
Xgrid and Cross Platform Grid Computing

Abstract:

This paper will provide an overview of grid computing including its history, current development, terminology, and use. The scope of this project is geared towards the implementation of Apple's *Xgrid* technology in a university environment, or in a home environment for power users such as architects. Later in this paper we will implement an Xgrid, and will demonstrate its cross platform abilities, while exploring its revolutionary take on grid computing. Benchmarks of this grid will be done using light ray tracing software called *Pov-Ray* in combination with Architectural CAD software called *Sketchup*, and will produce a series of complex graphical scenes. Each computer on the grid will be responsible for rendering at least one scene, and rendering times will be recorded.

Since the *Xgrid* technology is new to the market there are still many flaws and bugs which will make this goal very challenging. However, this project has made the Linux Xgrid agent functional, which until now has remained dysfunctional, and through a combination of *Apache* and *PFTP* has been made to work almost flawlessly with Xgrid. An automated installation script has also been created that will download and install all the necessary dependency requirements for the Xgrid agent. This script will allow administrators to quickly and efficiently distribute the Linux Xgrid agent between PC's.

Through the examination of this technology we should be able to draw some conclusions as to the state of cross platform grid computing using Xgrid, and provide some recommendations of how UCFV could use this technology.

Stuart Bowness

440 Project Report • University College of the Fraser Valley • April, 2005

Table of Contents



Introduction	7
Literature Review	9
Grid Computing Terminology and Technology	9
History	13
Current Work in Grid Computing	15
Project Justification:	25
Brief Descriptions of Project Products:	26
Summary of Project Deliverables:	26
Determination of Project Success:	27
Definition of Hardware Resources:	27
Definition of Software Requirements:	28
Xgrid Overview	29
Introduction to Xgrid	29
Xgrid in Action	31
Under the Hood of Xgrid	33
Xgrid Security	37
How Job Submission Works	38
Job Types	40
The Xgrid User Interface	44

Xgrid Implementation	45
Implementing Xgrid on OS X	45
Installing a Cross Platform Xgrid	49
Hardware Setup	51
SSH and Remote Installation	54
Cross Platform Problems and Solutions	55
Apache 2.0 and PFTP	57
Benchmarking and Testing	59
Pov-Ray	59
Sketchup	61
Results	65
CGI, Blosxom, and RSS for Displaying Xgrid Output	73
Conclusions	75
The Future of Xgrid	75
Recommendations	76
Final Conclusion	78
Glossary	81
Appendixes	83
Appendix 1 - Darwinports and Pov-Ray	83
<i>Introduction</i>	83
<i>Installing XCode Tools</i>	83
<i>Installing DarwinPorts on OS X</i>	85
<i>Finally Installing Pov-Ray on OS X</i>	87
<i>Installing DarwinPorts and Pov-Ray on Linux</i>	87
Appendix 2 - Xgrid User Interface	93

<i>Controller and Client Setup</i>	93
<i>The Xgrid.app</i>	98
<i>Conclusion</i>	104
Appendix 3 - Xgrid Installation Files	105
Appendix 4 - Choosing the Linux Distribution	106
<i>Choosing our Linux Flavor</i>	106
<i>Ubuntu Linux</i>	107
<i>Fedora Core 3</i>	116
<i>Linux Conclusions</i>	123
Appendix 5 - Installing the Xgrid Linux Agent	125
Appendix 6 - Xgrid Linux Agent Segfault Issues	129
Appendix 7 - Editing the xgrid.config.xml File	133
Appendix 8 - Automating the Installation of the Agent, DarwinPorts, and Pov-Ray	134
Appendix 9 - Custom Pov-Ray Libraries	146
Appendix 10 - The Rendering Script	147
Appendix 11 - Custom Apache2.conf	150
Appendix 12 - Gantt Chart	151
End Notes	153
Bibliography	156



List of Figures

Figure 1 - Industries where HPC is being used

Figure 2 - Xerox Palo Alto Research Center

Figure 3 - The SETI@Home project

Figure 4 - A sample Condor / Globus network utilizing Intel / Linux Hardware

Figure 5 - Virginia Tech's supercomputer was assembled in 3 weeks by many volunteers

Figure 6 - The final product - a supercomputer

Figure 7 - The Cooling system used

Figure 8 - The rack-mount system

Figure 9 - World's 3 fastest supercomputers 2003

Figure 10 - Cost in millions for the 3 fastest supercomputers in the world

Figure 11 - A screenshot of the Xgrid Blast application with one computer at 1.5GHz connected

Figure 12 - . The locations of major contributors to Xgrid at Stanford

Figure 13 - The roles and interactions of computers within an Xgrid

Figure 14 - The submission of cal (calendar) jobs to Xgrid

Figure 15 - Sample output of the first cal job

Figure 16 - The submission of a custom Pov-Ray job

Figure 17 - The network structure

Figure 18 - The submission of a Pov-Ray job taking a file list as an argument

Figure 19 - The first Pov-Ray rendered scene

Figure 20 - The first 3D architectural drawing created using Sketchup for OS X

Figure 21 - The room.pov render

Figure 22 - The author hard at work on a 3D architectural drawing, in the background is a Dual Processor PowerMac which was used in exploding the final drawings

Figure 23 - A wireframe outline of the final 3D architectural drawing

Figure 24 - One of the many final Pov-Ray renders from the architectural drawing

Figure 25 - The time to render 10 identical scenes on a single 1.5GHz machine

Figure 26 - The scene used for the 10 identical renders

Figure 27 - Time taken to render 10 identical scenes using 6 computers

Figure 28 - Time taken to render 10 identical scenes using 3 fastest computers

Figure 29 - A screenshot of the 1.5GHz machine finishing the last task in the Pov-Ray render

Figure 30 - The final result depicting the times taken to render 10 identical scenes

Figure 31 - The time taken to render 7 scenes on a 1.5GHz machine

Figure 32 - The time taken to render 7 scenes on 4 computers

Figure 33 - A comparison of standalone vs. Xgrid benchmark times on a set of 7 scenes

Figure 34 - The blog used in displaying the Pov-Ray output files

Figure 35 - GridObjects is a web-based Xgrid management system

INTRODUCTION

Computing technology has grown exponentially in recent years, particularly in the fields of microprocessors, mass storage, and networking. This growth in power has buried the 21st century deep into the world of technology, and has launched computing into homes and businesses worldwide. With society's new-found dependence on computing, our reliance on the speed of processing has also increased as we have begun to do far more with our computers than we would have ever done in the 1980's and 1990's. Processing dependent computing has flourished under the wealth of software applications that have been written to take advantage of these new modern processors, and as a society, we have eagerly integrated these advances into our daily lives. Businesses perhaps now more than ever rely on computers to process transactions, data, and a plethora of other information, and home users have begun using their computers for much more than typing and games. Universities have also capitalized on this growth in computational power and are just beginning to realize the potential in harnessing massive amounts of computational power for aiding research.

Grid computing is the concept behind harnessing the resources of multiple computers over a network, and using that power to solve compute-intensive problems. Grid computing is currently a very hot topic amongst corporate and academic circles at the moment, and is most prominent in fields where massive amounts of processing are done on a daily basis. Some of these these fields include biomedical research, mathematical and statistical research, and graphic rendering.

One of the problems presented to individuals who wish to implement grid computing is that computers in most Universities or business offices do not run the same operating system or have the same hardware. In order to demonstrate how grid computing can be used in everyday activities on a variety of operating systems, this paper will examine how such technology can be used to benefit modern architects. A popular 3d CAD program called *Sketchup* will be used in conjunction with light ray rendering software to produce a series of environments which the grid will compute. Each of these environments will be intensive enough to take a considerable length of time for a single computer to process,

hence the advantage of using grid technology. A cluster of Apple and Linux based machines will be assembled, and the final product will be a series of high quality rendered architectural scenes. The purpose behind this research paper is to provide a detailed overview of grid computing, its benefits, and how it is being used today and will show how this technology can be used in real life cross platform situations. Through research and the demonstration of these grid-computing technologies it will be possible to compare these findings with how the technology could be used in a University setting for research purposes.

This paper is highly recommended reading for anyone interested in the growing field of grid computing, and is wondering how they could tie such technology into aspects of their work or research. Anyone with tasks that take significant amounts of processing time and have a method for breaking these tasks into smaller tasks will definitely be able to take advantage of this technology, and will find this document an invaluable asset to their project goals. Architects using *Sketchup* in combination with *SU2Pov* and *Pov-Ray* will also find this paper useful as it details considerable amounts of information on how to distribute 3d scenes. University researchers are also one of the primary groups which this paper is geared towards. Researchers will find that this document explains how grid computing can benefit their research work and will develop a better understanding of what this technology is capable of, and the direction in which it is going.

LITERATURE REVIEW

Grid Computing Terminology and Technology

The term “grid computing” seems to be something of a cliché phrase that you hear discussed everywhere from the world of academia to large corporations, film studios and graphic workshops. It is a term that has been so loosely coined that it can mean any number of things. Below are some of the ways in which grid computing has been defined:

- Transparent, secure, and coordinated resource sharing and collaboration across sites.
- The ability to aggregate large amounts of computing resources which are geographically dispersed to tackle large problems and workloads as if all the servers and resources are located in a single site.
- A hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to computational resources
- The flexible, secure, coordinated resource sharing among dynamic collections of individuals, institutions, and resources.

For the purposes of this paper, grid computing is loosely defined as: The ability to form a heterogeneous network of computers capable of distributing a maximum of available resources to tackle large complex problems and workloads.

The term heterogeneous in this context is used to outline that networks do not need to consist of single homogeneous operating systems, and can consist of computers running either Windows, Linux, BSD, Unix, or Mac OS X. This is imperative to the definition of grid computing because, in a real life environment, it is not frequent that every system available on the network is running the same operating system.

There are four aspects of grid computing that will appear throughout the course of this paper: high performance computing, high throughput computing, cluster computing, and internet computing.

High performance computing (HPC) is generally referred to as an aspect of super computing where grids of processors are utilized to build a system capable of enormous processing power. Applications of HPC can be found in weather modeling, drug development, and financial forecasting. HPC is most effective in computing either “embarrassingly parallel” or “deeply parallel” problems. Embarrassingly parallel problems, like the aforementioned SETI@Home project, are where a massive amount of data can be broken up in any number of ways, and to which analysis is applied. Deeply parallel problems are those where most of the calculations in a problem are integrated with another part of the process either as an input or a potential side effect. Deeply parallel problems are often referred to as non-linear and the problems often take on a hierarchical structure. HPC is essential for both embarrassingly parallel and deeply parallel tasks as the processing of data quickly is of utmost importance. High performance computing is often measured in terms of floating-point operations¹ per second, and is used in an environment where every second counts. Some of the industries in which high performance computing is currently being used is depicted [Fig. 1] below:

- Digital Content Creation
 - Semiconductor
 - Automotive
 - Others
- Database
 - Geophysics
 - Software
- Telecomm
 - Weather and Climate Research
 - Information Processing Service

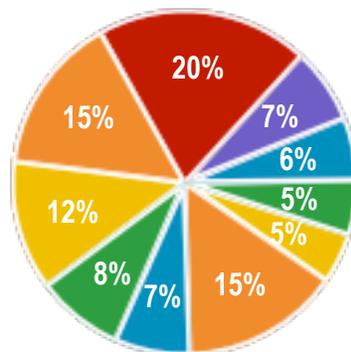


Fig. 1 - Industries where HPC is being used.

¹ See Glossary [1]

High throughput computing (HTC) is very similar to high performance computing but its goals are slightly different. Instead of processing being intensely done over very short intense periods of time, processing is done over very long periods of time and as such, HTC is measured in terms of floating-point operations per year. This type of computing is particularly of use to scientists, engineers and researchers who are looking at what they can accomplish over N months/years instead of over N days. Below is an example of HTC in action as discussed by Miron Livny during an interview with HPCwire [1].

“If you must manufacture a chip, you have a window of about three months to run as many simulations as you can before bringing the product to market. Essentially, this is an HTC problem. If you have a high-energy physicist who is reconstructing events and enriching them with Monte Carlo data, the project has a year or two to complete, but the more computational resources that can be brought to bear in that time, the greater will be the statistical significance attained. This too is an application basically limited by throughput rather than response time.”

The Condor Project, which will be mentioned more in depth later, is a software application which provides this kind of HTC power.

Cluster computing came about as a response to the ridiculous price of supercomputing. Clusters run on freely available software, and usually on free operating systems such as FreeBSD, or Linux. There is some overlap between cluster computing and HPC as they both share similar outcomes in their goals to create a supercomputer that can be used for massively parallel applications, however HPC is geared around the creation of a highly customized supercomputer, whereas cluster computing is geared towards the creation of a supercomputer through the use of publicly available hardware components. Clusters are made up of individual computers, each of which is called a node. Technically each computer in an office could act as a node, but in a true clustering environment, nodes would consist of dedicated machines which are interconnected through a fiber optic, or gigabit ethernet cable. The explosion in the speed of microprocessors over the last few years has really made cluster computing a reality. This year, at Doshisha University in Japan they have just finished building a Gigabit ethernet cluster of 512 1.8Ghz AMD Opteron ma-

chines with a computational power equivalent to 1169 Giga-Flops [2]. This has given them a 298th place listing on the Top 500 listing of supercomputers in the world, and although it is certainly not a first place finish, it is a respectable result for a supercomputer that is comprised of off the shelf components.

With the sudden growth in both the size and speed of the internet, the availability of computing power available to researchers has risen exponentially. The type of computing power discussed here is not the power of large supercomputers, or even interwoven networks of high performance clusters, rather, it is the power of the home computer, and is known as the field of internet computing. Internet computing occurs where jobs are parallel enough that they can be broken down into individual tasks, but instead of being distributed between hundreds of processors on a giant mainframe or cluster, they are distributed to clients all over the world via the internet. The basic theory behind internet computing is that it empowers individuals who own computers and are connected to the internet to donate their computers to any compute-intensive project. Users can download a lightweight client which will periodically connect to a centralized server to download data and begin new tasks, or to upload data, and finish old tasks. The central server in this manner is not only responsible for providing clients with uploads and downloads, but it also is responsible for gathering and compiling the data returned from these clients into a result. Internet computing has become a very powerful way to acquire a vast amount of computational power, but it does have certain limitations and constraints. The market for freely available computational power is potentially massive, however, a large majority of home users do not choose to allow their computers to work on such projects nor are they aware that such programs exist, and many projects which have poorly defined goals, or require complex setup and installation instructions will find that the amount of users they can gather will be relatively small. If internet computing was to be used for small to medium scale research projects, the results would not be nearly as efficient as they might be within the confines of a dedicated grid. However, with a large scale research project and solid project goals, this form of computing definitely has gross computational advantages over the long term especially since it is essentially free.

Next this paper will discuss the history and evolution of the grid computing field, and what other organizations around the world are doing to promote and integrate its use.

History

Grid computing started in the 1970's when the cost of processing was so high that every effort was made to save CPU cycles. In those days machines came in the form of big mainframes and cost hundreds of thousands of dollars. As a result, methodologies were created for maximizing the use of spare cycles, and the concept of a Grid was given birth [3]. Some of the earliest experiments done with grid computing included programs called *Creeper* and *Reaper*. These programs ran on the ARPA-net which was the forerunner to the world wide web we use today. These projects were “worm” programs in which the creeper would replicate itself from machine to machine, and the reaper would act as a destructor and remove the creeper from the machines [4]. In 1973, the Xerox Palo Alto Research Center [Fig. 2] made some major breakthroughs that heavily influenced the development of distributed computing. Some of these developments included the development of the individual computer workstation, the installation of the first ethernet network



Fig. 2 - Xerox Palo Alto Research Center

which allowed autonomous machines to communicate with each other, and lastly the PARC developed the first distributed file server so that users could share common files. At the PARC another “worm” was created by scientists John F. Schoch and Jon A. Hupp. The purpose of this worm was to move from machine to machine using idle resources for beneficial purposes. In another similar effort Richard Crandall (Currently an

Apple employee and a professor at Reed College USA) networked NeXT computers which performed computations and combined the efforts of multiple machines [5]. All of

these developments and inventions helped pave the road for the future of distributed computing.

With the invention of the modern internet in 1991, distributed computing has elevated to much more than computing over local area networks, and grids now operate across multiple platforms, and across geographically dispersed areas. Two distributed computing efforts in particular have elevated themselves past levels anyone had anticipated and now have millions of contributing computers. The first of these projects is called *distributed.net* and, when it first started it had thousands of contributors who helped crack encryption

algorithms. The second project is the much more infamous SETI@Home [Fig. 3] project which searches for extraterrestrial intelligence through the analysis of radio signal fluctuations. SETI@Home is the largest, most successful distributed computing project to date and has well over 2 million people running its software. These 2 million people field 3 mil-

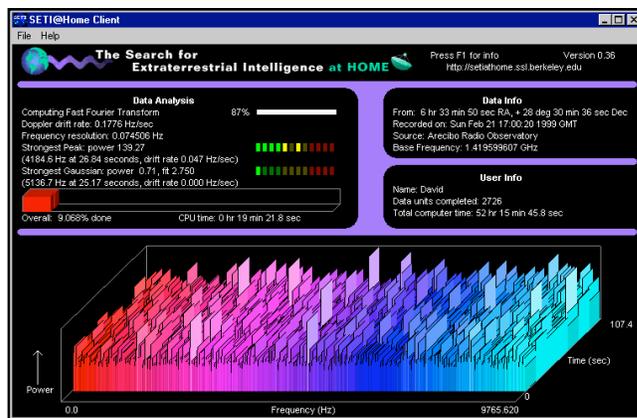


Fig. 3 - The SETI@Home project

lion computers that average about 14 teraflops (14 trillion floating point operations per second) and has completed around 500,000 years processing time in the past year and a half [6]. The advantages are purely cost-effective, as the cost of running a supercomputer capable of the same output would cost millions of dollars. Since the creation of SETI@Home and the realization of its rapid success, other projects have similarly appeared in order to capitalize on the new-found apparent wealth of processing power suddenly available. Folding@Home is another program that was created with the hopes of attracting enough processing power to assist research in understanding protein folding, protein aggregation, and various other related diseases and is sponsored by Stanford University. The Find - A - Drug project has also been recently launched and seeks computing power to aide their research in discovering new drugs that can be used to cure malaria, cancer, HIV, and to counteract bioterrorism. All of these emerging grids rely on their own

their own custom software to permit individuals all over the world to join to their network, and assist them in the processing of data and interpretation of results.

Grid technology has come a long way since its inception, but still has yet to bring the benefits of its power to the average artist, scientist, or researcher in a manner that is easy to setup, manage, and maintain. Certainly there are many distributed computing projects around which allow users from all over the world to donate their computers and resources to, but there are far fewer projects which allow individuals or small to medium sized organizations to easily create their own distributed systems. Most grid systems are difficult to setup, require complex command-line knowledge, and often require strong scripting knowledge in order to submit “jobs” to the network. Most researchers, and certainly most graphic artists, do not necessarily have the skills or the background in order to utilize these tools, however things are beginning to change and several major endeavors are underway to bring these tools to the public.

Current Work in Grid Computing

One of the largest grid endeavors that has the aspiration of bringing grid computing to the masses is Sun Microsystem’s *Sun Grid Utility*. At the time which this report was written, this utility has not yet been released, but is expected to reach consumers within the next six months. According to Sun:

“The Sun Grid compute utility provides customers with fully virtualized cpu memory and high-performance storage connected through a secure networked Grid, at a price of \$1/cpu-hr. Customers can use it for jobs such as Monte Carlo simulations, protein modeling, reservoir simulations, mechanical CAD simulations and similar non-transactional workloads. The Sun Grid compute utility will deliver a standard computing Grid, powered by AMD Opteron processor-based systems, Solaris 10 OS and N1 Grid Engine, to help provide customers optimal performance, functionality and security” [7].

Or in other words, Sun provides the hardware, a pre-configured grid, and pay-as-you-go pricing. This concept is fairly revolutionary, and comes at a cost which both researchers,

scientists, and graphic artists, under heavy time constraints, will all applaud. Not only will this save heavily on infrastructure costs, but it will also allow scientists to use a grid on an as needed basis. Better than this yet is Sun's *Grid Storage Utility* which supports customer's grid jobs at the price of \$1 per Gigabyte/Month. This allows their customers to store their jobs, upload and download data, and not have to worry about hardware issues, configuration, or software setup. The Sun *Grid Storage Utility* is also secure, and can be rented for any period of time.

IBM has also begun development to embrace the power of grid computing and is currently working on a special product line called the *IBM @ Server* [8] which will form a solid platform for grid development and grid management. IBM has also created a DB2 line of products which sport tools to enable the efficient setup of grids, and the creation of complex data infrastructures. This, in combination with the *@ Server* line, will create a whole range of solutions for businesses looking for more computational power to analyze statistical data, project forecasts, or analyze markets. Outside of these solutions IBM also plans in the future to grid-enable many of its systems and software. Grid-enabled applications will automatically submit work to systems connected to the grid to compute whenever the nature of the task is parallel enough to be worthwhile distributing.

Microsoft is also looking towards the future of grid technology team, and has one of their infamous skunk-works teams developing a product called *Bigtop* which has been kept very secretive over the last year. According to an unidentified source at Microsoft-Watch.com "Microsoft is working on a skunk-works project that is designed to allow developers to create a set of loosely coupled, distributed operating-systems components in a relatively rapid way" [9]. *Bigtop* will have methods for automating the creation of highly parallel processes, and will introduce its own customized programming language called "High Wire" for this specific purpose. A concept known as "Bigparts" will permit the integration of a PC device to be used as a special purpose server in this environment, and will feature web based management as opposed to local management so these servers can be managed remotely. It is likely that the public can expect to see a preview release of the *Bigtop* around the first quarter of 2006, and a stable release closer to the end of the decade.

Also, away from the world of the typical “big guns” of the computing industry there are some exciting grid projects that much smaller companies are bringing to the table. Globus Alliance is perhaps one of the most active and perhaps largest ongoing grid researchers out there and is funded by some of the big name companies previously mentioned, including, IBM, Microsoft, and Cisco Systems. The Globus project itself, however, is largely maintained, and developed by a group of academic individuals from around the U.S. and Europe. These individuals include contributors from several universities and laboratories, and also the National Institute of Advanced Industrial Science Research Center in Japan. Interestingly the Max Planck Institute for Gravitational Physics (Albert Einstein Institute) is also involved, though finding any information on what exactly they do for Globus was not possible. Globus’ research focuses not only on the development of grid infrastructures, but also on the development and design aspect of creating applications that can utilize grid services. Globus Alliance provides a toolkit which can be used for building grids and includes software used in the security, information infrastructure, resource management, data communication, fault detection, and portability of grids. This toolkit can be used independently, or in conjunction with other applications to develop grid services. As every organization is different and has different network topologies and issues, the Globus toolkit is built to work seamlessly with existing infrastructure. Its core services, interfaces, and protocols allow users to access remote resources as if they were located on their own machine while also preserving control over who can use these resources and when they can access them. One of the best things about Globus is that it is a completely open source (Similar to the Linux operating system) and, as a result, does not require its’ users to wait for vendor provided upgrades or updates because the source is readily available. According to Globus, this “encourages broader, more rapid adoption and leads to greater technical innovation, as the open-source community provides continual enhancements to the product” [10]. Globus Alliance is catering to large scale grid deployments, and offers all the control that serious grid geeks demand.

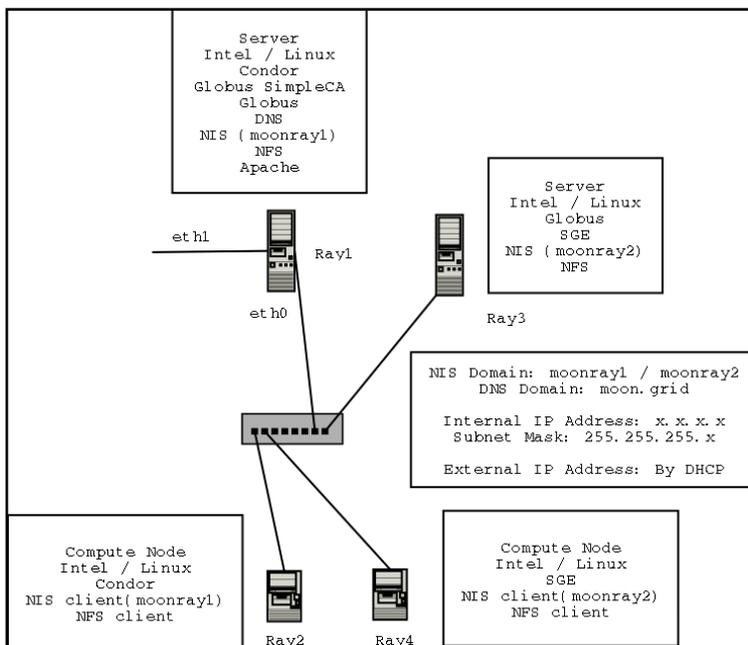


Fig. 4 - A sample Condor /Globus network utilizing Intel/ Linux hardware

The Condor Project is very closely aligned with the Globus Alliance project but is distinct in more than a few ways. Condor is designed to be a specialized work management system for computationally intensive jobs. Similar to other grid software Condor provides a job queuing mechanism, scheduling, policy, priority schemes, resource monitoring, and resource management. It functions in

a manner which permits users to submit their parallel computing tasks, monitor them, and of course permits notification upon their completion. What Condor does differently from most other grids is that it does not require a computer to become a dedicated grid computer in order to participate in grid activity. A single machine can join the Condor grid only when it is idle, and will stop tasks once the user returns to the computer. This is a very powerful resource for universities like UCFV in particular as it harnesses the maximum computational power available to researchers, whilst not interrupting ongoing work. Condor is also fully capable of being integrated with resources managed by a Globus enabled grid [Fig. 4], and as such, has a lot of expandability. One of the great things about Condor is that it is at a stable 6.7 release, and is offered as a completely cross platform Grid. Condor will run on most Linux distributions, Mac OS X, and of course Microsoft Windows. Unlike the technologies that have been discussed earlier, with the exception of Globus, this technology is available now, and is being put to good use by both educational environments and businesses alike [11].

Apple has also recently launched itself into the research world on a number of fronts, and hopes to once again become a dominant force in education and research. Virginia Tech has become the fore-runner for the latest Apple technology as they were in the market for a high performance supercomputer at the same time that Apple was looking to demonstrate some of its engineering prowess. What Virginia Tech got, was not only a supercomputer that landed in the top 5 supercomputers in the world, but it also achieved the ranking of the fastest university supercomputer in the world. But what does supercomputing have to do with grid technology? In Virginia Tech's case, their supercomputer consists of 1100 Dual 2 Ghz Powermac G5's, so in the traditional sense of the word, it is not a single mainframe supercomputer, it is a series of networked desktop computers that combine to form a supercomputer, or perhaps a better word for it might be a super grid. Due to the excellent documentation available on how they setup this super grid, this paper will take a brief look at what Virginia Tech has done, how much it has cost, and how they are using this technology. Although UCFV may not have the funds to create a grid this powerful, it is an excellent model.

Virginia Tech first started the project to create a world class supercomputer in September 2003. Srinidhi Varadarajan, PhD, was appointed director of this project, and found quickly that the IBM's G5 which Apple uses in its PowerMacs was a perfect fit for the architecture and roles of their system. The G5 which Virginia Tech is currently using sports a 64 bit processor, 2 floating point units, supports up to 8 GB of memory, and speeds of 1 Ghz on the front - side bus for each processor which offers a staggering 16GBps (Giga-bytes per second) throughput on a dual 2Ghz PowerPC processor setup. Bandwidth is further optimized through utilizing a 400Mhz, 128 bit memory bus with a low latency hypertransport interface that connects the PCI-X controller, and the I/O subsystems to a system controller. This is one of the many reasons behind the blazing speeds that their supercomputer has achieved. For the first time, according to Varadarajan, "clusters finally have the capacity to go toe to toe against the fastest custom designed supercomputers"[12]. and he has been very impressed with the fact that there really is not much you can't do on his cluster that you can do with a supercomputer. His 1100 PowerMacs have provided 2200 CPU's which have delivered over 10 teraflops of processing power to re-

searchers at Virginia Tech. Virginia Tech is also currently in the middle of a major upgrade involving a move from Apple's Dual 2GHz PowerMacs to the Apple Dual 2.5GHz XServes. This they hope will provide them with something in the neighbourhood of 30 teraflops of computing power.



Fig. 5 - Virginia Tech's supercomputer was assembled in 3 weeks by many volunteers.

The machines and the PCI - X network cards arrived in early September, and within 3 weeks [Fig. 5] the entire cluster was assembled. Initial benchmarks proved very impressive and much praise was given to the PCI-X network cards. These PCI - X network cards plug into a PCI -X expansion slot which features a

133 Mhz bus and is much faster than the standard and sluggish 33Mhz PCI bus that is standard in average desktop machines. This makes the PCI - X network card ideal for high performance networking, and delivers very high network bandwidth with very low latency. These cards were then used in conjunction with fiber optic cable to interconnect all of the G5 nodes (individual PowerMac G5 computers) into a single super computer. By December they had finished benchmarking the system, and were ready to see some results.

“It was really nice to have the same platform running on my desktop that I can use to check email, and also at the same time connect to the super computer, which is one of the fastest supercomputers in the world.”

- Srinidhi Varadarajan, PhD - Director, Terascale Computing Facility, Virginia Tech



Fig. 6 - The final product - a supercomputer



Fig. 7 - The cooling system used

Originally Virginia Tech had hoped for a supercomputer within the top 10, the top 5 being optimistic, but they were elated to find themselves placed with the worlds third fastest supercomputer. Both Apple and Virginia Tech were very proud of their results, and is an excellent demonstration of a leap in super-computing performance, super-computing manageability, and supercomputing price. Price of course is always a determining factor when considering the development of a supercomputer [Fig. 6].



Fig. 8 - The rack-mount housing

The price for these 1100 PowerMac G5's, their rack-mount housing [Fig. 8], the cooling system [Fig. 7], and of course the miles of fiber-optic cable that were needed came at the mere price of \$5.2 million US. Now this may seem like it is a very large price to pay for a supercomputer, but in actuality the price of the system is very small. Consider the chart on the following below.

Here is a comparison of the worlds fastest 3 supercomputers [Fig. 9]. Virginia Tech is about 30% slower than the Los Alamos Lab, and more than 3 times slower than NEC's Earth Simulator [13].

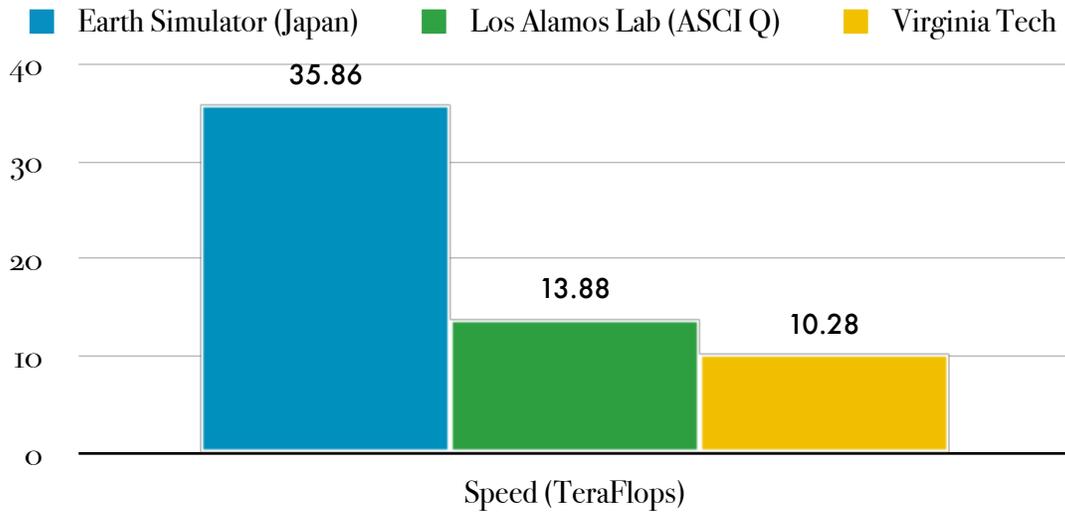


Fig. 9 - World's fastest 3 supercomputers (2003)

However when considerations additional cost in millions that these organizations have paid, Virginia Tech is the clear winner. The cost savings of implementing an Apple Grid Cluster (\$5.2M) [Fig. 10], and using it as a supercomputer are immense in comparison.

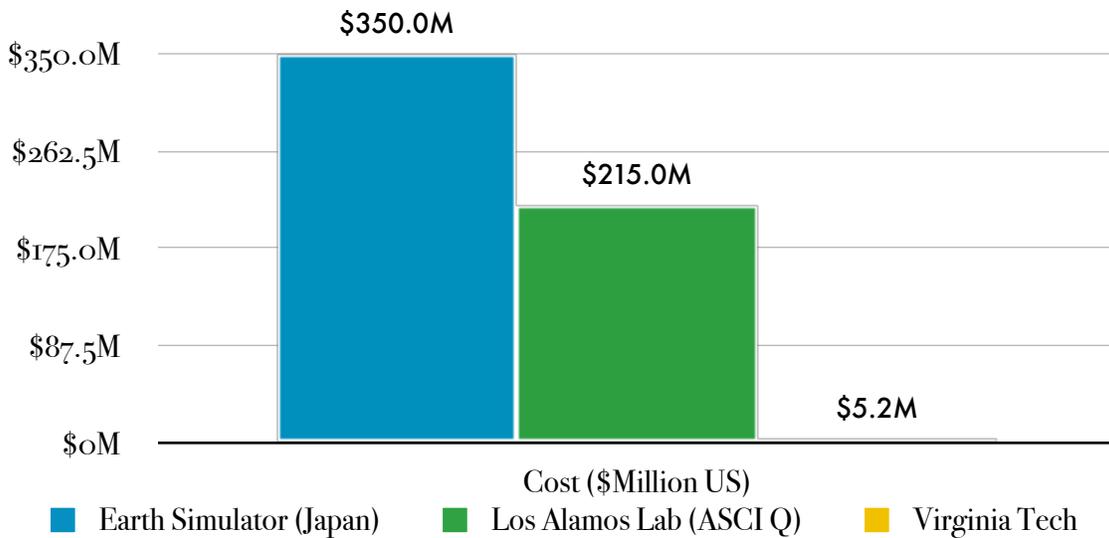


Fig. 10 - Cost in millions for the 3 fastest supercomputers in the world

So not only is grid computing powerful, but it is also much more cost efficient than custom built supercomputers. Virginia Tech had the aspirations of becoming one of the top 30 research universities in the United States, and now with the fastest university supercomputer in the world, they are a step closer to this goal. Attracting researchers to work at the university is now much easier as all the infrastructure is in place for compute intensive research.

What Virginia Tech has done is absolutely amazing, but how much more power would they have been able to harness if they had been able to add the power of every single PC in the University as auxiliary network? Obviously connecting up every odd machine to their existing grid wouldn't necessarily assist its speed as issues like network bandwidth and latency must be considered. For example, computers across campus would not be able to work as quickly as systems in their PowerMac cluster, and would bog down the system as the grid would always be waiting on these often much slower machines to finish processing. However, it should be possible to take all of these extra machines, and using software such as Condor or Apple's *Xgrid* connect them in such a way that they could perhaps add a 3 -5 teraflop addition to their already existing grid. This would be a very interesting project to engage in, as not only would it test theories of cross-platform computing, but it would also test concepts such as grid integration. In this scenario, these two grids could be informally connected for administrative purposes, and the second network could largely be used by secondary researchers such as graduate students and undergraduates who may not be given access to the supercomputer.

The previous paragraph briefly touched on Apple's *Xgrid* software as a possible system for cross platform grid computing. Xgrid is a brand new player to the grid market that has yet to make a 1.0 release, and is of right now still a preview technology. Xgrid comes with two types of software packages intended for different uses, one is named Xgrid and the other is known as Xgrid Blast. Xgrid Blast is Apple's open source biotechnology applica-

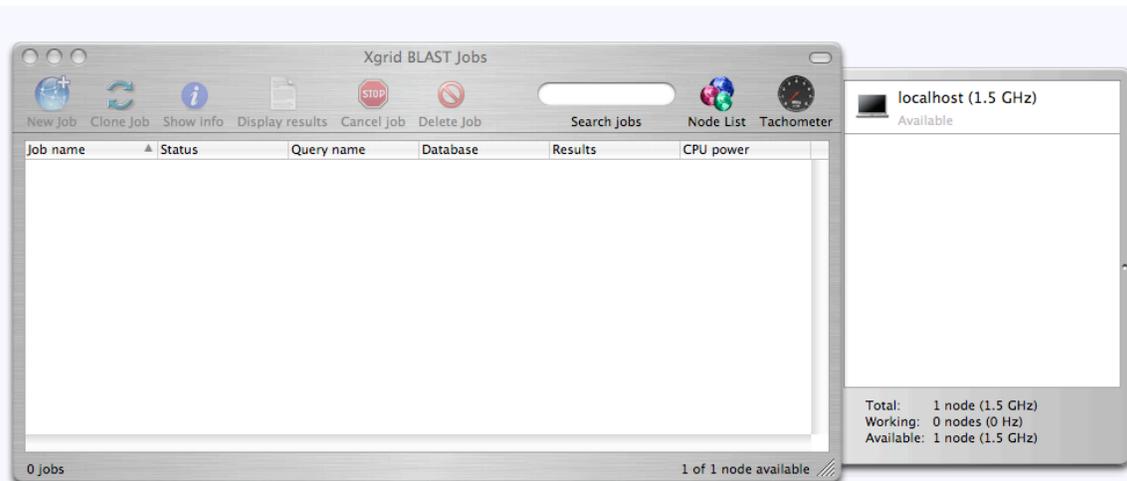


Fig. 11 - A screenshot of the Xgrid Blast application with one computer at 1.5GHz connected

tion and will only be quickly discussed in this paper. Blast is used by life science researchers to find matches in DNA and protein sequences. For example Richard H. Scheller PhD who is vice president of Research at Genetech uses blast to query DNA sequence files for matches against multi-gigabyte genomic databases on a cluster of four dual-processor Xserves [14]. Xgrid Blast connects to existing Xgrid clusters, and provides much more complex reporting and managing than that of the simple Xgrid application, however it is limited strictly to biotechnology research, and as such it will not be a major component of this research paper. The Xgrid application itself however is capable of much more than dissecting gene sequences and even has very preliminary linux support. Xgrid is a computational clustering technology and is designed to bring the power of grid computing to the masses. Xgrid allows anyone to interconnect servers, workstations, and dedicated machines to an Xgrid cluster, and does so in a way which is simple and easy to configure and distribute. Xgrid uses technology similar to Condor's "idle" technology where computers which are inactive will only compute in the grid when they have become idle. This allows an individual or an organization to maximize grid performance, without affecting any of the users connected to the grid. The market is screaming for more powerful grid tools, and Xgrid delivers this in a way that only Apple could invent.

PROJECT JUSTIFICATION:

With the rise in demand for more and more computational power, particularly by large research institutions, the dawn of grid computing is upon us. Gone are the days where a central mainframe system would do all the work for scientists, 3d artists, and mathematical theorists. Certainly these mainframes will still be kept and will still do computation, but computers located throughout the organization working together or separately will do a much larger amount of computation. Grid technology is very much a cutting edge topic and the software that is available to do Grid computing is still very much in its early stages.

In every organization there are computers of all speeds, and they often sport different operating systems. Take for example a typical University. There will be Windows systems, Macintosh systems, and of course the odd Linux box (particularly in computer science departments). The problem posed is that in order to implement a grid that will do computations on all these varying systems, the grid must work in a cross-platform manner. The whole purpose of this project is to explore the possibility of cross-platform grid computing, and determine if it is not only feasible, but also a much faster way of doing large tasks that would take a single computer much longer to compute.

This grid will be tested through the use of Pov-Ray. Pov-Ray is a high-quality, freely available ray-tracing software package that is available for PC, Macintosh and UNIX platforms. Pov-Ray allows a programmer to create complex graphics through the use of its own custom language. It employs a concept known as ray-tracing to do this which allows a programmer to create light rays. These rays custom programmed into specific scenes and can be refracted by mirrors, glass, or undergo various other contortions, all of which result in a single pixel of the final image. Obviously this is a very intensive process that would take a single CPU many hours (or days) to finish; however, this paper suggests that this time can be shortened greatly through employing a grid. Benchmarks will be provided between processing time difference of an individual computer versus the processing time over the grid.

Brief Descriptions of Project Products:

- An overview of grid computing and its benefits
- Review of possible Grid Platforms (Xgrid and Condor) (time permitting)
- A review of possible Linux distributions that will be used in the project
- An analysis of the Xgrid interface
- Install Linux and Mac OS X
- Implement the Xgrid Agent on the Linux Platform
- Implement the Xgrid Controller on OS X
- Create a remote retrieval script for distributing files
- Create a Pov-Ray scene for benchmarking the grid
- Install and configure a 12 port switch for use by the Grid
- Add as many systems as possible to the Grid
- Determine optimal job size for Pov-Ray renders
- Explore the use of SSH for remote administration of the Grid (time allowing)
- Provide a brief report on the future of Xgrid and grid's in general
- Provide some recommendations for how this may be used in a university setting. (Time providing)

Summary of Project Deliverables:

(See Work Breakdown Structure and Gantt chart Appendix 12)

Determination of Project Success:

The success of this project will be defined by the analysis that the project provides on cross platform Grid Computing. As this field is still very cutting edge at the moment there is no way to anticipate whether a Pov-Ray scene in a cross platform manner can be rendered. Thus the success of the project cannot be simply based on the ability to successfully render something in a cross platform manner. Rather, determine the project's success through its ability to provide exposure on cross platform grid computing, its successes, failures, and its direction.

Definition of Hardware Resources:

The following hardware resources have been acquired for the scope of this project:

- One 1.5 GHz Powerbook G4 w/ 512MB RAM and 128 MB Video
- One 1.2 GHz iBook G4 w/ 512MB RAM and 32 MB Video
- One DUAL 1.2GHz PowerMac G4 w/ 1,5 GB RAM and 64 Mb Video (Sketchup Only)
- One 2GHz AMD PC w 1GB RAM and 128Mb Video
- One 1.6 GHz AMD PC w/ 256 mb RAM and 32Mb Video
- One 400 MHz PC w/ 386Mb RAM and 8 Mb Video
- One 200 MHz PC w/ 96Mb RAM and 8Mb Video
- 12 Port 10/100 Switch
- 4 Port 10/100 Wireless Router

More hardware may need to be acquired if adequate performance results cannot be obtained. There are at least 2 partially built systems, which are available and would require processors, memory, video cards, and hard drives in order to become functional. In addition to this 5 or 6 additional computers may be loaned from friends if necessary.

Definition of Software Requirements:

Xgrid requires Mac OS X 10.2.8 or later (including 10.3), with a minimum of 128MB of RAM but 256 Mb is recommended. The package may be freely downloaded, and is easy to install.

The Xgrid agent for Linux requires a Linux installation with the following packages installed:

1. libxml2
2. glibc
3. roadrunner (the BEEP library) and its required library glib-2.0 (and libxml2)
4. Xgridagent.c Xgridagent.h Xgridagent-profile.c Xgridagent-profile.h
Xgrid.config.xml (see below for download)

In order to install Pov-Ray the following are required:

Linux - Requires a glibc-2.2 based GNU/Linux system running on x86 hardware. In order to have Pov-Ray run in an identical manner to OS X it may be necessary to install DarwinPorts.

Windows - The 32-bit version requires at least Windows 95, Microsoft HTML Help and at least Internet Explorer 3 (for the HTML Help engine). Internet Explorer 4 or later is strongly recommended so as to ensure the correct functioning of the documentation.

Mac OS X- Requires at least Mac OS 9.2 or at least Mac OS X 10.2.8. Using Mac OS 9.2.2 is strongly recommended. For Mac OS CarbonLib 1.6 is required.

In order to install WebDav the following are required: A working Apache 2.0 installation with sufficient permissions to edit the httpd.conf file. Some modules may also be required.

XGRID OVERVIEW

Introduction to Xgrid

Xgrid is a Mac OS X application designed to bring “Apple’s legendary ease of use to parallel and distributed high-performance computing” [15], or as Steve Jobs often states, high performance computing built for “mere mortals.” One of the biggest problems in the grid computing industry is that installing and configuring your own grid can be a very complex and time consuming process. Apple’s *Xgrid* technology is aimed squarely at the backyard researcher, graphic artist, or scientist, but can scale remarkably up to levels that major researchers need. This technology is still in its preview (beta) stages, but is expected to be at a stable release within the next 6 months.

In brief, Xgrid allows anyone to take a group of nodes (individual computers) and network them into a cluster or grid. Users on the cluster are allowed to create long running computations known as “jobs” and submit these jobs to a controller which is a dedicated machine on the network which handles the management of these jobs. Xgrid can then create multiple tasks for each job, and distribute those tasks among multiple nodes. A concept known as “Desktop Recovery” is used to describe the action of coupling individual computers into a computational grid when those machines would normally be idle. Desktop recovery is a big feature of Xgrid as it allows computations to run only when those machines would otherwise be idle, and as a result does not impact the individual who would normally be working on that machine. Xgrid also allows systems to run as dedicated nodes, which means that they will dedicate 100% of their processing power to jobs as they are placed on the grid. This combination of desktop recovery and dedicated machines allows a researcher or individual to maximize the use of available processing power and dedicate it to their work.

As far as constraints and limitations are concerned theoretically Xgrid has none. In theory, Xgrid should be able to allow any number of systems to connect to the network, resulting in N Gigahertz of effective power. During the Xgrid preview however, the maximum number of machines that can connect cannot exceed around 300-500 Ghz. Al-

though this is definitely a fairly low figure in terms of raw computational power, for the average lab, teacher, student, or researcher, the power that Xgrid can harness is more than adequate.

If more computing power than this is required, other grid projects such as the Condor Project are designed for large-scale computational grids. They feature more advanced administrative tools, flexible resource management utilities, and built-in features like checkpointing which allow individual jobs to be stopped, and resumed by a completely separate machine. These grid projects are designed to be fully featured frameworks for grid deployment in large computing environments and are complex to setup, complex to maintain, and require in-depth command line knowledge.

Although Xgrid does not provide all the functionality that projects such as Condor, or Globus [16] have to offer, it does have much that these projects do not. Some of these current benefits include:

- ▶ A fast and efficient client setup process that requires little configuration in order to allow a single individual machine to participate.
- ▶ GUI based grid submission tools, and GUI based grid monitoring.
- ▶ Command line knowledge is not necessary, although it can be used by those who are more comfortable with it. Xgrid is capable of being fully managed through a terminal.
- ▶ The utilization of Apple's Rendezvous² technology which permits individual computers on the network to connect to a controller without having to know the IP address.
- ▶ Hiding complex issues such as data distribution, job execution, and result aggregation from the user.

² See Glossary [2]

- ▶ Using Xgrid's *BEEP*³ framework it is theoretically possible to secure controller/client communications through a SSL tunnel.
- ▶ The ability to create your own customized plugins that save both time in job submission, and allow a grid administrator to quickly distribute a customized task.
- ▶ Allows for a diverse range of input and will permit any UNIX style operation to be run.
- ▶ Aims to support all types of computational needs ranging from graphical rendering to scientific and mathematical calculation.
- ▶ Using cocoa it is possible to integrate Xgrid right into your applications to help with computing intensive tasks.

Xgrid in Action

Despite Xgrid is still a preview technology, its revolutionary approach to grid computing, and its ease of use, have made it a success in many academic fields. There are currently many ongoing research projects around North America which are using this technology, and are integrating it right into University laboratories, a few of these projects are listed below.

Xgrid at Stanford University - Dr. Charles Parnot, and Brian Kobilka, M.D work in the Molecular and Cellular Physiology department at Stanford and are utilizing Xgrid to accelerate a better understanding of pharmacology. They are currently running large calculations that need massive amounts of computing power to model the conformational changes of the “beta 2 adrenergic receptor.” This is done through modeling G-protein coupled receptors. This complex research could provide further understanding on how these receptors could react to medications used to battle various diseases related to the heart, coronary system, and asthma. Naturally since this type of work requires the proc-

³ See Glossary [3]

essing of enormous volumes of data, and with numerous parameters per model, this task is ripe for a grid. Using the combined power of around 200 - 400 Apple computers located all over the world [Fig. 12] they have been able to reduce calculation times for models from 1 year to 1 week [17]. Currently they have between 100 - 200 agents located within North America, and an estimated 200 - 300 agents in Europe. See the diagram below for a depiction of agent locations:



Fig. 12 - The locations of major contributors to Xgrid at Stanford

Xgrid at Reed College - Under the guidance of Professor Richard Crandall, and given government funding, the students at Reed College have utilized an Xgrid in their studies in Epidemiology. The computational team was part of a much larger overall research team, and was responsible for the construction of combinational models of epidemics with a particular focus on an original theory of the fractal character of survivor sets. The team has written software to use these models, and has produced a paper on the implication of this research on fractal characteristics on vaccination strategy. Their grid ranged in size from 10 Ghz to 100 Ghz and managed to reduce their computations from an estimated 30 CPU years to something the length of a “CPU summer” [18].

North Carolina State University - Sammie Carter has been working on Grid computing for the student masses, and has a very fresh perspective on what grid computing can be. "Wolfgrid is a project whose aim is to build a computing community at NC State where the individuals are the computer. In this community everyone helps, supports, and sustains each other. It is a project hoping to bring people with vastly different computing needs together. NC State intends to use this system to support the various demands of both scientists and artists, while encouraging communication and collaboration within the Wolfgrid community. Wolfgrid aims to be a community super computer where the community is the computer" [19]. Their system currently is running at around 20 - 30 Ghz and consists of systems all over the campus. Students donate spare CPU cycles on their computers to assist their colleagues on research projects.

University of Utah - James Reynolds has created a grid with speeds of up to 500 Ghz which has been made available to both arts and science students. The arts students have used the grid for rendering 3d scenes they have created using a very popular graphics program known as *Maya*. The science students have used the grid for working on specific research and projects, and have found the power of the grid to be a definite aid in their studies.

Simon Fraser University - SFU has employed Xgrid to take advantage of the unused CPU time available on Macintosh computers throughout the university. As of May 2004 SFU's Xgrid cluster consists of over 80 machines, running at over 55 GHz combined on the Xgrid Tachometer [20]. Research using Xgrid has so far been concerned primarily with problems in computational mathematics where solutions are "embarrassingly parallel", such as stochastic and independent exhaustive searches. Some of the projects have included turan conjecture, fermat number factoring, Mahler's measure of polynomials, and least autocorrelated binary sequences.

Under the Hood of Xgrid

The technology behind Xgrid is not much different that many other grids that can be found today on the market, however Apple's implementation of this technology is quite

unique. In order to discuss further how Xgrid works we must first take a look at the basic manner in which Xgrid functions. Below [Fig. 13] is a schematic of the roles which computers can take within an Xgrid.

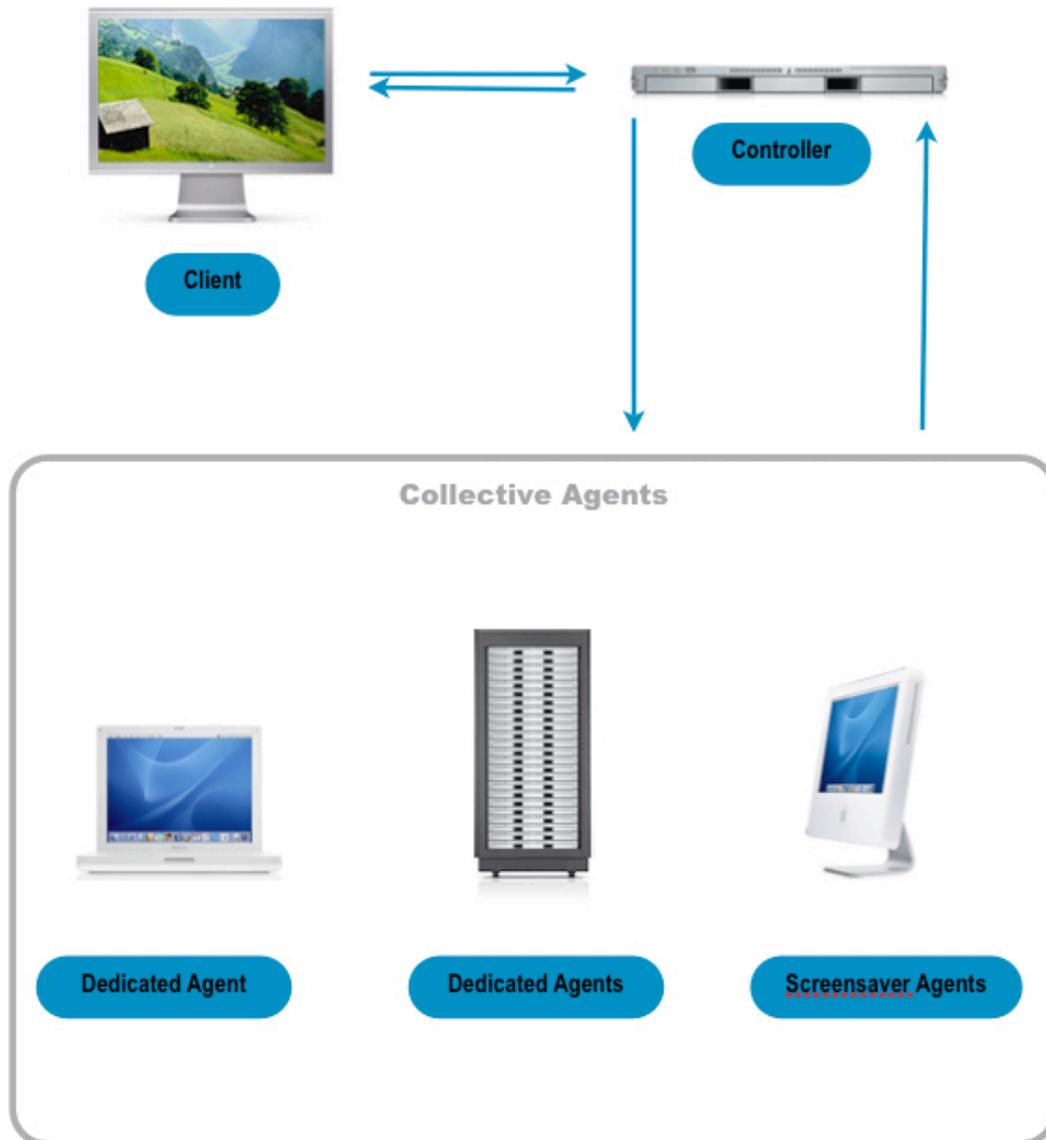


Fig. 13 - The roles and interactions of computers within an Xgrid

As we can see on the previous page there are three types of roles which computers can take: agents, controllers, and clients. These components each assume different roles in the formation of an Xgrid and each is an essential part of creating the grid.

Agents:

Agents are the workhorse of the Xgrid, any computer running either Linux (unsupported) or Macintosh OS X (supported) can run as an agent. Under OS X the agents are located at `/Library/StartupItems/GridAgent/GridAgent`, and under Linux they are available in whatever installation directory you specify. Agents are responsible for receiving jobs from the controller, processing the jobs, and returning the jobs to the controller again. Agents can either run as dedicated agents, meaning that they will always accept jobs, and will dedicate maximum processing jobs to those jobs upon their arrival, or agents can run in screen-saver mode, meaning that they will only accept jobs if the user has been away from the computer for a specified period of time. Linux agents at the time of this report unfortunately are still very experimental and can only be run as dedicated agents. Agents in an XServe cluster as demonstrated in the middle box in the above illustration (Fig. 13) are also capable of joining the Xgrid. In order for a OS X system to run as an agent it is necessary to specify how the agent will connect. It is important to note at this point that the agent can be forced to provide a password to the controller as part of a mutual authentication process. This is done through the agent password preference pane of Xgrid.app. Connecting to the controller can be done in three different ways:

- (1) Binding to the first available as discovered through Rendezvous.
- (2) Binding to a specific service name (ex. domain.com)
- (3) Binding to a specific host (ex. 192.168.0.100)

Once the agent has connected to the controller and begins processing a job it reports its processing speed to the controller which in turn reports it to the client for output on the tachometer (See Clients section for details). This is done either through a call to `/usr/sbin/system_profiler` under OS X or through `/install/dir/Xgridagent/Xgridagent.xml` under Linux.

In terms of what is copied to agents the command will be copied to a temporary directory located in: (Ex.) `/tmp/xgagent.GSIOZ6o4/bin/sh` and runs under the user “nobody”. The working directory which is used to hold temporary files or downloads is located in: `/tmp/xgagent.bUDiNL4g`. This is very important to note because it is sometimes necessary to check the status of these folders in order to inquire as to why an agent crashed, or whether certain files downloaded correctly.

Clients:

Any OS X system can be an Xgrid client if it has the Xgrid application installed, and has a functional network connection to the grid (A password may also be necessary). The client provides access to the controller, and allows any local user to submit jobs to the controller using either the command line Xgrid command, or a graphical user interface. The user can define any number of parameters through the client (See Appendix B. The Xgrid User Interface) and once submitted, these are sent to the controller. When the job has been completed the client is notified and can retrieve the results from the controller. One of the best features of the client is the ability to create a custom job called a “plug-in” that is stored on the controller, and can be utilized later to run similar instances of the same type of job. Clients can connect either through Rendezvous or IP address.

Controller:

The Xgrid controller is the heart and soul of the grid as it is responsible for managing communication and resources of all the available agents on its cluster. The controller receives job submissions from the client(s), breaks them up into tasks, dispatches each task to an agent, and provides the job submission results back to the client. There can only be one controller per grid, however there are allowed as many controllers per subnet as there are IP addresses. Although Apple claims that there is no theoretical limit to the number of agents that are permitted per controller, there is certainly a practical limit which is based partly on network bandwidth, the extent to which a job can be paralleled, and of course the issue of the controller itself crashing which seems to happen at random intervals from time to time. That mentioned, Apple has gone on to state that no controller can currently handle more than 10,000 connections which is quite confusing as it certainly isn't the “no theoretical limit” that they advertise. The controller process can be found in the location: /Library/StartupItems/GridServer/Gridserver and can be executed via the command line from that location. The controller listens on port 4111 and possibly other non-privileged ports, and broadcasts via Rendezvous. The controller is also capable of giving passwords to agents, or requiring passwords from clients which is really its only form of security at the present point in time.

Xgrid Security

Now that this report has considered some of the basics of how Xgrid works, it needs to consider another aspect of grid computing that has been briefly mentioned in some of the above sections, and that is the issue of security. As mentioned in the agent section, Xgrid processes run on the agent as the standard UNIX user “nobody”, and can execute anything that the world user can execute including (but not limited to) applications or scripts located in the following directories:

- /Applications
- /bin
- /sbin
- /usr/bin
- /usr/sbin

This “nobody” user can read anything the world user is typically allowed to read which can include very confidential files such as those located in the /etc directory. This directory contains things like hostconfigs, authorization preferences, and http server configurations. Things only get worse from here as the world user is allowed read access to preferences located in the /Library folder, and even personal user folders such as those located in /Home or /Users. The world user is also given write permissions to anything located in /tmp, /var/tmp, and /Volumes. So what does all this talk about read and write access mean in terms of actual security vulnerabilities? Well the answer is that unless you are running in part of a highly trusted Xgrid cluster, joining an unknown Xgrid cluster as an agent would certainly be a very foolish thing indeed as any administrator could write a script that could easily utilize the grid’s distribution methods to compromise your system, or acquire personal information about you. The *Xgrid* agent is provided with very little protection from a malicious individual utilizing a Xgrid client, and as such it is highly recommended that you only join an Xgrid cluster when you are absolutely positive that the administrator is responsible, and their networks are properly secured. Hopefully this will change as Xgrid matures into a full 1.0 version, but the current preview release 2.0 does not incorporate any security as far as file permissions go. In terms of security available to the controller and the client, all connections to the controller can be required to be authorized by a password which is encrypted. Controllers can force clients to provide these passwords prior to being able to submit jobs, and as of the date of this research paper there are no known exploits to bypass these restrictions. The remaining content exchanged between the controller, agents and clients is not encrypted, but this report has discovered that it should be theoretically possible to pass Xgrid jobs

through an SSL connection as SSL is supported within the BEEP framework that Xgrid runs on, but unfortunately there is no current way to activate it.

How Job Submission Works

When a client submits a job to Xgrid there is a particular order of operations that occurs.

```
Praetorian-OSX:~ optio$ /usr/bin/cal
  March 2005
  S M Tu W Th F S
      1 2 3 4 5
  6 7 8 9 10 11 12
 13 14 15 16 17 18 19
 20 21 22 23 24 25 26
 27 28 29 30 31
```

In order to best demonstrate how job submission works it is best to provide an example. In the example we will run the command `/usr/bin/cal`. This will run the standard Unix calendar command, and given no arguments it would normally print out a list of all the

days in the month.

In our case though we will be passing parameters to it so that more than 1 computer can work on it at a time. For a list of the arguments we are going to pass, see [Fig. 14] below:

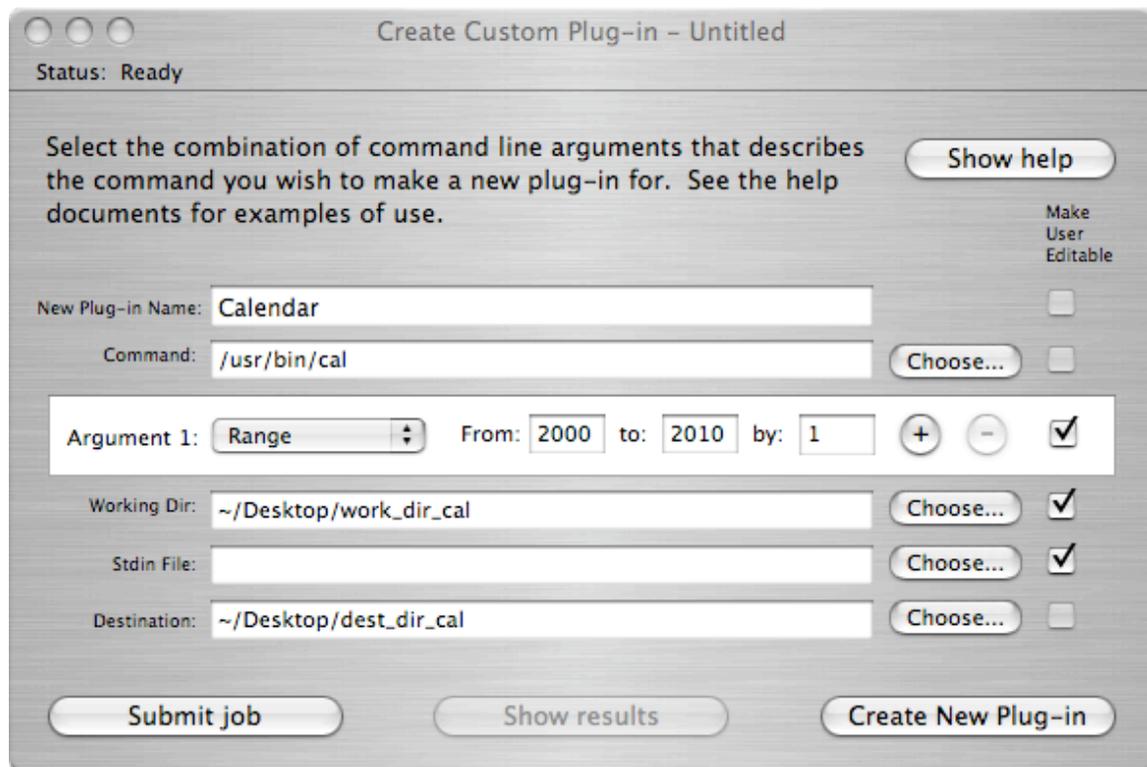


Fig. 14 - The submission of cal (calendar) jobs to Xgrid

As it is seen here, the cal command is going to be passed the parameters 2000 to 2010 by 1. Hence in this particular example there are 11 jobs. Since each agent on the grid will only receive 1 job, each agent may have to connect more than once depending on how many agents are connected. Once the submit button is pressed a confirmation box will appear to tell us that we are about to submit 11 jobs and that the input size for these tasks is 225280 bytes, and do we want to submit the job. Upon pressing submit the following actions will occur:

1. The client compresses everything inside the working directory into a tarball. In our case nothing is inside of this directory but normally you would copy your script command to that directory unless it is available on every machine like /usr/bin/cal is.
2. The tarred data is sent with all the folders in the hierarchy (so if there are any sub-folders inside the working directory they will be submitted as well) to the first agent along with the first argument.
3. Each additional agent will also receive a tarball of the working directory along with the next argument. This loop until there are either no more agents available, or no more arguments for additional agents to take. Agents will take these jobs in order of speed, so in our example if we have six (2 GHz) agents, and five (1 GHz) agents, and five (500 MHz) agents we will see that first the six (2 GHz) Gigahertz agents will take jobs, and then the five (1 GHz) agents will take jobs, but then all the jobs will be taken so the 500 MHz machines will receive no jobs. XGrid identifies the speed of connecting agents through a XML configuration file that specifies the speed in MHz of the processor of the agent. Negotiation of which agents get tasks is completely moderated by the controller. This is done so that jobs will be computed using the maximum power of the available resources. For much more complex jobs this becomes even more apparent as slow agents will often bog down the completion of tasks.
4. The agent receives the tarball and extracts it to /tmp/xxxx where x is a string of letters unique to that particular job. This folder is then the working directory for the current machine.

5. Since the tarball will extract, in our case the actual working directory will be `/tmp/xxxx/Users/myuser/Desktop/work_dir_cal`
6. The command executes from this working directory and takes the given argument and passes it to the executable.
7. Once the task is done, the output is captured to `STDOUT` and is placed in a text file and returned to the server. If there was any further output (Later in the paper we will look at Pov-Ray rendering) it will compress everything in the folder up in a tarball and upload that back to the controller.

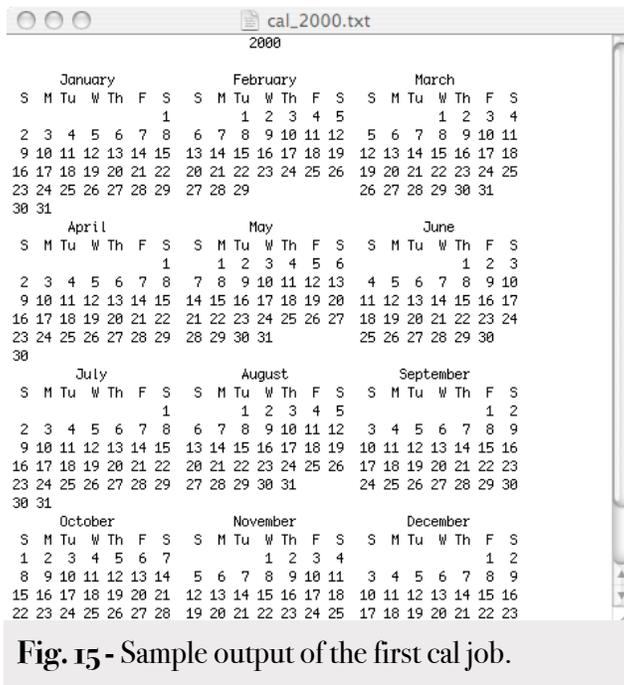


Fig. 15 - Sample output of the first cal job.

8. A button is displayed on the client indicating that the job is done and the results can be viewed. Upon clicking this button it will then open the destination directory (In this case it is `~/Desktop/dest_dir_cal`) and will display the results. If you are not using the command-line client, the results are simply found in the destination directory specified. See [Fig. 15] for sample output of the first job.

There are also several ways in which to submit jobs, and there are many different types of jobs that can be submitted. The next section will look at some of these job types, and what each type has to offer.

Job Types

There are several methods for submitting your jobs to Xgrid. Once the client has been started a screen will be presented for a new job. At this point the user will see a listing of

different job types that the grid can perform. Notably there are 5 primary job types for submitting jobs: Shell, Xfeed, Create Custom Plugin, Factor, and MPILaunch. There also is a Mandelbrot job which computes complex fractals but this is really used much more as a testing measure for the grid than as an actual submission method worthy of mentioning here. Each of these 5 methods serves a different purpose, and each provide different functionality.

The Shell job type is best used to demonstrate a single task being submitted to a single agent. The user can specify any standard Unix command, and upon submitting the command, the fastest agent currently connected to the grid will retrieve the job and execute it. Upon my first time running this job type, I had assumed that once I submitted a unix command to the grid that all agents within the grid would run the job, however it is strictly only the fastest computer on the grid which will retrieve and execute this task. In fact for all tasks that are not passed to the controller with arguments, the fastest computer on the grid will always grab the job.

The XFeed job type allows you to specify either a command, or a path to a script which you have written. XFeed is different than Shell jobs in that it also allows you to specify arguments which the command or script will take. Arguments can either be a literal, a range of integers, or a random integer. Arguments can also be stacked so, for example, you can specify both a literal, and a range argument when you pass the command to be executed. It will execute everything in sequence, and in the example it would deal with the literal first, before processing the range arguments. Both ranges and random integers allow you to specify the starting value, the finishing value, and the amount to increment each value by. It is important to note here that if you specify a range argument (or a random argument), and provide a range greater than one, that each agent on the grid will retrieve a single argument and process it. This is a key concept in grid computing, and one that took a considerable length of time to discover. I had previously assumed that somehow the controller knew how to take any given argument and divide it up between the connected agents, but in fact single arguments will only ever be processed by single agents. Agents cannot share arguments, and cannot just simply specify a command with no arguments and expect that it will run on every machine on the grid. Even if it could

do this, there really wouldn't be much point to this functionality as each agent would return the same results so all that would end up happening is that each agent would return identical (or very nearly identical) results to the controller depending on the nature of the command specified. XFeed also lets you specify the location of a source executable script if you are using one, a location for inputs, and a location for where you would like the output of your commands to go.

The Create Custom Plug-in job type is perhaps the most powerful of the available methods, and it allows you to submit your own complex scripts or jobs to the controller. Plug-ins that you create can be saved, and recalled later so you do not need to retype commands. Since every Xgrid task that you create is a command-line string that is distributed to agents, providing the functionality to save these custom strings is a great way to make complex tasks accessible in the future whilst requiring a minimum of input required back from the user. Bash, Shell (sh), Python, or Perl scripts are all fully capable of being run via either XFeed or the Custom Plug-in and can significantly extend the power of Xgrid tasks. In fact, plug-ins provide much of the same functionality that XFeed jobs provide, but extend that functionality by allowing a much more diverse range of allowable arguments. For a more complete listing of every option a custom plug-in is capable of submitting, please see Appendix 2 and look at the page discussing the Xgrid custom plug-in. The Custom Plug-in was used extensively in the creation and testing of Xgrid for this report.

The Factor job type is a very simplistic job that is best used for grid testing purposes. The factor job allows you to submit a batch of factoring tasks as a single job. A factor job takes three parameters: mod, exp, and machine passes. It then computes the results based on the data you enter in the form of: $2^{\text{exp}} + \text{mod}$ where special efficiency accrues when $\text{mod} = -1$ (Mersenne numbers) or $\text{mod} = +1$ (Fermat numbers). For example, when the arguments mod:1, exp: 128, and machine passes: 4 is entered, there are 4 text files that get generated and each contains the results of the factor being run. An example of the first machine pass can be found below:

```
Factoring on Praetorian-OSX.local:  
340282366920938463463374607431768211457
```

```
Sieving...
Commencing Pollard rho...
Commencing Pollard (p-1)...
Commencing ECM...
Choosing curve 1, with s = 1578126186, B = 1000, C = 50000:
Choosing curve 2, with s = 737139265, B = 1000, C = 50000:
Choosing curve 3, with s = 490722994, B = 1000, C = 50000:
Choosing curve 4, with s = 1692156077, B = 10000, C = 500000:
Choosing curve 5, with s = 1402156263, B = 10000, C = 500000:
Choosing curve 6, with s = 1299376378, B = 10000, C = 500000:
Choosing curve 7, with s = 2114855085, B = 10000, C = 500000:
Choosing curve 8, with s = 864277417, B = 10000, C = 500000:
Choosing curve 9, with s = 1310676679, B = 10000, C = 500000:
Choosing curve 10, with s = 651493669, B = 10000, C = 500000:
Choosing curve 11, with s = 650143996, B = 100000, C = 5000000:
```

Although this particular job type is not very useful in anything to do with real world applications, it does demonstrate one of the many ways in which Xgrid can be used, and it is useful as a benchmark.

The MPILaunch job type is the last of the useful job types and allows a user to run an Message Passing Interface (MPI) job by specifying an executable compiled with MacMPI. MPI is best used when communication between agents is necessary, and as such it is a very powerful tool. The solution is MPI, the Message Passing Interface. Developed over the last several decades as an alternative to shared memory architectures (such as OpenMP), MPI enables developers to efficiently program "tightly coupled" algorithms which require nodes to communicate during the course of a computation. MPI is used by many distributed programmers as an alternative to shared memory architectures such as OpenMP and allows developers to create highly dependent algorithms which require agents to communicate throughout computations. MPI consists of a standard set of API calls, that can be implemented as a cross - platform communication library that will manage all aspects of communication and data transfer between agents. MPI provides a layer of abstraction that works well with existing networking architecture (ex. Ethernet, Myrinet, InfiniBand), while allowing developers to recompile their programs for any platform and hence providing a layer of cross platform support. Different implementations can be run either standalone, or embedded within scheduling solutions like Xgrid. Although

MPI is outside the scope of this research paper, it is important to note that Xgrid's ability to execute these tasks can really add significant power to tasks that are dependent on one another. MPI and Xgrid are an excellent combination for many scientists and researchers.

The feature set and job submission tools that Xgrid brings to table of grid computing is quite significant, and will only increase with further releases of this technology. While using these tools, one will quickly notice that Xgrid also takes one of the weakest elements of the field of grid computing which is the usability and the user interface aspect, and transforms it into one of its strongest elements. Xgrid is truly a breakthrough in grid usability and the following section will detail more on how the user interface works for Xgrid.

The Xgrid User Interface

As with all things that are designed by Apple, Xgrid has been designed with the end-user in mind. Other grid applications such as Condor require command line configuration or have extremely complex interfaces which certainly hinder administrative tasks and confuse users. Xgrid has been chosen primarily for this project because it takes something as complex as grid computing, and breaks it down into an application that is both robust and easy to use. Alan Cooper, author of "The Inmates are Running the Asylum" states in his book that "The essence of good interaction design is to devise interactions that let users achieve their practical views without violating their personal goals" [21]. In the case of most users, they want to be able to do their work without feeling inadequate or stupid. Complex and confusing interfaces achieve just that, but Apple's Xgrid interface is surprisingly simple considering the wealth of power available at your finger tips. This is technology that everyone can use, and it doesn't require a PHD in computer science in order to figure out, which is the beauty of it. For a more thorough overview of the Xgrid user interface see Appendix 2. The next section details how to implement this Xgrid on OS X.

XGRID IMPLEMENTATION

Implementing Xgrid on OS X

Prior to discussing this paper's implementation of Xgrid, and how this paper has used this technology to assist in real-world tasks, it is worthwhile to point out that until I started working on this project I had no experience in any of the technologies that have been used. In fact the only thing that was remotely familiar with me throughout the entire course of the project was the general use of the Linux operating system itself. I have been using OS X since January of 2004, and have used Linux since July of 2003. I have never touched anything to do with grid technology, bash scripting, perl scripting, CGI-scripting, XML, C, GCC compiling, 3d rendering, architectural CAD, customized Apache server configurations, or writing code for distributed environments. Neither have I ever written a report so extensive or complete as I hope this one will be. So needless to say just about everything I am about to embark on is completely foreign to me.

As this project was constructed from the ground up, I started off with only 2 operational systems, both running Mac OS X. The primary focus at this point was to get the network operational, and then setup Xgrid to attempt to render a single Pov-Ray file. One system was designated as the primary Xgrid controller is a 1.5 GHz PowerBook G4 (Currently one of Apple's top of the line laptops). The other system was a 1.2 GHz iBook laptop which was designated as a full time agent. Both of these systems were connected to a 5 port DLINK 504P Wireless-G router which had been purchased in December 2004 along with a high gain antenna. This router was purchased specifically for the purposes of this project, as I was quite interested in seeing how the grid could perform under wireless conditions. The omnidirectional high gain antenna was purchased to increase the signal strength of the router as I initially ran into connectivity issues where the laptops would not be able to connect to the network properly due to poor network signal strength. As with all wireless networks, security is always a consideration, and as I wanted my controller to function wirelessly I took the time to properly setup WEP encryption using a 128-bit hex key. This was done manually through my routers' web based interface. Each laptop

was then set through the Apple's Airport interface to connect to the router, and utilize the WEP password that I provided. After reading several online articles on wireless security it became apparent that using a very simple WEP key was poor practice so I found an open-source WEP key generator, and set a password that consisted of much more irregular characters. I also wanted to lock the network down further by only allowing systems to lease IP addresses if their MAC address matched the MAC address filter on the router. Unfortunately due to DLINK's poor support for network security this feature refused any computer access to the router as long as WEP was enabled. So the option at that point was to either run an unsecured wireless network with MAC address filters, or a WEP encrypted wireless network with no MAC filters. I chose the latter because I would much rather run a secured wireless network than one that anyone could join if they could clone a MAC address. As this report is not a report on wireless security further details on this setup will not be disclosed, however these concerns were worthy of brief mention as they are an aspect of our implementation.

The next stage of the setup for OS X involved downloading Xgrid from Apple's High Performance Computing website here: <http://www.apple.com/acg/xgrid/> . Once this was downloaded Xgrid can be installed by double clicking the downloaded .dmg. It will then auto-mount the Xgrid installation volume to your hard drive, and by double clicking on the Xgrid installable the setup process will begin. To see a list of what folders and files Xgrid creates please see Appendix 3.

Once Xgrid is installed we can start our grid. The Xgrid controller is used to start the grid services and can be started either through the Xgrid preference pane or through the command line by running:

```
%sudo /Library/Xgrid/Scripts/server_on  
%sudo /Library/Xgrid/Scripts/server_start
```

The agents can be started either through the Xgrid preference pane, or through the command line by running:

```
sudo /Library/Xgrid/Scripts/agent_on
```

```
sudo /Library/Xgrid/Scripts/agent_start
```

The agents can either bind to the first available controller through Rendezvous, or to a specific IP address. In this case it will just bind to the first available controller since there is only one controller running. It is worthwhile to note here that sometimes the controller crashes, and I have found that specifying the controllers' IP address to be the only way for the agents to reconnect once the controller is restarted. For exact details on how to start the controller or agent through the GUI see Appendix 2.

Upon getting the grid started I ran through some sample jobs such as Mandelbrot (which computes complex fractals) and I ran the calendar job (/usr/bin/cal) that has been already discussed earlier in this paper. These methods are great for testing preliminary connectivity of the grid, and for establishing that all the OS X agents can in fact retrieve jobs.

I wanted to determine very early on that I could in fact render Pov-Ray files across the grid, and as such I realized that I needed to obtain a command line version of Pov-Ray that I could use on OS X. The reason for this is that for Xgrid to distribute jobs a set of command line parameters need to be passed to the agents to process, and if Pov-Ray cannot run via the command line, the agents cannot run Pov-Ray jobs. Through research found on both James Reynolds website (University of Utah), and Daniel Cote's website (University of Toronto) it was hinted that DarwinPorts needed to be installed in order to obtain a command-line version of Pov-Ray. DarwinPorts is a command line application that can fetch open-source software that is "ported" specifically to the PPC (Power PC) platform. A large amount of Linux software has been ported to OS X, and is available through DarwinPorts. For a more complete overview of DarwinPorts, and its installation, see Appendix 1. Once DarwinPorts was installed it was only a matter of running the following command in order to install Pov-Ray:

```
%port install Pov-Ray
```

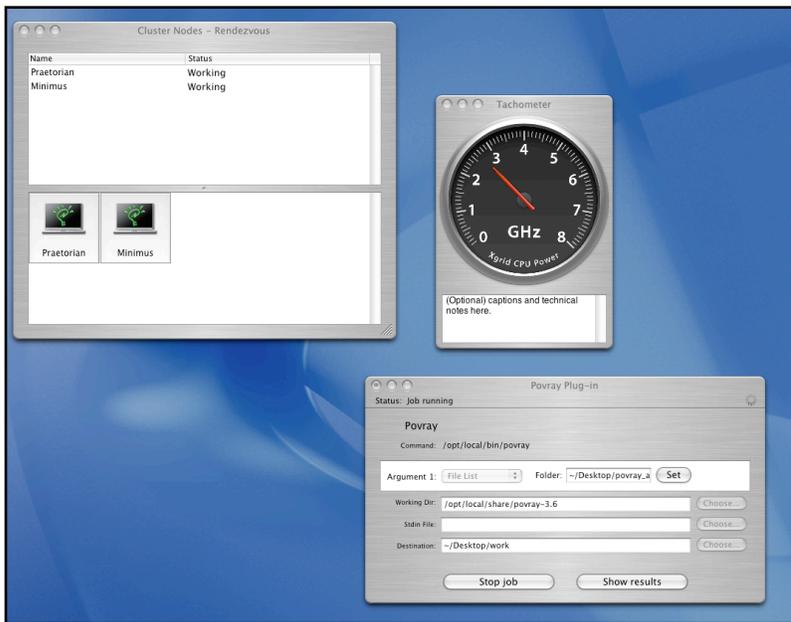


Fig. 16 - The submission of a custom Pov-Ray job

Now that Pov-Ray is installed I wanted to try rendering scenes on the homogenous Xgrid. To do this I created a custom plugin that utilized the command `/usr/local/bin/Pov-Ray` and took as an argument a folder of Pov-Ray files [Fig. 16]. These Pov-Ray files were just sample scenes that are included with any installa-

tion of Pov-Ray. The two systems then proceeded to render the scenes I specified, and everything was returned to the destination directory. I noticed at this point that the output included not only the Pov-Ray files and the text files of the command output, but also a copy of the working directory and the folder I specified in the arguments list. This is because a tarball of the entire working directory is returned to the controller, which the controller then untars to the destination folder. The discovery of what exactly the controller does, and how the agents respond was key to understanding how Xgrid could be implemented on Linux. Although it appears that setup and configuration of Xgrid on OS X is fairly straight forward it still took a good deal of time to install the tools necessary to install Pov-Ray and the setup of both systems and the network hardware took close to 10 hours. That being said, the ease of setting up Xgrid on OS X is certainly easy enough for any enthusiast to follow through with the help of a good manual. Next, this paper will look at taking Xgrid and implementing it in a cross platform manner.

Installing a Cross Platform Xgrid

The task of attempting to setup a cross platform Xgrid is very much a wild and unexplored frontier. It is wild simply because the agent that is available to Linux architectures is still very much in its experimental stages, and is still a prototype. The agent was developed by Daniel Cote (PhD) who works at the Biophotonics group at the Ontario Cancer Institute (University of Toronto) with help of the Xgrid team and Ernest Prabhakar at Apple. This agent is not supported by Apple or Mr. Cote, and is merely provided as semi-functional beta/proof of concept. The inspiration behind creating the agent for Unix architectures was so that researchers who do not have access to a large number of Apple computers could use their PC's to join the grid. The controller still must be an OS X based machine, however there is no limit to the number of Linux agents that can connect.

A very large portion of time in this project was allocated to getting Xgrid to run on Linux, and more importantly, getting the agents on Linux to run something useful without crashing. As the whole purpose of this project is to provide exposure on cross platform computing, the implementation of this agent is absolutely critical to the project.

The first step in this roller coaster ride, was to choose a Linux distribution that would match both the project's goals, while maximizing performance and speed. After doing some research the Linux distributions I selected were Ubuntu which is a Debian based distribution, and Fedora Core 3 which is Redhats' well known community project. Both of these distributions ran into problems meeting requirements for the installation of Xgrid, and as such a utility known as APT⁴ had to be used to retrieve the packages necessary in order to build the list of dependencies that Xgrid requires. For further details including the installation of these distributions, see Appendix 4.

Prior to this project I was very used to installing applications on Linux using either an rpm or a deb package, but did not have much experience compiling things manually using tarballs or gzips. Since the Xgrid agent largely required compiling packages by hand, this was a very interesting learning experience. Essentially there are three steps in install-

⁴ See Glossary [4]

ing any .tar or .tar.gz under Linux. The first step is to configure the build which is done through issuing the ./configure command. This will go through a script and determine whether or not you have all the necessary libraries and dependencies⁵ in order to build the application, and finally it will create a makefile. The next stage of the installation requires typing the make command, which executes the makefile, which actually compiles the source code into binary. Lastly, by typing the make install command the installation script will install the applications files to the system or to any directory you specify using the --PREFIX=/install/dir during the configuration phase. These steps needed to be completed on 3 essential packages. The first of these is the LibXML package which provides support for the XML language under Linux, the second was glib which provides necessary libraries for GCC compiling, and the third was RoadRunner which provides the communication framework for the linux Xgrid agent. Around 30 - 40 hours of the time working on the project was spent determining dependency requirements, rewriting configuration scripts, and building automated installation scripts. For a complete overview of the steps required in compiling the Linux Xgrid agent please see Appendix 5.

I managed to find instructions on how to setup the Xgrid Linux agent at Daniel Cote's website [22] and managed to get to the point where Xgrid would run, but the second that Xgrid would connect to the grid the agent would segfault and halt. I tried recompiling the libraries in a number of different ways, but every time I would presented with the same problem where it would segfault and halt. At this point I was completely stumped and the possibility of running Xgrid in a cross platform manner seemed fairly slim. I posted the segfault problems to the Xgrid user list, and was lucky enough to have Daniel Cote respond with some simple steps I could take to debug the application. After running through the output generated from a debug, I could not narrow the issue down further than finding that the agent seemed to keep crashing with a call to the XMLStrnCat function in the xgridagent.c file. Not having much experience in C, I posted a series of questions to the Xgrid user list, and with a few hints I was able to narrow the search down to one specific area in the code, which needed to be removed in order for the application to run properly. For more details on the specific steps taken in debugging the output, and the

⁵ See Glossary [5]

lines that needed removing from the `xgridagent.c` file see Appendix 6. Once this was done I created my own version of the Xgrid Linux agent that contained this patched file so that I would not have to manually apply the fix to each Linux system prior to running it.

Due to the fact that I wished to install the Linux agent on as many as 5 PC's I needed to come up with a way to install the agent quickly and efficiently without having the hassle of having to manually type every aspect of the installation process. I had begun to play around quite a bit with bash scripting at this point, and the power of this technology really began to become apparent as I thought about how I could write a script that would completely automate the task of installing the Xgrid Linux agent. Throughout the course of installing the agent on both Fedora and Ubuntu (Appendix 2) systems I had carefully documented the dependencies that were needed, and recorded all the steps required in setting the system up properly. This was then tied in with the installation of DarwinPorts and Pov-Ray (Appendix 1) to create a completely automated installation of every aspect of my project. To see the complete script see Appendix 8.

At this point the single Linux agent was tested to make sure it could in fact render Pov-Ray files, and a sample render of a simple Pov-Ray library scene was conducted by running:

```
povray -benchmark
```

Once I had the automated installation script fully functional on one computer, it was time to get the script tested on as many PC's as I could muster. A large amount of hardware was acquired for the purposes of this project, and the setup of this hardware is detailed in the next section.

Hardware Setup

Although the resources listed in the hardware resources section of the project justification section had all been acquired, much of the hardware had yet to be assembled and made functional. The following details the steps taken in assembling the systems for the grid:

- ▶ The 1.8 Ghz system had to be built from scratch. This involved installing the motherboard, processor, RAM, hard drive, CDROM, heatsink, and cabling. Fedora Core 3 was then installed on this system [See Appendix 4 for more details).
- ▶ The 200 Mhz system required a bit of a RAM upgrade as it only had 32Mb of EDO RAM at the start of the project, this was boosted to 96Mb to bring it up to a somewhat functional state. A customized Ubuntu installation was necessary on this system. This involved a minimal installation with no XWindows, and then only including the bare minimum set of packages that the agent would require in order to run. This machine during the course of this project was entirely utilized via the command line.
- ▶ The 2Ghz system did not need any additional hardware configuration. A few settings in the BIOS were tweaked to enhance the performance of the processor, but no notable additions were made. Ubuntu was installed on this system as well.
- ▶ The 400 Mhz system required a new hard-drive to be installed as the one that came with it failed mid-way into the project. Ubuntu was installed to this system.
- ▶ The 1.2Ghz iBook system required the addition of an extra 256Mb of RAM and the manual installation of OS X and was tailored to include DarwinPorts and Xgrid. (See Appendix 1 for more details)
- ▶ The 2Ghz PowerBook system was the main workhorse for the entire project, and was purchased 3 months before the project began. This system did not require any additional hardware, however the configuration of the system to obtain a static IP was necessary. The router leases this static IP to this system based on its MAC address. The airport (Apple's Wireless Card) was also required to obtain a static IP from the router through the same manner. This was done on both interfaces because this system was intended to be the primary controller. I did not want the controller changing its IP address mid-job, and after taking a look at the /var/log I noticed that the airport interface in particular goes up and

down quite a bit, and as such I wanted to ensure that it would always grab the same IP address.

- ▶ A 12 port 3Com SuperStack II switch was purchased from Ebay and arrived in early January. This switch has ten 10 base T ports, and two 10/100 ports. I had initially wished to get a 12 port switch with gigabit ethernet support but this was quite cost prohibitive. On arrival the switch was ripped apart and the two fans inside it were taken out as they produced around 50 decibels of noise which was far too loud for my environment. The switch was connected to the 5 Port DLink router through 20 feet of CAT5E network cable and was tested to make sure the ports were functional.
- ▶ 100 feet of Cat5E cable was purchased, along with 25 RJ45 jacks. From this I hand-made two 20 foot lengths of cross-over cable, and two 5 foot lengths of straight-through cable. The network structure is detailed below: [Fig. 17]



Fig. 17- The network structure

SSH and Remote Installation

The most time consuming process in setting up a grid was the installation of the software on all the machines connected. In my project I quickly realized that a large amount of time was being wasted trying to use 6 different keyboards, and 6 different mice to install software on each of the computers connected. When errors occurred it was necessary for me to move from computer to computer to look at the issues. This was both time inefficient, and often confusing as keyboards got confused, and commands were frequently typed into the wrong terminals. The solution for all these issues was the installation of SSH which permits remote secure shell access to any computer running the SSH daemon.

The SSH daemon is installed and enabled by default on any Fedora Core 3 machine. This is a bit of a security risk particularly if you don't have very strong passwords, but it does have the advantage of requiring no setup. On Ubuntu the SSH daemon is not installed by default and as such it must be downloaded using APT⁶. This can be done by typing:

```
sudo apt-get install openssh-server
```

Once APT has retrieved and installed the package, the daemon is auto-started and the computer can now be accessed via SSH. In my case I used the PowerBook to completely control the rest of the network remotely. This was done by opening up several terminal windows, and then typing "ssh" followed by the IP address of the computer I wished to control like so:

```
ssh 192.168.0.101
```

After a username and password have been entered, terminal access was granted and I began my script-automated installation of Xgrid, DarwinPorts, and Pov-Ray. See Appendix 8 for further details.

⁶ See Glossary [4]

Once this was completed I also needed to update the Pov-Ray libraries which needed to contain a custom library set that is not installed with the Pov-Ray default libraries. These custom libraries contain very particular information required in the rendering of my Pov-Ray scenes. The retrieval and installation of these libraries was done through another automated installation script. See Appendix 9 for further details on how this was done.

Cross Platform Problems and Solutions

From the very start of my experience with the Xgrid Linux agent I have been presented with problems. The first issue that has already been discussed was the challenge of actually getting the agent to run without segfaulting, but the second challenge proved to be an ever larger one. After submitting my first Pov-Ray job to Xgrid I found out that the controller would crash if anything more than 15k was uploaded from the agent after the job completed. After doing some research, I found that Mr. Cote had documented this issue on his website:

*“3. **Very important bug:** if the message sent by the agent is larger than 15k, it will hang. This is a problem due to my poor understanding of BEEP. See code `xgridagent-profile.c` for problem description in the function `xgridagent_SengMSG()`. This means that with the Custom plugin, if you generate data in the working directory and it is larger than 15k (tarred and zipped), the agent will hang. Most useful cases fall in that category. Fix the code if you know how, because I don't.”*

-Daniel Cote

This proved to be a massive obstacle because the text files that were being returned to the controller were often over 15k, and the output of the Pov-Ray renders always totaled more than a megabyte per job. I took a look at the `xgridagent-profile.c` file, but the level of complexity which is involved in the programming behind the agent would have taken far more time to correct than the scope of this project could allow. At this point I almost gave up hope, and began to research implementing a Condor grid and scrap the whole concept of Xgrid being able to work in a cross-platform manner. As I began researching Condor, I still kept persisting with questions on Xgrid as I was sure there was something I was missing, and it was during this period of questioning that I stumbled into finding a

solution. This solution not only makes it possible to run Xgrid in a Cross-Platform manner, but it works very well.

During the documentation of how Xgrid handles job submission, I discovered that jobs are run as the user “nobody” which in the Linux world is given read and write permissions to the /tmp directory, and can read anywhere that the world user is given access. The problem with my original method of job submission was that I was passing the controller a file list which it took, and then dolled out one file per agent, and passed the povray execution parameters. The screenshot below [Fig. 18] details how this was done:

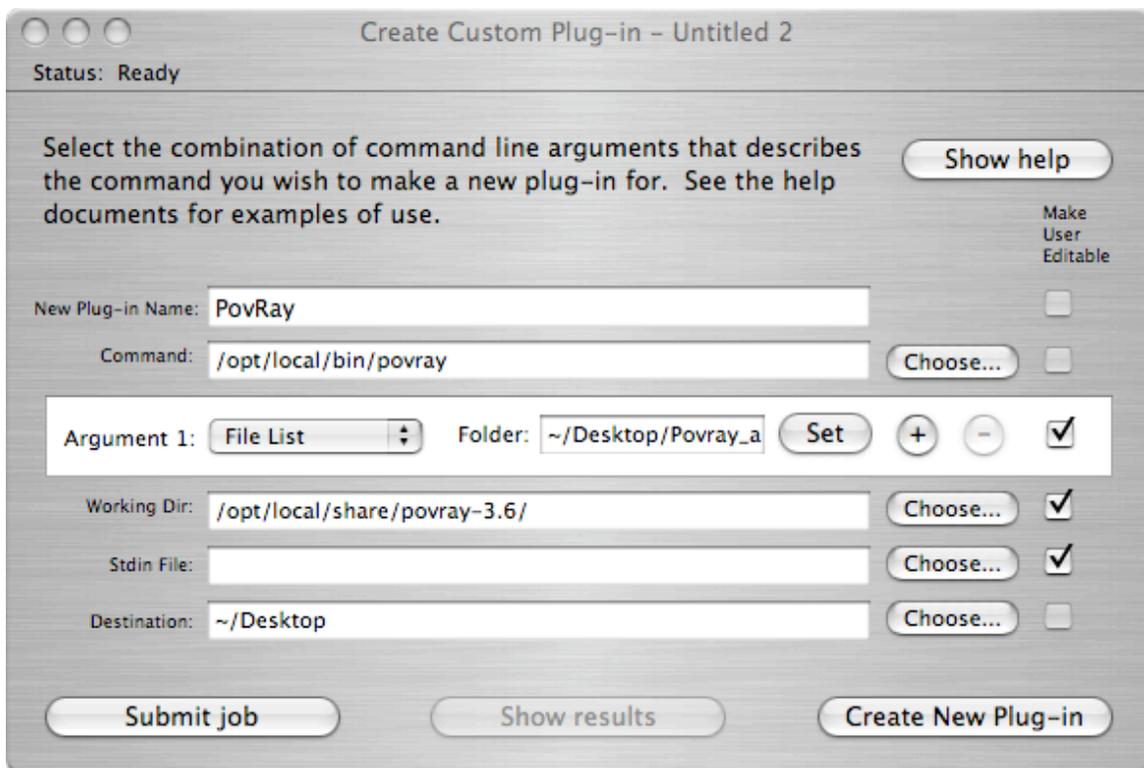


Fig. 18 - The submission of a Pov-Ray job taking a file list as an argument

The problem with this is that the agent takes the entire list of files, and tars them along with the contents of the working directory. This creates a tarball that can be upwards of a few hundred kilobytes in size. The agent then receives this tarball and extracts it to its own folder within /tmp. Once everything is finished rendering the final render, and the contents of the working directory in /tmp/123df on the agent are uploaded back to the controller. In the case of the Linux agent this will crash the controller every single time

the agent uploads the tarball back to the server. This method of submitting pov-ray jobs will however work on all the OS X platforms on the grid.

The solution I came up with to solve the 15k issue was to have the agents download a script from the controller that will force the agent to use a web-server for uploading and downloading, rather than the controller. The script will inform the agents to change its working directory, download everything it needs from my own customized Apache 2.0 web-server, and upload all the results back to my own PFTP server. Originally I had planned on including the Pov-Ray output in the STDOUT but the output for large scenes is often well over 15k so I piped the output to a different text file instead and had the script upload this to the FTP server as well. For more information on the script that was created see Appendix 10.

This solution solves the 15k problem because it forces the agent to work outside of its default working directory. Work is done within the /tmp/pov-script folder instead of the /tmp/123df folder which means that when the /tmp/123df folder is tarred and returned to the controller, the contents of this directory will not include anything. The only thing that is returned to the controller will be the results from the STDOUT which in my script will only be a small handful of echo and pwd commands. The resulting rendered graphics, and their output files are uploaded to my ftp server, which completely avoids utilizing the controller whatsoever so the end result is that nothing over 15k is ever returned to the controller.

Without in-depth knowledge of what exactly is going on behind the scenes in Xgrid this discovery would never have been possible, Through this discovery it is very probable that I am currently the only person in the entire world that is successfully using a number of Xgrid Linux agents as fully functional members of my grid.

Apache 2.0 and PFTP

During the time I spent trying to resolve the 15k upload issue I came upon some work that James Reynolds had done at the University of Utah involving using a web-server, and a series of CGI scripts to handle distribution of jobs amongst the grid. Although Rey-

nolds' scripts will only work in an OS X environment, it provided me with the insight of using a web-server in conjunction with an FTP server to avoid utilizing the controller for transferring files.

Initially I had utilized my own online-web-space (www.obelix.ca) which runs on a high end server down in the states, but once I began rendering complex Pov-Ray Scenes I realized quickly that using my slow ADSL connection to download and upload files to the server took far too much time away from processing as my ADSL connection only uploads at 60kb/sec, and downloads at 160kb/sec. My final Pov-Ray scenes totaled over 100Mb to download, and the resulting renders were around 250kb each. As a result running a web-server and a FTP server on the local area network seemed to be a much better solution as it would be far less bandwidth intensive and would allow each computer to fetch the next job faster.

Apache 2.0 was downloaded as part of an OpenWeb (<http://openosx.com/openweb>) package available to Mac OS X which included some great OpenSource utilities such as WebMin. The default installation of Apache was not sufficient for my needs, and I decided to re-write the HTTPD.conf file specifically for the use of this project (See Appendix 11 for Details). This was done largely for security reasons as the default setup of Apache is far to relaxed for my liking and I wanted to tailor the security settings specifically for local area network use.

A program called PFTP which is an FTP program for OS X was utilized in creating an FTP server which the Linux platforms could communicate properly with. Originally I had been using the default FTP server included in OS X but for some odd reason the Fedora Core 3 machine on the grid could not connect to the server. After some diagnosis it appeared that the Fedora Core 3 machine was running into authentication issues with some of the protocols being used on the FTP server. Below is the output of this problem to prove its existence:

```
220 localhost FTP server (tnftpd 20040810) ready.  
502 RFC 2228 authentication not implemented.  
502 RFC 2228 authentication not implemented.
```

```
KERBEROS_V4 rejected as an authentication type
```

```
Connected to 192.168.0.110.
```

At this point I tried diagnosing the problem but many fruitless searches revealed little about the nature of the issue. At this point I switched to PFTP and after configuring the allowed users and user directories, the Fedora Core 3 machine was able to upload to the server via the command line. A FTP upload script was created to handle this functionality (See Appendix 10 for more details).

BENCHMARKING AND TESTING

Pov-Ray

Once the grid was proven to be fully operational, the task of developing a complex Pov-Ray scene capable of demonstrating the power of grid computing became a central focus of the project. As stated in the project justification section of this paper, Pov-Ray is a high-quality, freely available ray-tracing software package that is available for PC, Macintosh and UNIX platforms. Pov-Ray allows a programmer to create complex graphics through the use of its own custom language. It employs a concept known as ray-tracing to do this which allows a programmer to create light rays. These rays custom programmed into specific scenes and can be refracted by mirrors, glass, or undergo various other contortions, all of which result in a single pixel of the final image. The reason that Pov-Ray has been chosen as the primary benchmark for the grid is because the speed at which Pov-Ray scenes render is almost exclusively tied to the processing power of the computer. Factors like RAM and GPU (Graphics Processor) speed play a very minimal role in the creation of Pov-Ray scenes. It is also worth while to note that learning Pov-Ray proved to be an infinitely more complex task that I had ever imagined. Documentation on the language is lacking in good examples, and the quality of the documentation when compared to other languages I am familiar with such as PHP or JAVA is certainly very poor. As a result countless hours were spent working on very simple scenes which I quickly realized did not provide nearly the level of complexity I needed. For example, my first scene I created was rendered in 1 second. Below is the code:

```
#include "colors.inc"  
camera {location <0,2,-3>look_at <0,1,0>}  
light_source {<6,6,-2> color White}  
plane {<0,1,0>0pigment {color Yellow}}  
sphere {<0,1,0>1pigment {color Red}finish {phong 1}}
```

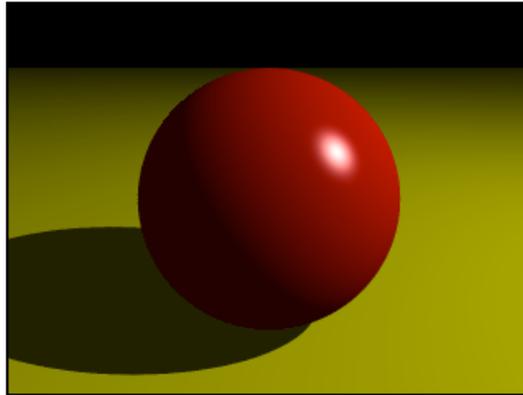


Fig. 19 - The first Pov-Ray rendered scene

I won't go into great detail here, but I will run through what each aspect of the code does. The first line includes the `color.inc` file which contains a list of predefined colors that can be applied to any surface using the `color` keyword. The second specifies the camera which is a point in space that determines what part of any object you are looking at. The location is specified in `x,y,z` coordinates. The light source is specified in the same way, and is used to define where the light is, so that Pov-Ray will know how to shadow the image. The last two lines define a horizontal plane with a yellow colour, and a red sphere centering on the origin. Also in the sphere line the `finish` keyword is used to identify the level of reflectivity that the object will have. As this paper's focus is on grid computing, more information of Pov-Ray will not be provided however more information can be found at Pov-Ray's website www.povray.org.

Several other scenes were created, but all required less than a minute to render. At this point I decided I needed to look at other options, and had seen Virtual Lego being used in combination with Pov-Ray to produce what seemed to be much more complex scenes. James Reynolds at the University of Utah uses his Xgrid precisely for this purpose. I spent a good week or two experimenting with Virtual Lego in a book I purchased which details how to use it with Pov-Ray, but ultimately I decided I wanted to incorporate something that had more real world value. The real

world value I had been seeking came in the form of an architectural CAD program called Sketchup.

Sketchup

I chanced upon Sketchup whilst meeting with a friend who is an architect, and as he explained the tedious chore of waiting for renders to finish using Vue D' Esprit (A 3D rendering program), I had an idea. Would it be possible to use Sketchup to design a complex scene, and then export that scene to Pov-Ray? This was something he was unfamiliar with, but I was sure that it must be possible, so I went to the Sketchup forums and began to research if this could be done. Plug-ins can be developed for Sketchup using the Ruby programming language, and it was in the Ruby forums that I found a utility called SU2Pov made by Didier Bur that converts Sketchup drawings to Pov-Ray scenes. I then began working steadily on Sketchup whilst experimenting with Pov-Ray plug-in. As I had never used any CAD software of any kind the learning curve was fairly steep, but Sketchup is very well designed, and once I had figured out a few key concepts and worked through the tutorials I began to get the hang of things really quickly. My first simple drawing [Fig. 20] was little more than a box which contained a few pieces of furniture, and some indoor lighting components. These lighting components export the light source into Pov-Ray, and without them the scene would come out completely black (As can be seen in one of the windows in the above screenshot). Once the model is exported, it can then be rendered in Pov-Ray either through the Mac OS X GUI client, or through the command line using:

```
/opt/local/share/povray-3.6/povray room.pov +W1024 +H768
```

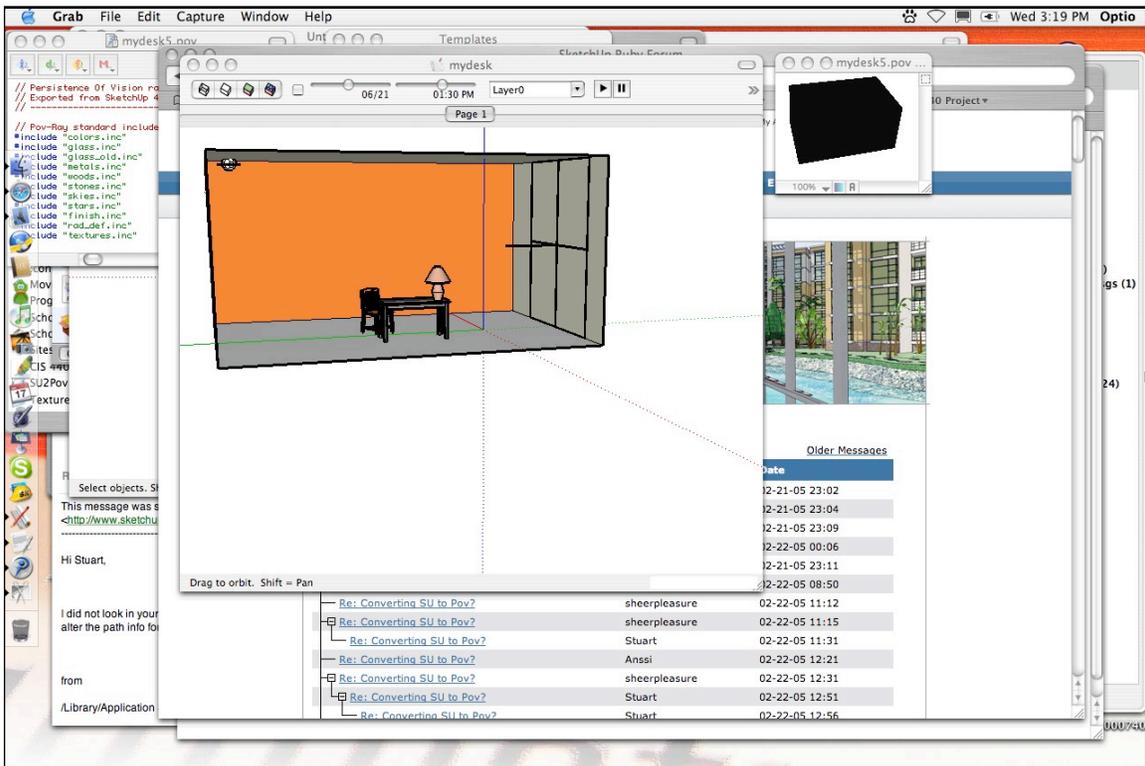


Fig. 20 - The first 3D architectural drawing created using Sketchup for OS X



Fig. 21 - The room.pov render

In this command the scene “room.pov” is rendered at 1024 x 768. The initial test was very spectacular considering the time it would have taken to develop the scene using Pov-Ray alone. To the left is the resulting scene [Fig. 21]. (Note that the lamp was removed and a plant added instead)

In terms of learning time I be willing to place at least 100 hours on the learning of Sketchup and the SU2PoV alone. Sketchup is an immensely powerful program, and documenting what I did within it is far outside the scope of this project. I found the challenge of creating a scene that would take a computer several hours to render to be immensely challenging, and in the latter portions of the project

the scene I developed became so huge that in order to work on it I had to take a trip to Victoria to utilize a friend's Dual 1.2 GHz PowerMac in order to complete the drawing. Below is a photograph [Fig. 22] of the development environment for my final Sketchup scene:



Fig. 22 - The author hard at work on a 3D architectural drawing, in the background is a Dual Processor PowerMac which was used in exploding the final drawings.

Developing the final scene took almost 2 days of solid work, but the result was astounding. On the following page is a wire-frame view of the model [Fig. 23] in Sketchup that was created for the benchmarking of the grid. This model incorporated many aspects of Sketchup and demonstrates a large number of its components and capabilities.

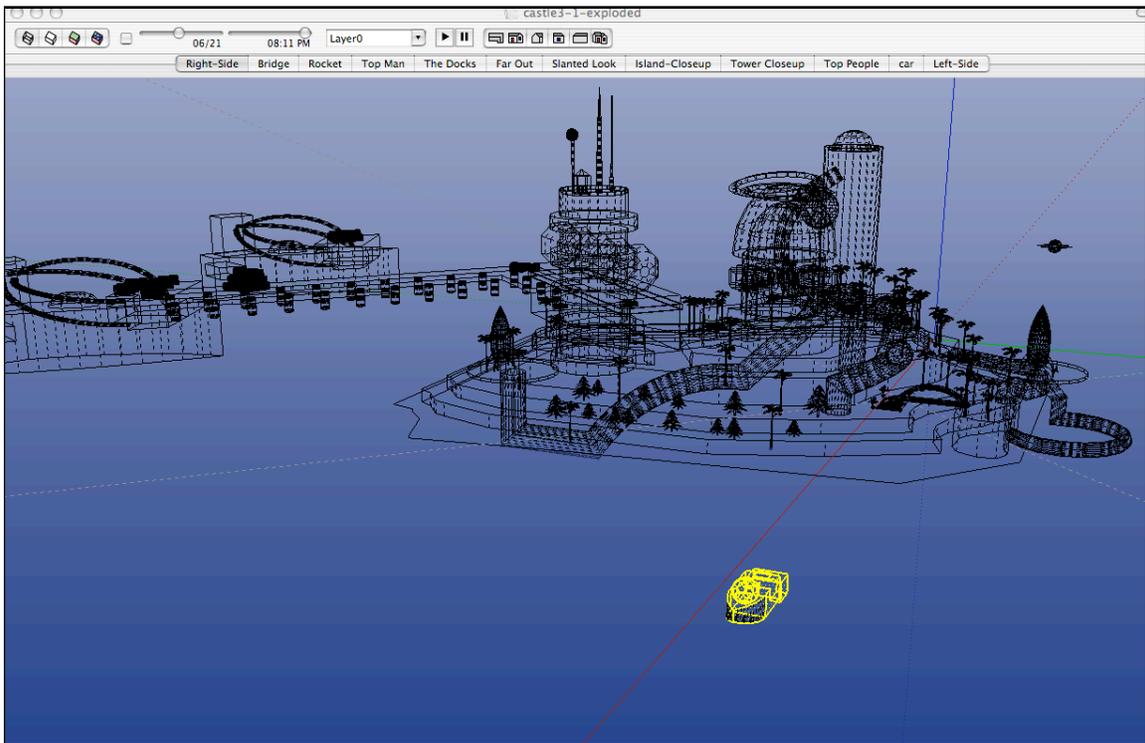


Fig. 23 - A wireframe outline of the final 3D architectural drawing.

At 1024 x 768 it took my PowerBook 3 hours and 14 minutes to render the scene. The reason the model took so long to render is due to a number of factors.

- The level of reflectivity I demanded required a mirror-like polish on every scene
- Radiosity-Final was used which increases particle effects and soft shadowing
- Realistic water and sky effects were employed. This was done manually at first, but in a later revision of SU2Pov this became integrated with the plug-in. Some of my own custom programming was still left in
- Pov-Ray surfaces were used. Some of these are visible in the final render in the form of metallic finishes, windows, rock finishes, and spiraling.

The design of the final scene was based around the need for a very random drawing that employed the use of curved surfaces and complex shadowing. Other scenes were developed with more of an architectural look and feel, but this scene proved to be the ultimate benchmark. One of the actual scenes is depicted on the following page.



Fig. 24 - One of the many final Pov-Ray renders from the architectural drawing.

After the final of this scene I was now ready to fully test the grid and see what could be accomplished.

Results

Now that a sizable Pov-Ray scene had been produced, the hardware has been setup, and the Linux agents are capable of running on the grid without crashing, it is time to benchmark the grid. As written in the project proposal the purpose of this project is to prove that employing a grid to distribute renders in a cross platform manner is not only feasible, but it is also much faster than stand-alone rendering. To make the interpretation of these results easier, names will be assigned to each of the computers connected to the network. These names of each of these machines are listed below:

- Centurion (2.5GHz AMD Barton - Ubuntu)

- Legionaire (1.8GHz AMD Athlon -Fedora Core 3)
- Praetorian (1.5GHz PowerPC - OS X)
- Minimus (1.2 GHz PowerPC - OS X)
- Protus (400 MHz Pentium II - Ubuntu)
- Percival (200 MHz Pentium I - Ubuntu)

The first benchmark which was conducted required rendering 10 identical Pov-Ray scenes at 640x480 with anti-aliasing, on a single 1.5Ghz PowerBook G4 machine. This scene was not very intensive, and only took a short time to complete. The results of these renders are depicted in the graph [Fig. 25] below:

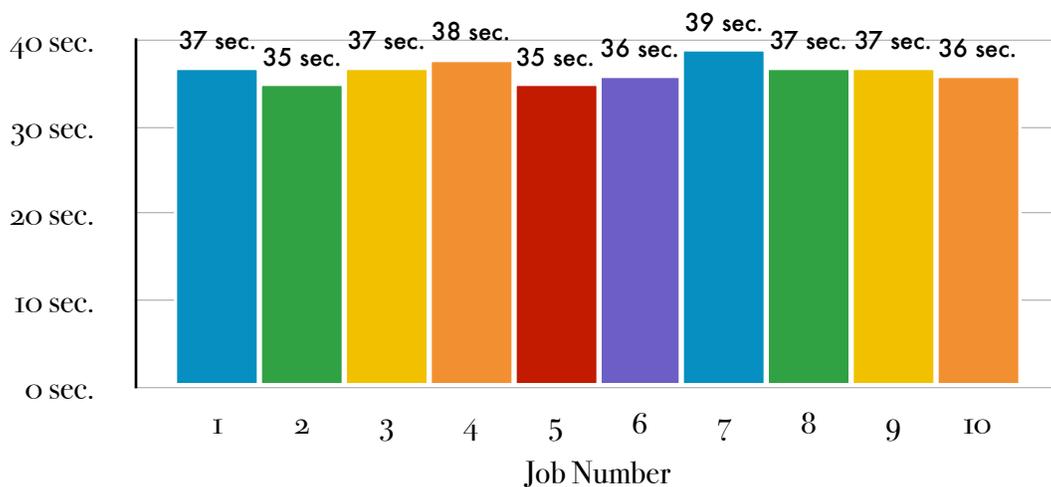


Fig. 25 - The time to render 10 identical scenes on a single 1.5GHz machine

The average time it took the computer to render these scenes was 36.7 seconds. The total time required in rendering the scenes was 367 seconds or 6.12 minutes. The rendered scene [Fig. 26] is depicted on the following page.

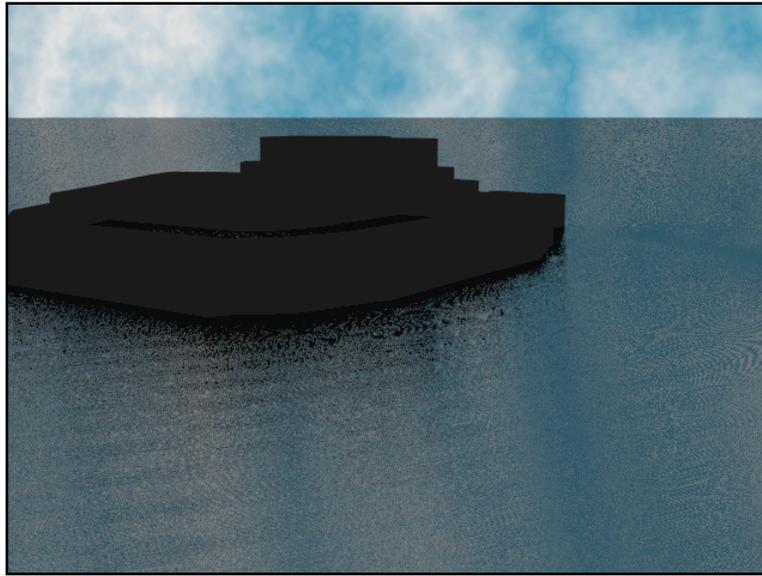


Fig. 26 - The scene used for the 10 identical renders

The second benchmark also rendered the same 10 identical Pov-Ray scenes at 640x480 with antialiasing, but this time the grid was employed in distributing the renders. The results of this benchmark are depicted below [Fig. 27]:

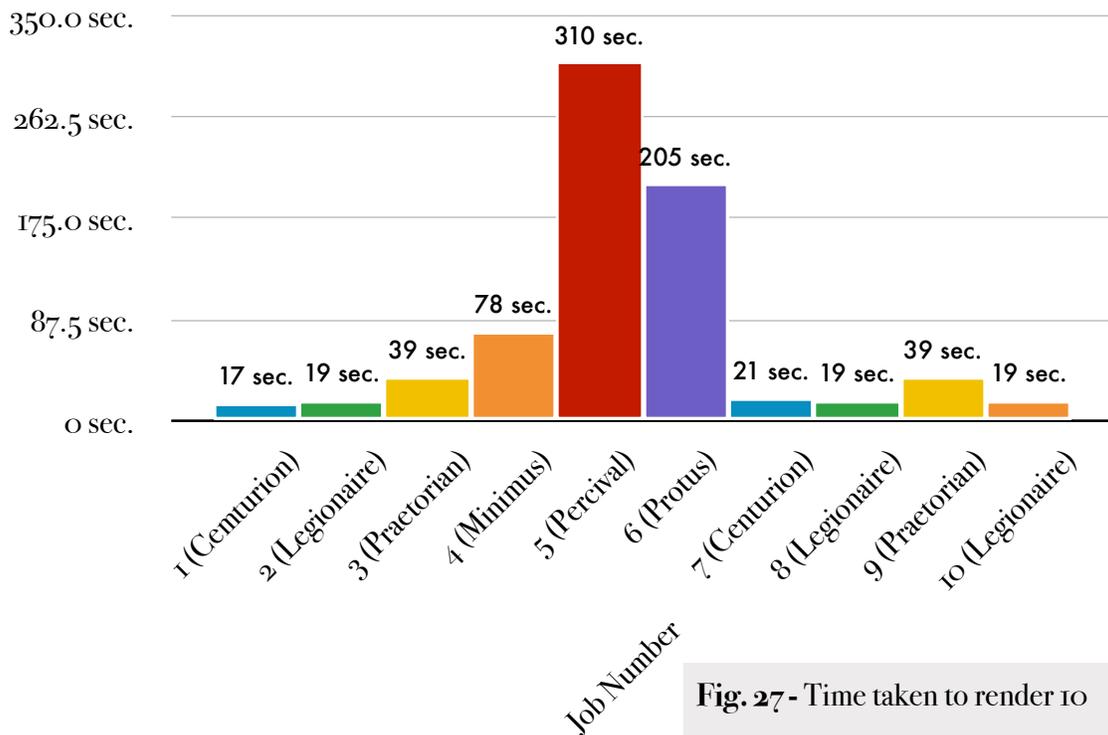


Fig. 27 - Time taken to render 10 identical scenes using 6 computers

These results are difficult to interpret because if we take the total time value we would find that it took the grid 766 seconds to complete the task. This is incorrect however because some of these jobs were run concurrently. In the case of Centurion, Legionaire, and Praetorian, each computer finished its first job, and grabbed the next. In Legionaire's case it grabbed 3 jobs because it finished its second job slightly before Centurion finished. Factors such as network latency and delay also come into play here as I had expected Centurion to grab 3 jobs. In the case of this graph however, the total time for the entire task to complete was 310 seconds (5.17 minutes) which was the time it took Percival to complete rendering. Percival was rendering for 105 seconds past the next slowest system (Protus) and completed well after the rest of the computers were long done rendering. At this point I realized that having the slow systems as functional members of my grid did more damage than it did use. Even on small jobs such as these they completely bogged down grid performance. That being said the grid still completed the task 57 seconds faster than it took the 1.5Ghz Praetorian to complete on its own.

At this point I was interested to see what would happen if I took the slowest computers out of the Grid and allow only Centurion, Legionaire, and Praetorian to participate. The results [Fig. 28] on the following page detail this discovery.

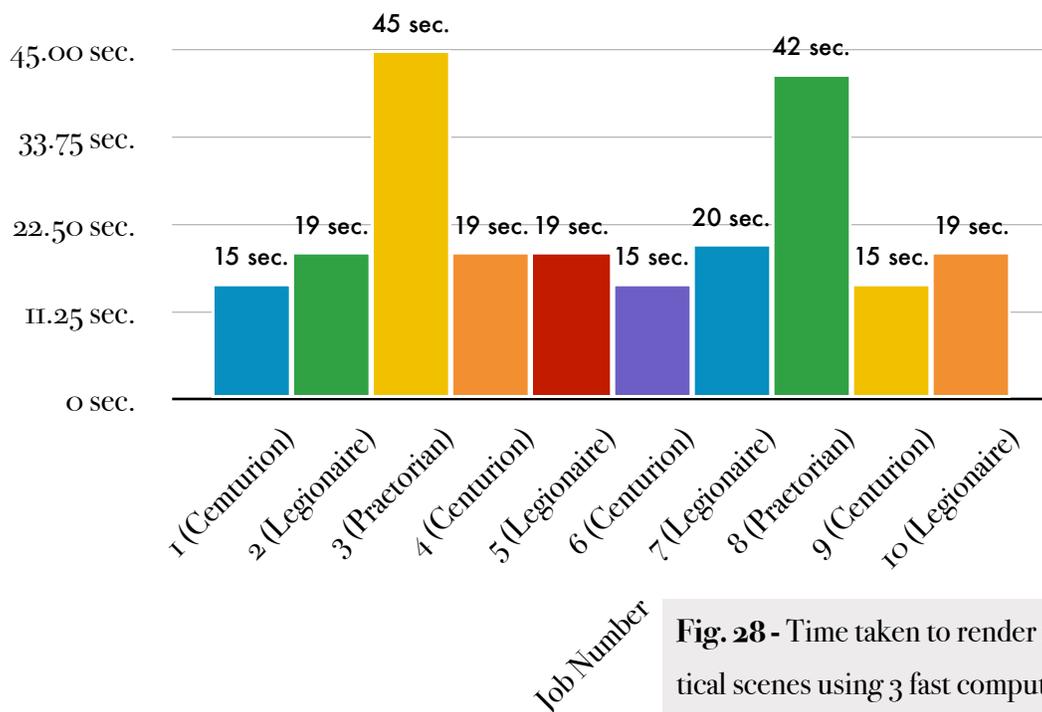


Fig. 28 - Time taken to render 10 identical scenes using 3 fast computers

These numbers are quite promising as we have cut down massively on the time to complete the job. The task was started at 11:29:20 by Centurion, and finished at 11:30:40 with Praetorian. Below is a screenshot [Fig. 29] of Praetorian finishing the task. The total time to finish the job was 80 seconds (1.33 minutes) which is nearly 5 times faster than the initial benchmark. This proves that slower machines on a HPC network are more of a hinderance than a use in most cases.

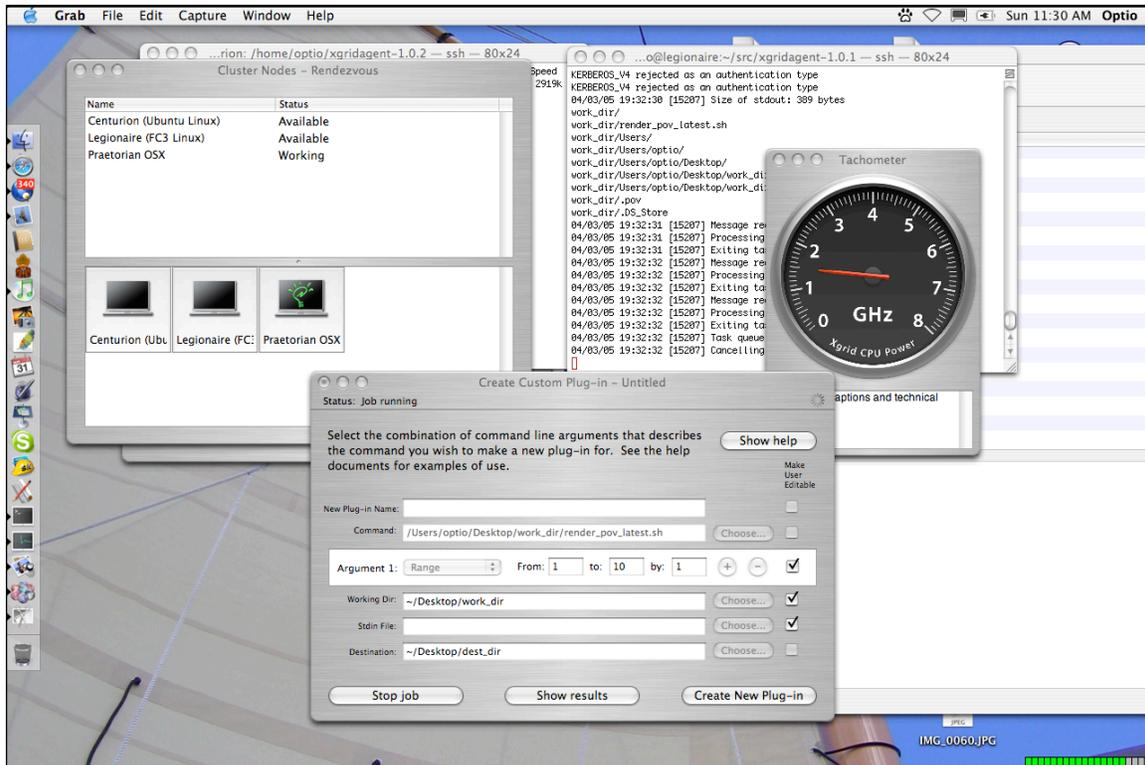


Fig. 29 - A screenshot of the 1.5GHz machine finishing the last task in the Pov-Ray render

The chart on the following page depicts the final results for the low quality rendering times. [Fig. 30]

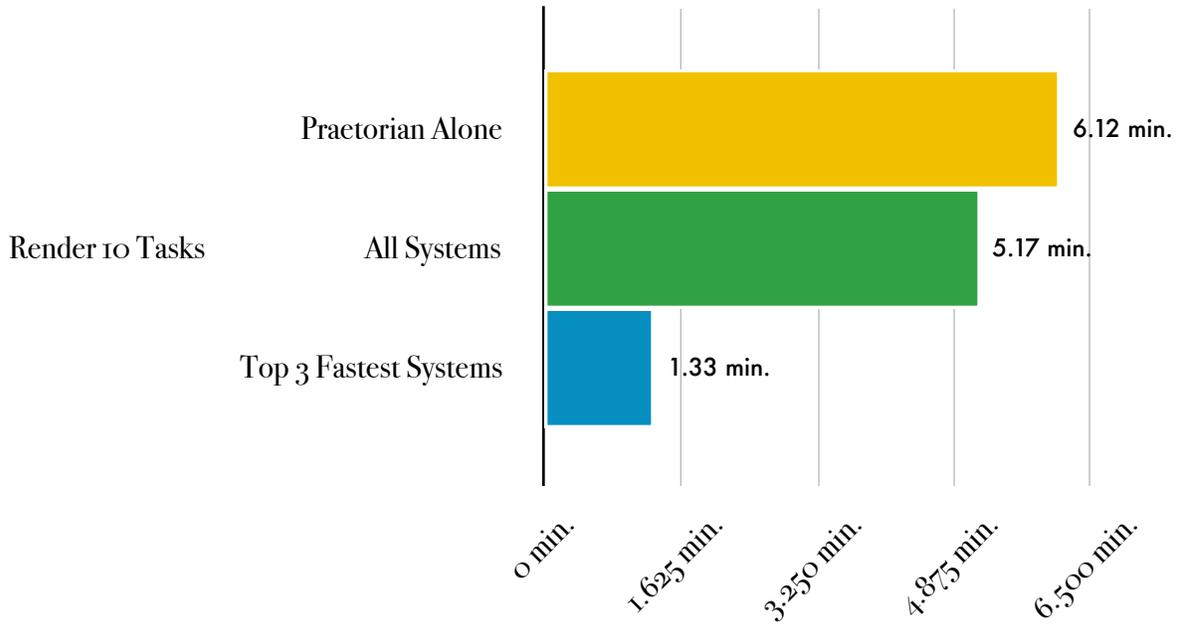
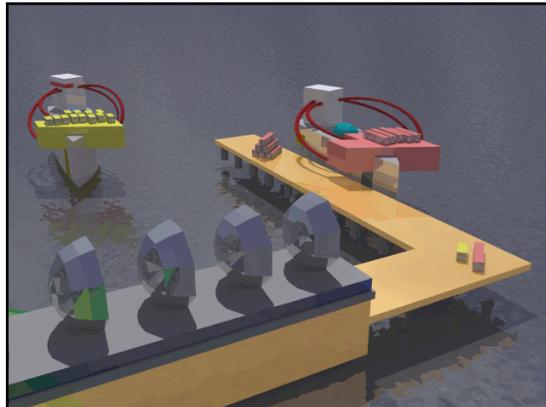
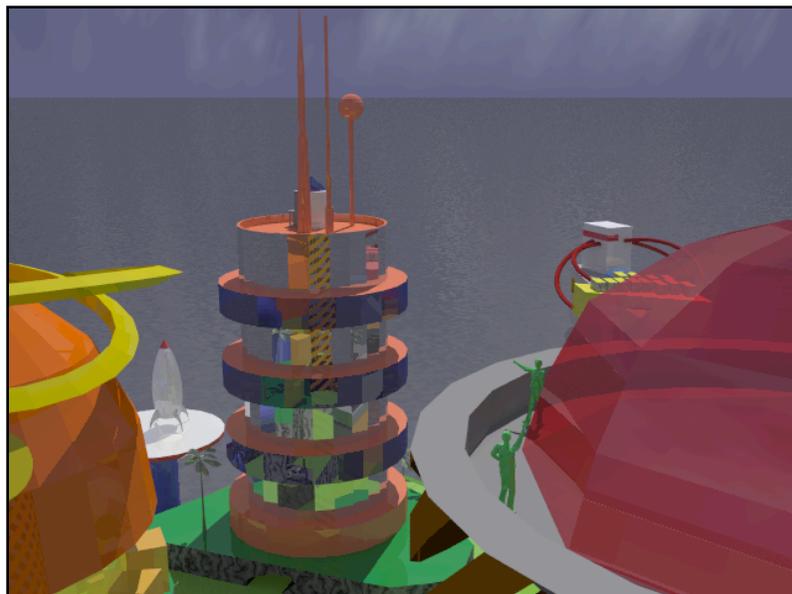
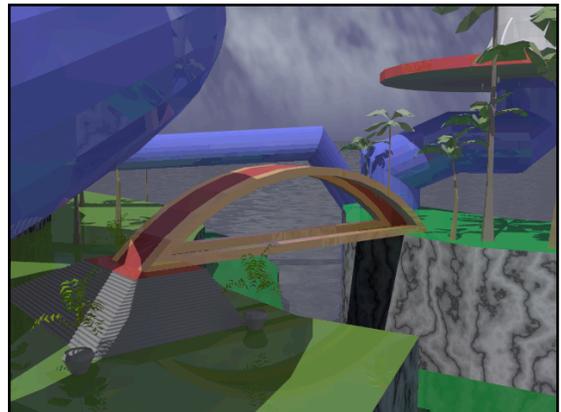
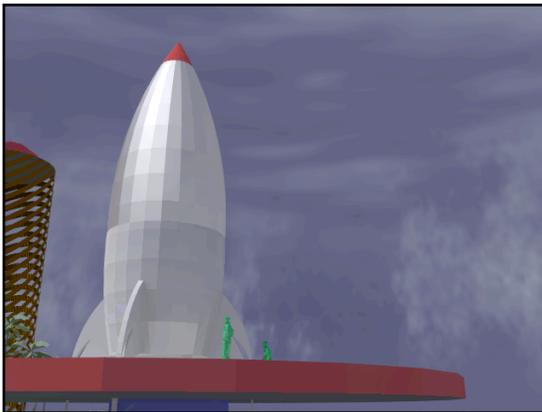
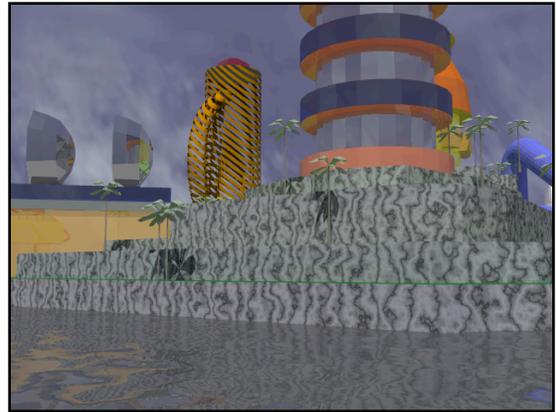
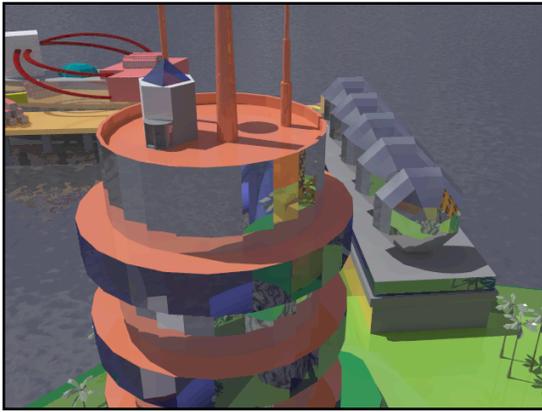


Fig. 30 - The final result depicting the times taken to render 10 identical scenes

The first major test of the benchmark was using our final Pov-Ray scene on Praetorian alone. 7 scenes from different perspectives of my model were used to obtain this benchmark. These scenes are depicted below:





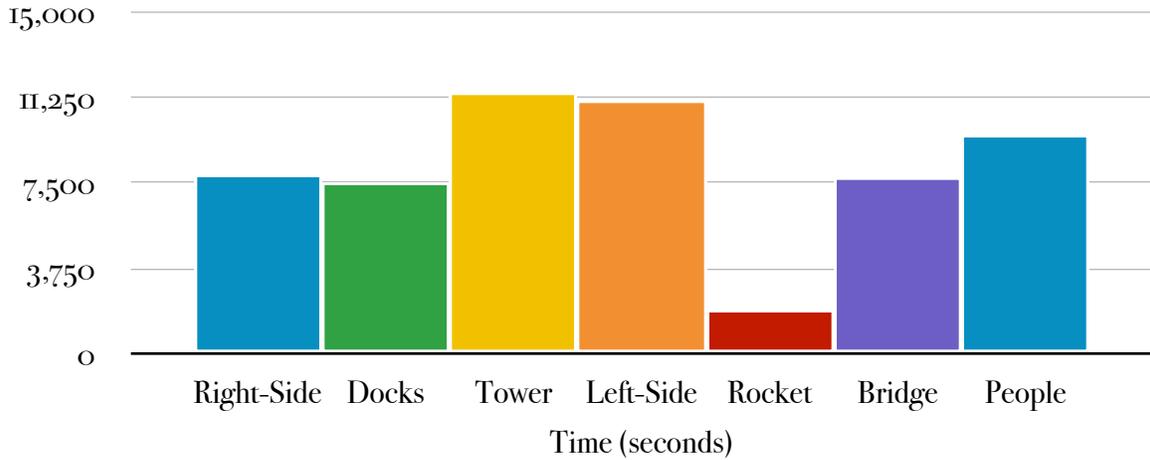


Fig. 31- The time taken to render 7 scenes on a 1.5GHz machine

The total time required to process all the scenes in our benchmark series took 57573 seconds or 959.55 minutes or 15.99 hours on our single PowerBook G4 [Fig. 31]. However, under our grid performance greatly increased [Fig. 32].

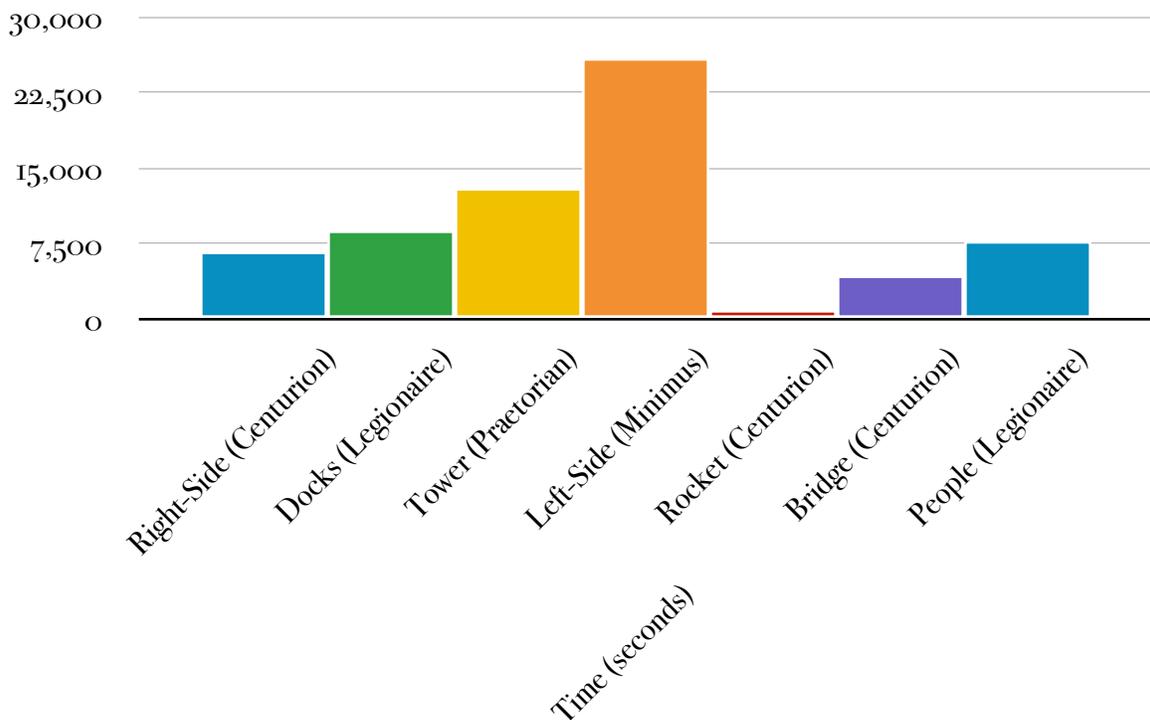


Fig. 32- The time taken to render 7 scenes on 4 computers

Xgrid definitely appears to deal excellently with job scheduling, and as we can see here Centurion takes 3 jobs since it completed the rocket scene in a very short period of time. Praetorian never got a chance at a second job since Legionaire took the remaining job because it finished first. The time to complete the job was the time for Minimus to complete the Left-Side render, as all the other machines were finished well before Minimus completed, which in this case was 7.20 hours (25929 seconds total). So as we can see here already we have witnessed more than a 50% reduction in the time taken to complete renders. This result is very impressive and I felt that this accurately demonstrated the power of using a cross-platform Xgrid. Below is a final comparison of the results of this discovery.

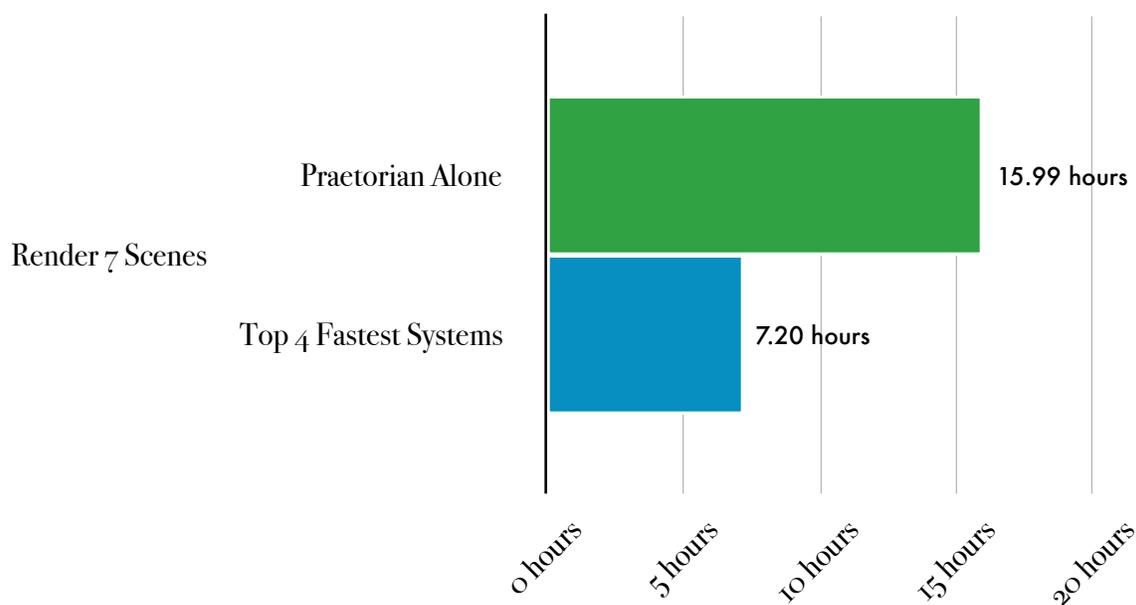


Fig. 33- A comparison of standalone vs. Xgrid benchmark times on a set of 7 scenes.

CGI, Blossom, and RSS for Displaying Xgrid Output

In order to better display the output of my Pov-Ray renders I utilized open-source blogging software called Blossom to take the results of my pov-ray text output, and display it

on a website. Bloxom runs using a combination of HTML, RSS and CGI, and was the primary reason behind creating a CGI-Bin on the Apache web-server. The main goal behind this concept is that it permits an administrator to view the entire output of a submitted job without having to be anywhere near the grid. With this kind of technology, a grid administrator can leave the grid running at work, and when he returns home he can monitor the output of the grid very easily via a browser, or through an RSS reader.

In the submit script I created (See Appendix 10) text files are uploaded to a directory within my Apache web server which Bloxom then time-stamps and posts to my blog. Further details are not provided as this is an additional feature outside the scope of this report. Setting up Bloxom and creating the website it ran on took approximately 5 hours. Below is a screenshot of the site:

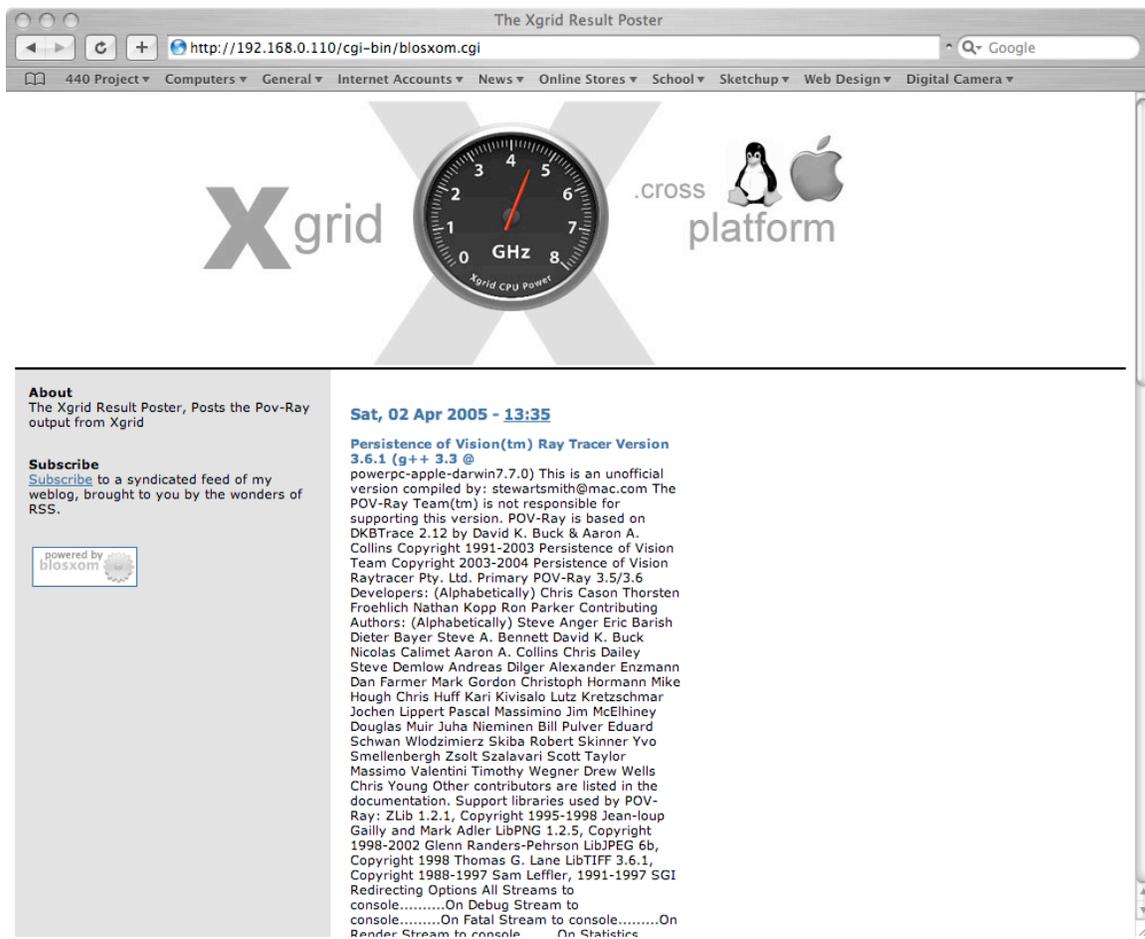


Fig. 34 - The blog used in displaying the Pov-Ray output files

CONCLUSIONS

The Future of Xgrid

Xgrid Preview certainly appears to be a step in the right direction, and for the purposes of this project, has certainly proved its worth. With the upcoming release of *OS X Tiger*, Xgrid will reach a 1.0 release, and is expected to contain many fixes and new features. Some of these features are listed below:

- Supports up to 10,000 queued jobs, 10,000 tasks per job, and 2GB of data submitted per job, 1GB of data per task, and over 10 GB of total returned results.
- The agent will be included as bundled software on both desktops and servers
- In addition to the command - line client, there will be a new Cocoa API for monitoring nodes and submitting Xgrid jobs.
- The controller will be integrated directly into the OS X 10.4 Server Admin Panel via a new Xgrid Admin panel. Also a lightweight controller will be introduced for desktop versions.
- Mutual authentication for both the client to controller, and client to agent.
- Email notification of tasks completed
- Single sign-on using Kerberos and Open Directory Services.
- A new software developer's kit to allow individuals or organization to create their own applications and custom plug-ins for Xgrid.

The primary goal of Xgrid 1.0 is to provide scientists and researchers with all the tools they need to easily build a super-powered computational cluster, but the uses for this technology are boundless.

One of the most interesting projects relating to Xgrid that is being developed is some

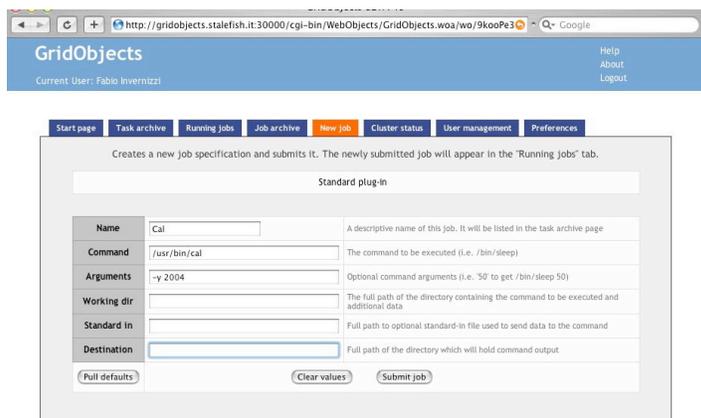


Fig. 35 - GridObjects is a web-based Xgrid management system.

web-based software called *GridObjects*. GridObjects is an application in the works by Fabio Invernizzi. This software has not been released yet, but several screenshots have been provided and it looks as though it is a complete web based administration utility for the remote administration, monitoring, and submission of Xgrid jobs. More

screenshots of this software can be viewed at:

<http://www.stalefish.it/GridObjects/GridObjects.html>

Xgrid certainly appears as though it has a promising future, and the fact that people are already developing software for it in anticipation of its 1.0 release is only fuel to the fire that it is likely to be quite successful.

Recommendations

Considering that the UCFV CIS department has 3 laboratories each with an estimated 25 computers in each lab, and each with a 2GHz processor, the total power available to the CIS department alone exceeds 150GHz. This grid could be found especially useful for students or professors in the chemistry, physics, engineering or mathematics faculties who wish to experiment with compute-intensive theories, protein modeling, genome research, or architectural rendering. but this is not the only use which the grid could be used for. UCFV could lead British Columbia in offering the first ever grid computing course where students would learn about grid technology in both theory and application. The labs could be used as a permanent training ground for students who wish to experiment in a technology which is destined to be one of the most hottest sectors in the technology industry. UCFV could also provide services such as an “open-grid” concept whereby staff

from other Universities could be invited to UCFV to run their simulations or calculations. This could not only increase UCFV's prestige in the academic community, but it could also increase the research output of countless projects, and could potentially provide UCFV with further funding from the government or corporations who are willing to invest in the technology.

The power and ease of use that Xgrid brings to the grid computing scene is certainly miles ahead of the competition, and with the expected release of Xgrid 1.0 in Tiger, things will only get better. One of the biggest selling features of Xgrid in my opinion is the ease of the setup and submission of jobs to the grid. This is particularly important to science students or professors who may not have experience in computing but want to distribute their work. These individuals will find this far easier to do on an Xgrid than any other grid technology available on the market today.

For UCFV to implement an Xgrid, a few Apple systems would of course need to be purchased for the controller to reside on. I would recommend a series of PowerMac G5's for this task (The same systems that ran on Virginia Tech's Super Computer), as their dual CPU architecture is perfect for handling large volumes of compute intensive related work. PowerMac's are very reasonably priced at \$1899 but with UCFV's educational discount (All canadian universities are eligible) the starting price of these systems begins at \$1699 per system. A number of these systems could be purchased depending on the manner in which UCFV would like to distribute the grid controllers. If funding is not available for systems of this calibre, Apple has also recently introduced a product called the Mac Mini which is priced at \$599 with the educational discount. The Mac Mini is a tiny machine with only a 1.4 GHz G4 processor, however for the purposes of being a controller it would be adequate. If UCFV were to order these systems in May, OS X Tiger will be included in the package. Although OS X Tiger will have a lightweight controller on non-server versions of the OS it is recommended that UCFV purchase OS X Tiger Server as further Xgrid functionality will be available on the server version only. OS X Tiger Server is priced at \$599 Canadian with the educational discount.

The main bulk of the grid's computing power would be provided by the systems already existing in our labs. Using my automated installation scripts, setting up Xgrid on either Fedora or Ubuntu is a breeze, and within a day or two, UCFV's computers could be connected to the grid. There are still a fair number of flaws in the Linux agent, but we have a number of very talented programmers on staff who could work on increasing the stability of the Linux agent. In this quest I am sure both Apple and the aforementioned Daniel Cote would be very willing to provide assistance or answer technical questions if serious development effort was put into the software. This could put UCFV on the roadmap of a very cutting - edge field, and if UCFV were to produce a fully functional Linux agent, there would be a large number of academic institutions across the globe would be very interested in it.

If UCFV decides that it does not want to look at implementing Xgrid and is disinterested in Apple technology, another option is Condor. Condor has the advantage of being truly cross platform since it can run on OS X, Linux, and Windows, but it is not easy to setup, nor is it easy to submit tasks to. A large amount of research was done on Condor during the time this report was produced but it has not been included in this report due to time constraints. UCFV could definitely implement a Condor grid without having to purchase any further hardware, but the time and money required in setup, trouble shooting, and research may be worth far more than it would cost to purchase Apple hardware.

The future is now, and the time to capitalize on grid computing has come. UCFV cannot afford to sit and wait as this technology provides an opportunity to establish ourselves as a research institution. Virginia Tech has done it, SFU is doing it, and UCFV can do it too. Grid computing is a field which is rapidly expanding and experience in this field would certainly be an asset to any individual graduating from our CIS program, or any of the science programs in general.

Final Conclusion

This paper has taken a in-depth look at the field of grid computing, some of its history, its terminology, and its practical application. Details have been provided as to the current

state of grid computing and the paper has outlined information on future products expected to be introduced to the field by companies such as Sun Microsystems, IBM, Microsoft, Globus, and Apple.

In order to demonstrate how grid computing can be used in everyday activities on a variety of operating systems, this paper has examined how such technology can be used to benefit modern architects. A combination of Apple's *Xgrid* technology, *Pov-Ray*, and *Sketchup* was used to accomplish this. Sketchup, a popular 3d CAD program, was used in conjunction with light ray rendering software called Pov-Ray to produce a series of environments which the grid (Xgrid) computed. Each of these environments was intensive enough for the grid to spend considerable time processing the scenes. During the low quality benchmarks, it was observed that adding slower machines to the grid only marginally increased performance by 18% but by removing these systems from the grid and only using systems of similar speed it was found that the grid could increase performance by as much as 460%. The high quality benchmarks that were used, found that grid-computing can enhance computational performance by more than 50% on the graphical scenes, taking 7.20 hours to render 7 scenes that took a single computer 15.99 hours to render. These scenes were rendered over a cross-platform Xgrid running on a mixture of Fedora Core 3, Ubuntu, and Apple OS X operating systems. Through these results it is possible to see the application of this technology for both home users and research universities.

The road to creating a cross-platform Xgrid has not been easy, and has witnessed many bumps along the way. The scripts and products which are detailed in this paper's research will make future installations and uses of this technology infinitely easier. The automated setup script for Linux in particular allows for any administrator to quickly and easily distribute the Xgrid linux agent among many computers, and is one of the primary products of this paper.

The purpose behind this research paper was to provide a detailed overview of grid computing, its benefits, and how it is being used today and has shown how this technology can be used in real life cross platform situations. Through this research it is easy to see how

this technology could be utilized in a university setting. Since any shell task or command line parameter can be executed as a job over the network, the possibilities of this technology are almost limitless in their application. Researchers at universities can use technologies such as PERL or MPI in writing programs that will help automate compute intensive tasks, and by using some of the examples provided in this paper can see how it is possible to distribute their projects in a cross platform manner. It is hoped that through the information and documentation provided within this project that students and staff at UCFV, and other universities around the world will not only be able to create their own cross platform Xgrid networks, but that they will also see the vast range of opportunities that this technology can bring to their research.

GLOSSARY

1. Floating point operations are when you take two numbers that might not be integers (i.e. 5.4 or -1.1) and add / subtract / multiply / divide them. The reason it comes up in assessing the performance of computing is that there are some kinds of computing problems, like doing permutations and combinations, that mostly involve integer math, and some that involve mostly floating math (like calculating distances and angles and orientations in a 3D environment). So over time, computers came to do integer and floating point math in separate parts. But especially for high-performance scientific computing, it is the floating part that is the computer intensive part. So the number of floating point operations per second (flop) became the benchmark of speed.

2. Rendezvous - Apple's revolutionary network technology that creates instant IP based networks with no configuration required. For more information on Rendezvous, visit Apple's rendezvous website at: <http://www.apple.com/macosx/features/rendezvous/>

3. BEEP - BEEP is a framework for writing application protocols that are message-oriented, which may, or may not, use XML. Further information on BEEP can be found at <http://www.bEEP.org>

4. APT- An advanced packaging tool that creates a rapid, practical, and efficient way to install packages, manage dependencies automatically, and take care of their configuration files while upgrading. APT is built into all Debian based distributions, and can also be installed on Fedora Core 3 through <http://apt.freshrpms.net/> . Below are a few of the commands you will see in use throughout this report:

- `apt-get update` : This command updates the apt package repository list, and gets the latest list of packages available from sources specified in the `/etc/apt/sources.list` file.
- `apt-get upgrade`: Searches sources for updated versions of every single package that is currently installed on the system. Once updates are found they are automatically downloaded and installed, old versions are removed

and the new versions are installed in their place. This is an excellent utility for keeping your operating system up to date.

- `apt-get dist-upgrade`: This method can be used to upgrade an entire installation system at once. This will include all kernel upgrades and other upgrades to do with the system that `apt-get upgrade` does not download or install. APT will download these packages automatically and replace the old kernel version with the new one. Kernel recompiles are a thing of the past with this excellent utility.

5. Dependencies - During the installation of Linux applications or packages, certain packages often “depend” on the presence of other libraries or applications. Usually if things fail within the configure script it is because you do not have the elements necessary to compile the application. At this point you can scroll up through the print out and determine what exactly is missing from the requirements. In some cases this is much easier said than done, and can require many hours of scouring the net to find out how to acquire the necessary dependencies to solve these issues. Sometimes even after installing a particular dependency the configure script will still fail. This can sometimes be solved by making sure the configure script knows where to find these dependencies. This can be done using the: `--with-application_name-includedir=/path/to/dependency_location`

Ex.

```
./configure  
--with-roadrunner-includedir=$HOME/roadrunner-1.0
```

APPENDIXES

Appendix 1 - Darwinports and Pov-Ray

Introduction

In order to install a command line version of Pov-Ray it is necessary for us to install a command-line utility called DarwinPorts. The DarwinPorts project goal is to provide a method for installing open-source software products on Darwin, Mac OS X, FreeBSD, or any Linux system. They have recently just finished successfully porting 2000 applications into their repositories and this list grows continually. In our case we will be installing Pov-Ray through the ports in order to obtain its command line functionality. By default the Pov-Ray package that is built for OS X is not available via the command line and as a result we would not be able to run the render through our grid.

After some research and many thanks to James Reynold's from the University of Utah which has published some great articles on using Xgrid and Pov-Ray together, I found DarwinPorts to be the only possible solution to my problem. It is important to note here that command line Pov-Ray is essential to our project, as when the Xgrid client issues the command to the Xgrid controller to run an executable called Pov-Ray with a bunch of parameters, all agents on the grid must have Pov-Ray installed in the same manner. If the agents do not have the Pov-Ray libraries and executables available in the same directories, the agents will fail to render anything that the controller passes to them. DarwinPorts is the only logical solution to this issue, as it will install Pov-Ray to the same directories with the same paths regardless of whether it is running on a Unix machine, and OS X machine, or a Linux machine.

Installing XCode Tools

Before installing DarwinPorts can be installed on OS X we need to acquire and install the XCode tools. We need these tools because they provide the GCC compiler, autoconf,

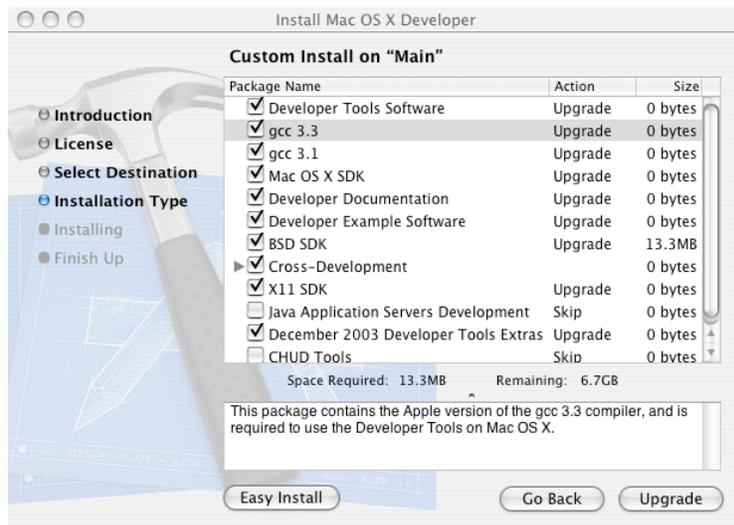
CVS, and other tools which make this all work. XCode tools are available to download from Apple's developer website here:

<http://developer.apple.com/tools/macosxtools.html>

Once the tools are downloaded, extracting the package and opening the Developer.mpkg will install the tools. Before installing make sure you customize the installation and ensure that the appropriate packages we need are selected. A screenshot on the following page will detail what packages we need to install. I checked off installing the cross-platform development packages as I wasn't sure what version of the GCC package DarwinPorts required and by choosing this option the 2.x version of GCC is installed. I also installed the X11 SDK as I was pretty sure that any applications that we download in the future that are cross platform may require some of the X11 libraries. It is also good to note that without the Developer Tools Software we will not be able to use CVS (Concurrent Versioning System) and since DarwinPorts is only provided via CVS this makes XCode Tools all the more necessary.

It is also worthwhile to note that installing DarwinPorts is made much easier by the presence of X11. X11 is the complete suite of the standard X11 display server software, client libraries and developer toolkit, and is built specifically for OS X.

A screenshot of the XCode Tools installer is below.



Installing DarwinPorts on OS X

In order to install DarwinPorts we must now open a terminal and login to the DarwinPorts CVS Server. This can be done by opening a bash shell and making sure we are in our home directory (cd ~). Then typing the following:

```
bash% cvs -d
:pserver:anonymous@anoncvs.opendarwin.org:/Volumes/src/cvs/od
login
```

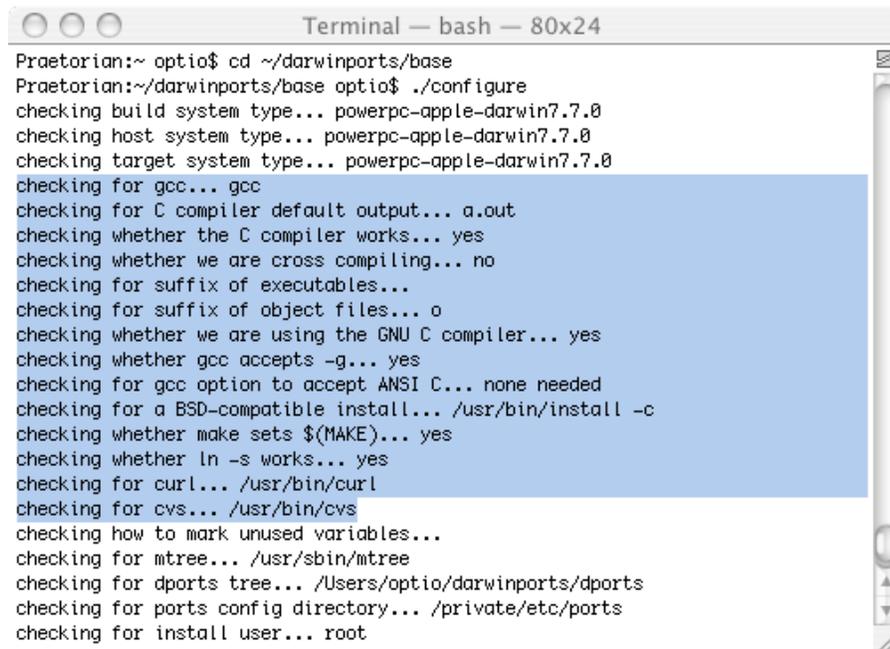
It will then prompts for a password but there isn't one set so just hit enter and continue. Now that we've logged in we can download the nightly CVS snapshot by entering the following command:

```
bash% cvs -d
:pserver:anonymous@anoncvs.opendarwin.org:/Volumes/src/cvs/od co
-P darwinports
```

Now it will retrieve all the necessary files from the CVS server. When it has finished it will return us to the console and we are now ready to begin the installation. First we will enter into the directory it downloaded to and then we will configure the installation.

```
bash% cd ~/darwinports/base
bash% ./configure
```

At this point it will run through the configure script and check to make sure we have all the necessary libraries in order to compile the application. Pay specific attention to the highlighted areas in the screenshot below as it will tell you whether you have all the necessary requirements to install DarwinPorts (Notably CVS, GCC, and Curl)

A terminal window titled "Terminal — bash — 80x24" showing the output of the ./configure script. The output lists various system checks and their results, such as "checking build system type... powerpc-apple-darwin7.7.0" and "checking for gcc... gcc".

```
Praetorian:~/optio$ cd ~/darwinports/base
Praetorian:~/darwinports/base optio$ ./configure
checking build system type... powerpc-apple-darwin7.7.0
checking host system type... powerpc-apple-darwin7.7.0
checking target system type... powerpc-apple-darwin7.7.0
checking for gcc... gcc
checking for C compiler default output... a.out
checking whether the C compiler works... yes
checking whether we are cross compiling... no
checking for suffix of executables...
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ANSI C... none needed
checking for a BSD-compatible install... /usr/bin/install -c
checking whether make sets $(MAKE)... yes
checking whether ln -s works... yes
checking for curl... /usr/bin/curl
checking for cvs... /usr/bin/cvs
checking how to mark unused variables...
checking for mtree... /usr/sbin/mtree
checking for dports tree... /Users/optio/darwinports/dports
checking for ports config directory... /private/etc/ports
checking for install user... root
```

Next we need to use the make command, and the make install command to install DarwinPorts. You will need superuser access to do this:

```
bash% make
bash% sudo make install
```

Now in order to be able to run the port executable without being in the directory in which it is installed we need to add the executable path to our .profile like so:

```
bash% vi ~/.profile
```

Now hit (shift-i) which will force vi to make an insertion. We will add this line to the file:

```
export PATH=$PATH:/opt/local/bin
```

Hit (:wq) to write and quit the vi text editor. If you find this all too much hassle you can enter that command manually every time a terminal window is started for the same effect.

Finally Installing Pov-Ray on OS X

Providing that everything has compiled correctly you can now install Pov-Ray for command line rendering. This is done by entering the command below:

```
bash% sudo port install Pov-Ray
```

This should install Pov-Ray to /opt/local/share/Pov-Ray-3.6 and setup the executable in /opt/local/bin. Once Pov-Ray is installed you should be able to run Pov-Ray through the command line by typing the following:

```
bash% Pov-Ray [+/-]Option1 [+/-]Option2
```

Installing DarwinPorts and Pov-Ray on Linux

Having installed DarwinPorts Installing Pov-Ray on Linux turned out to be a much more complex operation than the straight forward installation on OS X, and almost immediately I ran into issues. Below are the initial steps I took to creating what I thought would be a working DarwinPorts installation.

On the Linux client it will be necessary to install APT which stands for “Advanced Package Tool” and is freely available for most distributions, and pre-installed on Ubuntu. In order to install we must run the following commands:

```
apt-get update  
apt-get install tcl  
apt-get install tcl-devel  
apt-get install tcl-html
```

Now we can try installing darwinports:

```
bash% cvs -d  
:pserver:anonymous@anoncvs.opendarwin.org:/Volumes/src/cvs/od  
login
```

```
bash% cvs -d
:pserver:anonymous@anoncvs.opendarwin.org:/Volumes/src/cvs/od co
-P darwinports
```

Once it has finished downloading it should place DarwinPorts inside your home directory. Change paths to the DarwinPorts base directory where the installation files are kept and then run the configure script.

```
bash% cd ~/darwinports/base
bash% ./configure
```

Here we will notice something interesting happening within the content of the configure script when it is executed. I had originally glossed over this, but as we will see, this is the start of some problems.

```
checking md5.h usability... no
checking md5.h presence... yes
configure: WARNING: md5.h: present but cannot be compiled
configure: WARNING: md5.h: check for missing prerequisite headers?
configure: WARNING: md5.h: proceeding with the preprocessor's result
```

Everything appeared as though it had configured correctly with the exception of the above few lines so I then tried:

```
bash% make all
```

This then tries to compile the application and fails miserably. The below is the result of the attempt:

```
==> making all in src/programs/mtree
make[3]: Entering directory `/home/optio/darwinports/base/src/programs/mtree'
gcc -DHAVE_CONFIG_H -I. -I. -DNO_MD5 -DNO_RMD160 -DNO_SHA1
-DHOST="i686-pc-linux-gnu" -g -O2 -c compare.c -o compare.o
```

In file included from compare.c:80:

```
/usr/include/md5.h:27: error: syntax error before "UINT4"
/usr/include/md5.h:30: error: syntax error before '}' token
/usr/include/md5.h:38: error: syntax error before "PROTO_LIST"
/usr/include/md5.h:39: error: syntax error before "PROTO_LIST"
/usr/include/md5.h:41: error: syntax error before "PROTO_LIST"
/usr/include/md5.h:43: error: syntax error before "PROTO_LIST"
make[3]: *** [compare.o] Error 1
make[3]: Leaving directory `/home/optio/darwinports/base/src/
programs/mtree'
make[2]: *** [all] Error 1
make[2]: Leaving directory `/home/optio/darwinports/base/src/
programs'
make[1]: *** [all] Error 1
make[1]: Leaving directory `/home/optio/darwinports/base/src'
make: *** [all] Error 1
```

As this has failed there is no point in running the “make install” command and we are back to the drawing board on how to install Pov-Ray. I then appealed to the DarwinPorts user groups and to numerous online forums and received nothing but **poor feedback stating that DarwinPorts was intended for OS X installation only**, and although it technically states that it “can” be installed for Linux, it does not actually work.

A few days later I received an email from an individual named Dave Serpa who works for freeshell.org, and he stated that apparently he had got DarwinPorts to work operationally with his Fedora Core 2 Linux installation. After several emails that flew between our inboxes he was able to advise me on how I could correct some of the issues I was having. After doing some playing on my own with the determination of dependencies that were necessary to build DarwinPorts I came up with a long installation script that will automate the complete installation of DarwinPorts for me. Below is this script:

```
#!/bin/bash
```

```
##First get all the necessary libraries and programs we need
sudo apt-get update
sudo apt-get upgrade
sudo apt-get dist-upgrade
sudo apt-get install automake
sudo apt-get install autoconf
sudo apt-get install libtool
sudo apt-get install flex
sudo apt-get install bison
sudo apt-get install gcc
sudo apt-get install gcc-doc
sudo apt-get install g++
sudo apt-get install libgtk1.2-dev
sudo apt-get install libpng-dev
sudo apt-get install curl
sudo apt-get install libxml2
sudo apt-get install glib2
sudo apt-get install gawk
sudo apt-get install perl
sudo apt-get install indent
sudo apt-get install g77
sudo apt-get install gtk2-devel
sudo apt-get install openssl
sudo apt-get install libssl0.9.7
sudo apt-get install libssl-dev
sudo apt-get install tcl8.4-dev
sudo apt-get install tcl8.4-doc
sudo apt-get install libpgtcl-dev
```

```

sudo apt-get install libpopt-dev
sudo apt-get install cvs
sudo apt-get install libgrypt7
sudo apt-get install libgrypt7-dev
sudo apt-get install libgrypt-doc
sudo apt-get install ssl-cert

##Now it is time to install darwinports and Pov-Ray

##we need to add the wheel group to the system so that we can
##install darwinports properly, only do this if you are using a
##debian based distribution

cd /usr/local
mkdir src
cd src
#Login to CVS
cvs -d
:pserver:anonymous@anoncvs.opendarwin.org:/Volumes/src/cvs/od
login
#Downloads darwinports to /opt/local/src via CVS
cvs -d
:pserver:anonymous@anoncvs.opendarwin.org:/Volumes/src/cvs/od co
-P darwinports
PREFIX="/opt/local"           # default prefix
cd darwinports/base
./configure --prefix="$PREFIX" --mandir="$PREFIX/share/man" \
    CPPFLAGS=-I/usr/include/openssl
make all
sudo make install
cd ../../bin

```

```
./port install Pov-Ray

#If you run into the following:
    #/home/user/darwinports/bin # ./port install Pov-Ray
    #can't find package darwinports
#    while executing
#"Package require darwinports"
#    (file "./port" line 34)
#sudo addgroup --system wheel
```

At this point you should have a fully functional DarwinPorts installation, and Pov-Ray should now be able to run simply by typing the command:

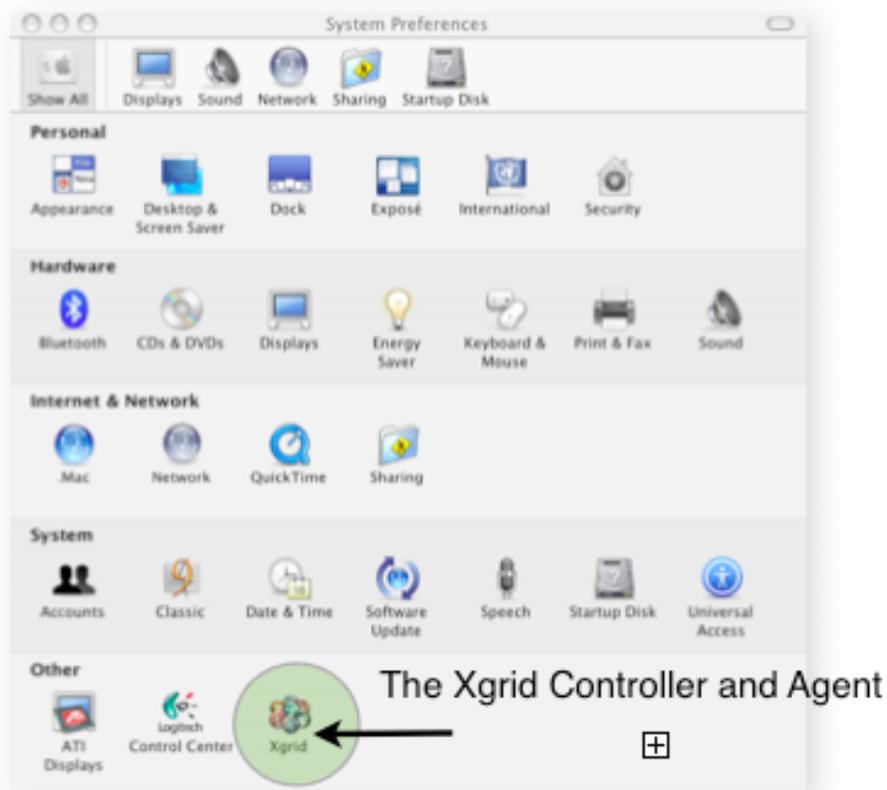
```
bash%: Pov-Ray
```

Overall the installation of DarwinPorts for Linux took the better part of 20 hours to get working within the Linux environment, and required not only in-depth knowledge of command-line linux, but also a knowledge of linux packages and dependencies. Furthermore being able to debug command line errors and create workarounds, and knowing how to decipher the technical advice that could be found in newsgroups and forums was definitely a major challenge. It is also important to note here that this is the first time DarwinPorts has ever been installed on Ubuntu Debian based distributions, or Fedora Core 3. The installation of DarwinPorts for Linux was critical in getting Pov-Ray to work in a cross platform manner with Xgrid.

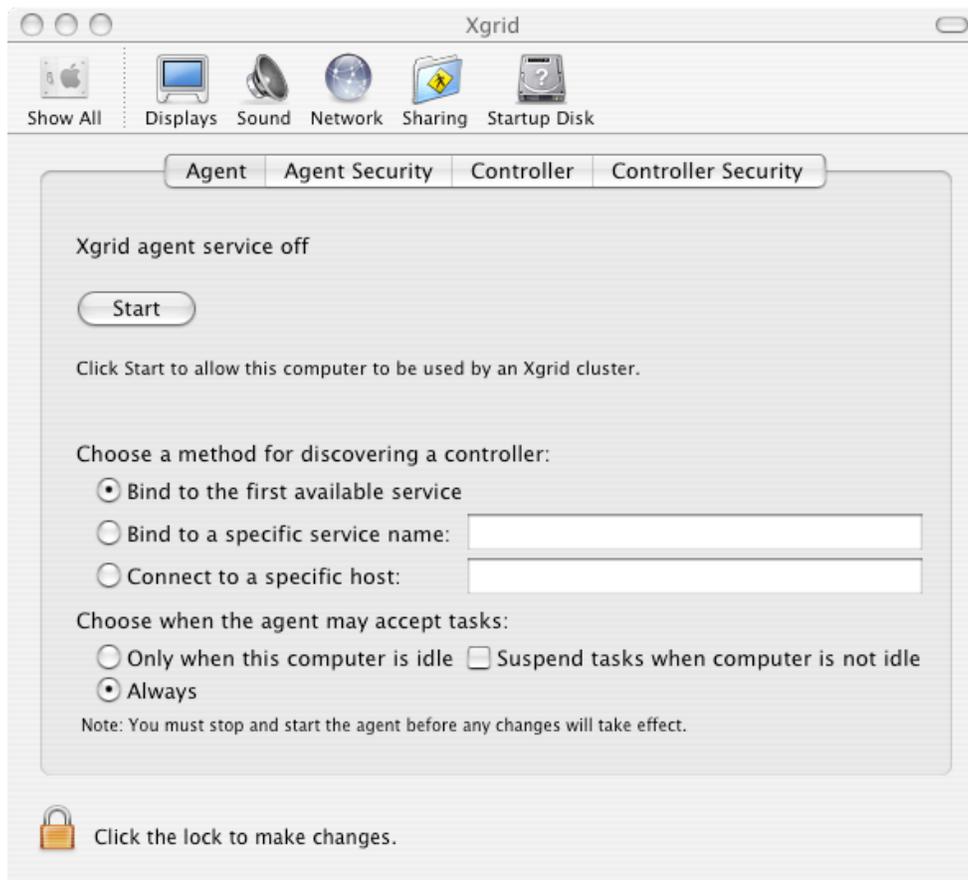
Appendix 2 - Xgrid User Interface

Controller and Client Setup

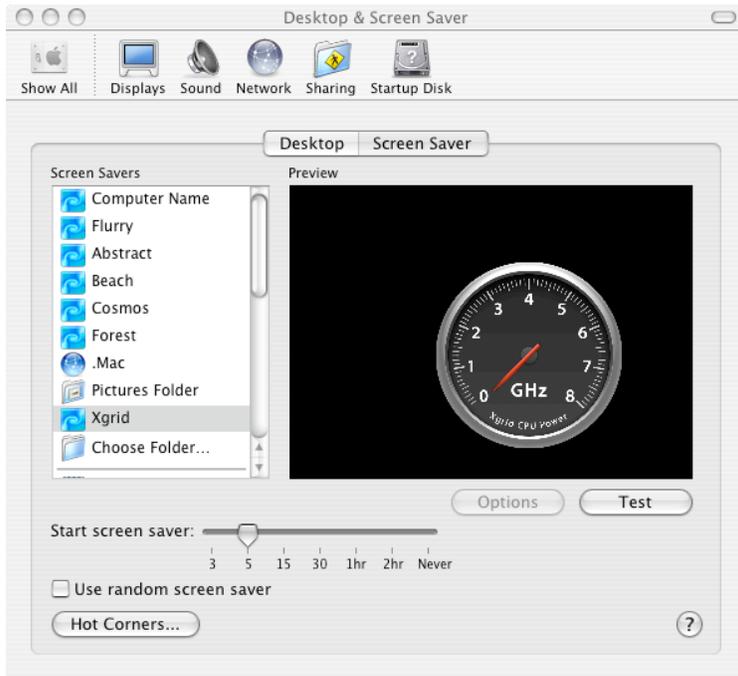
We will begin our analysis of the Xgrid interface by taking a look at the controller and client setup. Once Xgrid is installed, an Xgrid icon appears in the system preferences pane of your OS X client.



Once the user has clicked on the Xgrid icon in the preferences pane, they are presented with 4 tabs across the top depicting control options for either the agent or the controller. The first screen that appears is the configuration for the Xgrid agent. The agent may be started or stopped by clicking the “Start” button and then again by clicking the “Stop” button once the agent has been started.



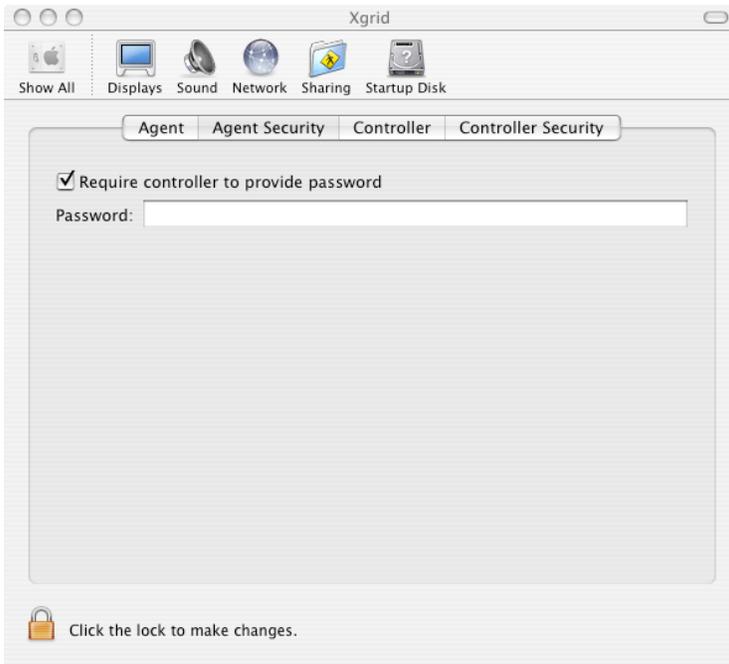
Options for connectivity include binding to the first available service, binding to a specific service name, and connecting to a specific host. In our case since we will only be running one Xgrid controller, binding to the first available service is fine, although we could go and specify the controller's IP address and connect to it manually just as easily.



The user can also specify when the agent may accept tasks that the controller has available. If you choose to only allow the agent to do jobs when the computer is idle, the agent will only become active when the screen-saver becomes active (Left). Tasks can optionally be suspended and withheld until the computer is idle

again and ready for processing jobs.

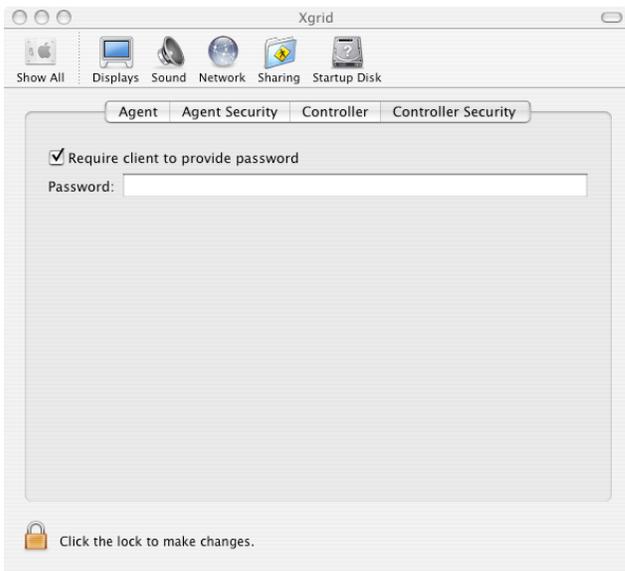
Lastly there is an option to allow the computer to always accept tasks. This is the setting I will be using as I want all of my systems to be available whenever I have jobs that need completing.



The security options for the agent are very simplistic, and provide a single option to force the agent to check whether or not the controller has supplied a password. By default Xgrid requires passwords to be specified.

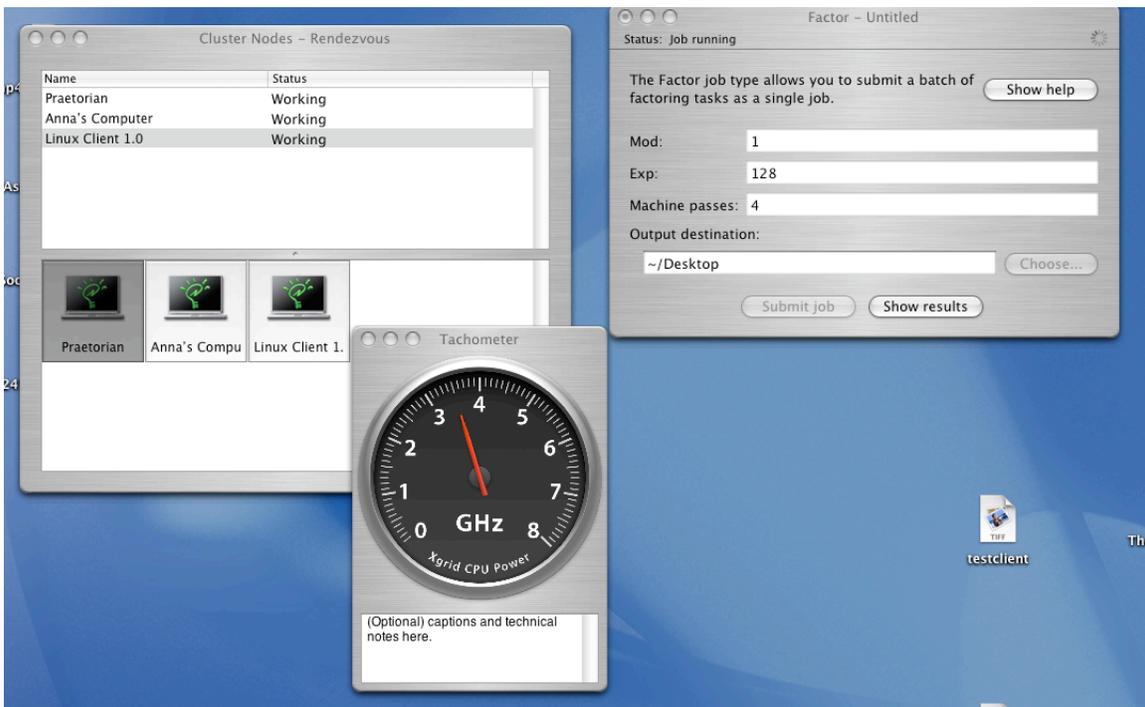


Once the controller tab is clicked options are presented for the configuration of the controller. Again we see a very simple configuration with a start/stop button and a field for specifying a password for the agents to connect to.



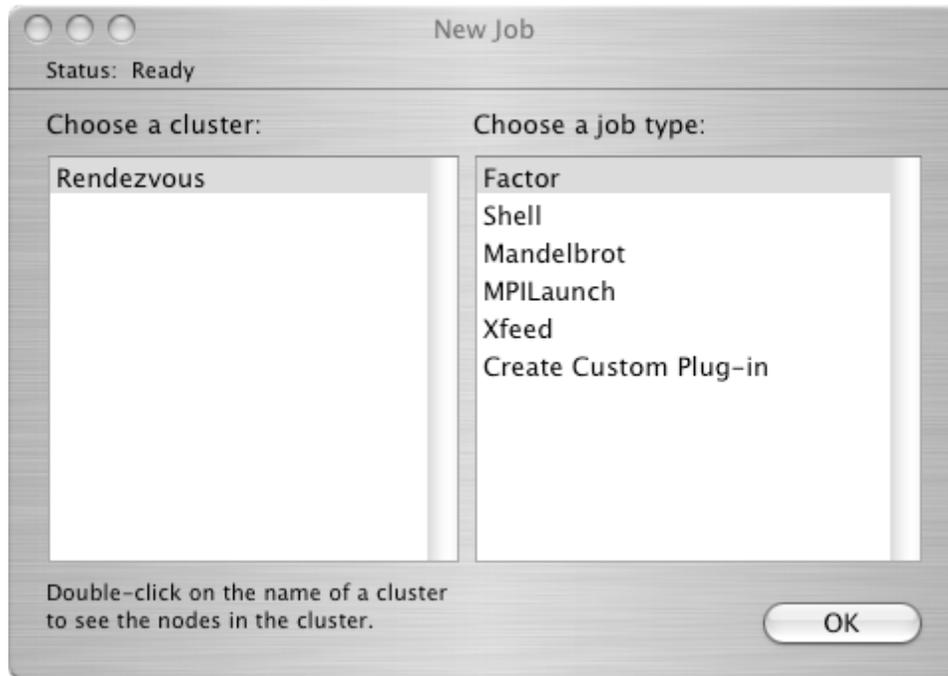
Once the controller and The last tab is for the controller security. An option exists to force the agent to provide a password. If the password is not specified the agent will not be allowed to join the grid.

Once the controller and agent configurations have been set, the Xgrid application can be started up. Below is a screenshot of Xgrid in action working on a factor job.

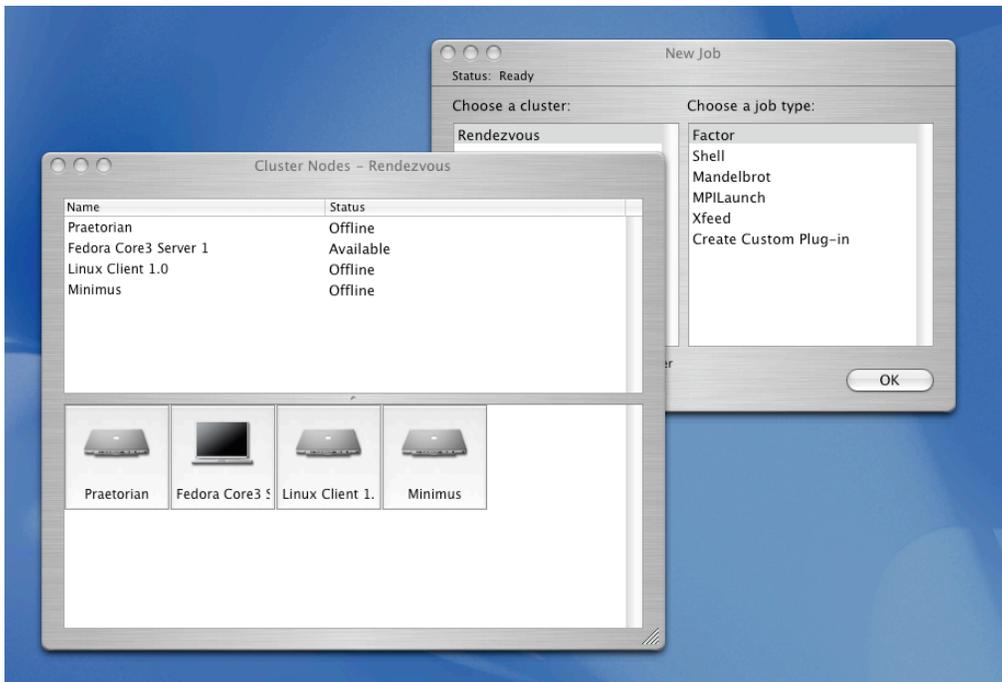


The Xgrid.app

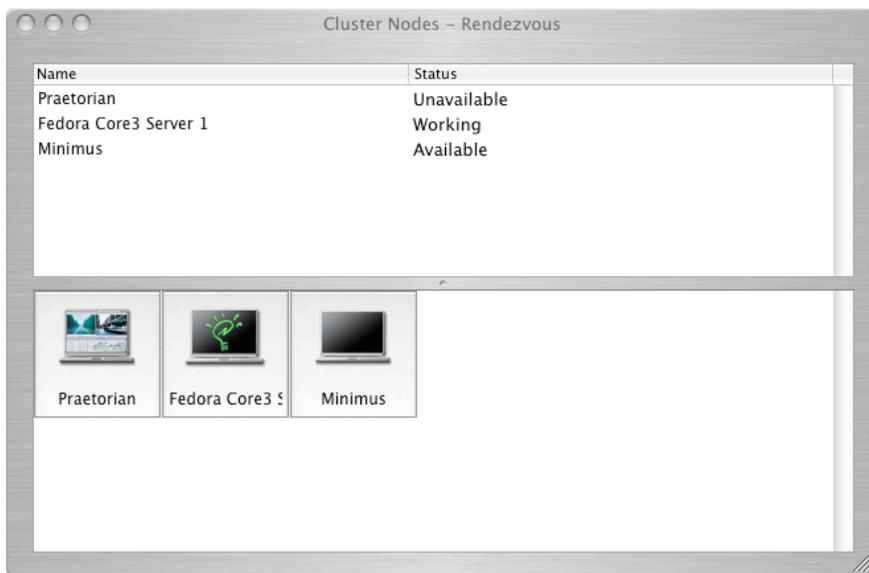
The Xgrid application in itself is extremely modular. Below is a screenshot of what the application looks like at startup.



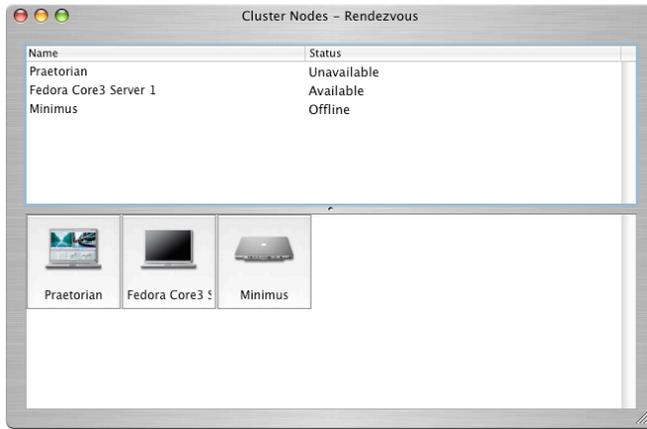
As we can see here the interface is very simple. Design simplicity is something that is heavily emphasized in Alan Cooper's book, and is something that is very obvious in this particular application. The beauty of this application is in its modular structure, and its intuitive interface. The user is presented with a list of available clusters on the left, and possible job types to perform on the right. Upon clicking on the "Rendezvous" cluster another screen pops up which lists the agents connected to the grid, and their status. This makes life very simple, as both the options are self explanatory, and more information on an item is easily available by either double clicking on a item, or single clicking and pressing the "Ok" button.



As we can see above, we have just double clicked on the Rendezvous cluster and we are now viewing the systems that comprise our cluster. Currently there are 3 offline systems and one online Linux agent. Not only are the systems presented in terms of their names and status, but they are also represented graphically by the little computer icons on the bottom of the pane. These icons are updated dynamically as their status changes. A screenshot below depicts the states of these systems.



This easily viewable display makes administrative tasks quite easy as a single glance at the display will be able to tell you the status of the grid, how many systems are connected to it, and what the status of each system is.



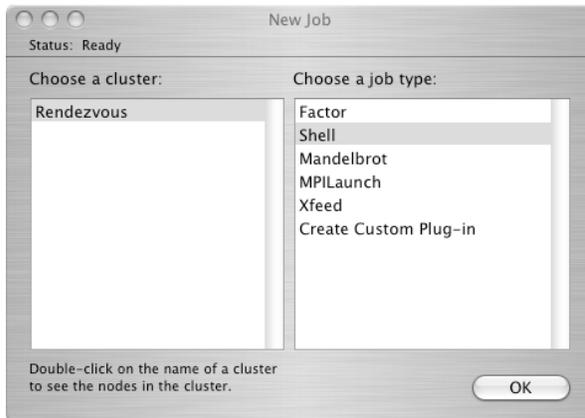
As jobs are finished each graphic is updated to the “available” status, unless they become offline or unavailable. Unavailable systems only become unavailable when they are no longer idle, this is a feature we talked about earlier that can be enabled in the Agent tab of the Xgrid preferences pane.

One of the coolest features of Xgrid is the tachometer. This meter measures the current grid processing power in gigahertz as it is applied to jobs. For instance before a new job is created, or while the network is idle the tachometer registers at 0.

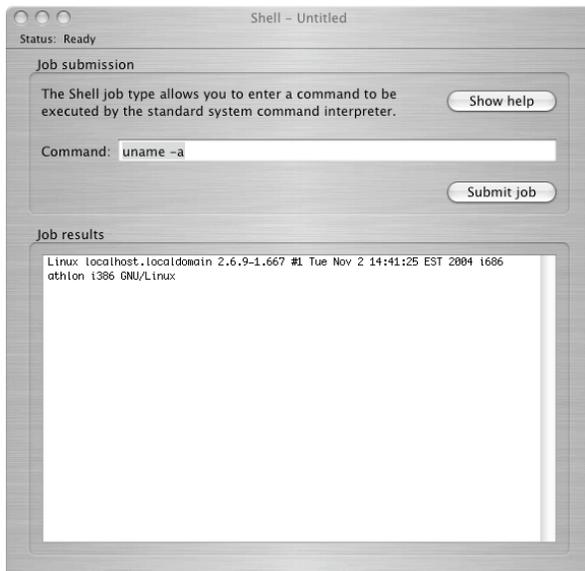


As jobs are submitted the tachometer will drop or rise due to either a shortfall in processing, or a sudden burst of processing by all available machines on the grid. This tool is merely eye-candy, yet it is very useful in deciphering which computer is doing work as it is tied directly to the effects viewed on the graphical status icons we discussed earlier.

Although there is no need as far as this project is concerned in delving into all the plugins available to XGrid we shall review two built-in job types which we will be using frequently to submit jobs to the grid. The first is a “Shell” job which allows any standard unix command to be submitted to the grid.



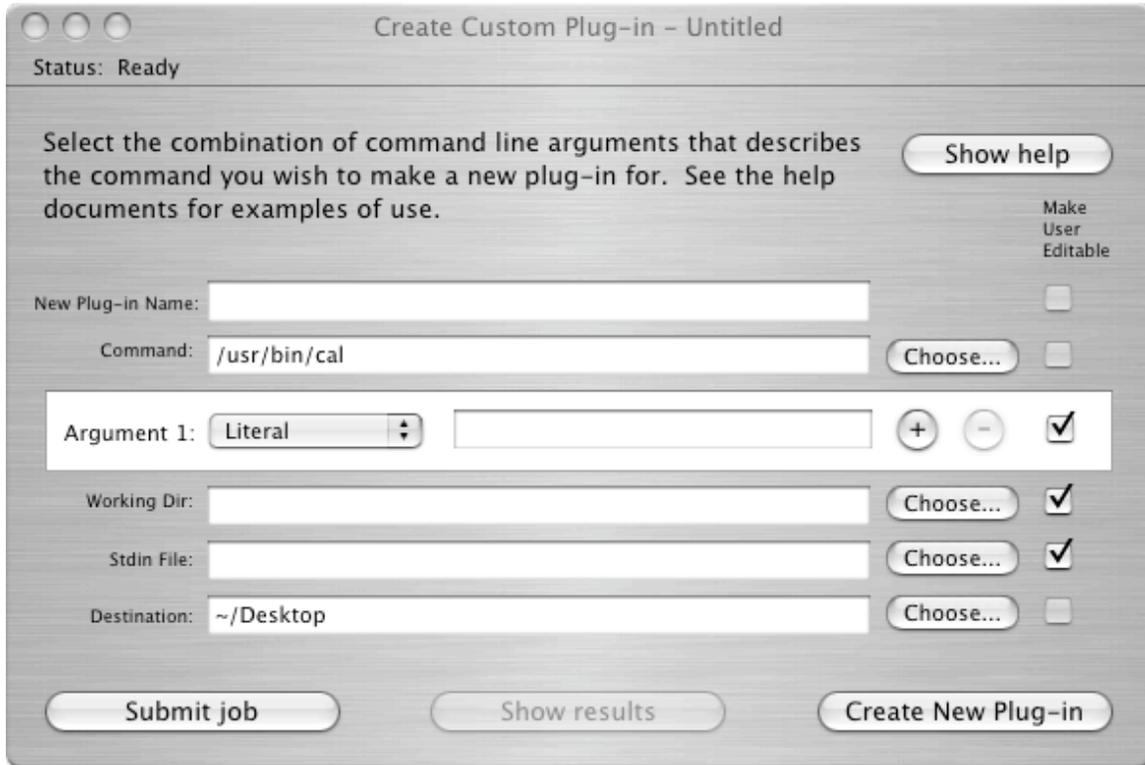
Once this option has been selected hitting the ok button will bring us to another screen which will permit us to enter shell commands. This particular function was found to be highly useful when needing to debug particular commands being submitted to the grid, or to perform single command line operations.



As we can see here a unix command has been given (`uname -a`) and the results from that command have been provided. Everything has been designed simply, yet elegantly, with maximum effort put into not adding more buttons or menus than are absolutely necessary. The results of our query are even selectable so that one can include them in reports or journals if necessary. It is the little things in this application that make it such a great success.

```
Linux localhost.localdomain 2.6.9-1.667 #1 Tue Nov 2 14:41:25 EST 2004
i686 athlon i386 GNU/Linux
```

The only other job-type we will review is the custom plug-in. This will be the mainstay of our job submission process so it is important to review how it works.

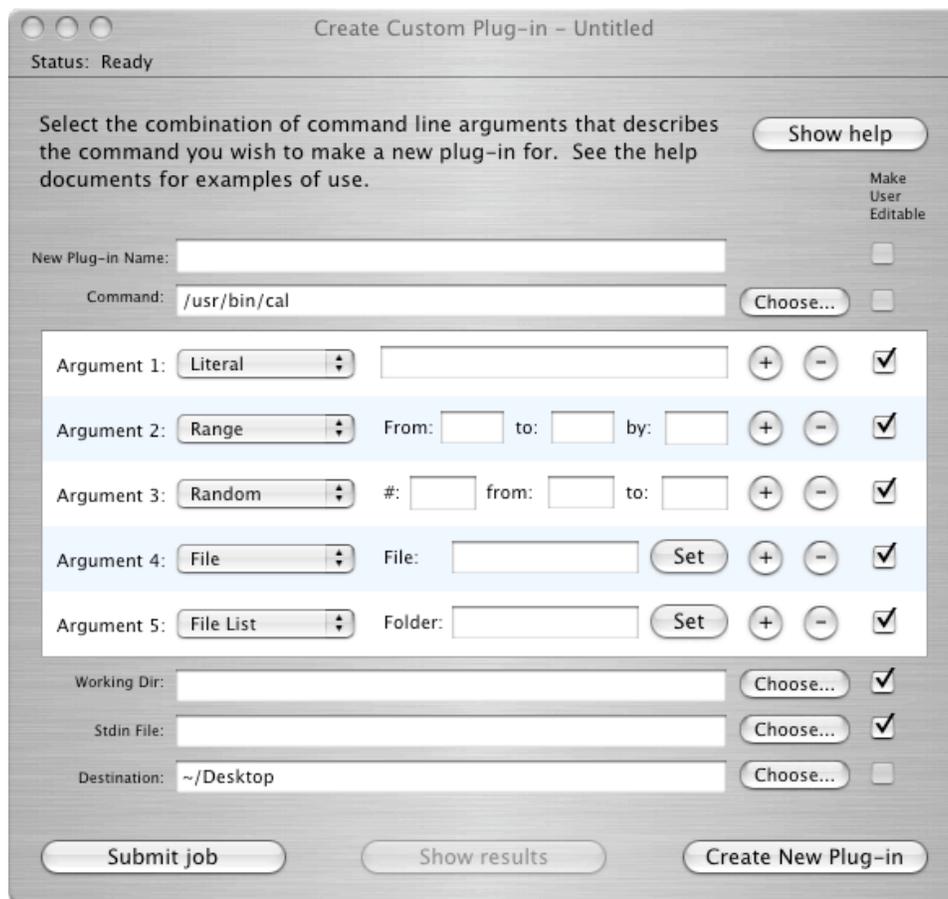


In addition to using this plug-in to submit complex jobs, you can use this plug-in to create additional custom Xgrid plug-ins without writing any code or changing any user interface settings. The simple idea behind this plug-in is that every Xgrid job task is a command-line string sent to an agent. By specifying the executable name and the number and types of the arguments, you can create a template that will create a list of command line strings and will be usable as a plug-in in the future.

The interface for this dialogue has been kept very clean, although the use of abbreviations such as “Working Dir” and “Stdin File” is somewhat confusing to the user. On the other hand the assumption is that most people using this application are not your typical user, and should be able to figure out what those abbreviations mean.

The most important aspect of this whole job-type is the argument specifier. This allows the administrator to specify what arguments need to be passed through to the clients in addition to the commands, and are expandable. By clicking the little plus icon on the same line as Argument 1, you can add as many additional arguments to your command as you deem necessary. This is particularly useful if you need to pass multiple arguments to a single command.

A screenshot of the types of possible arguments that can be added to a single command is posted below:



As we can see here there are a maximum of 5 different types of argument specifications, but you are allowed to pass unlimited numbers of these arguments in your job. The interface here has clearly been well thought through as options are not presented to you unless you require them through either clicking the plus or the minus buttons. This makes this

particular plug-in very easy to work through and figure out. Also the “Show Help” button is very useful as in-depth documentation on every aspect of the plug-in is presented.

Conclusion

The Xgrid interface is certainly well polished, and from an administrative standpoint is almost a grid-administrators dream come true. A few more features would have been nice though such as the ability to disconnect remote agents without having to restart the controller, or an option in the menu to restart the controller when it crashes as it frequently does. As Xgrid is still a technology preview I expect that some of these functionality issues will be ironed out prior to its debut in OS X Tiger but there are of course no guarantees of this.

Certainly there are other grid interfaces out there, but all of them pale in comparison to the usability that Xgrid has to offer. Considering that it will be possible to integrate this technology right into key applications such as Maya or Photoshop in the future this tool will certainly be a favorite amongst non-techies and techies alike who are looking for easily manageable grid software. The interface certainly falls well within the guidelines that Allan Cooper states in his book on usability design, and I certainly feel as though the interface really does a great job of taking a technical program and turning it into a very user friendly application.

Appendix 3 - Xgrid Installation Files

Xgrid installs itself to several locations on OS X. Below is a list of all the locations which Xgrid installs itself to, and some information as to what is stored in the most important directories.

/Applications/Xgrid BLAST.app	(This is the Xgrid BLAST application)
/Applications/Xgrid.app	(This is the Xgrid application)
/Library/Application Support/Xgrid	
/Library/PreferencePanes/Xgrid.prefPane	
/Library/Preferences/com.apple.xgrid.agent.plist	(This contains agent preferences)
/Library/Preferences/com.apple.xgrid.controller.plist	(This contains controller preferences)
/Library/Screen Savers/Xgrid.saver	(This is the Xgrid ScreenSaver)
/Library/StartupItems/GridAgent	
/Library/StartupItems/GridServer /Library/Xgrid	(Contains most of the Xgrid files)
/etc/xgrid	
/usr/bin/xgrid	(Location of the command line tool)
/usr/libexec/xgrid	(Location of the server processes)
/usr/share/man/man1/xgrid.1	(Manual entries)

Appendix 4 - Choosing the Linux Distribution

Choosing our Linux Flavor

With the object of this project being to run a cross-platform grid of some variety it is highly necessary for us to choose a Linux distribution that will match our project goals closely whilst providing a stable platform to work from. Some of the key factors in choosing the Linux distribution we will use involve the following:

- A large repository of available programs and libraries that are easily downloadable.
- A minimal amount of dependency issues particularly surrounding the use of GCC, Curl, and Tcl.
- Sufficient network hardware support
- Sufficient graphical hardware support
- An installation process that does not take days to get through
- A simple, clean user interface

As I have been using Linux for almost 3 years now I have had great success in the Redhat Linux flavor, and have found that the support for the product, and the number of available packages to be far superior than other Linux distributions. With the release of Fedora Core 2 however, I began to have my doubts about how committed Redhat seemed to maintaining their home-user product line (Fedora). In Fedora Core 2 I found that dependencies were often broken, the Redhat update engine seldom worked, and finding packages that were built for the platform to be something of a nuisance as their default package manager (Yum) had no where near as many available packages as the APT repositories that I had been used to on Redhat 9.

With this in mind we will be taking a look at a new distribution called Ubuntu and comparing it to that of Redhat's latest Fedora distribution. We will evaluate the distributions

by gaging their usefulness in aiding the project's goals, and we will determine which distribution we will use during the project.

The hardware specifications of the testing machine follow:

- AMD 2500+ Processor
- 1 GB PC3200 DDR
- 120 GB Western Digital Hard Drive
- ATI Radeon 9200 SE
- ASUS A7N8X NForce2 Motherboard
- Integrated Sound and LAN

Ubuntu Linux

Ubuntu is a newcomer to the Linux arena and has received wide acclaims as the Linux distribution that “just works”. Ubuntu is an ancient African word meaning “humanity to others” which coming from a community that has long been hailed as a techies dream come true and an end users nightmare is quite the irony. The Ubuntu community has been built around 4 pillars which seem to describe quite well their overall goal. The first pillar is that all Ubuntu releases including their enterprise edition will always be free of charge. The second is that they want to offer Ubuntu with as many accessibility features and languages as possible in order to make it the most accessible distribution ever. The third is that they plan to release a stable release every 6 months. Lastly they are committed to free and open source development and will continue to improve this software and distribute these improvements freely. Bearing these things in mind it certainly appears as though the goals of Ubuntu really are to offer humanity to the open source movement.

Ubuntu is based off the Debian distribution which was started more than a decade ago and now has a strong following within the Linux community. Debian also has one of the largest package management systems offering an estimated 17,000 packages freely over the Debian package manager. Debian is arguably the fastest Linux distribution available on the market, and is highly customizable to particular hardware architectures. This how-

ever does come at a cost, and the cost is the degree of difficulty involved in the installation. Some time ago I tried to install Debian and run into issues to do with the compatibility of my Nvidia graphics drivers with XWindows. A week later and after a complete kernel recompile with the integrated Nvidia drivers I still was unable to run XWindows. These issues are not at all uncommon to most Debian users, and the frustrations of installation often mar what would otherwise be an excellent distribution. Ubuntu, however, promises a greatly eased installation process, and looks to create an installation that is quick and easy to configure.

The Ubuntu Linux distribution is available from multiple sources the world over and comes in the form of an ISO CD image that must be written to a CD. The ISO is available here:

<http://www.ubuntulinux.org/download/>

Once the CD has been written we will reboot the computer and make sure that the first boot device is the CDROM. Once this is done hit enter once the graphical boot loader appears and the installation we begin.

The installation is fairly straightforward in the beginning, and prompts us for the usual keyboard, and language configurations. The installer will automatically determine our partition setup, and will present us with the option of either erasing the entire disk or manually editing the partition table. We will at this point choose to manually edit the partition table ourselves and setup the system to run on 10 Gigabytes of empty space I left at the end of the drive. We will divide these resources as follows:

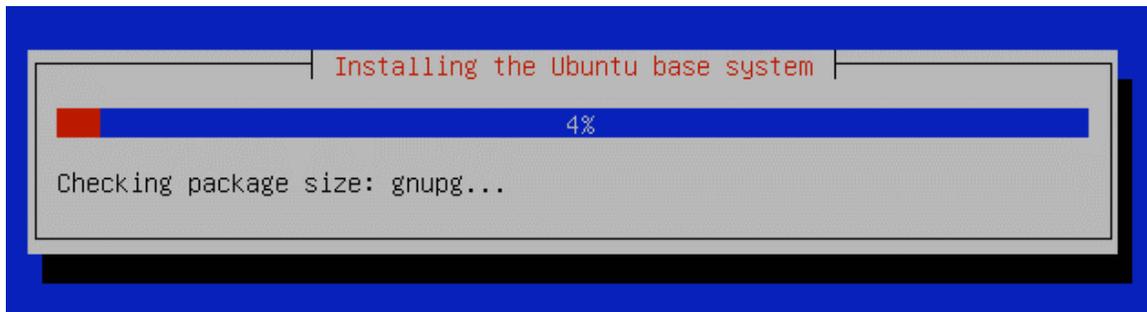
SWAP Partition (2GB)

/ Partition (7GB)

/Home Partition (1GB)

The SWAP partition is typically allocated double whatever the amount of memory in the system is, and is allocated this to allow the operating system to page to the hard drive. The / partition is the root partition and will contain all the operating system files, folder struc-

ture and libraries that are necessary in order for the system to run. The /Home partition will contain all my documents and downloads. It is important to note here that I created the /Home partition separate from the root partition in the case that if the system fails I can reinstall the operating system without having to format over the data located in that partition. Once these steps have been taken and the new partition table written the operating system will begin its installation process. It is important to note here that Ubuntu only provides a base installation, this installation is not customizable until the operating system is running, at which point we can install whatever we desire.



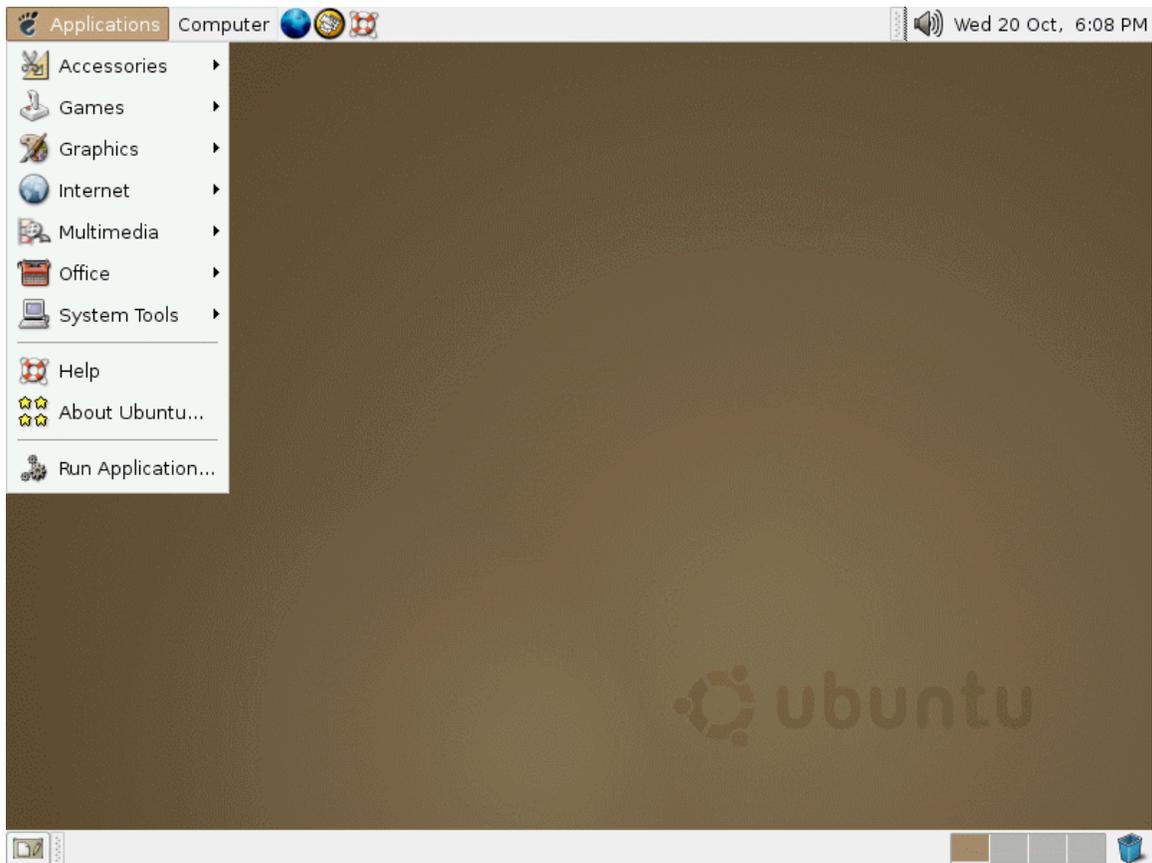
Once the installation has finished the system will reboot and will start XWindows. I was extremely surprised at this point as I had anticipated it to have crashed just as Debian did. Not only did it find all my hardware including my integrated 3Com Ethernet controller. As we can see below it successfully configures the network interface.

```

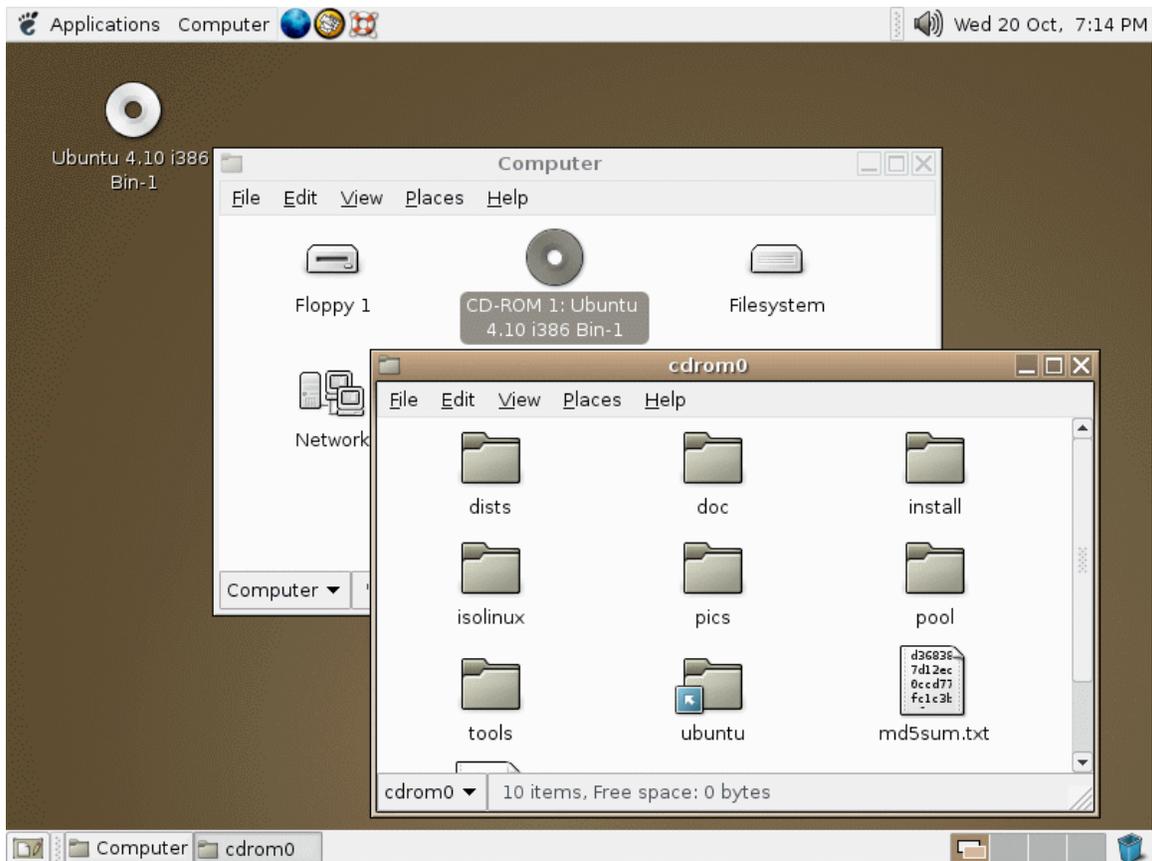
* Mounting a tmpfs over /dev... [ ok ]
* Creating initial device nodes... [ ok ]
* Setting disc parameters... [ ok ]
* Checking root file system...
/dev/hda1: clean, 21146/497984 files, 213346/995896 blocks [ ok ]
* Setting the System Clock using the Hardware Clock as reference... [ ok ]
* Calculating module dependencies... [ ok ]
* Loading modules... [ ok ]
* Creating device-mapper devices... [ ok ]
* Starting RAID devices... [ ok ]
* Setting up LVM Volume Groups... [ ok ]
* Starting Enterprise Volume Management System... [ ok ]
* Checking all file systems... [ ok ]
* Mounting local filesystems...
* Running systemd to make sure resolv.conf is ok... [ ok ]
* Starting hotplug subsystem... [ ok ]
* Configuring network interfaces... [ ok ]
* Setting up general console font...
* Setting the System Clock using the Hardware Clock as reference... [ ok ]
* Synchronizing clock to ntp.ubuntu.org... [ ok ]
* Initializing random number generator... [ ok ]
* Entering runlevel: 2
* Starting system log daemon... [ ok ]
* Starting kernel log daemon... [ ok ]

```

There are a few more configurations to pass through after this point, and it asks us to configure user names, passwords, and time-zones. Finally it prompts us to check for new packages and update them as necessary. I said yes to this as I was tempted to see whether it would actually download all the latest packages for everything it installed. This proved to be just the case as APT updated basically every single package that was installed. I thought for sure that this would break dependencies and that I would end up with a non-functional installation, however this was not the case at all and XWindows started up normally which was pleasantly surprising.



The interface appears very clean and I was pleasantly surprised by the layout and thought put into making this an operating system that is fairly easy to understand. The menus are clearly categorized, and unlike many linux distributions it does not provide us with such a massive list of installed programs that we are overwhelmed by the availability of choice. Some of the things that you often take for granted in Windows or OS X such as the auto-mounting of a CDROM are things that often do not work with Linux. This was not the case with Ubuntu however and I am quite impressed that the simple things like CDROM mounting in Nautilus (The file browser) actually worked.



There are many little things that make Ubuntu a nice distribution including fully featured media support, excellent hardware detection, and a repository system that doesn't seem to break dependencies as easily as most other available Linux distributions. From my initial observations I made the call that in fact Ubuntu might be well suited to the projects goals, however, it was time to test this as there can be no uncertainties in my Linux distribution of choice. The first step in the testing procedure was to download and begin the installation of all the libraries that are necessary for Xgrid. As discussed earlier it is necessary to ensure the presence of GCC which is the GNU Compiler Collection for Linux and is necessary in order to successfully compile XGrid. GCC provides front ends for C, C++, Objective-C, Fortran, Java, as well as standard libraries for all of those languages. Other necessary packages included Curl which is a command-line tool for remote retrieval (FTP, HTTP, WebDav etc.), and Tcl which is also necessary to compile some of the other programs we will be utilizing in our project. Tcl stands for "Tool Command Language" and provides graphical user toolkits that are highly portable and run cross-platform.

Using APT (Advanced Package Tool) is a very familiar thing for me coming from a Red-hat 9 world, and I had no problems in figuring out what commands would be necessary in order to retrieve the necessary applications. APT as I have talked about earlier is a package management system which contains repositories of libraries and applications that are available freely for download. The beauty of APT is that it maintains a version list of every library and application installed on the system and by doing a simple “apt-get upgrade” it is possible to upgrade ones entire system to the latest versions. APT also uses this list to make sure you do not install the same application twice, so if you tell APT to install a certain package, it knows whether or not this package is already installed and will let you know if it is. APT is a great and easy way of dealing with Linux dependency issues as well because when you give it the command to install something it will automatically look at the dependencies for that library or application and download them. In our case the first thing we need to do is obtain Tcl. This can be done by entering the following in the terminal:

```
bash% apt-get install tcl
```

This then proceeded to download the Tcl libraries from the APT repository and install them in the system. I then tried the next package

```
bash% apt-get install curl
```

After a quick download it installed, and I tried using curl through the command line to verify that it was working and it did in fact return with a result

```
bash% curl -O http://obelix.ca/index.gif
```

% Total		% Received		% Xferd	Average Speed			Time	
Curr.					Dload	Upload	Total	Current	
Left	Speed								
100	48788	100	48788	0	0	70299	0	0:00:00	0:00:00
0:00:00	97452								

In this command we can see that curl will retrieve the necessary file and that it is in fact working. I found that a lot more libraries were necessary in order to get Ubuntu to the state in which it would be ready for my 440 project. By default none of the programming tools that are necessary to develop anything in Linux are installed on Ubuntu, so the process of installing the necessary tools for compilation, and installation of most programs is a long and tedious task. Below is a list of the commands that were necessary to download dependencies for both the Xgrid agent, and darwinports. (See Appendix 8 for details)

Before we do anything it is always a good idea to run “apt-get update”. This command runs through the online servers and retrieves the latest lists of software and updates these lists on your system so that when you want to get a certain package, it automatically downloads the most current version.

```
bash% sudo apt-get update
```

Now lets do a “apt-get upgrade” and an “apt-get dist-upgrade”. These commands will automatically download the latest and most stable versions of every single piece of software we have installed on our machine. This is a great tool as it allows an administrator to quickly and easily update an entire system through running two commands.

```
bash% sudo apt-get upgrade
```

```
bash% sudo apt-get dist-upgrade
```

Now lets start downloading and installing all the other tools, libraries, development kits, documentation, and applications that we will need to install both darwinports and the linux agent.

```
bash% sudo apt-get install automake
```

```
bash% sudo apt-get install autoconf
```

```
bash% sudo apt-get install libtool
bash% sudo apt-get install flex
bash% sudo apt-get install bison
bash% sudo apt-get install gcc
bash% sudo apt-get install gcc-doc
bash% sudo apt-get install g++
bash% sudo apt-get install libgtk1.2-dev
bash% sudo apt-get install libpng-dev
bash% sudo apt-get install curl
bash% sudo apt-get install libxml2
bash% sudo apt-get install glib2
bash% sudo apt-get install gawk
bash% sudo apt-get install perl
bash% sudo apt-get install indent
bash% sudo apt-get install g77
bash% sudo apt-get install gtk2-devel
bash% sudo apt-get install openssl
bash% sudo apt-get install libssl0.9.7
bash% sudo apt-get install libssl-dev
bash% sudo apt-get install tcl8.4-dev
bash% sudo apt-get install tcl8.4-doc
bash% sudo apt-get install libpgtcl-dev
bash% sudo apt-get install libpgtcl-dev
bash% sudo apt-get install cvs
bash% sudo apt-get install libgrypt7
bash% sudo apt-get install libgrypt7-dev
bash% sudo apt-get install libgrypt-doc
bash% sudo apt-get install ssl-cert
```

Once all of this has been done I found that I could get both the Xgrid agent and darwin-ports to compile. Determining the requirements for the installation of the above packages, the order in why they should be installed, and the dependencies which these packages had was a process that took a number of hours.

Overall I have been very impressed with Ubuntu. It is a very fast Linux distribution, and with a few hours of configuration can be made to work precisely with our projects goals.

Fedora Core 3

The Fedora project is sponsored by Redhat and is a community-supported open source project. After Redhat 10, Redhat decided to no longer provide support for its home-user products and instead chose to offer a free and non-Redhat supported distribution called Fedora. Fedora is funded by Redhat, but all of the support available for the product comes from the community of users that use it. Message boards, IRC groups, and email lists are the forms of support that are available to Fedora users and all of them come with the cost of \$0 both for you the user and for Redhat. Redhat in this manner maintains its image as a major Linux provider for non-enterprise consumers, and also gains a large testing ground for its custom linux applications and environments. As mentioned earlier I have been less than satisfied with the Fedora Core 2 distribution which has left much to be desired, and I hope that Fedora Core 3 will prove to be a much more stable, useful distribution that its predecessor.

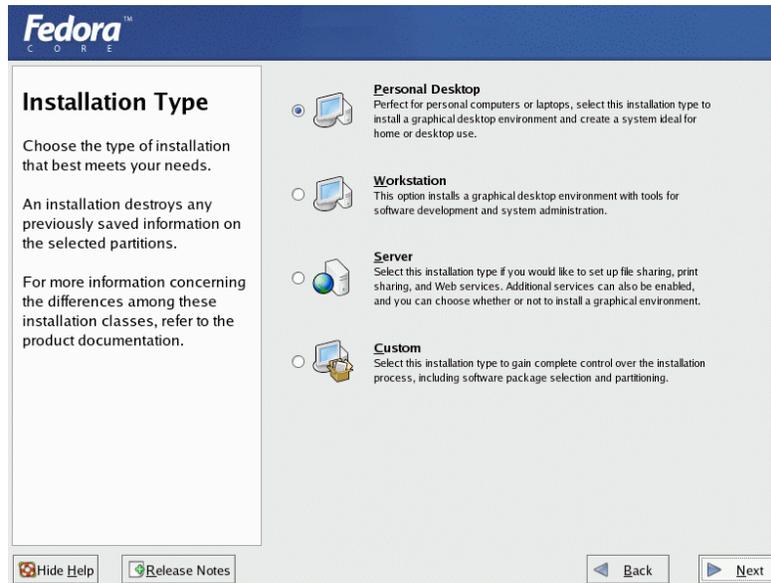
The Fedora Core 3 Installation Process

Fedora has an absolutely massive list of packages that come on 3 CD's and are freely available to download at the following site:

http://download.fedora.redhat.com/pub/fedora/linux/core/3/x86_64/iso/

Interestingly Fedora is also offered in a DVD format which is actually the one I used for the installation. The DVD weighs in at 2.46GB and includes almost every package we

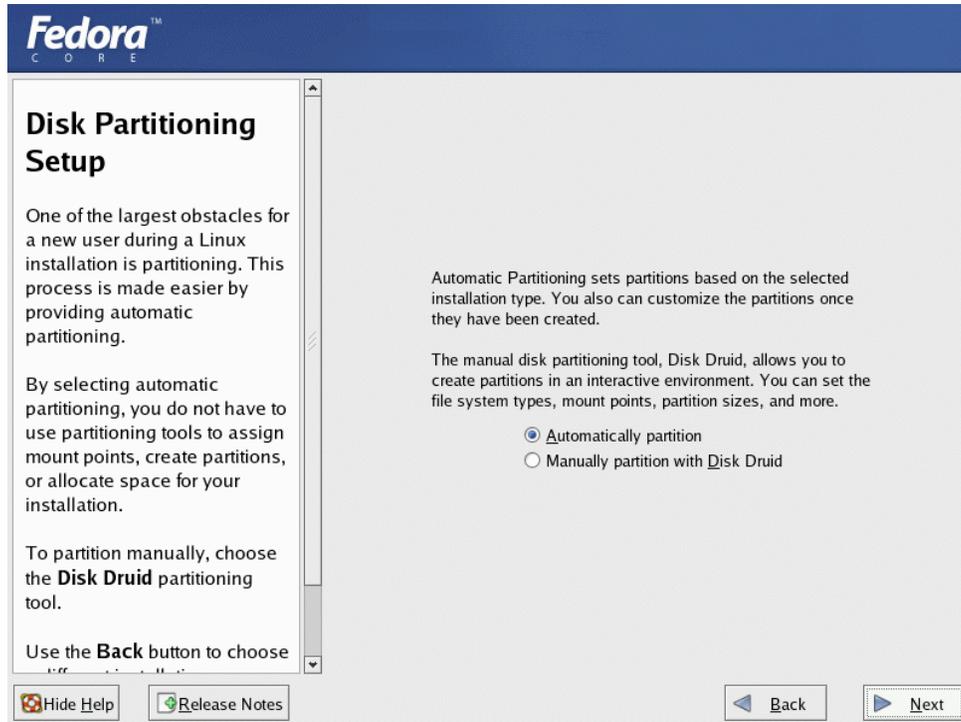
need for the project. The installation itself is much more graphical than Ubuntu's text based installer, and for end-users is certainly a much easier method of installation (especially where partition tables are concerned). The installation starts with the same standard setup of languages, keyboards etc. that we had in Ubuntu. It then presents us with options as to what type of installation we want to use, and we will select custom.



At this point we are presented with a list of all the available packages and we will make sure that we select all the development packages (particularly GCC) and the packages for both kernel development and X development. My reason for selecting these packages is that I am never 100% sure what packages I will be needing and it is easiest to install everything at once during installation than it is to get them later. There have been times in the past where network cards refuse to work and I have been required to do a complete kernel recompile so that I can manually add my own ethernet card modules. It is always best to be prepared for the worst case scenario when working in Linux in my opinion.

Partitioning comes next and the graphical installer makes this quite easy. If we choose automatic partitioning it will setup the drive and remove any other partitions that are currently in use. However as is the case with the Ubuntu installation we need to manually

partition everything ourself as we are running this system as a dual boot system with Windows 2003 Server.



We will setup Fedora Core 3 in the same manner as the Ubuntu installation with the following partitioning setup:

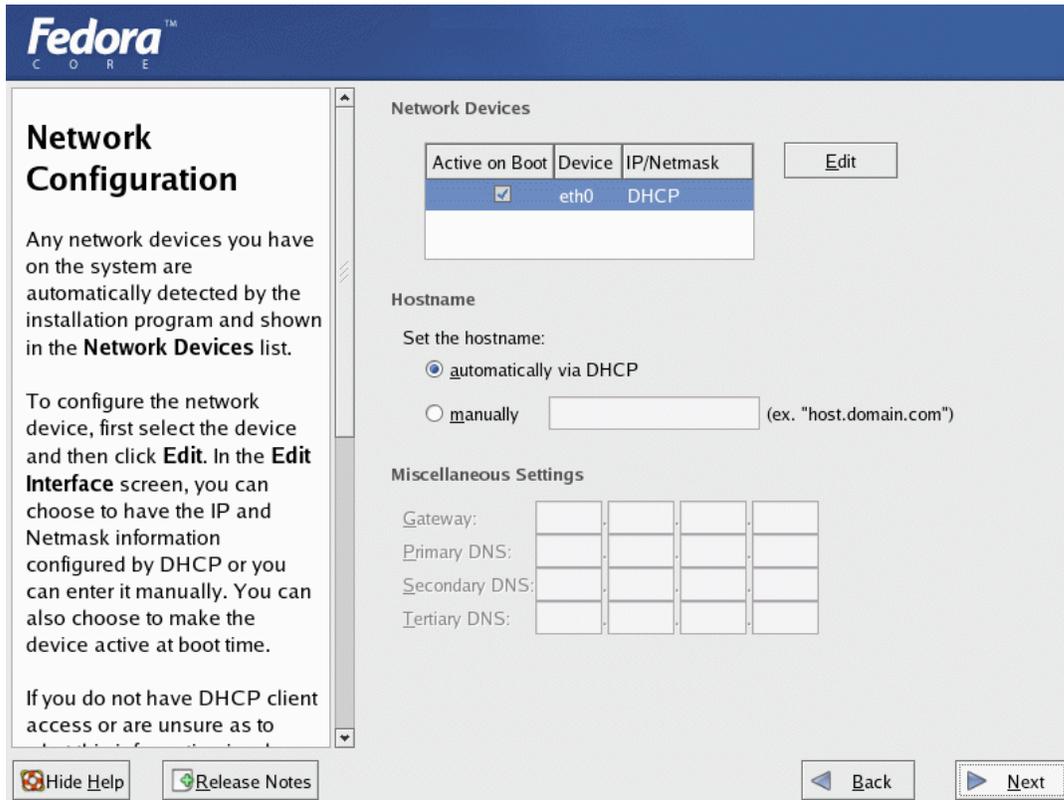
SWAP Partition (2GB)

/ Partition (7GB)

/Home Partition (1GB)

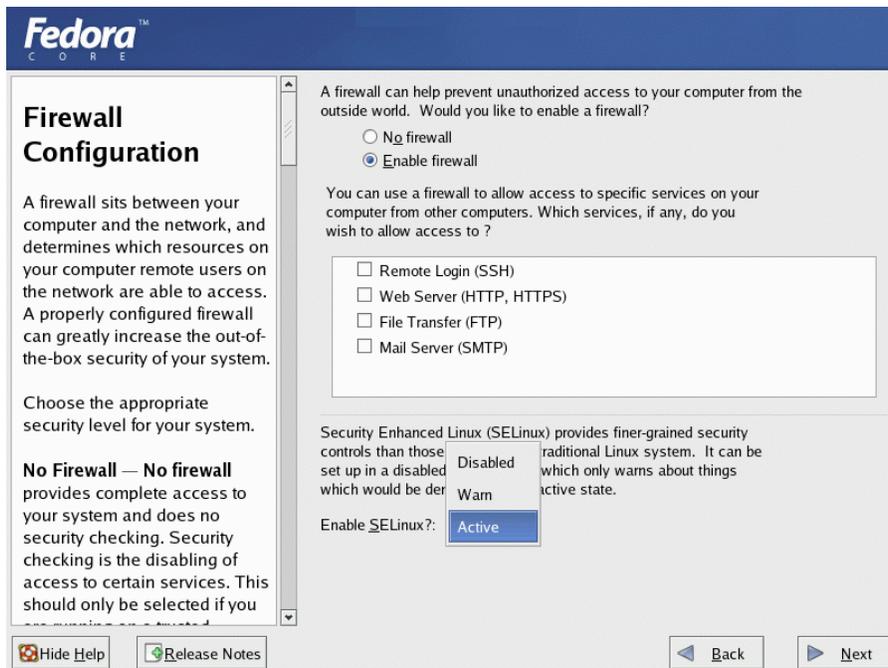
Once this has been done we will configure our bootloader and ensure that our Windows 2003 Server partition is set as the default boot partition. This is only necessary because I do not want Fedora booting by default. This can be undone later by editing the grub.conf file located within the /boot/grub directory if need be.

The network configuration wizard during the Fedora Core 3 installation



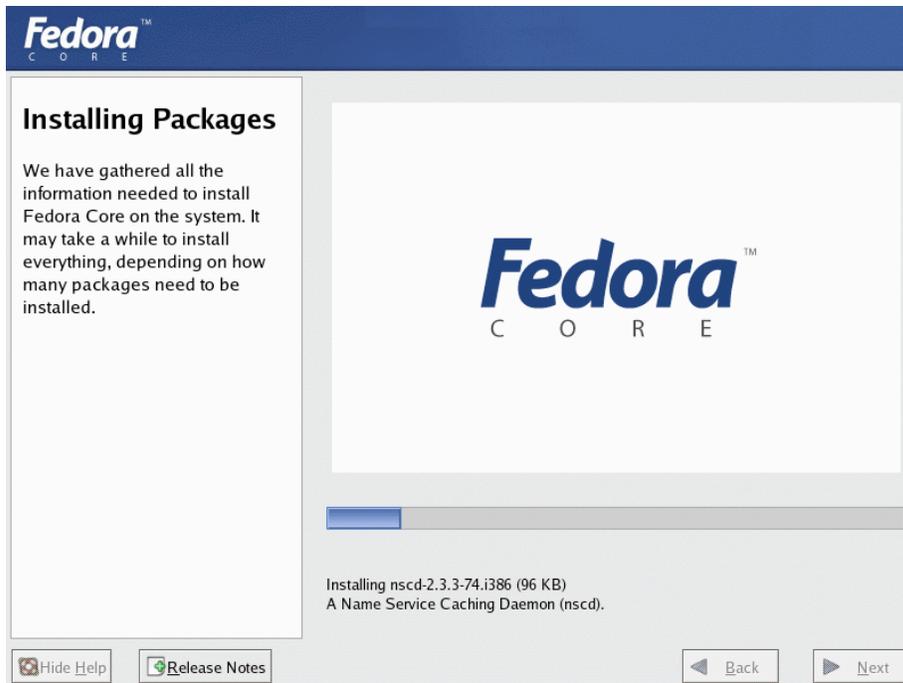
At this point we are presented with the option of configuring our ethernet card which it has successfully detected and we have the option of either specifying our own IP address, gateway, and DNS, or using a DHCP setting. As my router utilizes DHCP I will use the default DHCP setting that is provided.

Another thing that is a very nice feature of Fedora that was no present in the installation of Ubuntu is the option to install a firewall. This is a very important aspect of networking that is missed sorely by Ubuntu, and is something that should definitely come enabled by default. In Fedora not only is there a firewall installed but there is also an option to install SELinux which stands for Security Enhanced Linux. SELinux was created by the National Security Agency (NSA) in the US to help combat the problems Linux has with hacking.. In short, SELinux provides an access control architecture to confine processes to only the files they need to complete their actions. Security is outside the scope of this review but it is important to keep it in mind when choosing a linux distribution, and this is something that we will enable.



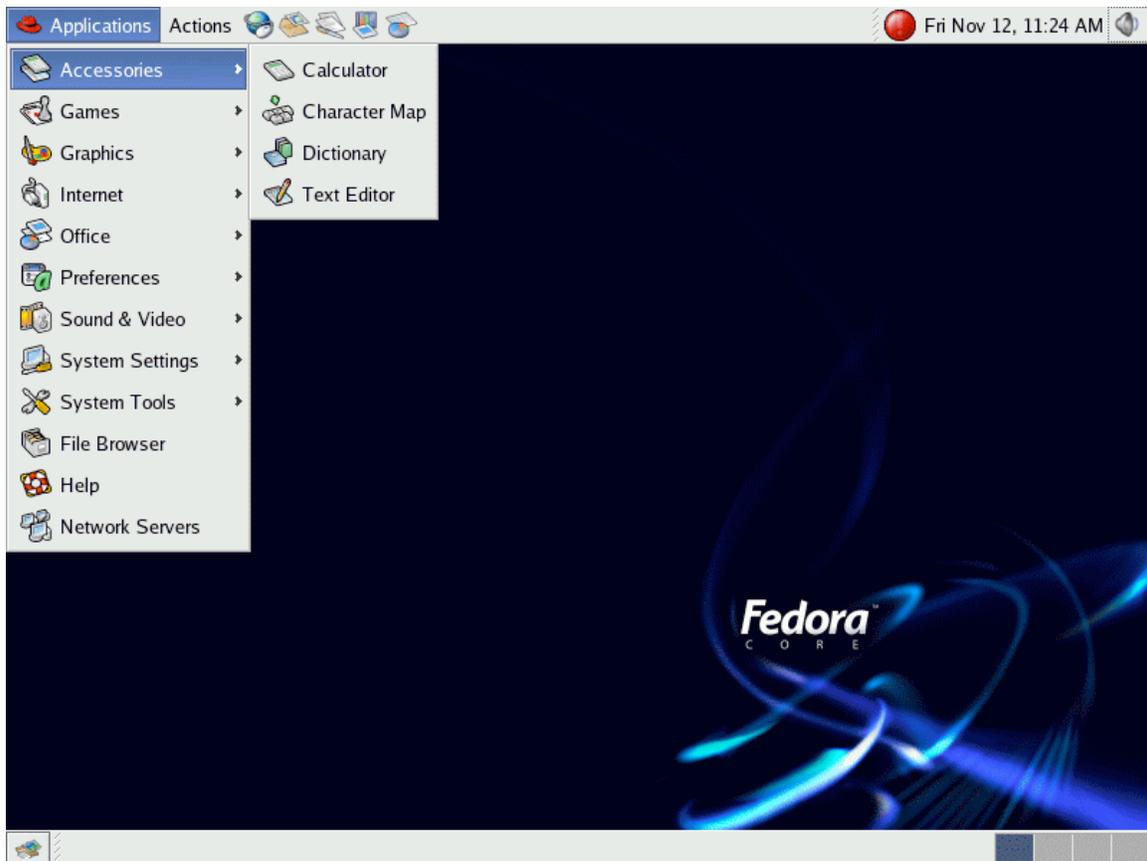
A screenshot of the firewall configuration and the process of enabling Security Enhanced Linux.

Next comes the setting of the root password, installation of additional languages, and final configuration of the time-zones. This is all very straightforward and the installation will now begin.



A screenshot of Fedora Core 3 installing.

The rest of the setup process is very straightforward and there is no reason to cover it in any detail. Upon restart you have to agree to the license agreement, set the date and time, set the default display settings for your monitor, create a user, and lastly test to make sure the sound card they have chosen works. The sound in our case is not an issue we even need to care about so I skipped this step entirely. XWindows then loaded and I was quite pleased to find that it had successfully detected both my video card, and my network card so no kernel recompiles would be necessary.



As we can see in this screenshot the menu system in Fedora Core 3 is quite simplistic and options are fairly well placed. I personally preferred the way that the Ubuntu menu system was organized, as system settings and preferences to me are things that in my opinion should be amalgamated into one single menu so as not to confuse the user. The menu system however isn't where we will be spending most of our time, and as what we really need is solid package support we will now test the distribution by trying to install the necessary packages. The default package manager that comes with Fedora is still Yum and I still find it a much worse system than using APT. With that in mind I went and found an APT build for Fedora which was available at FreshRPMS.net here:

<http://freshrpms.net/apt/>

Once APT was downloaded and installed I was able to successfully run and install the following:

```
bash% apt-get install tcl
```

```
bash% apt-get install curl
```

There was no reason to get the GCC package as Fedora Core 3 had already installed that for me during the OS installation process. After a brief check through the APT repository list that they now have for Fedora Core 3, they certainly appear as though they have been busy between releases and there is now a greatly increased number of available packages to the distribution. This tied with all the software that comes on the DVD and the relatively simple installation process make Fedora Core 3 a good operating system for the purposes of this project. On the down side their redhat-auto-updater which is supposed to download the latest packages and fixes still does not function properly, and crashed several times during updates.

Linux Conclusions

For the purposes of this project it is absolutely essential that the distribution I choose must be easy to install, and although Ubuntu did not have a graphical installer, the text-based installation was relatively painless to get through. Despite having no GCC compiler or any programming or development tools installed, it was not too painful to acquire these through its Advanced Package Tool. The “apt-get upgrade” feature which fetches all the latest packages and updates actually worked, and for that reason alone I’d certainly recommend it over Fedora Core 3. For the home user Ubuntu also has several advantages as things like multimedia support are fully functional by default and require no special configuring unlike the multimedia packages that come with Redhat. There are certainly things to appreciate about Ubuntu but it definitely does require much more configuration than Redhat does in order to work with the goals of this project. I will leave the current PC machine running the Ubuntu Distribution as I am very satisfied with its performance and features.

Our final choice for distribution with the remaining machines however will be to use the Fedora Core 3 installation for due to its very large complement of packages available on the DVD, and due to the fact that the APT system for Redhat does contain all the available packages we need as well. The reason we are choosing to stick with Redhat is not because it is the better distribution, but because it requires much less setup time so it is much faster to rapidly deploy in our grid environment. In respect to which is the better distribution, I would certainly pick Ubuntu as the clear winner as its interface is a lot snappier, the multimedia applications which are installed “just work” and it has a much less cluttered menu system that would be much easier for beginners to use.

Appendix 5 - Installing the Xgrid Linux Agent

The first step in installing the Xgrid Linux agent is to acquire all the packages necessary in order to build the agent. For simplicities sake, these packages have been downloaded to my own webserver to simplify the retrieval process as some of the online sources for these libraries are no longer available. Open a linux terminal and make a directory called “src” within your home folder, and then type the following commands:

```
curl -O http://www.obelix.ca/xgrid-files/glib-2.4.1.tar
curl -O http://www.obelix.ca/xgrid-files/libxml2-2.6.9.tar
curl -O http://www.obelix.ca/xgrid-files/roadrunner-0.9.1.tar
curl -O http://www.obelix.ca/xgrid-files/xgrid-patched.tar
```

All the libraries we need will be remotely retrieved. Next we need to extract these packages like so:

```
tar vxf libxml2-2.6.9.tar
tar vxf glib-2.4.1.tar
tar vxf roadrunner-0.9.1.tar
tar vxf xgridagent-patched.tar
```

These packages all have an extensive list of dependencies, and it is very important to make sure that all the necessary dependencies are present. I cannot count the numbers of countless hours that were spent resolving dependency issues, and recompiling to ensure each dependency had been met. If dependencies are not met, the installation will fail. To deal specifically with this issue an automated configuration script was made (See Appendix 7 for details). This is very important to note because it allowed me to install the Xgrid agent without having to manually type each line that is contained within Appendix 5.

The first package that will be compiled is the libxml package. You will notice that we add the --PREFIX to the ./configure line. This is because we want to install all the libraries

and files within our home folder so we do not need to use **ROOT** access in order to utilize the Xgrid agent. Below are the steps required:

```
cd libxml2-2.6.9
./configure --prefix=$HOME
make
sudo make install
```

Providing there are no errors here, we will move onto the glib installation:

```
cd ~/src/glib-2.4.1 --prefix=$HOME
./configure
make
sudo make install
```

Double check the `/home/current-user/lib` folder to make sure that both the libxml and glib files are present. If they are not, something went wrong in the installation phase and you are probably missing dependencies. Providing that glib successfully installed we can now install the RoadRunner communication framework by typing the following into the terminal:

```
cd ~/src/roadrunner-0.9.1
./configure --prefix=$HOME
make
make install
```

Next comes the complex part of the installation phase which involves configuring the Xgrid agent. If you just try to configure the agent with `--prefix=$HOME` the installation will fail because the configure script is incapable of finding the RoadRunner libraries without explicit direction as to where it can find these files. To install the agent type the following commands:

```
cd ~/src/xgridagent-1.0.2
```

```
./configure --prefix=$HOME\  
--with-roadrunner-includedir=$HOME/include/roadrunner-1.0\  
--with-roadrunner-libdir=$HOME/lib
```

The “\” characters at the end of the line simply allow you to continue typing more parameters on a new line. When the configure script is running make sure you look at the “locate libr” line and make sure the output is “Found” and not “Missing” otherwise the Xgrid agent will not run. If you did not download my patched version of the Xgrid agent (See Appendix 6 for more details) then you will manually need to open the xgridagent.c file and comment the following lines starting on line 1647:

```
/*#if USE_XML_STRNCAT_NEW*/  
  
tmp =  
(xmlChar*)xmlStrncatNew(BAD_CAST("\n"),BAD_CAST(inBuf),inLength);  
  
/*#else  
  
tmp =  
(xmlChar*)xmlStrncat(BAD_CAST("\n"),BAD_CAST(inBuf),inLength);  
#endif*/
```

Once this is done the binary can be compiled by typing:

```
make
```

Note: One of the biggest problems I was having during the first month of the project was the installation of Linux and the XGrid agent itself. I decided that I needed to know whether the agent could in fact function in a Linux Environment. During my first installation of Ubuntu I was unable to obtain the GCC library (using APT) necessary to compile the roadrunner library (BEEP) that the XGrid agent framework runs from. This proved very problematic as it prevented the XGrid agent from running on Ubuntu at all. At this point I removed Ubuntu and installed Fedora Core 3. I was able to obtain the necessary GCC libraries using YUM and the roadrunner binaries installed just fine.

When the Xgrid agent is run the following message appears:

```
./xgridagent error while loading shared libraries:  
librr-0.9.so.0: cannot open shared object file: no such file or  
directory
```

This meant that it was unable to load the shared libraries that roadrunner provides. After doing some research about libraries and how they were linked, I found that it is necessary to set the path to this library so that the agent knows where to find it. This library can be defined by setting the LD_LIBRARY_PATH environment variable in either the .profile file in the current users' home directory, or in the .bashrc file located in the current users' home directory. This is done like so:

```
export PKG_CONFIG_PATH=$PKG_CONFIG_PATH:$HOME/lib/pkgconfig  
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/lib
```

No make install is necessary in this phase as the make command will compile everything that is necessary. The Xgrid agent should now run by typing the command:

```
./xgridagent 192.168.0.100
```

Where the IP address above is the IP of the Mac OS X controller. The Linux Xgrid agent should now connect to the controller and be able to process any standard unix commands passed to it. If you run a job and notice that the speed of the Linux agent is 2Ghz even though the CPU is not running at that speed you will need to edit the xgrid.config.xml file (See Appendix 7 for further details)

Appendix 6 - Xgrid Linux Agent Segfault Issues

The next issue I ran into was that of the XGrid agent failing to run. I would set the agent to run at the controller's IP address, and unfortunately it would crash with a segmentation fault just after entering the `newnode()` function.

```
[optio@localhost xgridagent-1.0.2]$ ./xgridagent 192.168.0.102
11/05/04 10:48:45 [2662] Initialized pending tasks
11/05/04 10:48:45 [2662] Entering newnode()
Segmentation fault
```

This was very perplexing and after stepping through the debug by running:

```
[optio@localhost xgridagent-1.0.2]$ gdb ./xgridagent 192.168.0.102
```

I then found a number of issues were presenting themselves. Below is the result from running the debug on the application.

```
This GDB was configured as "i386-redhat-linux-gnu"...Using host
libthread_db
library "/lib/tls/libthread_db.so.1".
```

```
(gdb) run 192.168.0.102
Starting program: /home/optio/xgridagent-1.0.2/xgridagent
192.168.0.102
Error while mapping shared library sections:
: Success.
Error while reading shared library symbols:
: No such file or directory.
[Thread debugging using libthread_db enabled]
[New Thread -150736768 (LWP 3101)]
Error while reading shared library symbols:
```

: No such file or directory.

Error while reading shared library symbols:

: No such file or directory.

[New Thread -150738000 (LWP 3104)]

11/05/04 11:37:27 [3101] Initialized pending tasks

11/05/04 11:37:27 [3101] Entering newnode()

[New Thread -161244240 (LWP 3105)]

Program received signal SIGSEGV, Segmentation fault.

[Switching to Thread -150736768 (LWP 3101)]

0x004a1d6c in memcpy () from /lib/tls/libc.so.6

(gdb) bt

#0 0x004a1d6c in memcpy () from /lib/tls/libc.so.6

#1 0x00257f9d in xmlStrncat (cur=0x804d5d2 "",

add=0x8086b30 "<?xml version=\"1.0\"?>\n<!DOCTYPE plist PUBLIC
\"-//Apple

Computer//DTD PLIST 1.0//EN\"

\"><http://www.apple.com/DTDs/PropertyList-1.0.dtd>\")>\n<plist>\n<dict>\n

<key>correlationID</key>\n <string>0</strin"...,

len=764) at xmlstring.c:460

#2 0x0804c7be in ConvertLineBreaksToNewBuffer (

inBuf=0x8086b30 "<?xml version=\"1.0\"?>\n<!DOCTYPE plist PUBLIC
\"-//Apple

Computer//DTD PLIST 1.0//EN\"

\"><http://www.apple.com/DTDs/PropertyList-1.0.dtd>\")>\n<plist>\n<dict>\n

<key>correlationID</key>\n <string>0</strin"...,

inLength=764) at xgridagent.c:1645

#3 0x0804c838 in WriteXMLToTextBuffer (inMessage=0x804d5d1)

```

at xgridagent.c:1675

#4 0x0804c268 in RegistrationRequestMessage (outMessage=0x-
feec31a0)

at xgridagent.c:1372

#5 0x08049ecf in xgridagent (simple=0x8082550, error=0xfeec324c)

at xgridagent.c:227

#6 0x080499d0 in main (argc=2, argv=0x804d5d1) at
xgridagent.c:51

```

As we can see here on Line #1 the function `xmlStrncat` is broken. This function can be found in the `xgridagent.c` file and is solely responsible for the crash. After doing some research, and many thanks to the mac users group at North Carolina State University I found that this function is deprecated and needed to be removed. The original section of the file we were concerned with looked like this:

```

if USE_XML_STRNCAT_NEW

tmp =
(xmlChar*)xmlStrncatNew(BAD_CAST("\n"),BAD_CAST(inBuf),inLength);

else

tmp =
(xmlChar*)xmlStrncat(BAD_CAST("\n"),BAD_CAST(inBuf),inLength);

endif

```

Unfortunately the `xmlStrncatNew` never gets called because it is not defined. Thusly it runs the old deprecated function and causes the program to crash. By commenting out those lines and forcing the program to use the new function this can be solved:

```

/*#if USE_XML_STRNCAT_NEW*/

tmp =
(xmlChar*)xmlStrncatNew(BAD_CAST("\n"),BAD_CAST(inBuf),inLength);

/*#else

```

```
tmp =  
(xmlChar*)xmlStrncat(BAD_CAST("\n"),BAD_CAST(inBuf),inLength);  
#endif*/
```

The agent then ran and I was able to successfully pass a simple test to the grid which ran and returned the name of every computer connected to the grid at that time. The total time that it took to get the agent working was around 7 hours.

Appendix 7 - Editing the xgrid.config.xml File

Xgrid utilizes the XML language as a method for storing settings and preferences about each agent that connects to the controller. When an agent connects, it passes some of this relevant information to the controller such as the agent's name and its CPU speed. On OS X this is done automatically, however on Linux there is no such luck and the configuration must be edited manually in order to set these values. XML stores variables as key, value pairs and as such we are looking to edit the ServiceName and MaximumCPUPower string values. It is necessary to edit this file in order to correctly setup each Linux agent that connects to the controller. If this is not done the controller will crash because there will be more than one agent trying to access the grid with the same name. It is also worth while to note here that the debug level can also be set here so that an administrator can monitor the status of the currently running agent. Adjusting the MessageLevel string will specify what level of debug should be outputted. Below is the Xgrid Linux agents' configuration file:

```
<plist version="1.0">
<dict>
    <key>ServiceName</key>
    <string>Linux Client 1.0</string>
    <key>MaxTaskCount</key>3
    <string>1</string>
    <key>MaximumCPUPower</key>
    <string>2000</string>
    <key>MessageLevel</key>
    <string>3</string>
</dict>
</plist>
```

Appendix 8 - Automating the Installation of the Agent, DarwinPorts, and Pov-Ray

As I had never created an automated Linux installation script of any kind, this was a bit of a major undertaking and involved considerable learning. Constant thought had to be given to how best to break the configuration into several steps, whilst ensuring that each step provided adequate feedback so that if a particular section were to fail, the user could simply re-run that section of the installation script.

One of the issues in designing these scripts was the concept of the “working directory.” I often found myself trying to keep track of where certain things were being stored in the filesystem, and how I could design the system so that it would be simple to move swiftly through the file hierarchy. I decided that a “src” folder would be placed within the users home directory to which all libraries and files would be downloaded, extracted, and installed. Sometimes the script would fail at multiple points with very terse error messages, and it would require going through the script step by step until the error was found. It would have been impossible to document every little challenge I had in creating this script as the list would be endless, but this is the first script in the entire world that automates the installation of the Xgrid agent on both the Fedora Core 3 and Ubuntu platforms and I think that says enough in itself.

In order to begin the installation process two things must be done. First APT must be downloaded and installed. APT⁷ is built into Ubuntu, but is also available for Fedora Core 3 at <http://apt.freshrpms.net> . For the context of this project I downloaded apt and made it available on my own server here: <http://www.obelix.ca/xgrid-files/apt-i386.rpm> ,so that all my machines would have immediate fast access to the download. Once this is downloaded it can be installed by issuing the command:

```
sudo rpm -ivh apt-i386.rpm
```

⁷ See Glossary [3]

Once this is done Curl which is a command line utility for downloading remote files, must be downloaded. This can be done by running the following commands:

```
sudo apt-get update
sudo apt-get install curl
```

Once this is done I used the curl command to retrieve the first scrip,make it executable, and run it like so:

```
curl -O http://obelix.ca/xgrid-scripts/getAllScripts
chmod +x getAllScripts
./getAllScripts
```

The first script I created was called “getAllScripts” and its job was to retrieve all the subsequent scripts that would be required to install everything. This was done by creating a bash shell script with the following contents:

```
#!/bin/bash
curl -O http://www.obelix.ca/xgrid-scripts/downloadScript
curl -O http://www.obelix.ca/xgrid-scripts/StageOneInstallScript
curl -O http://www.obelix.ca/xgrid-scripts/StageTwoInstallScript
curl -O
http://www.obelix.ca/xgrid-scripts/StageThreeInstallScript
echo "Now making these executable...."
chmod +x downloadScript
chmod +x StageOneInstallScript
chmod +x StageTwoInstallScript
chmod +x StageThreeInstallScript
sleep 2
echo "Starting downloading script to install dependencies"
sleep 2
```

```
./downloadScript
```

The last line in this script will start the downloadScript which will download and install all the dependencies that are needed to install the Xgrid agent, DarwinPorts, and Pov-Ray. Determining these dependencies was a matter of sifting through the configuration outputs of every single compile, and ensuring that each dependency was added to the script. The downloadScript is listed below:

```
#!/bin/bash

##First get all the necessary libraries and programs we need this
isn't

##always necessary and largely depends on your linux distro

##You will need to install apt in order for this to work. This is
##available here: http://apt.freshrpms.net/

echo "Getting everything we need to install Xgrid agent and Dar-
winports"

sleep 5

echo "First updating APT"

sleep 1

sudo apt-get update

echo "Now upgrading existing repositories"

sleep 2

sudo apt-get upgrade

echo "Now installing necessary components of Xgrid and DPorts"

sleep 1

sudo apt-get install automake
sudo apt-get install autoconf
sudo apt-get install libtool
sudo apt-get install flex
sudo apt-get install bison
```

```
sudo apt-get install gcc
sudo apt-get install g++
sudo apt-get install libgtk1.2-dev
sudo apt-get install libpng-dev
sudo apt-get install curl
sudo apt-get install libxml2
sudo apt-get install glib2
sudo apt-get install gawk
sudo apt-get install perl
sudo apt-get install perl-dev
sudo apt-get install indent
sudo apt-get install g77
sudo apt-get install gtk2-devel
sudo apt-get install openssl
sudo apt-get install libssl0.9.7
sudo apt-get install libssl-dev
sudo apt-get install tcl
sudo apt-get install tcl-devel
sudo apt-get install tcl8.4
sudo apt-get install tcl8.4-dev
sudo apt-get install libpgtcl-dev
sudo apt-get install libpgtcl-dev
sudo apt-get install cvs
sudo apt-get install ssl-cert
sudo apt-get install make
sudo apt-get install python
sudo apt-get install python-dev
sudo apt-get install gettext
```

```
sudo apt-get install patch
sudo apt-get install xlibs-dev

sleep 5
##Now make the directory that we will install everything to
echo "Now making the /src directory in your home folder..."
sleep 3
mkdir ~/src
cd ~/src

echo "Now downloading the necessary Xgrid files from
Obelix.ca..."

sleep 2
##Download the packages that XGrid requires
curl -O http://www.obelix.ca/xgrid-files/glib-2.4.1.tar
curl -O http://www.obelix.ca/xgrid-files/libxml2-2.6.9.tar
curl -O http://www.obelix.ca/xgrid-files/roadrunner-0.9.1.tar
curl -O http://www.obelix.ca/xgrid-files/xgrid-patched.tar
curl -O http://www.obelix.ca/xgrid-files/.bashrc

echo "....."
sleep 1

echo "Now starting the StageOneInstallScript press ctrl-c to can-
cel if
you do not wish to continue"

sleep 10

cd ~

./StageOneInstallScript
```

The purpose of the “sleep” command in the above script is just so that the program will hang for a few seconds and allow the user to view the installation process at key points. The downloadScript will also sleep for 10 seconds at the end of the script so that the user can cancel out if a manual install is preferred, otherwise the StageOneInstallScript is started.

The StageOneInstallScript basically deals exclusively with the setup of the Linux Xgrid agent. This script first extracts all the libraries and Xgrid to a src directory, and then compiles it all.

```
#!/bin/bash

echo "Now extracting the tar files."

cd ~/src

sleep 5

##Extract these packages

tar -vxf libxml2-2.6.9.tar

echo "Done for libxml2"

sleep 2

tar -vxf glib-2.4.1.tar

echo "Done for glib2"

sleep 2

tar -vxf roadrunner-0.9.1.tar

echo "Done for roadrunner"

sleep 2

tar -vxf xgrid-patched.tar

echo "Done for all"

sleep 5

##Setup libxml

echo "Now setting up libxml"
```

```
sleep 2

cd libxml2-2.6.9

./configure --prefix=$HOME

make

make install

echo "Done for libxml"

sleep 2

echo "Now setting up glib2"

sleep 2

##Setup glib

cd ~/src/glib-2.4.1

./configure --prefix=$HOME

make

make install

echo "Done for glib2"

sleep 2

echo "Now I am setting the pkg_config_path and the ld_library_path"

sleep 1

export PKG_CONFIG_PATH=$PKG_CONFIG_PATH:$HOME/lib/pkgconfig

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/lib

sleep 3

echo "Now setting up roadrunner"

sleep 2

##Setup roadrunner
```

```

cd ~/src/roadrunner-0.9.1
./configure --prefix=$HOME
make
make install
echo "Done for all"
sleep 2

echo "Now setting up XgridAgent"
sleep 2
##Setup XGridAgent
cd ~/src/xgridagent-1.0.1
#cp -R xgridagent-1.0.1 ../
#cd ../xgridagent-1.0.1/
./configure --prefix=$HOME \
--with-roadrunner-includedir=$HOME/include/roadrunner-1.0 \
--with-roadrunner-libdir=$HOME/lib
echo "Done for all"
sleep 2

echo "....."
sleep 1

echo "Now starting the StageTwoInstallScript press ctrl-c to cancel if
you do not wish to continue"
sleep 10
cd ~
./StageTwoInstallScript

```

Once the StageOneInstallScript has completed the StageTwoInstallScript will begin. In this stage the environment variables are specified, so that the agent will run. Without these environment variable the agent will crash at runtime. It is best to also add the environment variables to the bashrc file. Below is the StageTwoInstallScript:

```
#!/bin/bash

cd ~/src/xgridagent-1.0.1

sleep 5

#Note if you download the non patched xgrid you will need to edit
it

#Add this to ~/.bashrc

#export PKG_CONFIG_PATH=$PKG_CONFIG_PATH:$HOME/lib/pkgconfig

#export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/lib

curl -O http://obelix.ca/xgrid-scripts/bashrc

#cp bashrc ~/.bashrc

sleep 2

echo "Line added to the bashrc file: export
PKG_CONFIG_PATH=$PKG_CONFIG_PATH:$HOME/lib/pkgconfig"

echo "Line added to the bashrc file: export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/lib "

sleep 2

echo "Starting make"

make

sleep 1
```

```

echo "Now you have installed the Xgrid agent barring no errors"
echo "You can run it by navigating to ~/src/xgridagent-1.0.1"
echo "Then run ./xgridagent 192.168.0.100 substitute your IP"
sleep 8

echo "....."

sleep 1

echo
"*****"
"*****"

echo "Now starting the StageThreeInstallScript this will install
DarwinPorts and Pov-Ray"

echo
"*****"
"*****"

echo "Note: this is not required to run Xgrid it is merely an ad-
dition"

echo "DarwinPorts is useful because we can setup our system to
match OS X \

    very closely"

echo " "

echo "Press ctrl-c in the next 60 seconds to cancel if you do not
wish to continue"

sleep 60

cd ~

sudo ./StageThreeInstallScript

```

Lastly this will call the StageThreeInstallScript which is responsible for the setup of DarwinPorts and Pov-Ray. It is necessary to run this script as the superuser because there are certain processes within the script that require the permissions of the superuser. It is also

important to not that the group “wheel” is created in this script because without it, DarwinPorts cannot install. Below is the StageThreeInstallScript:

```
#!/bin/bash

##Now its time to install darwinports and Pov-Ray

##we need to add the wheel group to the system so that we can install

##darwinports properly, only do this if you are using a debian based

##distribution

#If you run into the following:

#/home/user/darwinports/bin # ./port install Pov-Ray

#can't find package darwinports

#   while executing

#"Package require darwinports"

#   (file "./port" line 34)

sudo addgroup --system wheel

sudo chmod -R 777 /usr/local

mkdir /usr/local/src

cd /usr/local/src

#Login to CVS

echo "*****"

echo "The password to the CVS is blank so just hit enter"

echo "*****"

sleep 3

cvs -d
:pserver:anonymous@anoncvs.opendarwin.org:/Volumes/src/cvs/od \

login
```

```

echo "Successfully logged in, now about to download..."

sleep 2

#Downloads darwinports to /opt/local/src via CVS

cvs -d
:pserver:anonymous@anoncvs.opendarwin.org:/Volumes/src/cvs/od \
    co -P darwinports

sleep 1

echo "Downloads Done, install commencing"

sleep 1

#default prefix
PREFIX="/opt/local"

echo "Now getting ready to configure, press ctrl-c to stop"

sleep 10

cd darwinports/base

./configure --prefix="$PREFIX" --mandir="$PREFIX/share/man" \
    CPPFLAGS=-I/usr/include/openssl

make all

sudo make install

export PATH=$PATH:/opt/local/bin #This can go in .bashrc too

cd /opt/local/bin

./port install Pov-Ray

```

Once this has completed the user should have a completely functional version of Xgrid, DarwinPorts, and Pov-Ray without having to type a single command. This has not only

allowed me to distribute my setup on my 4 PC's but if other systems were added to my grid it would be very simple to set them up with access.

For copies see the following files on the project CD-ROM:

- *bashrc*
- *downloadScript*
- *getAllScripts*
- *StageOneInstallScript*
- *StageTwoInstallScript*
- *StageThreeInstallScript*

Appendix 9 - Custom Pov-Ray Libraries

The default installation of Pov-Ray lacks many of the custom libraries that SU2Pov (Sketchup to Pov-Ray) requires. Several steps were taken in setting up the distribution of these libraries:

- Since Sketchup and the necessary libraries were only installed to my PowerBook, I first tarred the povray include directory located in
`/opt/local/share/povray-3.6/include`
- This tar was then uploaded to my online web-server using an FTP program for OS X called Transmit
- A bash script called “povray-includes” was created to download the new includes, remove the old includes, and extract the tar of the new includes in their place. The script is detailed below:

```
#!/bin/bash
```

```
cd /opt/local/share/povray-3.6
```

```
sudo curl -O http://obelix.ca/povray/include.tar
sudo rm -rf /opt/local/share/povray-3.6/include
sudo tar -xvf include.tar
```

- SSH sessions were then started with every agent on the network, and the following commands were run:

```
curl -O http://obelix.ca/povray/povray-includes
chmod +x povray-includes
./povray-includes
cd /opt/local/share/povray-3.6/includes
cp LENS.INC lens.inc
```

Once this was completed each system was individually tested by submitting a small sample Xgrid job that ran a Pov-Ray command and utilized some of these libraries.

Appendix 10 - The Rendering Script

Below is the bash script called “render_pov.sh” that was used to handle the rendering of Pov-ray scenes without crashing the controller. The script uses my own local Apache 2 web server to download from, and calls an upload script to handle the ftp process.

```
#!/bin/bash
filename=$1
dir=/tmp/pov-Script

MACHINE_TYPE=`uname -s`

if [ "$MACHINE_TYPE" == "Darwin" ]; then
    echo "Yay, you're running a Macintosh"
    echo "This is the $filename argument"
```

```

    pwd

    curl -O
http://192.168.0.110/xgrid/povray-files/\$filename".pov"

    /opt/local/bin/povray $filename".pov" +W640 +H480

    sleep 1

    pwd

    ./ftpUpload $filename".png"
elif [ "$MACHINE_TYPE" == "Linux" ]; then

    mkdir /tmp/pov-script

    chmod 777 /tmp/pov-script

    echo "Yay, you're running a Linux box"

    pwd

    cd /tmp/pov-script

    curl -O http://192.168.0.110/xgrid/ftp-client/ftpUpload

    chmod +x ftpUpload

    curl -O
http://192.168.0.110/xgrid/povray-files/\$filename.pov

    /opt/local/bin/povray $filename.pov +W640 +H480 +A
    &>$filename.txt

    sleep 1

    pwd

    ./ftpUpload $filename.png

    ./ftpUpload $filename.txt

    echo "Hopefully finished rendering scripts, they are avail-
able on the server"

fi

```

The FTP script used called from within the render_pov.sh script handles the FTP uploading, and takes a filename as an argument. It was necessary to have the FTP script separate as it would produce end of file exceptions if it was included within the context of the last script and I could not figure out a way to resolve the issue.

```
#!/bin/bash
uploadFilename=$1
_host=192.168.0.110
_user=optix
_pass=34123

ftp -n $_host <<EOF
user $_user $_pass
cd xgrid/results/scenes
put $uploadFilename
bye
EOF
```

Originally I did not have to use the FTP commands and was using CURL to upload everything, but I found that occasionally CURL would fail to connect to the server, and was far less stable in general when uploading. The command I used to do this was:

```
curl -T $filename".png" -u user:pass -O \  
ftp://obelix.ca/public_html/xgrid/$filename".png"
```

Appendix 11 - Custom Apache2.conf

Below is the customized Apache2.conf file used throughout the course of the project. Of particular note here is the directives used in creating a CGI-bin which is used by the Bloxom web blog which will handle the output of the Pov-ray renders. All traffic to the web-server is denied to anyone outside of the local area network for security reasons, and FancyIndexing is enabled to permit much more robust display of directories and their contents.

```
ServerRoot "/usr/local/apache2"

DocumentRoot "/Library/WebServer/Documents"

PidFile "/private/var/run/httpd.pid"

ErrorLog logs/error_log

ServerAdmin stuart@obelix.ca

ServerName 192.168.0.110

User nobody

Group #-1

Listen 80

ScriptAlias /cgi-bin/ "/Library/WebServer/CGI-Executables/"

DirectoryIndex index.html index.htm index.php index.php3
welcome.html welcom.htm index.html.en

TypesConfig conf/mime.types

DefaultType text/plain

#LoadModules

LoadModule access_module modules/mod_access.so

LoadModule auth_module modules/mod_auth.so

LoadModule mime_module modules/mod_mime.so

LoadModule autoindex_module modules/mod_autoindex.so

LoadModule cgi_module modules/mod_cgi.so
```

```
LoadModule dir_module modules/mod_dir.so
LoadModule userdir_module modules/mod_userdir.so
LoadModule alias_module modules/mod_alias.so
LoadModule php4_module modules/libphp4.so
AddType application/x-httpd-php .php

#Directories
<Directory "/Library/WebServer/Documents">
    IndexOptions FancyIndexing VersionSort FoldersFirst
    AllowOverride None
    Order deny,allow
    Deny from all
    Allow from 127.0.0.1
    Allow from 192.168.0
</Directory>
<Directory "/Library/Webserver/CGI-Executables">
    AddHandler cgi-script .cgi .pl
    Options +ExecCGI
    Order deny,allow
    Deny from all
    Allow from 127.0.0.1
    Allow from 192.168.0
</Directory>
```

Appendix 12 - Gantt Chart

See the following page for a full Gantt chart and timeline of product deliverables.

ID	Task Name	Jan 2, '05	Jan 16, '05	Jan 30, '05	Feb 13, '05	Feb 27, '05	Mar 13, '05	Mar 27, '05
1	Cross Platform Grid Computing with Pov-Ray	2	6	10	14	18	22	26
2	Scope							
3	Determine project scope							
4	Define preliminary resources							
5	Define system requirements							
6	Scope complete							
7	Research							
8	Research Grid Computing							
9	Review Possible Grid Platforms							
10	Xgrid							
11	Condor							
12	Review Possible Linux Distributions							
13	Ubuntu							
14	Fedora Core 3							
15	ColLinux for Windows **							
16	Research Apache on OS X							
17	Research how to setup WebDav							
18	Determine Available Resources**							
19	Acquire necessary hardware and determine requirements							
20	Acquire all necessary software							
21	Create an Analysis of the XGrid Interface							
22	Testing Stage							
23	Install Linux on main testing PC							
24	Install both Ubuntu and Fedora and determine which will suit th							
25	Compile Xgrid Agent for Linux							
26	Install Xgrid							
27	Install Xgrid and setup Xgrid controller on OS X Server							
28	Test Xgrid Between 2 OS X Machines							
29	Test Xgrid in Linux							
30	Install WebDav running on an Apache Server running on OS X							
31	Create Remote Retrieval Script							
32	Create Pov-Ray Scene							
33	Test Pov-Ray							
34	Pov-Ray on OS X							
35	Pov-Ray on Linux							
36	Pov-Ray on Windows **							
37	Determine how to match Linux pov-ray structure with OS X str							
38								
39	Hardware Implementation							
40	Install and configure 12 port switch							
41	Linux Implementation							

END NOTES

1. Beck, A., (June 27 1997). High Throughput Computing: An Interview with Miron Livny. Retrieved March 4, 2005, from <http://www.cs.wisc.edu/condor/HPCwire.1>
2. Mannheim, U. O., Tennessee, U. O., (November 2004). Top 500 List 11/2004. Retrieved March 4, 2005, from <http://www.top500.org/sublist/System.php?id=6714>
3. Brown, M. C., Grid computing -- moving to a standardized platform. Retrieved February 24, 2005, from <http://www-128.ibm.com/developerworks/grid/library/gr-stanplat.html>
4. Saskatchewan, U. O., Further Key Developments. Retrieved February 24, 2005, from http://www.cs.usask.ca/resources/tutorials/csconcepts/2002_7/static/tutorial/introduction/history/keydevelop.html
5. Grid.org, GRID.ORG, Ñ - Grid Computing: The Evolution. Retrieved February 24, 2005, from <http://www.grid.org/about/gc/evolution.htm>
6. Grid.org, GRID.ORG, Ñ - Grid Computing: SETI@home. Retrieved February 24, 2005, from <http://www.grid.org/about/gc/seti.htm>
7. Today, G., GRIDtoday: SUN LIGHTS UP \$1/CPU/HR HOUR 'SUN GRID'. Retrieved February 24, 2005, from <http://www.gridtoday.com/05/0207/104571.html>

8. IBM, IBM Grid Computing - IBM and grid - Products and services. Retrieved February 24, 2005, from http://www-1.ibm.com/grid/about_grid/ibm_grid/products_services.shtml
9. Foley, M. J., (December 27 2004). A Peek Under Microsoft's Secret 'Bigtop'. Retrieved February 24, 2005, from <http://www.microsoft-watch.com/article2/0,1995,1746291,00.asp>
10. Globus, About the Globus Toolkit. Retrieved February 24, 2005, from <http://www-unix.globus.org/toolkit/about.html>
11. Team, C. P., What is Condor?. Retrieved February 24, 2005, from <http://www.cs.wisc.edu/condor/description.html>
12. Apple, Apple - Hardware - Video - Virginia Tech, Supercomputer. Retrieved February 2, 2005, from http://www.apple.com/hardware/video/virginiatech/virginiatech_480.html
13. 500.org, T., TOP500 List 11/2003. Retrieved February 2, 2005, from <http://www.top500.org/list/2003/11/>
14. Today, G., GRIDtoday: APPLE STORMS MARKET WITH XGrid. Retrieved April 1, 2005, from <http://www.gridtoday.com/04/0112/102479.html>
15. Guide to Apple Xgrid. Retrieved February 12, 2005, from <http://images.apple.com/acg/xgrid/pdf/xgridguide.pdf>
16. Parnot, C., Xgrid@Stanford. Retrieved February 24, 2005, from <http://cmgm.stanford.edu/~cparnot/Xgrid-stanford/html/projects/projects.html>

17. Warrene, B., Mac News: Hardware : Stanford University Lab Builds an Xgrid. Retrieved February 24, 2005, from <http://www.macnewsworld.com/story/35786.html>
18. Crandall, R., Studies in Epidemiology. Retrieved February 24, 2005, from <http://academic.reed.edu/epi/>
19. Carter, S., Wolfgrid: the ncsu community supercomputer. Retrieved April 3, 2005, from <http://packmug.ncsu.edu/wolfgrid/>
20. Adams, N., xgrid@sfu - Research. Retrieved April 3, 2005, from: <http://www.hpc.sfu.ca/xgridatsfu/research.php>
21. Cooper, A., (January 2004). The Inmates are Running the Asylum., 150. Retrieved March 13, 2005.
22. Cote, D., Simple.: XGrid agent for Unix architectures. Retrieved January 12, 2005, from <http://unu.novajo.ca/simple/archives/000026.html>

BIBLIOGRAPHY

- Adams, N., xgrid@sfu - Research. Retrieved April 3, 2005, from <http://www.hpc.sfu.ca/xgridatsfu/research.php>
- Apple, Apple - Hardware - Video - Virginia Tech, Supercomputer. Retrieved February 2, 2005, from http://www.apple.com/hardware/video/virginiatech/virginiatech_480.html
- Beck, A., (June 27 1997). High Throughput Computing: An Interview with Miron Livny. Retrieved March 4, 2005, from <http://www.cs.wisc.edu/condor/HPCwire.1>
- Brown, M. C., (n.d.). Grid computing -- moving to a standardized platform. Retrieved February 24, 2005, from <http://www-128.ibm.com/developerworks/grid/library/gr-stanplat.html>
- Carter, S., Wolfgrid: the ncsu community supercomputer. Retrieved February 20, 2005, from <http://packmug.ncsu.edu/wolfgrid/>
- Cooper, A., (January 2004). The Inmates are Running the Asylum., 150. Retrieved March 13, 2005.
- Cote, Daniel. XGrid agent for Unix architectures. <http://unu.novajo.ca/simple/archives/000026.html>
- Crandall, R., (n.d.). Xgrid: 2 types of biology. Retrieved February 24, 2005, from <http://lists.apple.com/archives/Xgrid-users/2004/Dec/msg00025.html>

Foley, M. J., (December 27 2004). A Peek Under Microsoft's Secret 'Bigtop'. Retrieved February 24, 2005, from <http://www.microsoft-watch.com/article2/0,1995,1746291,00.asp>

Globus, (n.d.). About the Globus Toolkit. Retrieved February 24, 2005, from <http://www-unix.globus.org/toolkit/about.html>

Grid.org, (n.d.). GRID.ORG , - Grid Computing: SETI@home. Retrieved February 24, 2005, from <http://www.grid.org/about/gc/seti.htm>

Grid.org, (n.d.). GRID.ORG , - Grid Computing: The Evolution. Retrieved February 24, 2005, from <http://www.grid.org/about/gc/evolution.htm>

Guide to Apple Xgrid. Retrieved February 12, 2005, from <http://images.apple.com/acg/xgrid/pdf/xgridguide.pdf>

IBM, (n.d.). IBM Grid Computing - IBM and grid - Products and services. Retrieved February 24, 2005, from http://www-1.ibm.com/grid/about_grid/ibm_grid/products_services.shtml

Mannheim, U. O., Tennessee, U. O., (November 2004). Charts for November 2004 - Application Area. Retrieved March 4, 2005, from <http://www.top500.org/lists/2004/11/charts.php?c=0>

Parnot, C., (n.d.). Xgrid@Stanford. Retrieved February 24, 2005, from <http://cmgm.stanford.edu/~cparnot/Xgrid-stanford/html/projects/projects.html>

Saskatchewan, U. O., (n.d.). Further Key Developments.

Retrieved February 24, 2005, from

http://www.cs.usask.ca/resources/tutorials/csconcepts/2002_7/static/tutorial/introduction/history/keydevelop.html

Team, C. P., (n.d.). What is Condor?. Retrieved February 24, 2005, from

<http://www.cs.wisc.edu/condor/description.html>

Today, G., GRIDtoday: SUN LIGHTS UP \$1/CPU/HR HOUR

'SUN GRID'. Retrieved February 24, 2005, from

<http://www.gridtoday.com/05/0207/104571.html>

—, GRIDtoday: APPLE STORMS MARKET WITH XGrid.

Retrieved February 24, 2005, from

<http://www.gridtoday.com/04/0112/102479.html>

Warrene, B., (n.d.). Mac News: Hardware : Stanford

University Lab Builds an Xgrid. Retrieved February 24, 2005,

from <http://www.macnewsworld.com/story/35786.html>