2007

Maestro: a remote execution tool for visualization clusters

Aron Lee Bierbaum
Iowa State University

Maestro: A remote execution tool for visualization clusters

by

Aron Lee Bierbaum

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering

Program of Study Committee:
Carolina Cruz-Neira, Major Professor
Julie Dickerson
Chris Harding

Iowa State University

Ames, Iowa

2007

TABLE OF CONTENTS

LIST OF FIGURES v

ACKNOWLEDGEMENTS vii

ABSTRACT viii

1. INTRODUCTION 1

Research Problem 1

Statement of Purpose 2

Scope of Research 2

Define the Requirements for a Remote Execution Tool 2

Analyze Existing Remote Execution Tools 3

Design Maestro Based on the Defined Requirements 3

Develop the Initial Implementation 3

Perform Iterative Refinement of the Design 3

Discuss Results 3

Thesis Organization 4

2. BACKGROUND 5

Virtual Environment 5

Immersive Visualization Clusters 6

3. REQUIREMENTS 7

Security 7

Cross Platform 8

Interactive Remote Desktop 8

Complete Control of Execution Environment 8
Custom GUI for Application Execution 8
Screen Saver Management 9

4. EXISTING CLUSTER EXECUTION TOOLS 10

PsTools 10
Description 10
Strengths 11
Limitations 11
Rexec/SSH 11
Description 11
Strengths 12
Limitations 12
Parallel Program Trees 12
Description 12
Strengths 13
Limitations 13
REMOTE++ 13
Description 13
Strengths 14
Limitations 14

5. ARCHITECTURE OF MAESTRO 15

Software Library Structure 15
Core Components 16
Network Communication 16
Client GUI 17
Plug-in Management 17
Configuration 17

6. MAESTRO IN DETAIL 19
Network Communication 19
Authentication 20
Cluster Configuration 21
Application Configuration 22
Command 23
CWD 23
Environment Variable 23
Environment List 24
Argument 24
Choice 24
Group 25
Reference 25
Override 25
Add Options 25
Remove Option 25
GUI Components 26
Launch View 26
Reboot View 27
Process View 28
Resource View 29
Desktop View 30
Ensemble View 31
Stanza Editor 31

7. DISCUSSION 34
Event System 34
Stanza Reference Option 34
Automated Logon 35

Logon Desktop 35

8. CONCLUSIONS 36

9. FUTURE WORK 38

BIBLIOGRAPHY 39

LIST OF FIGURES

Figure 5.1 Maestro Components 16

Figure 6.1 Event Propagation 19

Figure 6.2 Example Ensemble File 22

Figure 6.3 Example Stanza File 23

Figure 6.4 Launch View 27

Figure 6.5 Reboot View 28

Figure 6.6 Process View 29

Figure 6.7 Resource View 30

Figure 6.8 Desktop View 31

Figure 6.9 Ensemble View 32

Figure 6.10 Stanza Editor 33

ACKNOWLEDGEMENTS

I would like to thank Dr. Carolina Cruz-Neria for her continued support throughout my college career. She has provided me with more opportunities than I could ever ask for. I would also like to thank my colleagues and friends Allen and Patrick. They have taught me more about software engineering than any book could. They have also provided valuable feedback and support throughout this research.

Finally, I would also like to thank my fiancée Lindsay for her support through this process. She has helped me in more ways than I can count, and I can't wait to spend the rest of my life with her.

ABSTRACT

In recent years immersive visualization systems have transitioned from running on large shared memory systems to clusters of commodity PCs. While there has been much research done to create middleware to manage application synchronization, there has been very little work done to allow easy execution of immersive applications on a cluster. Although there are existing remote execution tools, they are targeted at high performance computing (HPC) and enterprise administration. This thesis presents Maestro, a cross-platform remote execution tool designed specifically for visualization clusters. The goals of Maestro, a description of its design, and a detailed discussion of its implementation are provided. The design description gives explanations of the three components of Maestro: the core that handles networking and security, the user interface that controls the cluster, and the daemon on each cluster node. Maestro has been successfully deployed on numerous large visualization clusters, which have led to refinements and improvements to the tool.

1. INTRODUCTION

Research Problem

With the increasing performance of commodity hardware visualization systems have started shifting to clusters of PCs rather than large shared memory systems. There has been considerable research done to create middleware that allows these applications to be synchronized across these clusters. This research has continually skipped over the significant issues of executing these applications on the remote cluster nodes.

There are existing tools that allow a user to launch an application on remote nodes. But these tools do not address the specific needs of a clustered virtual reality (VR) application. There are two different sets of existing tools. The first is composed of administration tools that are designed to allow a system administrator to run applications on any individual nodes. The second group addresses the needs of job scheduling on computation clusters. Tools from both groups will be discussed in more detail in Chapter 4.

Users of graphics clusters should not have to sacrifice the simplicity of launching an application from the master node of the cluster. Rather than writing custom scripts and using remote shells to launch an interactive, immersive application users should be able to launch applications across all the nodes of a cluster in a simple, uniform manner. We believe that the current remote execution tools do not address the specific needs of launching a clustered VR application. The need for a better tool is the driving force behind this research.

Statement of Purpose

The research presented here begins by specifying the fundamental requirements for any remote execution tool that can be used on a visualization cluster. Using these requirements as the basis for comparison, we will analyze existing remote execution tools emphasizing their strengths and limitations. The results of this analysis will be used to define the needs of a tool that can be used to effectively execute remote commands on a visualization cluster. This tool, called **Maestro**, is the focus of the work presented in this document. Maestro is a cross-platform tool that allows execution of applications across a visualization cluster.

Scope of Research

To meet the research goals proposed in the statement of purpose, the work is structured in the following stages:

1. Define the requirements for a remote execution tool.
2. Analyze existing remote execution tools.
3. Design Maestro based on the defined requirements.
4. Develop the initial implementation.
5. Perform iterative refinement of the design
6. Discuss results.

Each of these steps is described in detail in the following subsections.

Define the Requirements for a Remote Execution Tool

Before researching any existing remote execution tools, the goals and requirements of the Maestro project had to be defined. By doing this first it simplified the evaluation of existing tools because key criteria could be used to determine the usefulness of each tool. These requirements have also been useful in identifying deficiencies in various iterations of development and stating goals for future work.

Analyze Existing Remote Execution Tools

Existing remote execution tools were investigated to determine if any one met the requirements of this research. The goal was to ensure that work was not being repeated and to define that contributions Maestro could make to the field. This stage of the research helped in designing Maestro because features offered by other tools could be included with those offered by Maestro.

Design Maestro Based on the Defined Requirements

Upon reviewing existing tools, we feel that our plan to create a new Remote Execution tool is justified. No other implementation offers the exact feature set we want. While there are many tools that address remote execution on computation clusters, none are designed specifically for visualization clusters. The design of Maestro was refined based on what was found in existing tools, but the primary goals remain the same.

Develop the Initial Implementation

A simplified initial implementation was written and tested on a visualization cluster. The initial work did not meet all of our goals and had problems that needed to be overcome. The first pass proved to be slow and hard to use, and therefore required many additional development iterations.

Perform Iterative Refinement of the Design

The design of Maestro has been refined continuously based on developer ideas, user feedback, and further research into new remote execution tools. Maestro has been demonstrated and installed on several large visualization clusters and input from users has been taken into consideration during development iterations. The initial design has been extended to allow complex authentication, better error handling, and complex application launching configurations.

Discuss Results

The results of using Maestro on large visualization clusters show that while Maestro is a very useful tool there are still problems. These problems have been taken into consideration for future iterations of

the development cycle.

Thesis Organization

The remainder of this text is organized as follows:

- Background information for remote execution and visualization cluster is provided in Chapter 2.
- Requirements for a remote execution tool are presented in Chapter 3.
- Previous remote execution tools are discussed in Chapter 4.
- A high-level description of Maestro's architecture is given in Chapter 5.
- The details of the architecture are presented in Chapter 6.

2. BACKGROUND

In this chapter we will describe some fundamental concepts that need to be understood before describing Maestro in detail. We will start with an overview of virtual reality and its current uses. Then we will describe the shift away from large shared memory systems towards clusters of commodity machines used to generate these virtual environments.

Virtual Environment

A "virtual environment" is defined by Stuart as:

> "Systems capable of producing an interactive immersive multisensory 3-D synthetic environment" it uses position-tracking and real-time update of visual, auditory, and other displays (e.g., tactical) in response to the user's motions to give the users a sense being 'in' the environment, and it could be either a single or multi-user system." [21]

The above definition points out several key components to the VR experience. First, a virtual environment must respond to user input through some set of hardware devices. Second, the user is immersed in the application such that their attention is focused on the sensory elements in the space. Finally, the environment is rendered in real time by the hardware while responding to the users actions at any given time. This is much different than pre-recorded animations. The combination of these components results in a VR application that is centered on the user.

Immersive virtual environments allow users to view, navigate and/or modify three-dimensional models with a first person perspective. The type of immersion achieved in these graphical systems allow users to behave in the 3D virtual scene the same way they would behave in a real environment. Immersive visualization (IV) systems are currently used in different applications such as scientific data

visualization (metabolic networks [6], fluid dynamics [23], information flows [14]), e-learning [9], collaborative design [7, 15] and computer games [13].

Immersive Visualization Clusters

Traditionally immersive visualization systems have been designed on dedicated, high-end, shared memory computers to generate interactive virtual environments. In recent years, this almost exclusive use of high-end computers for these purposes has shifted to commodity hardware. This is mainly because using commodity hardware has become a low-cost alternative [5, 12, 20]. Continuous, rapid improvements in commodity hardware have allowed designers of immersive visualization systems to employ high-quality graphics hardware, high-speed processors, and significant amounts of memory with much lower costs than would be possible with high-end, shared memory computers.

Graphics clusters pose unique challenges. Most obviously, the software run on graphics clusters is normally interactive. Users provide live input in order to see immediate results. Each node normally has one or more display devices (monitor or projector) connected to it. Furthermore, the processes running on each node normally work in concert to produce what appears to be a unified image rather than, say, each node working autonomously to compute a small part of a large problem. Thus, users of a graphics cluster are actively using all the nodes of the cluster simultaneously. This is in contrast to a computational cluster where a user sits down at the master node and starts up a computationally expensive job and then leaves the computers to run until the work completes.

3. REQUIREMENTS

In this chapter we present the components that make up the minimum requirements for a remote execution tool specifically designed for immersive visualization clusters. These components reflect our experience researching and evaluating existing remote execution tools, developing Maestro, and deploying Maestro on large clusters. The components presented are as follows:

- security
- cross platform support
- interactive remote desktop
- complete control of execution environment
- custom GUI for application execution
- screen saver management

Details for each of these components are given below.

Security

In order to execute an application on a remote machine, a tool must address a few security issues. First, the user must be able to authenticate with the remote node. This allows the software to ensure that the user has the correct rights to run the application on the cluster. Also authentication allows the remote application to access the user's secure files. Once we successfully launch an application we also need to ensure that all network traffic is encrypted. While a cluster is often located on an isolated network, it is still important to guarantee that an un-trusted individual could not get access to potentially classified information.

Cross Platform

Most existing remote execution tools are designed to run on UNIX based systems, with the exception of a few tools that have been ported to work on Windows®. These tools are limited because they are not designed to run on multiple platforms. A useful remote execution tool must allow the user to switch between platforms transparently. For example, it is important that a user can remotely execute an application on a cluster of nodes running Linux while using a client Graphical User Interface(GUI) on their Windows laptop.

Interactive Remote Desktop

The main difference between an immersive application and normal console application is that it needs to be able to open a graphics window on each cluster node. In order to open this window, the application must be able to interact with the current desktop. This requires the remote execution tool to grant the remote process additional access rights. The tool should allow an authenticated user to launch an interactive application without being logged into each cluster node.

Complete Control of Execution Environment

The remote execution tool should not require any changes to the application being executed. Since applications can receive input in many different ways the user should be able to control the complete execution environment. Users should be able to specify the command to execute, the current working directory, environment variables, and command line arguments. This information should be captured in a reusable application configuration file. In the past users have addressed this issue by using custom scripts that set up the correct environment for each application. We believe that this process is cumbersome and error prone.

Custom GUI for Application Execution

Most applications can be executed in different ways depending on their execution environment and command line arguments. We feel that it is the remote execution tool's responsibility to capture

these options and present them to the user in a meaningful way. For example, if an application can be launched full screen by specifying -f, the user should be presented with this option. Also if an application takes an input file the tool should allow the user to choose a file before executing the application. The goal is to present the user with an intuitive way to launch an application on the cluster.

Screen Saver Management

The purpose of a screen saver is to prevent damage to a monitor that is displaying the same image for long periods of time. Normally a screen saver changes stops displaying this image after a certain amount of time without user interaction. A problem occurs when running an application across a cluster because the user is not using the keyboard or mouse on each remote node. We feel that a remote execution tool should allow the user to manage the screen saver settings on each node in the cluster. This would allow the user to disable screen savers while running an application.

4. EXISTING CLUSTER EXECUTION TOOLS

PsTools

Description

One of the few remote execution tools that exist for Microsoft Windows® is a package called PsTools. This package is a collection of command line utilities for remote execution and system administration. The first utility, PsInfo is used to gain general information about a given machine on your network. This general information includes computer name, owner, processor speed, physical memory size, and hard disk configuration. Next, PsList is used to get a list of all processes running on a given machine. The user can query process details which include, but are not limited to, start time, priority, number of threads, memory usage, and CPU usage. PsKill is similar to the UNIX kill command. It can terminate processes on remote machines by name or process ID. Since it can also be desirable to suspend a process that is consuming a resource, PsTools contains PsSuspend which only suspends a process to be resumed later. The user can also use the PsShutdown utility to reboot or shutdown a remote machine.

Remote execution is accomplished using a utility called PsExec. This utility allows the user to launch a command on a remote machine. Unlike other Microsoft Windows® based solutions, PsExec allows the user to launch interactive applications and supports user authentication. This is all accomplished using the administrative shared ($C) to execute the remote command and tunnel all application output back to the user's terminal

Strengths

PsTools is the most powerful and flexible remote execution tool that we found that had native support for Windows®. It supports many of the requirements described in Chapter 3. It supports user authentication by passing a username and password on the command line. Since PsExec takes advantage of existing features it has the added benefit of not requiring any installation on the client or server.

Limitations

PsTools was designed for network administration, not remote execution. It does not support the concept of a cluster of nodes. This means that the user would need to write custom scripts that would run their application on each node in the cluster. PsTools also does not meet our requirement of being cross platform. Also, during testing we encountered problems with user authentication. PsTools is a very powerful tool that unfortunately was not designed with our specific needs in mind.

Rexec/SSH

Description

Many UNIX based clusters address the problem of remote execution using custom scripts and features of RSH and SSH which allow launching a remote process [16].This method is fairly straight forward since all needed tools are included with standard UNIX distributions. It is simple to run a single remote command using these tools. It becomes much more difficult when a user needs to manage connections to a large number of machines. Since we need to launch a process on each node in parallel, this solution would need to open a terminal window for each node in the cluster in order to show results of the operation. This solution does not scale well with the current increase in visualization cluster size. Also, using scripts tends to break down quickly because they are home grown custom solutions that don't take into account future needs.

Strengths

Much like PsTools, this method usually works correctly without any user intervention. Once the daemon is installed and started on a server, any client with the correct credentials can execute a command. Also using SSH has the benefit of using encrypted network traffic. This can be very important if the user is running a confidential application on the cluster and needs to protect the data being transferred.

Limitations

This solution is very popular but it does not address many of our requirements. Once again the major problem with this method is that it has very limited support on Windows®. There are a few SSH servers that exist for Windows®, but they lack integrated authentication and desktop interaction. This solution also requires the user to write custom application dependent scripts that become very error prone.

User authentication is supported, but requires the user to take one of three actions. First, they could type their password once for each node. This will quickly become an annoyance when using a large cluster. Next, the user could include their password in the launch scripts. Storing your password in plain text is very insecure and should be avoided at all costs. Last the user could set up SSH keys to allow access without passwords. Setting this up correctly requires the user to complete a lengthy process and run a SSH agent before launching the application.

Parallel Program Trees

Description

There has been work done within the Apache group to find a simple, but efficient solution that builds upon features of RSH and SSH. This approach differs in that the controlling machine does not make a connection to each node in the cluster. Instead it connects to some k number of nodes and tells them to start the process. These nodes at the first level then connect to k more nodes and inform them to start the process. As you can see this forms a tree structure of TCP connections.

Strengths

The parallel program trees method was successful in decreasing the amount of time that it takes to launch a process by more than an order of magnitude [16].

Limitations

Although this remote execution method is much more efficient than SSH, it still suffers from the same limitations. It was designed specifically to execute console applications on large computation clusters. It does not address any of our high level requirements such as screen saver management.

REMOTE++

Description

Another solution that exists for the Windows® operating system is Remote++ [11], which was developed as a replacement for REMOTE. Both tools are designed specifically for applications that require multiple runs to complete a single task. Time parallelization allows the application to be run on multiple machines at the same time with different input values to reduce the overall execution time of the simulation. Parallel Independent Replications (PIR) uses time parallelization by distributing the executable with different input values to a set of nodes. Unlike its predecessor, REMOTE++ uses standard remote shell (rsh) and remote copy (rcp) commands.

There are a few requirements in order to be able to run an application on remote nodes. First, all remote nodes must be running rsh and rcp daemons. Also, the master node needs to have a joblist.txt file containing a list of executables with their respective input and output files. Finally, the master node needs to have a file named hostlist.txt that lists all available nodes.

In order to run an application on each node the user runs remotepp which loads the list of jobs and remote nodes. First, each job is matched up with an available node. Next, the executable and input files are copied to the node using rcp and the application is executed using rsh. Finally, after the application completes the output file is copied back to the master node.

Strengths

REMOTE++ is one of the few tools that exist to execute applications remotely on Windows®. REMOTE++ also uses standard and well tested tools to accomplish its goal of remote execution. Since it uses these standard rcp and rsh it appears very familiar to UNIX users.

Limitations

REMOTE++ was designed specifically for PIR applications that run on Windows. While it does a good job of addressing it's goals, it does not meet many of our requirements. It is not cross platform even though it uses standard commands that are found on UNIX. It also does not provide an easy way to manage the applications to execute on the remote machines. The target application is also limited to taking one input file and getting one output file. This does not address our requirement of being able to specify the complete execution environment for each node in the cluster.

5. ARCHITECTURE OF MAESTRO

After reviewing the existing solutions described in the previous chapter, we felt that it was necessary to design a new remote execution system specifically targeted at immersive visualization clusters. Most existing systems are designed for either high performance computing (HPC) clusters or large-scale corporate network administration. Designing a system with the needs for cluster of commodity hardware running a virtual reality system will result in a more useful and robust system.

To overcome these limitations, Maestro was designed with the following goals in mind:

- Cross platform
- Separation of cluster configuration and application configuration
- Highly extensible
- Secure authentication methods
- Easy installation and configuration

The following sections describe the high level system concepts and the motivation behind the system design. The detailed design description is in the following chapter.

Software Library Structure

Maestro is separated into three different modules, Figure 5.1: the core library, the server side daemon, and the client side graphical user interface (GUI). The core library contains event handling, user preferences, error handling, user authentication, and plug-in management. The daemon module contains a platform independent server that provides an abstract view of the local cluster node. Finally, the

GUI module contains a client that provides an overview of the entire cluster by aggregating information from each node's daemon.

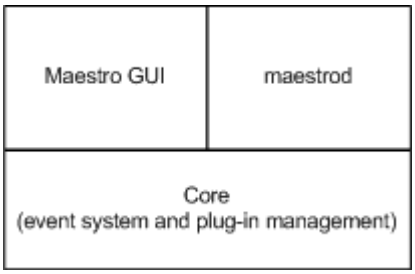

Figure 5.1 Maestro Components

We chose to write Maestro is in Python for two main reasons. The first is that it allows us to easily accomplish our goal of having a cross platform system. Python provides us with an abstraction layer that allows us to ignore the low level platform independent issues such as networking and endianness. The second reason is that Python is a scripting language and does not require any compile time. This allows us to do rapid prototyping and have shorter development iterations.

Core Components

Network Communication

When initially designing Maestro we needed to decide on a networking abstraction layer that would allow cross platform development. We also decided that we did not want to implement our own network protocol. Instead, we decided to use an existing Remote Method Invocation (RMI) framework. This allowed us to concentrate on making a useful tool rather than implementing a network protocol. The first version of Maestro used a rather simple RMI toolkit, Pyro [2]. After a few months of development we found Pyro did not meet our needs because it was not designed for asynchronous connections. We then chose to use Twisted's RMI module, ProspectiveBroker (PB) [8]. PB provided us with everything that we needed, including secure connections over SSL.

Client GUI

After reviewing many different GUI toolkits we chose Qt [3] for many reasons. First, the Python bindings, PyQt [1], are very complete and well tested. Also Qt's signal and slot mechanism fits very well with our own event system. Last, using Qt Designer we can manage the look and layout of the GUI separate from the application logic layer.

Plug-in Management

One of the main goals of Maestro was to allow users to easily extend the core functionality. We accomplished this by designing Maestro to use a sophisticated plug-in system that automatically finds and loads plug-ins at start up [22]. In order to extend Maestro a user only has to implement one of the plug-in interfaces and place their module in a location that Maestro can find it. Maestro then loads the plug-in at the appropriate time and attaches it to the core system.

Configuration

Another goal of Maestro is to separate the application configuration from the cluster configuration. This is accomplished by storing all cluster information in Ensemble files and all application specific configurations in Stanza files. The idea being that an Ensemble is a group of nodes much like an orchestra and a stanza contains application instructions much like a musical score.

An Ensemble configuration file is a list of cluster node elements, which contain the name, host name, and class of each node. The class attribute allows the user to break up a cluster into smaller groups. In turn this allows Maestro to launch the application slightly different on each smaller group of nodes. This is most commonly needed for applications that use a client/server network topology and need to run a different executable or pass extra arguments to the server node.

The Stanza file contains all information needed in order to launch an application on a cluster. This allows the Stanza files to be easily distributable with new applications. It is important to keep in mind that Stanza files are not meant to specify a static set of options used to launch an application. Instead they are meant to store how an application is launched and all of the different launching options. Maestro then uses this information to present the user with a custom interface that allows them to select

how to launch the application. For example, you could specify the command and a list of possible data files to load. When launching the application the user is then presented with this predefined list of data files to choose from while launching. More details about all configuration options and how they effect the GUI will be discussed in Chapter 6.

6. MAESTRO IN DETAIL

Building on the background given in the previous chapter we now present the implementation details of Maestro. We will begin by describing the network communication between the client GUI and the cluster nodes. The authentication system architecture is then discussed in detail. Next, the details of cluster and application configuration are explained. Finally, the chapter concludes with a description of the Maestro GUI components.

Network Communication

As described in the previous chapter, when designing Maestro we decided to use an existing Remote Method Invocation (RMI) framework instead of implementing our own network protocol. We wanted to design a network communication layer that was dynamic and easy to extend. In order to achieve this goal we modeled the communication after the Observer Pattern [10]. This allows the Maestro GUI to register callbacks to receive generic events when a cluster node's state changes.

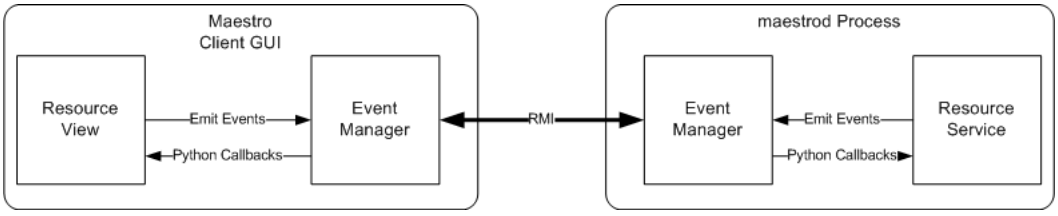

Figure 6.1 Event Propagation

Both the Maestro GUI client and the maestrod server have an `EventManager` that handles emitting events to the correct remote node and routing incoming events to the correct python callback. When the Maestro client GUI connects to a cluster node, Figure 6.1, it uses RMI to register it's `EventManager` to receive events from the server. During this process the server also registers its `EventManager` to receive events from the client GUI. This allows events to pass back and forth be-

tween the Maestro client GUI and the cluster nodes transparently. The routing of theses events depends on a destination host name. If an event should be sent to all cluster nodes the event's destination is set to '*'.

Maestro plug-ins emit events and register callbacks with their local `EventManager` object. A plug-in can emit a signal by specifying a destination host name, an event id string, and a list of arguments. The plug-in can also register callbacks to receive events with a specific event id string. For example, the Resource View can register to receive 'settings.cpu_usage' events whenever a cluster node reports it's current CPU usage. It can then emit a 'settings.get_cpu_usage' event to all cluster nodes. When maestrod receives this event it will get the current cpu usage and emit a 'settings.cpu_usage' event with the corresponding data. The Resource View then receives this event and updates it's display to show the current usage.

Authentication

User authentication is an important requirement for a remote execution tool. It is very important for Maestro to handle authentication correctly because Maestro is a service that executes code and commands on behalf of users. Authentication is the first phase of the connection process between the Maestro GUI and maestrod, the Maestro daemon. If authentication fails then the GUI user cannot perform any actions on the server side, even those that do not strictly require authentication.

Authenticating users proved to be a difficult task because of the large differences in authentication on different operating systems. In order to address this difficulty, Maestro uses a plug-in architecture. For each authentication method there is a client side plug-in and server side plug-in that handle their authentication independently of the other methods. Below, we list the current set of authentication plug-ins:

- **User Name and Password:** This authentication method is supported on all platforms, and includes support for domains on Microsoft Windows®. The implementation of this plug-in for non-Windows® plug-ins currently relies upon PAM [19]. For Windows®, the win32security.LogonUser() function is called with the given user name, password, and domain to get a user handle.

- **Active Directory:** Domain-level authentication using Kerberos 5. Direct use of Active Directory for authentication via credentials forwarding is available only on Microsoft Windows®. Other platforms may be configured to use Active Directory for authentication behind the scenes (through PAM, for example). For nodes that are trusted for delegation, users can forward their existing credentials to the Maestro service and never have to enter their user name or password.

- **NT LAN Manager (NTLM):** Basic Windows® challenge/response authentication. This is only supported for Windows® clients and servers. The use of NTLM for this supports single-hop credentials forwarding, meaning that a user can authenticate with the Maestro service on a remote node but cannot access remote resources from that remote node.

The authentication process is broken into a two-phase process. The first phase is negotiating a list of authentication methods that the client GUI and maestrod support. The second phase iterates through the list of authentication plug-ins trying each one to successfully authenticate.

Cluster Configuration

Ensemble files list all nodes that are in a particular cluster. As shown in 6.2 the file structure is very simple. Each node has three attributes, class, host name, and name. The class is a comma-separated list of identifiers indicating the role of the node in the cluster, which comes into play with the stanzas and application launching. For example, a node could be put into a master or slave class. By default, every node in the ensemble has its operating system name as part of its class, so this need not (and should not) be specified in the ensemble file. The host name is the IP address or machine name for the node. This is what the Maestro GUI uses for making the connection to the remote node. Finally, the name is a “friendly” identifier for the node that is used whenever the nodes of the ensemble are displayed in the Maestro GUI.

Ensemble files can be edited using the Maestro GUI’s Ensemble View, which will be discussed in more detail below. Generally these configurations would only need to be modified once when setting up a cluster. An administrator could then put the ensemble file in a place that is convenient for accessing by all users of the cluster.

```
<?xml version="1.0"?>
<ensemble>
  <cluster_node class="" hostname="node0" name="Node 0" />
  <cluster_node class="" hostname="node1" name="Node 1" />
</ensemble>
```

Figure 6.2 Example Ensemble File

Application Configuration

As discussed in the previous chapter, Stanza files contain all information about how to launch an application on a cluster. This is not limited to a static set of options, but also includes a description of application specific launching options. This allows the stanza author to describe the application in a dynamic way that allows future Maestro users to have more control over how the application is launched.

Each stanza file is composed of stanza items that are used to describe the execution environment and launch options. These options can be comprised of any of the following stanza items: choice, group, argument, environment variable, command, current working directory (CWD), reference, override, add options, remove options, or environment list. Each of these option types will be discussed in more detail below.

Each stanza item can have a class attribute that determines if the option should be used on a given cluster node. As described above each cluster node has a class attribute that is used in a matching process that decides if the option should be used. The matching is fairly simple. If all the class tokens of a stanza item match up with class tokens of the ensemble node, then the stanza item is used with that node. The more class tokens used on a stanza item, the more specific the matching with ensemble nodes will be. A stanza item with an empty class setting therefore matches all ensemble nodes. For example, Figure 6.3 shows an example of a stanza file that specifies different commands for nodes with the “master” and “slave” classes.

Example 4.3. Application Stanza Using Classes for Master and Slaves

```
<?xml version="1.0"?>
<stanza>
 <application name="myapp" label="My VR Application">
  <command class="master" name="master_cmd">
   ${APP_DIR}/masterApp
  </command>
  <command class="slave" name="slave_cmd">
   ${APP_DIR}/slaveApp
  </command>
 </application>
</stanza>
```

Figure 6.3 Example Stanza File

Command

The command option specifies the location of the application to be executed on each remote node. This command can varied depending on the options's class attribute. For example, a common cluster configuration is to have one master node and many slave nodes. Using two different command options you could specify one command for the master node and another to be run on each of the slave nodes. You could also specify a different command to be run on each operating system in your cluster.

CWD

The current working directory option specifies the directory that the application should be executed from. It is not required that this be the directory that contains the application since we can specify the full path to the application in the command option. This option addresses the needs of applications that use relative paths to access their data. For example if all images are loaded from a './data' directory the application must be executed from the parent directory.

Environment Variable

The environment variable option provides a method to apply a setting to the execution environment before starting the application. It is composed of a key value pair representing the setting name and value. These settings are applied to each cluster node before executing the specified command. It is

important to note that Maestro starts with a clean environment before starting an application. So in order to run an application all dependencies must be specified in a PATH ot LD_LIBRARY_PATH environment variable.

Environment List

The environment list option is a list of environment variables. In some cases, an application may use a large collection of environment variables, and creating individual environment variable items in the stanza can be quite tedious. Instead, an environment variable list can be used to capture multiple environment variables in one place. Each environment variable in the list is made up of a (the name of the environment variable) and zero or more values. If the stanza author allows the user to change the value of the environment variable, multi-valued variables will be presented using a combo box or an editable combo box when appropriate.

Argument

Command line arguments are the most common method to provide input to an application. The argument option provides a method of describing the command line arguments that can be used when launching. Maestro can then use this information at launch time to ask the user if they want pass the argument when launching, and what value to place with it. For example, an argument option could have the flag -i and value of data.txt. When launching, Maestro will ask them if they want to use this input file or specify another.

Choice

This option allows the stanza author to describe a choice that the user has when launching the application. For example, Maestro could force the user to choose between a predefined list of input data sets. Optionally the choice can be designated as being mutually exclusive. When using a mutually exclusive choice, the stanza author can choose to represent the choice with either radio buttons or a combo box.

Group

Creates a structural association of related items. This is useful to collect multiple settings behind a common facade. For example if there are four input files that conceptually form two distinct units the user could create two groups, “Input 1” and “Input 2.” If a choice is placed above these groups it forces the user to choose between the two units.

Reference

A reference allows reuse of options in other stanzas. The referenced option can be refined by using add, remove, and override operations. A reference identifies the XML element in the other stanza using a syntax similar to XPath [4]. This allows global options that define common settings or choices to be reused.

Override

While referencing an option from another stanza file it can be useful to change something about that option. An override allows the user to change the attributes and/or CDATA of the referenced option. For example one application may accept configuration files with “-i” flag while another may not require a flag at all. If the later application wants to use the argument option from the first stanza file the user can use an override to change the flag attribute to be empty.

Add Options

An add option allows the stanza author to place additional child options under the referenced option such as groups, choices, arguments, and environment variables. This allows the author to use an option described in a global stanza file but, for example add another possible value under a choice.

Remove Option

A remove operation deletes a child of the referenced structural item. The reference identifier uses the same path syntax as its parent reference.

GUI Components

The Maestro GUI is composed of a main window with a set of view plug-ins. In this chapter we will describe each of these view plug-ins in more detail.

Launch View

This view, Figure 6.4, provides the end user interface for remote application execution. The interface itself is built dynamically depending on the application stanza files described above. This means that each application will have a unique interface that allows the user to choose between different options specified in it's stanza file. This interface is composed of a basic set of controls. The layout of these controls will reflect the nesting and grouping of the options specified in the stanza editor. The basic set of controls are the following:

- Text fields where users enter values, strings, paths, etc. In the case of a file path, the stanza author may choose to indicate that a file chooser dialog box can be opened. Should that information be present in the stanza, a button will be placed to the right of the text field for opening the file chooser dialog.
- Radio button groups are used for controlling which mutually exclusive choice the user can make among different cases.
- Pull-down menus are used for long lists of mutually exclusive options in order to save space in the GUI layout. The stanza author is responsible for making the decision about when a pull-down menu is more appropriate than a group of radio buttons.
- Check boxes are used for groups of options that are not mutually exclusive.

When an application is launched, any output that it would normally write to a console window is streamed back to the Maestro GUI and displayed in the tabbed panel of log windows. Note that each tab can be detached from the Maestro GUI and moved and resized independently. This allows the output from multiple nodes to be seen simultaneously if so desired.

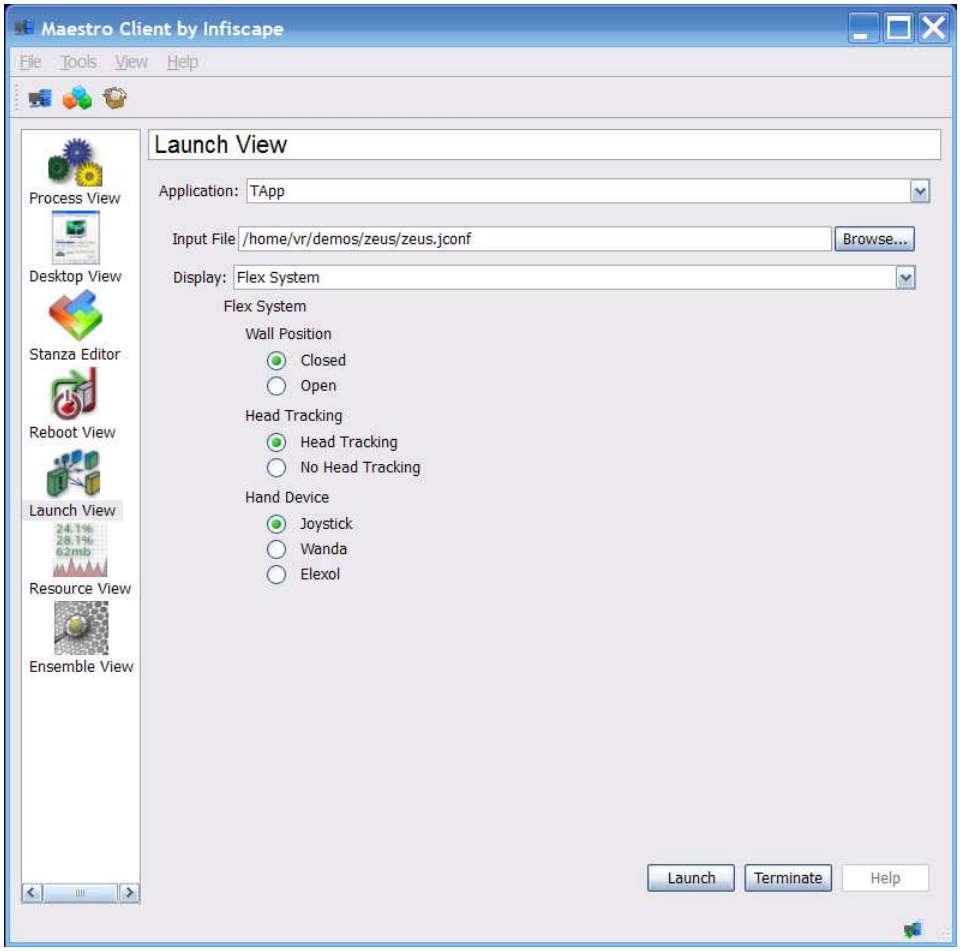

Figure 6.4 Launch View

Reboot View

The Reboot Viewer, Figure 6.5, allows the user to reboot individual nodes of the ensemble or all the nodes of the ensemble. When possible, the Reboot Viewer also allows the GUI user to specify the target operating system when rebooting. This can be done for all nodes at once using the buttons at the top of the view panel, or it can be done for individual nodes. To change the boot target for an individual node, click on the table item that identifies the current OS, and a pull-down menu will be activated that allows the target OS to be selected.

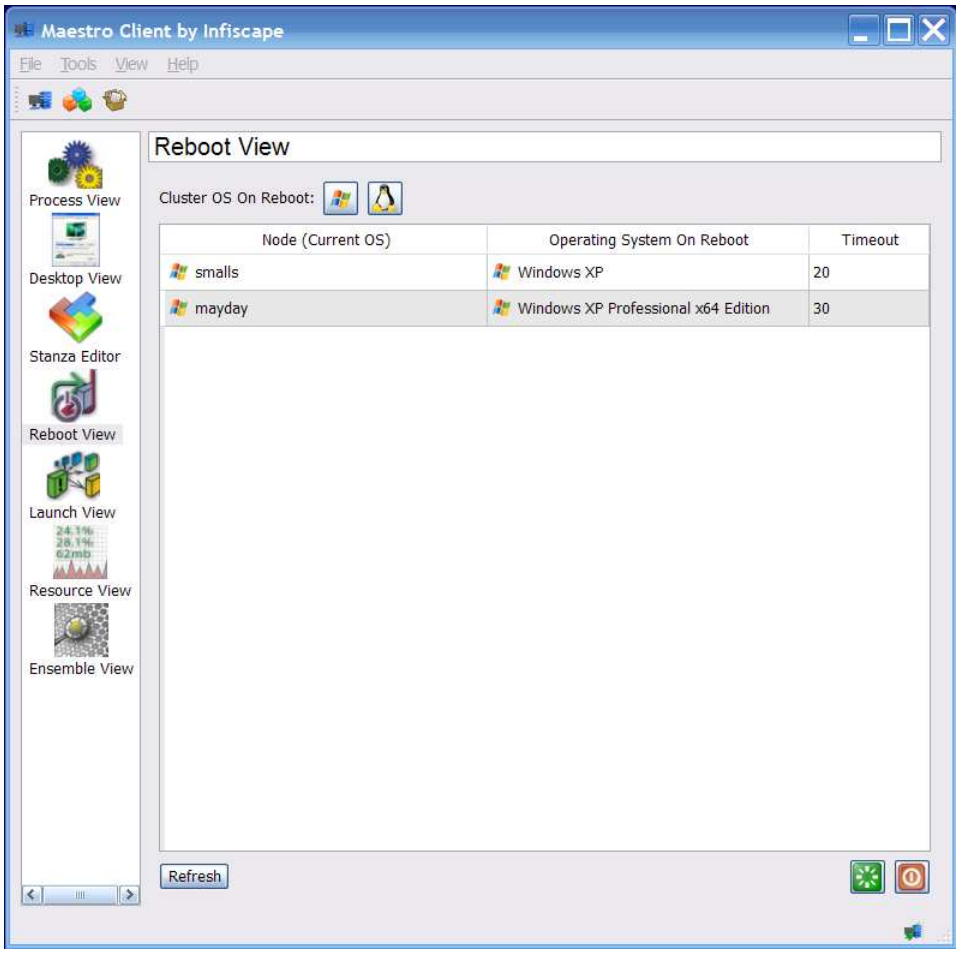

Figure 6.5 Reboot View

Process View

This view provides a list of all processes, Figure 6.6, currently running on the cluster as well as a means to terminate individual processes. This becomes very useful when a previously launched application fails to terminate. When first loaded, the Process Viewer displays nothing. To populate the process list, click the Refresh button. Queries will be sent to each of the ensemble nodes requesting the current list of the running processes. To update the list, click the Refresh button again. To kill a process, select it from the list and click the terminate button. You can terminate more than one processes by selecting multiple list items.

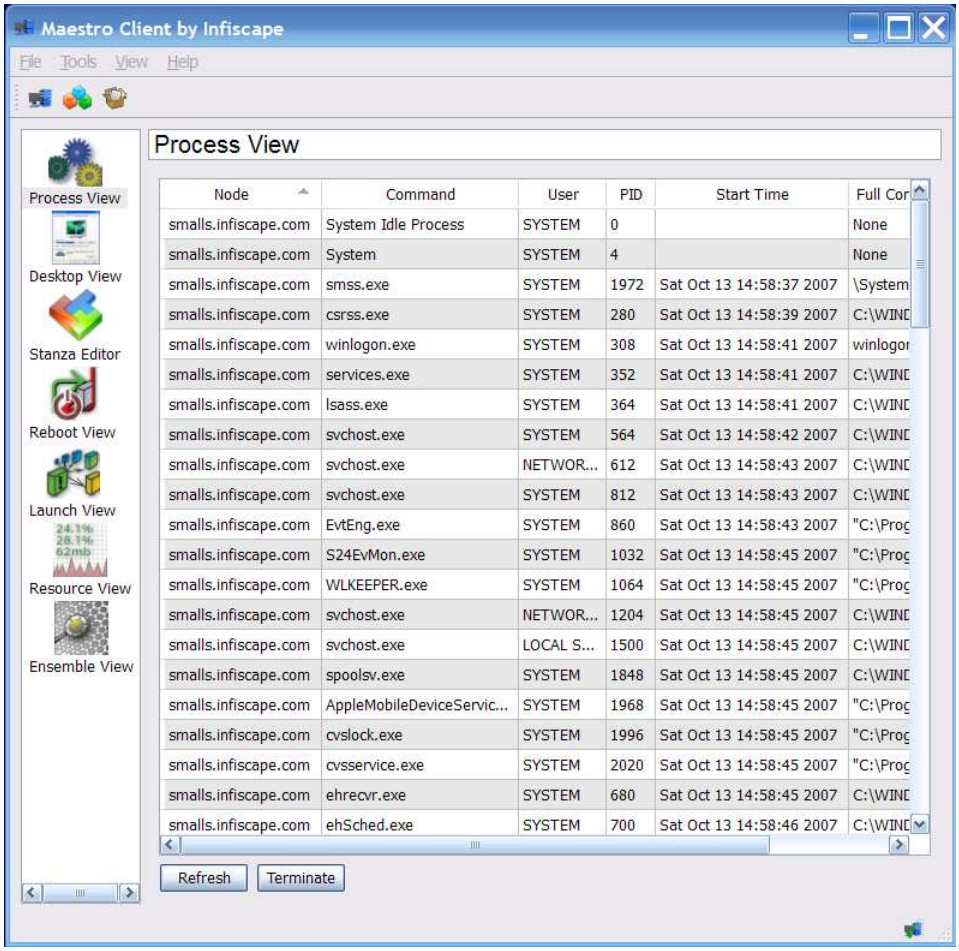

Figure 6.6 Process View

Resource View

The Resource Viewer, Figure 6.7, provides the user with a representation of the current resource consumption on each node in the cluster. Currently, the resources tracked are CPU and memory usage, and the graphs show this usage as a percentage of the total available. When first loaded the view does no show the state of any nodes. There are two methods to gain the resource consumption of a node. First, by clicking on the refresh button each node will report it's instantaneous usage. You can also request to receive continuous updates for any set of cluster nodes. In order to get continuous updates, right-click on a node and select the desired update rate from the context menu. To stop the updates, choose the Off item from the context menu.

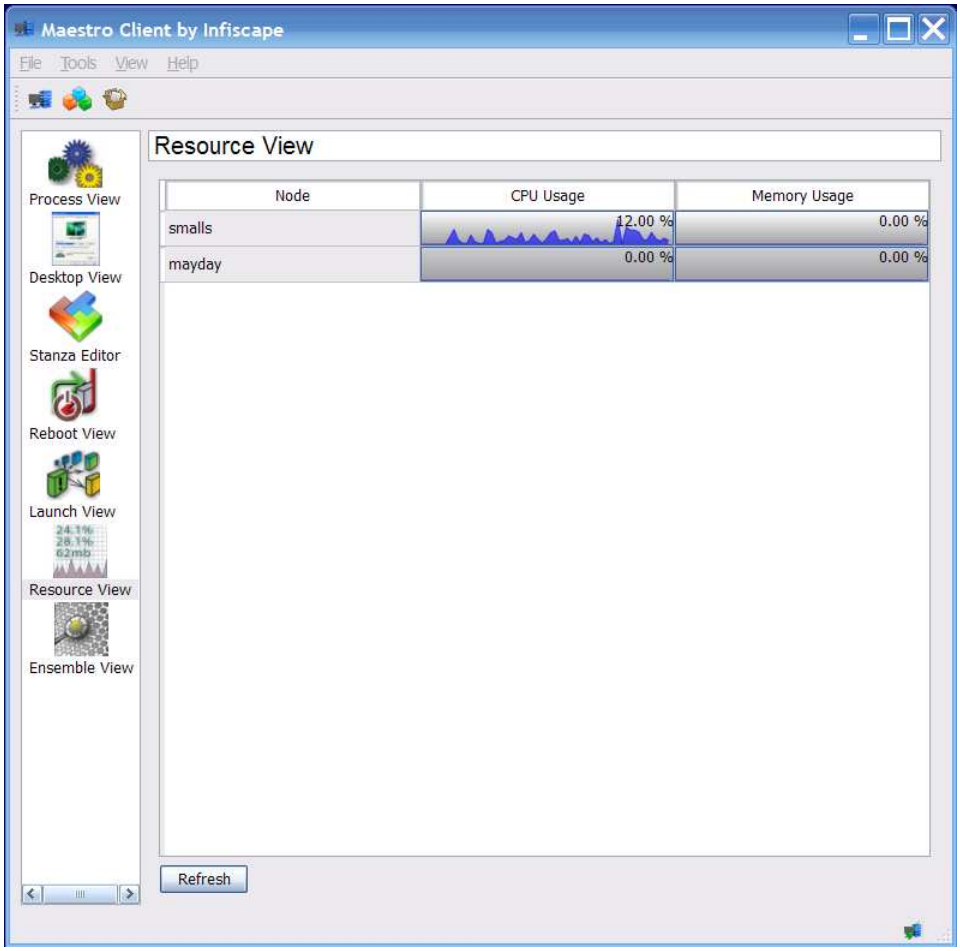

Figure 6.7 Resource View

Desktop View

The Desktop Viewer is where users can control the state of screen savers and power management for each node in the cluster. At the top of the view there is a pull-down menu that allows the user to select either all nodes or an individual node. The user can select a specific node to make changes for or select all nodes to apply the changes to every node in the cluster. There are three main functions that this view can perform. First by changing the state of the "Screen Saver Enabled" checkbox the user can disable screen savers. A running screen saver can be terminated by clicking the Stop Screen Saver button. This button also has the effect of unblanking a display and waking up a sleeping display if power management is still active. The user can change the current background image by specifying an image. The GUI will package the image up and send it to the remote cluster node to be used.

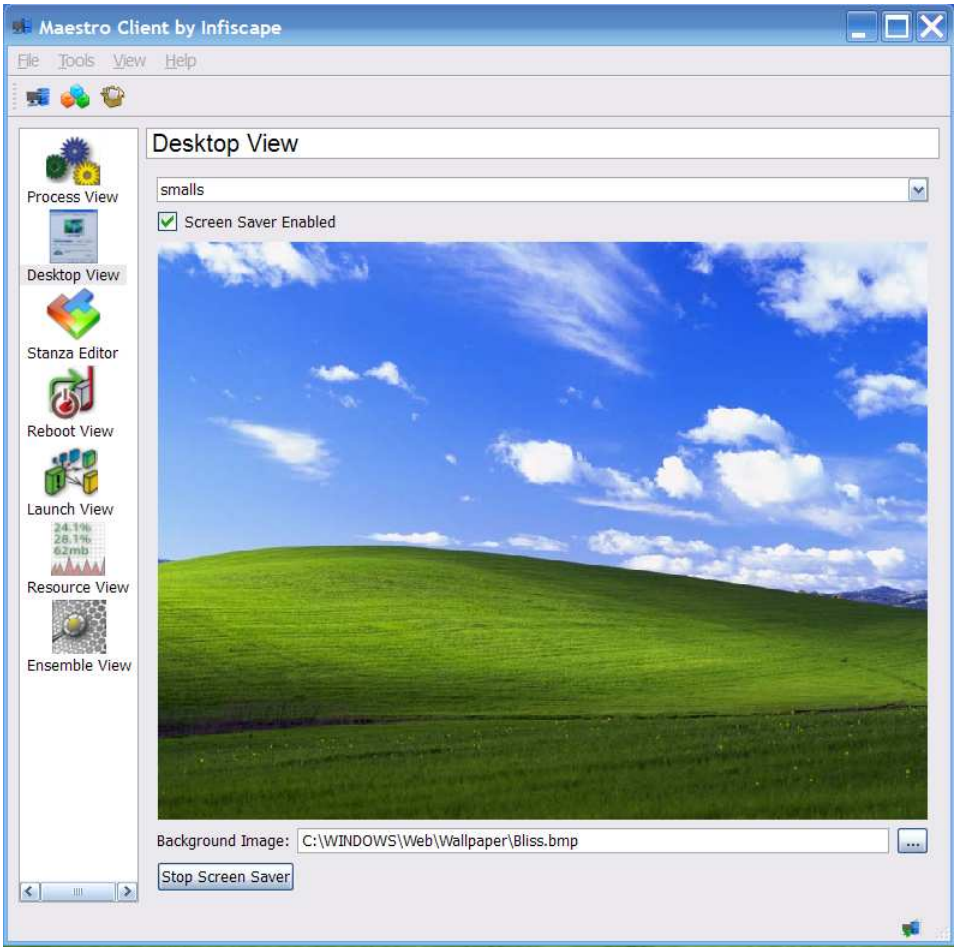

Figure 6.8 Desktop View

Ensemble View

The Ensemble view presents the user information about the currently loaded ensemble. As you can see in Figure 6.9, there is a list of all configured cluster nodes. Using this view the user can add, remove, or re-order the nodes in the ensemble. Also the user can browse information about each node by selected the node from the list. It displays the computer host name, IP address, up time, and other low-level details depending on the operating system being run on the node.

Stanza Editor

Since stanzas describe a hierarchy of options and environment variables, the idea behind this editor is to represent that hierarchy in an understandable yet powerful manner, Figure 6.10. A simple graph

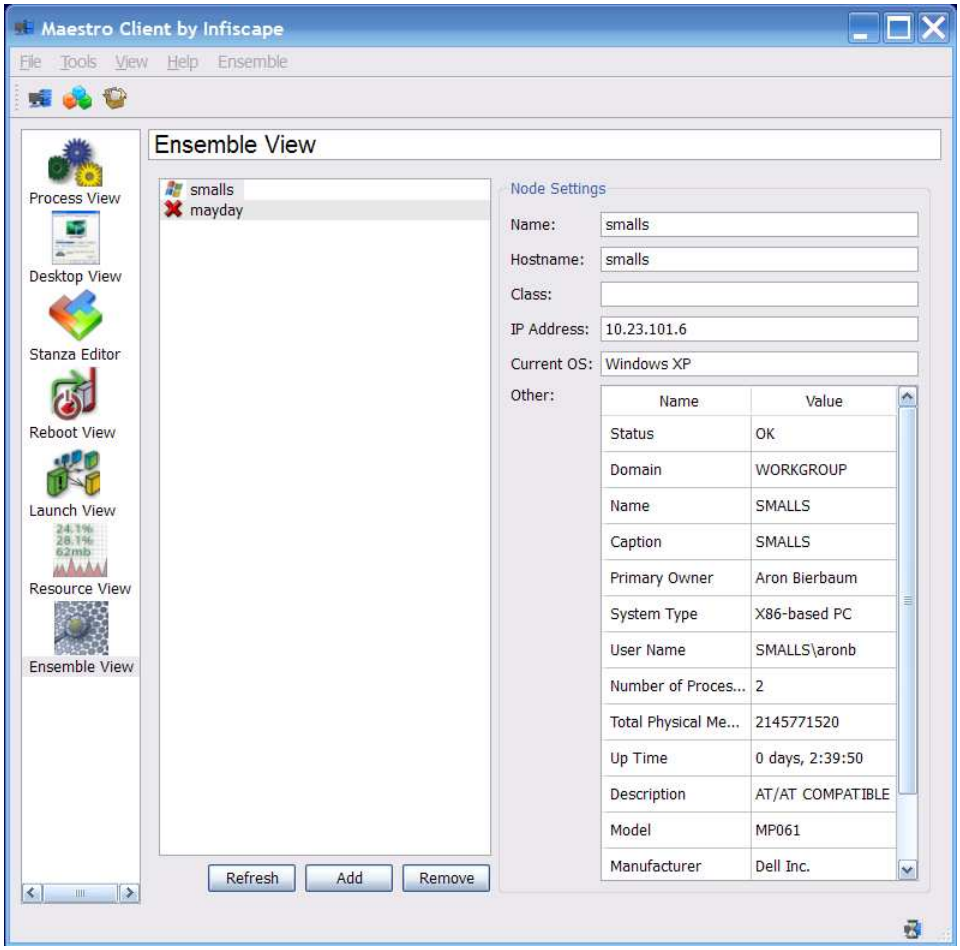

Figure 6.9 Ensemble View

structure is used to achieve this. The parent/child relationships are represented by arrows between nodes of the graph. Each node of the graph represents a stanza option, and the attributes of the option are editable in the dockable panel at the bottom of the stanza editor. This panel is dockable so that it can be detached from the Maestro GUI and resized independently to make editing easier. To add any one of these to the stanza being edited, click the desired tool bar item and drag it to the stanza editor canvas. At that point, connections can be made between the new structural item and existing items.

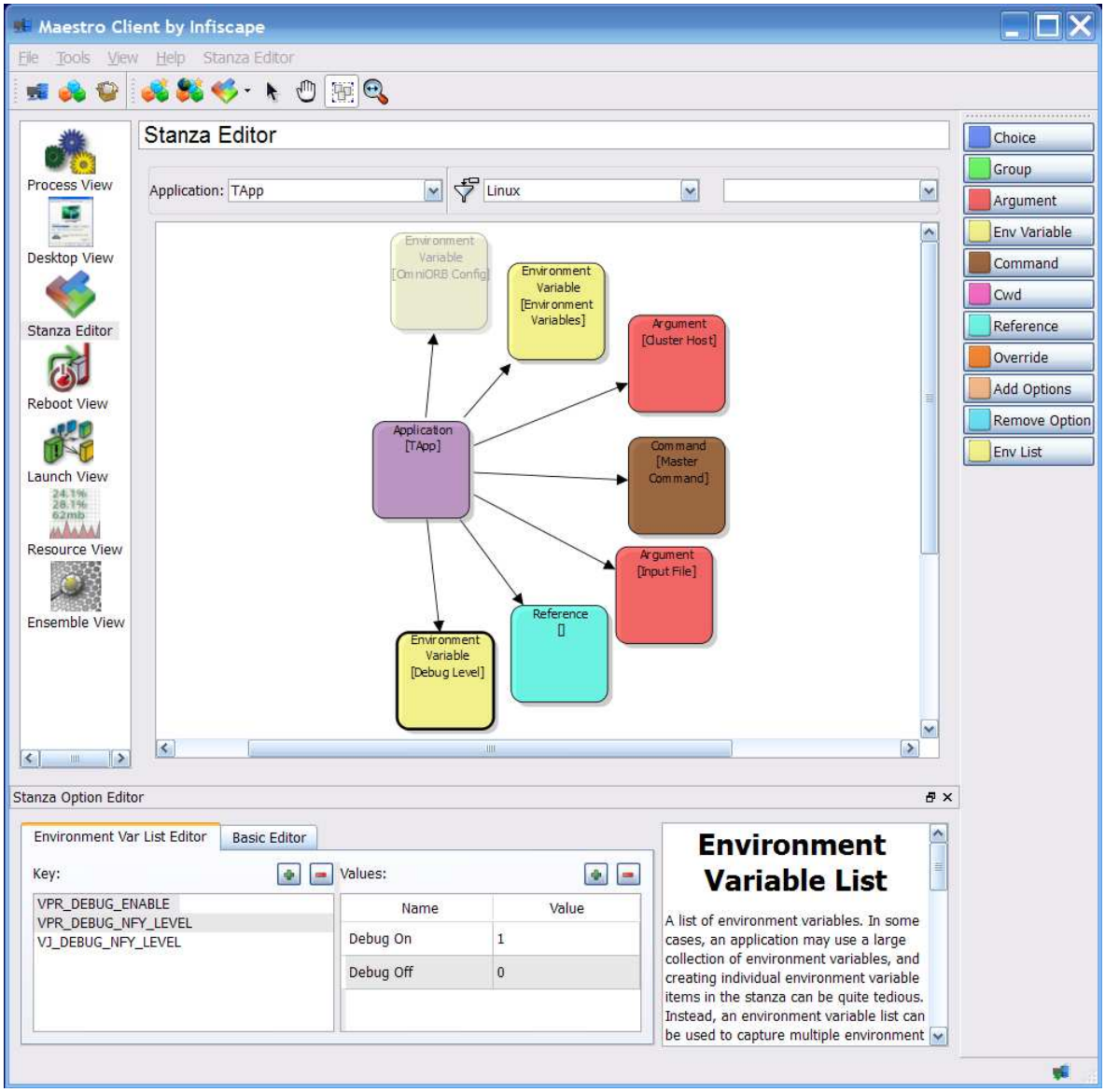

Figure 6.10 Stanza Editor

7. DISCUSSION

A few aspects of Maestro proved to be much more useful and unique than we had originally thought. There were also a few aspects of the research that proved to be very challenging and in some cases nearly impossible. This chapter discusses this aspects of Maestro in more detail.

Event System

At the beginning of this research designing a flexible and powerful event system was not a main design goal. After many iterations of development the current event system has proven to be very powerful and unique. The current design allows for a not only a client server architecture, but also a simple peer-to-peer network. Through a single method a client can send a dynamic event to any node in the cluster.

Stanza Reference Option

We also found that our current implementation of option references proved to be far more powerful than we originally thought. The current system can not only allow the stanza author to bring in stanza options from a common global set, but also modify these options as they are imported. When deploying a Windows® we also extended this to support command line arguments specified to the Maestro GUI client. This particular application took one command line argument that specified an input file. By passing override options to the Maestro client we could make file associations that would allow a user to right click on an input file and select to launch the application on the cluster with the given input file as a command line argument. This use case provides evidence that Maestro makes running immersive visualization applications on a cluster simple and user friendly.

Automated Logon

While developing Maestro we realized very quickly that typing a user name and password every time Maestro starts is counter productive. We tried addressing this issue on Windows® by using the Security Support Provider Interface (SSPI) to forward credentials between machines. Using SSPI we were able to add support for Active Directory and NTLM authentication methods. Although very useful, these two authentication methods come with severe limitations.

The use of NTLM only supports single-hop credentials forwarding, meaning that a user can authenticate with the Maestro service on a remote node, but cannot access network resources from that remote node. Also, delegation of credentials via Active Directory authentication requires that each node of the cluster be trusted for delegation. These security policies must be set up by the domain administrator on the domain controller.

This area of Maestro could still use a lot of research and development. We have still have a lot to learn about Windows® credential forwarding. We would also like to add support for credential forwarding on other platforms using a public key infrastructure (PKI) architecture similar to SSH. We will discuss this in detail in Chapter 9.

Logon Desktop

When executing a graphical application on a remote node the user must have the correct credentials to open a window on the current desktop. This has proven to be difficult on all platforms because the maestrod daemon must be executed with administrator rights and correctly keep track of system resources. It becomes even more difficult when we add the requirement that we want to execute an application on top of the user logon screen. On Windows® we were not able to find a method to gain the correct credentials to open a graphical window over the user logon screen. This means that each Windows® cluster node needs to have a user logged on before Maestro can execute applications. In order to work around this limitation the user can create a guest account with minimal security rights and set up Windows® to automatically logon this user each time it is started. Once again this area of Maestro requires more investigation in the future.

8. CONCLUSIONS

Immersive visualization clusters have gained popularity over the past few years. Before the introduction of Maestro, these clusters lacked an appropriate method to launch immersive applications. While there are existing remote execution tools, none address the specific needs of an immersive visualization cluster. This chapter will discuss how Maestro has addresses each requirement presented in Chapter 3.

Maestro was designed from the ground up with security in mind. The user is not allowed to connect to a maestrod daemon without first authenticating their identity. This authentication process attempts to authenticate the user with a set of authentication plug-ins. Basic user name and password authentication is supported on all platforms. Maestro also attempts to use credential forwarding to allow a user to use their existing credentials on the client machine when connecting to a cluster node. Maestro's plug-in architecture has proven to be more powerful and flexible than existing tools. Also since this architecture is extendable there are countless additions that can still be made. Some of these potential extensions will be discussed in Chapter 9.

Most existing remote execution tools are tied to a specific operating system. From the beginning of this research Maestro was designed to be completely cross platform. Currently Maestro has been tested and known to work on Microsoft Windows®, Linux, Mac OS X, and FreeBSD. Although it has only been tested on these four platforms, it should also work on any UNIX based operating system.

Immersive applications by definition always render graphics to a window on each cluster node. In order to do this the executed application requires special permissions to open an interactive window on the current desktop. Maestro transparently addresses this requirement on all supported operating systems by granting each remote process the correct access rights. There is currently no other remote execution tool that handles this requirement without user intervention.

Each application that we run on a cluster will interact differently with its execution environment. A useful remote execution tool should expect this and allow the user to have complete control over the execution environment. Maestro accomplishes this by allowing the user to specify everything including the command, current working directory, environment variables, and command line arguments. This is far better than existing tools that only let the user specify a command to execute.

The process of launching an immersive application is rarely a static operation. Instead the user launching the application usually needs to specify a set of command line flags that specify various input files and options. While launching an immersive cluster application the user should be presented with these options rather than having hard coded defaults. Maestro's solution to this problem is very unique. It allows the application developer to describe the application launching environment and all options that the user can specify while launching the application. Maestro then builds a dynamic user interface that presents the user with these options before launching the application.

Since an immersive visualization application is always displaying graphics it is very important that screen savers are not activated while running the application. Maestro addresses this requirement by giving the user a unified view of screen saver and power management settings on all nodes in the cluster.

The research presented in this paper explains the unique needs of an immersive application, describes how existing tools do not address these needs, and presents Maestro, our solution to this problem. We believe that Maestro successfully addresses each of the goals listed in Chapter 3. Maestro has evolved over time into a tool that is deployed on numerous large clusters and has proven to be indispensable.

9. FUTURE WORK

Immersive visualization is an evolving field, and similarly, Maestro is an evolving project. Based on user feedback and project goals, there are a few improvements that have been identified. This chapter will address each one of these individually.

The user authentication support in Maestro was designed to be extended through authentication plug-ins. Currently there is support for user name and password authentication on all platforms. There is also limited support for credential forwarding on Windows® using SSPI. We would like to create an additional plug-in that uses a PKI infrastructure similar to SSH. This would allow each cluster node to have a list of trusted public keys. The Maestro client could then use the user's private key to prove their identity. This could build on top of existing tools like ssh-agent to gain access to a user's private key list. Using this method we could eliminate the need for a user to type their user name and password each time they start the Maestro client GUI.

Administration of a large cluster is a difficult and tedious process. Maestro could be extended to allow more complete remote administration. This could include creating an interface to existing well tested tools. For example, Maestro could provide a method of starting a Virtual Network Computing (VNC) [18] client connection to any node in the cluster. This would allow the user to control the entire remote desktop. Maestro could also interface with existing cluster monitoring tools such as Ganglia [17] to provide the user with a better with of cluster resources.

This chapter has only discussed a few potential extensions. Since Maestro was developed to be highly extendable using a plug-in architecture, there are countless plug-ins that could be added in the future. It is our goal for Maestro to build a user community that continues actively developing these plug-ins in an open and cooperative environment.

BIBLIOGRAPHY

[1] "Pyqt website." [Online]. Available: http://www.riverbankcomputing.co.uk/pyqt/

[2] "Pyro: Python remote objects website." [Online]. Available: http://pyro.sourceforge.net/

[3] "Qt website." [Online]. Available: http://trolltech.com/products/qt

[4] "Xml path language (xpath)," November 1999. [Online]. Available: http://www.w3.org/TR/xpath

[5] J. Allard, V. Gouranton, E. Melin, and B. Raffin, "Parallelizing Pre-rendering Computations on a Net Juggler PC Cluster," in *IPT (Intl. Workshop on Immersive Projection) 2002 Proceedings*, Orlando, Florida, United States, March 2002.

[6] J. Dickerson, Y. Yang, K. Blom, A. Reinot, J.Lie, C. Cruz-Neira, and E. Wurtele, "Using virtual reality to understand complex metabolic networks," in *Proceedings of the Atlantic Symposium on Computational Biology and Genomic Information Systems and Technology*, September 2003, pp. 950–953.

[7] Z. Fan, M. Oliveira, C. Ma, and A. Kaufman, "A sketch-based interface for collaborative design sketch-based interfaces and modeling," in *Proceedings Eurographics Symposium 2004*, vol. VI, August 30-31 2004, pp. 1–5.

[8] A. Fettig, *Twisted Network Programming Essentials.* O'Reilly Media, 2005.

[9] W. Gallus, C. Cervato, C. Cruz-Neira, G. Faidley, and R. Heer, "A virtual tornadic thunderstorm enabling students to construct knowledge about storm dynamics through data collection and analysis," in *13th Symposium on Education*, January 11-15 2004.

[10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, ser. Addison-Wesley Professional Computing Series. New York, NY: Addison-Wesley Publishing Company, 1995.

[11] A. Hopkins, "Remote++: A script for automatic remote distribution of programs on windows computers," in *Proceedings of the ACM Southeast Regional Conference*, 2003.

[12] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, and J. T. Klosowski, "Chromium: A stream processing framework for interactive graphics on clusters," in *ACM SIGGRAPH 2002 Sketches and Applications*. Texas, United States: ACM Press, July 2002.

[13] J. Jacobson and M. Lewis, "Game engine virtual reality with caveut," *IEEE Computer*, vol. 38, no. 4, pp. 79–82, 2005.

[14] B. Karthikeyan, K. M. Bryden, and D. A. Ashlock, "Visualizing information flow in evolving graph-based population," in *Proceedings of International Conference in Smart Engineering Design (ANNIE-2003)*, St. Louis, United States, November 2003.

[15] C. Kim and J. Vance, "Collision detection and part interaction modeling to facilitate immersive virtual assembly methods," *ASME Journal of Computing and Information Sciences in Engineering*, vol. 4, no. 1, pp. 83–90, June 2004.

[16] C. Martin, "Parallel launcher for cluster of pc," 2001.

[17] M. L. Massie, B. N. Chun, and D. E. Culler, "The ganglia distributed monitoring system: Design, implementation and experience."

[18] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper, "Virtual network computing," *IEEE Internet Computing*, vol. 2, no. 1, pp. 33–38, 1998.

[19] V. Samar and R. Schemers, "Unified login pluggable authentication modules (pam)," RFC 86 (Informational), Open Software Foundation, October 1995. [Online]. Available: http://www.kernel.org/pub/linux/libs/pam/pre/doc/rfc86.0.txt.gz

[20] O. G. Staadt, J. Walker, C. Nuber, and B. Hamann, "A Survey and Performance Analysis of Software Platforms for Interactive Cluster-Based Multi-Screen Rendering," in *Proceedings of the Workshop on Virtual Environments 2003*. Zurich, Switzerland: ACM Press, 2003, pp. 261–270.

[21] R. Stuart, *The Design of Virtual Environments*. McGraw-Hill, 1996.

[22] C. Szyperski, D. Gruntz, and S. Murer, *Component Software: Beyond Object Oriented Programming*, 2nd ed., ser. Component Software Series. New York, NY: Addison-Wesley Publishing Company, 2002.

[23] T. Wasfy and A. Noor, "Visualization of CFD results in Immersive Virtual Environments," *Advances in Engineering Software*, vol. 32, pp. 717–730, 2001.