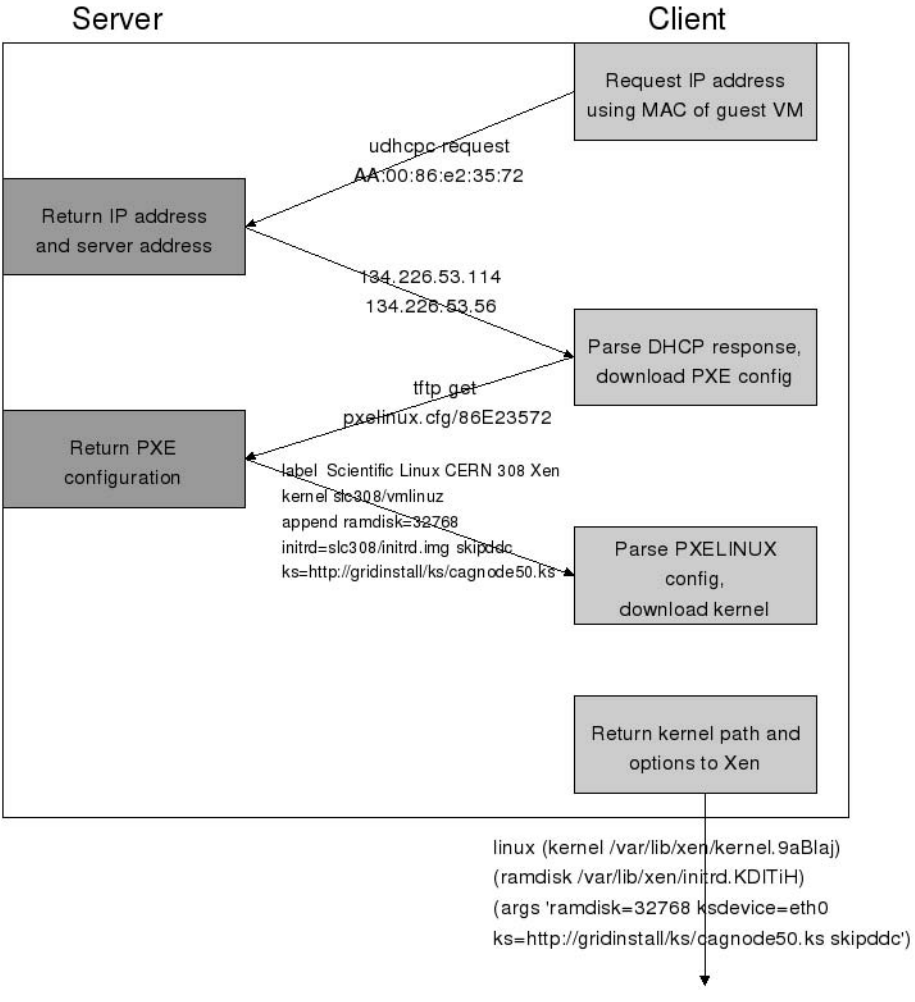

Fig. 2. Pypxeboot network boot cycle

1. Make a DHCP request using the MAC address that will be assigned to the guest VM.
2. Extract the address of the TFTP server and download the configuration for the guest VM.
3. Read the configuration received. If the VM is set to network boot, then download the kernel for the VM over the network. If the VM is set to local boot, drop back to a bootloader that will load the kernel from the VM's own filesystem. In either case, the location of the kernel retrieved is passed to Xen, which uses it to boot the VM.

DHCP client software normally reads the MAC address from a physical interface and uses that directly. With `pypxeboot`, the host VM will need to make a request

on behalf of the guest VM using the guest's MAC address. We modified the `udhcp` [10] client to read a user-specified MAC address from the command line, allowing us to send a DHCP request that appears to come from the guest VM. `udhcp` also retrieves other parameters provided by the server, the most important being the location of the server holding the node's boot configuration and kernel image. `pypxeboot` parses the output from `udhcp` to determine which server to use, and then downloads the configuration using a TFTP client.

In order for a Kickstart installation to succeed, installation tools such as Anaconda must be modified to run on the Xen virtual hardware. We are currently using images created by Linux Support at CERN [16] which include modified partitioning and hardware detection code that is compatible with Xen virtual hardware.

3.2 Quattor Management Component

We have implemented `ncm-xen`, a management component that runs on the host machine to translate the configuration specified in a machine's profile into a valid Xen configuration. This requires Xen global configuration options to be set, and also configuration files to be created for the guest VMs. `ncm-xen` can also create storage to be used by guest VMs and start VMs: these optional features are required to enable full unattended installation of Xen hosts and VMs from scratch.

Quattor components are normally written in Perl, and use the Configuration Cache Management (CCM) API to access the Quattor profile for the machine on which they are running. As shown above, most of the parameters for a VM can be translated directly into Xen configuration parameters: `ncm-xen` extracts these entries from the profile and writes them out to a configuration file for the VM.

A little more work is necessary to correctly configure the filesystems for a VM. In addition to creating the appropriate configuration entries describing the storage to be used by the VM, `ncm-xen` also supports the creation of file-backed storage and logical volumes. If a file or logical volume is specified in the VM's configuration entry but does not exist, `ncm-xen` will create it. This is particularly useful when host and guests are being installed from scratch: `ncm-xen` can automatically set up backing storage for a VM prior to its first instantiation. The details from the profile are also used to generate entries in the correct format for the Xen configuration file.

3.3 Putting It All Together

Having presented all the components of our integrated system, we can now describe a complete usage scenario where a combination of host and guest VMs, each with its own profile, are automatically installed on a physical machine.

Figure 3 shows the complete automatic installation of a host and guest VMs, from bare metal to a fully-configured set of VMs. The first step is to set up the DHCP entries and pxelinux configuration on the installation server: the AII

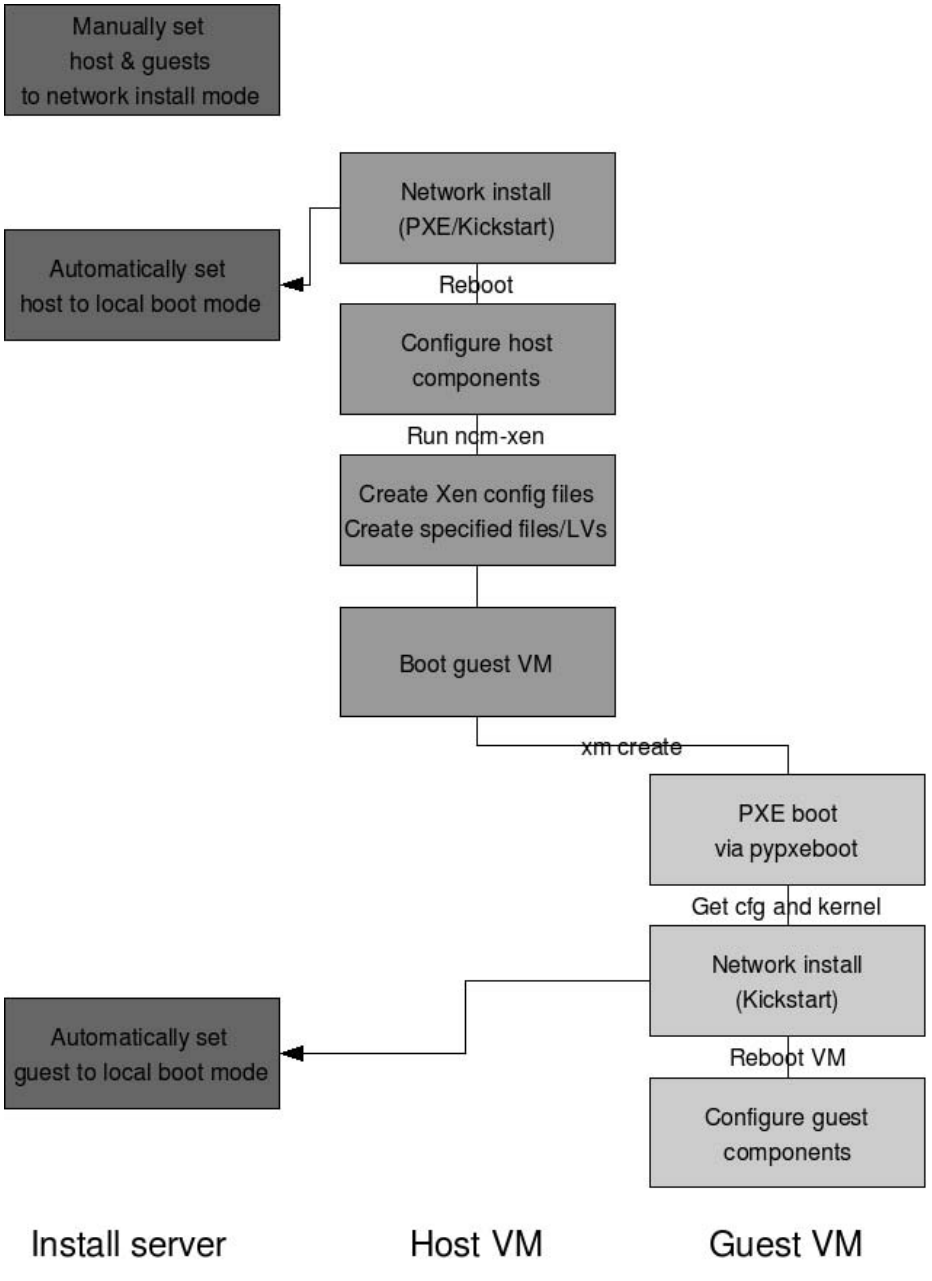

Fig. 3. Complete installation cycle

program provides options to translate information from machine profiles into DHCP entries, and then to set up the pxelinux configuration for the machines to cause a network installation.

The next stage of the process starts when the physical machine is powered on for the first time. The machine then boots into Kickstart via PXE and installation starts. Once a minimal set of packages has been installed, the machine restarts and boots from local disk. At this stage, Quattor packages are installed, the machine's profile is downloaded and a configuration run begins. As part of this configuration run, `ncm-xen` will be invoked to configure guest VMs. It will read the entries from the host's profile and then create filesystems and Xen configuration files according to this information. The guest VMs are configured to use `pypxeboot` as their bootloader, and when they are started they determine which kernel to boot via PXE, then boot into the same Kickstart installation procedure used by the host. They then reboot, download their Quattor profile and begin a configuration run to set up their own services. Once this completes, the host and guest VMs are all under Quattor control and future configuration changes can be made by deploying new profiles.

4 Related Work

The need to integrate VMs with existing management systems has been addressed by others. The developers of OSCAR [12], have presented the results of a project [2] to add support for Xen guests. However, as they did not solve the problem of network booting para-virtualised Xen guests, they were forced to use a cumbersome solution involving a two-stage install. In Xen-OSCAR, the guest VM initially boots into an install image that sets up the real filesystem for the VM and installs into it. As our installation uses a native network boot, the process is much simpler. The installation process for a VM is also very similar to a standard network installation, making it easier to debug.

XenSource provide the XenEnterprise [14] management suite for deployment and control of Xen VMs. Enomaly's enomalism Virtualized Management Dashboard provides similar functionality [13]. Both of these products provide a GUI-driven interactive management methodology, rather than the fine-grained, automated control provided by the Pan language and configuration database in Quattor. We see the two modes of management as complementary: in fact, we are also developing an interactive management interface targeted at the creation of custom test environments. This tool, known as GridBuilder, [7] uses template FS images and copy-on-write techniques to rapidly create transient VMs whose configuration need not be stored permanently in a central database. We hope to investigate closer integration of GridBuilder and Quattor in the future: for example, experimental configurations created in GridBuilder could be translated into Quattor configurations for deployment in a production environment.

5 Conclusion

The integration of Xen with a fabric management suite overcomes a significant obstacle to further deployment of VMs within the HPC community. Previously, those wishing to deploy VMs needed to develop their own tools, or to interface

external VM management tools to their existing fabric management system. The tools we have developed allow for seamless integration of VMs into an existing Quattor infrastructure, providing for unified management of physical and virtual machines.

`ncm-xen` is currently being used by Grid-Ireland to install Gird infrastructure servers hosted on VMs, and has been released to the Quattor community for testing. `pypxeboot` has been submitted for inclusion in the Xen release.

Future work will focus on extending the functionality of `ncm-xen`. For example, support for downloading and customising pre-installed images would allow for faster startup compared to from-scratch installation. It would also be good to modify the schema so that it is compatible with other VM technologies such as VMWare, qemu, and kvm.

This work makes use of results produced by the Enabling Grids for E-sciencE project, a project co-funded by the European Commission (under contract number INFSO-RI-031688) through the Sixth Framework Programme. EGEE brings together 91 partners in 32 countries to provide a seamless Grid infrastructure available to the European research community 24 hours a day. Full information is available at `http://www.eu-egee.org`.

References

1. Kickstart installations. `http://www.redhat.com/docs/manuals/linux/RHL-9-Manual/custom-guide/ch-kickstart2.html`
2. Vallée, G., Scott, S.L.: Xen-Oscar for cluster virtualization. In: Min, G., Di Martino, B., Yang, L.T., Guo, M., Rünger, G. (eds.) ISPA Workshops 2006. LNCS, vol. 4331, pp. 487–498. Springer, Heidelberg (2006)
3. Peter Anvin, H.: PXELINUX - SYSLINUX for network boot., `http://syslinux.zytor.com/pxe.php`
4. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the Art of Virtualization. In: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, ACM, New York (2003)
5. Berlich, R., Hardt, M.: Grid in a box - virtualisation techniques in Grid training. In: EGEE conference, Athens (April 2005), Available via: `http://www.ep1.rub.de/~ruediger/pandoraAthens.pdf`
6. Cassidy, K., McCandless, J., Childs, S., Walsh, J., Coghlan, B., Dagger, D.: Combining a virtual grid testbed and grid elearning courseware. In: Proc. Cracow Grid Workshop 2006 (CGW 2006). Academic Computer Centre CYFRONET AGH, Cracow, Poland (October 2006)
7. Childs, S., Coghlan, B., McCandless, J.: GridBuilder: A tool for creating virtual Grid testbeds. In: 2nd IEEE Conference on eScience and Grid computing, Amsterdam (December 2006)
8. Childs, S., Coghlan, B., O'Callaghan, D., Quigley, G., Walsh, J.: A single-computer grid gateway using virtual machines. In: Proc. AINA 2005, Taiwan, March 2005, pp. 761–770. IEEE Computer Society, Los Alamitos (2005)
9. Cons, L., Poznanski, P.: Pan: A high-level configuration language. In: LISA 2002: Sixteenth Systems Administration Conference, Usenix, pp. 83–98 (2002)

10. Dill, R., Ramsay, M.: udhcp client/server package (2002), http://udhcp.busybox.net/
11. Emeneker, W., Stanzione, D., Jackson, D., Butikofer, J.: Dynamic virtual clustering with Xen and Moab. In: Min, G., Di Martino, B., Yang, L.T., Guo, M., Rünger, G. (eds.) ISPA Workshops 2006. LNCS, vol. 4331, pp. 440–451. Springer, Heidelberg (2006)
12. Open Cluster Group. Open source cluster application resources., http://oscar.openclustergroup.org
13. Enomaly Inc. enomalism virtualized management dashboard., http://www.enomalism.com
14. XenSource Inc. Xen Enterprise 3.1 datasheet., http://www.xensource.com/files/xenenterprise_3.1_datasheet.pdf
15. Garcia Leiva, R., Barroso Lopez, M., Cancio Melia, G., Chardi Marco, B., Cons, L., Poznanski, P., Washbrook, A., Ferro, E., Holt, A.: Quattor: Tools and Techniques for the Configuration, Installation and Management of Large-Scale Grid Computing Fabrics. Journal of Grid Computing 2(4) (2004)
16. Jaroslaw Polok. Xenification of Scientific Linux CERN., https://twiki.cern.ch/twiki/bin/view/LinuxSupport/XenificationOfSLC
17. Segal, B., Robertson, L., Gagliardi, F., Carminati, F.: Grid computing: The european data grid project (2000)
18. Sollins, K.: The TFTP Protocol (Revision 2), RFC (1350) (July 1992)