University of Victoria
Faculty of Engineering
Fall 2006 Work Term Report

AutoXen: Automating Virtual Machine Deployment Within the Grid

Department of Physics and Astronomy
University of Victoria
Victoria, Canada

Angela Norton
0025775
Computer Science
anorton@uvic.ca

January 2, 2007

In partial fulfillment of the requirements of the Bachelor of Science Degree

Supervisor's Approval: To be completed by Co-op Employer

I approve the release of this report to the University of Victoria for evaluation purposes only.
This report is to be considered: NOT CONFIDENTIAL CONFIDENTIAL

Signature	Position	Date
Name	Email	Fax

Contents

1 Report Specification

1.1 Audience

The intended audience for this report includes my supervisors and any future developers on this project.

1.2 Prerequisites

Necessary prerequisites for understanding the material contained within are a good understanding of computer systems and some familiarity with grid computing and virtualization concepts.

1.3 Purpose

This report aims to give future developers a solid background and motivation for this project, a good overview of the architecture, and an understanding of the design decisions made. Further technical details are left to documentation on the group wiki.

2 Introduction

This paper describes the motivation for and architecture of our automated Xen domain deployment system, called AutoXen. The information on the wiki[5], under "Xen Project", is meant as a more technical supplement to this paper.

Xen is a Linux virtual machine environment that provides near-native performance[1] through a technique called *paravirtualization*. Essentially, paravirtualization allows the guest operating system (OS) some direct hardware access, instead of virtualizing the entire hardware interface. Xen requires both the host and guest operating systems to run modified kernels to support this.

The grid environment targeted by this project was GridX1, a joint Canadian endeavour involving the Universities of Alberta, Calgary, McGill, Simon Fraser, Toronto, and Victoria, as well as the National Research Council in Ottawa and the TRIUMF Laboratory in Vancouver.[3] GridX1 computers use the Globus Toolkit as middleware, to allow users to submit computationally intensive jobs from their workstations and receive the results back from whichever compute element was assigned the job.

3 Motivation

Many applications are compiled against specific versions of support libraries that exist on a certain version of a single operating system. An example of

[1]Depending on the application, performance can be up to 100% of native speed, but averages 90%.

such an application would be the ATLAS[1] software from CERN[2]. ATLAS is compiled to run on Red Hat Enterprise Linux 3. However, the WestGrid cluster runs SuSE Linux, which means that even if all other resources are busy and WestGrid has available resources, ATLAS jobs still cannot run there.

Few groups can take the time, effort, and money to make sure that their application runs on multiple different operating systems and all the revisions to each one. Support libraries can change dramatically between versions, forcing an application that is written to and compiled against one version to require that particular one, not previous nor later versions. Additionally, for intensive computations that rely heavily on the code being "correct" to produce the right answer, different versions of libraries or OS kernels may be unacceptable to the researchers.

All of these factors limit the true number of resources available to a job when it is submitted to the grid. One very promising solution to this is to integrate on-demand virtual machines into the grid. By installing Xen on worker nodes in the grid, we hope to homogenize the grid operating environment from the point of view of the application developer. If a developer is writing a program that runs on SuSE Linux, she can be sure it will run on a Xen-enabled grid. She doesn't need to worry about which OS she should be targetting in order to get maximum CPU time based on what is the most popular OS on grid machines.

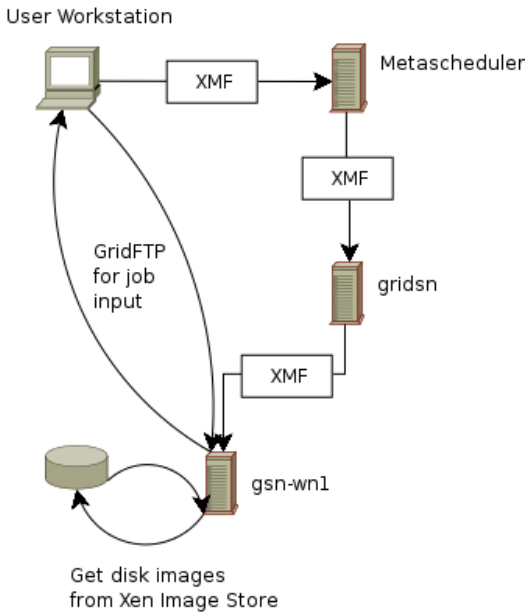

Figure 1: Original Architecture Plan

4 Architecture

The goal (see Figure 1) for this project is to implement a transparent system whereby a user could submit jobs to the grid, and with minimum extra effort, run them within a Xen guest domain. Job results would be staged back to the user's workstation just like regular grid jobs.

A "Xen Image Store" (XIS) would contain preassembled disk image files for Xen to boot. The XIS may be stored locally, or on another machine. Alternatively, the user can assemble their own disk image file, and upload it to the XIS.

Job input files must either be staged in using Globus' GridFTP service, or exist within a disk image that the guest domain will mount. For larger input sizes, or for jobs that will be repeatedly run using the same inputs, the

latter would use less bandwidth, given a local XIS.

4.1 AutoXen Script

AutoXen was designed to be a quick and simple way to automatically run a Xen guest domain given a set of configuration options. It is written in Perl, the language of choice for this group. It uses several publicly-available modules for XML validation and processing as well as command line option parsing. AutoXen takes an XML file, called a Xen Metaconfiguration File (XMF), as input. The XMF contains configuration options for the requested Xen guest domain and the job to run inside that domain.

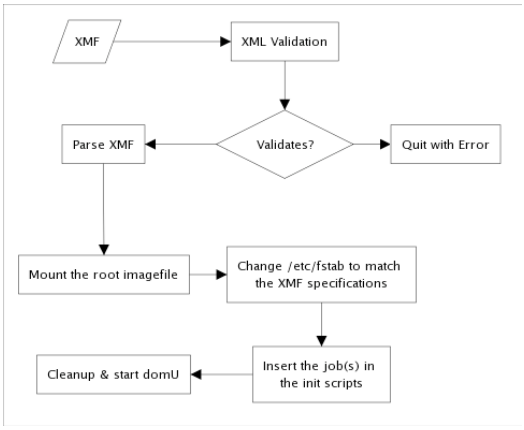

Figure 2: AutoXen Process

Figure 2 depicts the process AutoXen takes to customize and run a Xen domain for a job. The scripts starts with validating the given XMF file against the XML Schema, and if it passes, begins to process the various sections. Most of the entries in the XMF are reformatted and written to a temporary Xen configuration file. Others, such as *filesystemtype*, are needed

in the next step, which is mounting the root disk image and writing the appropriate */etc/fstab*. The job command to be run is written into a script in */etc/xenjobs.d* to run on bootup. Finally, AutoXen unmounts the disk image, deletes the mountpoint, and starts the Xen guest domain.

4.2 Xen Metaconfiguration File

The XML Schema that the XMF is validated against can be inspected below:

```
1   <?xml version="1.0" encoding="UTF-8" ?>
2   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3   <xs:element name="xenmetaconfig">
4   <xs:complexType>
5       <xs:sequence>
6           <xs:element name="general">
7               <xs:complexType>
8                 <xs:sequence>
9                 <xs:element name="memory" type="xs:integer"/>
10                </xs:sequence>
11              </xs:complexType>
12          </xs:element>
13          <xs:element name="imagefile" maxOccurs="unbounded">
14              <xs:complexType>
15                  <xs:sequence>
16                  <xs:element name="filesystemtype" type="string"/>
17                  <xs:element name="mountpoint" type="string"/>
18                  <xs:element name="blockdevice" type="string"/>
19                  <xs:element name="url" type="string"/>
20                  </xs:sequence>
21              </xs:complexType>
22          </xs:element>
23          <xs:element name="job">
24              <xs:complexType>
25                  <xs:sequence>
26                  <xs:element name="command" type="xs:string"/>
27                  </xs:sequence>
28              </xs:complexType>
```

```
29          </xs:element>
30      </xs:sequence>
31  </xs:complexType>
</xs:element>
</xs:schema>
```

This schema specifies the name and number of multiples allowed for each element in an XMF. The *general* section contains one element, *memory*, which is the amount of memory that the guest domain is to be allocated. The *imagefile* section, of which there may be more than one, specifies the location and type of a disk image file. The elements contained within an *imagefile* section must always occur in the above order: *filesystemtype*, *mountpoint*, *blockdevice*, *url*.

The element *filesystemtype* contains a string that specifies the format of the disk image: swap, ext3, xfs, etc. Because this string will be written to the guest domain's */etc/fstab*, it must be a valid mount type. The element *mountpoint* specifies where on the filesystem this image will be mounted (e.g. */home*). This value is also used in generating the guest domain's */etc/fstab*. The element *blockdevice* specifies the "device" that the guest domain will refer to it by (e.g. */dev/hda1*). Finally, *url* specifies where on the local filesystem the disk image is located.

The *job* section specifies what command should be run on startup of the domU. At this point, only one *job* section is allowed.

4.3 Issues

There are several issues still to be solved before AutoXen can be used in a production context. Most importantly, I/O from the job must be staged back to the user's workstation from the Xen guest domain. This may mean that the guest domains need to be grid-aware in order to use GridFTP to send the output back. Once this is resolved, the next issue is how to deal with guest domains that have finished running a job and are now idle. Currently, they run until manually shut down, but in a production system, this is not feasible, since each domain consumes resources which are then unavailable to the host.

Additionally, there are questions as to who will build the disk images: the user, or a designated support person. Users not familiar with the tasks of bootstrapping an OS into a disk image will likely be intimidated by the process. This could be a significant barrier to wider adoption of AutoXen. It may be possible for a system administrator to build a number of different images, each targetted at a particular application, and provide those to users. There are also security issues with allowing anyone to build and run arbitrary images, which points to the need for some sort of digital signing of disk images.

Closely connected with this last issue is the problem of shipping multiple gigabyte files around the grid. At these sizes, bandwidth used on average per job will increase, along with the start-up time costs for Xen jobs. If users have to wait an extra hour or two for their jobs to run, this could be an additional barrier to adoption of this system.

5 Summary

The AutoXen project proved that using Xen in a grid environment is feasible. Several challenges were solved this semester, including initial architecture design, disk image customization, and permission/sudo issues. AutoXen is a lightweight solution to the problem of how to integrate Xen into the grid. With a few additional feature additions, AutoXen will be ready to roll out to a real cluster and run real jobs for a select group of users who are knowledgeable enough to make and customize their own disk images.

References

[1] A Toroidal LHC ApparatuS (ATLAS); http://atlas.ch/, Accessed December 11, 2006

[2] Organisation Europenne pour la Recherche Nuclaire (CERN); http://public.web.cern.ch/Public/Welcome.html, Accessed December 11, 2006

[3] GridX1: A Canadian Computational Grid; http://www.gridx1.ca, Accessed December 18, 2006

[4] LHC; http://public.web.cern.ch/Public/Welcome.html, Accessed December 11, 2006

[5] GridX1 Wiki; http://wiki.gridx1.ca, Accessed December 18, 2006