

A Holistic Approach to High-Performance Computing: Xgrid Experience

David Przybyla
Ringling School of Art and Design
2700 North Tamiami Trail
Sarasota, Florida 34234
941-309-4720
dprzybyl@ringling.edu

Karissa Miller
Ringling School of Art and Design
2700 North Tamiami Trail
Sarasota, Florida 34234
941-359-7670
kmiller@ringling.edu

Mahmoud Pegah
Ringling School of Art and Design
2700 North Tamiami Trail
Sarasota, Florida 34234
941-359-7625
mpegah@ringling.edu

ABSTRACT

The Ringling School of Art and Design is a fully accredited four-year college of visual arts and design. With a student to computer ratio of better than 2-to-1, the Ringling School has achieved national recognition for its large-scale integration of technology into collegiate visual art and design education. We have found that Mac OS X is the best operating system to train future artists and designers. Moreover, we can now buy Macs to run high-end graphics, nonlinear video editing, animation, multimedia, web production, and digital video applications rather than expensive UNIX workstations. As visual artists cross from paint on canvas to creating in the digital realm, the demand for a high-performance computing environment grows. In our public computer laboratories, students use the computers most often during the workday; at night and on weekends the computers see only light use. In order to harness the lost processing time for tasks such as video rendering, we are testing Xgrid, a suite of Mac OS X applications recently developed by Apple for parallel and distributed high-performance computing.

As with any new technology deployment, IT managers need to consider a number of factors as they assess, plan, and implement Xgrid. Therefore, we would like to share valuable information we learned from our implementation of an Xgrid environment with our colleagues. In our report, we will address issues such as assessing the needs for grid computing, potential applications, management tools, security, authentication, integration into existing infrastructure, application support, user training, and user support. Furthermore, we will discuss the issues that arose and the lessons learned during and after the implementation process.

Categories and Subject Descriptors

C.2.4 [Computer-Communication Networks]: Distributed Systems – distributed applications.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGUCCS'04, October 10–13, 2004, Baltimore, Maryland, USA.
Copyright 2004 ACM 1-58113-869-5/04/0010...\$5.00.

General Terms

Management, Documentation, Performance, Design, Economics, Reliability, Experimentation.

Keywords

Macintosh OS X, Xgrid, Grid Computing, Cluster, High-Performance Computing, Rendezvous, Rendering.

1. INTRODUCTION

Grid computing does not have a single, universally accepted definition. The technology behind grid computing model is not new. Its roots lie in early distributed computing models that date back to early 1980s, where scientists harnessed the computing power of idle workstations to let compute intensive applications to run on multiple workstations, which dramatically shortening processing times. Although numerous distributed computing models were available for discipline-specific scientific applications, only recently have the tools become available to use general-purpose applications on a grid. Consequently, the grid computing model is gaining popularity and has become a show piece of "utility computing". Since in the IT industry, various computing models are used interchangeably with grid computing, we first sort out the similarities and difference between these computing models so that grid computing can be placed in perspective.

1.1 Clustering

A cluster is a group of machines in a fixed configuration united to operate and be managed as a single entity to increase robustness and performance. The cluster appears as a single high-speed system or a single highly available system. In this model, resources can not enter and leave the group as necessary. There are at least two types of clusters: parallel clusters and high-availability clusters. Clustered machines are generally in spatial proximity, such as in the same server room, and dedicated solely to their task.

In a high-availability cluster, each machine provides the same service. If one machine fails, another seamlessly takes over its workload. For example, each computer could be a web server for a web site. Should one web server "die," another provides the service, so that the web site rarely, if ever, goes down.

A parallel cluster is a type of supercomputer. Problems are split into many parts, and individual cluster members are given part of the problem to solve. An example of a parallel cluster is

composed of Apple Power Mac G5 computers at Virginia Tech University [1].

1.2 Distributed Computing

Distributed computing spatially expands network services so that the components providing the services are separated. The major objective of this computing model is to consolidate processing power over a network. A simple example is spreading services such as file and print serving, web serving, and data storage across multiple machines rather than a single machine handling all the tasks. Distributed computing can also be more fine-grained, where even a single application is broken into parts and each part located on different machines: a word processor on one server, a spell checker on a second server, etc.

1.3 Utility Computing

Literally, utility computing resembles common utilities such as telephone or electric service. A service provider makes computing resources and infrastructure management available to a customer as needed, and charges for usage rather than a flat rate. The important thing to note is that resources are only used as needed, and not dedicated to a single customer.

1.4 Grid Computing

Grid computing contains aspects of clusters, distributed computing, and utility computing. In the most basic sense, grid turns a group of heterogeneous systems into a centrally managed but flexible computing environment that can work on tasks too time intensive for the individual systems. The grid members are not necessarily in proximity, but must merely be accessible over a network; the grid can access computers on a LAN, WAN, or anywhere in the world via the Internet. In addition, the computers comprising the grid need not be dedicated to the grid; rather, they can function as normal workstations, and then advertise their availability to the grid when not in use.

The last characteristic is the most fundamental to the grid described in this paper. A well-known example of such an “ad hoc” grid is the SETI@home project [2] of the University of California at Berkeley, which allows any person in the world with a computer and an Internet connection to donate unused processor time for analyzing radio telescope data.

1.5 Comparing the Grid and Cluster

A computer grid expands the capabilities of the cluster by loosing its spatial bounds, so that any computer accessible through the network gains the potential to augment the grid. A fundamental grid feature is that it scales well. The processing power of any machine added to the grid is immediately available for solving problems. In addition, the machines on the grid can be general-purpose workstations, which keep down the cost of expanding the grid.

2. ASSESSING THE NEED FOR GRID COMPUTING

Effective use of a grid requires a computation that can be divided into independent (i.e., parallel) tasks. The results of each task cannot depend on the results of any other task, and so the members of the grid can solve the tasks in parallel. Once the tasks have been completed, the results can be assembled into the

solution. Examples of parallelizable computations are the Mandelbrot set of fractals, the Monte Carlo calculations used in disciplines such as Solid State Physics, and the individual frames of a rendered animation. This paper is concerned with the last example.

2.1 Applications Appropriate for Grid Computing

The applications used in grid computing must either be specifically designed for grid use, or scriptable in such a way that they can receive data from the grid, process the data, and then return results. In other words, the best candidates for grid computing are applications that run the same or very similar computations on a large number of pieces of data without any dependencies on the previous calculated results. Applications heavily dependent on data handling rather than processing power are generally more suitable to run on a traditional environment than on a grid platform. Of course, the applications must also run on the computing platform that hosts the grid. Our interest is in using the Alias Maya application [3] with Apple’s Xgrid [4] on Mac OS X.

Commercial applications usually have strict license requirements. This is an important concern if we install a commercial application such as Maya on all members of our grid. By its nature, the size of the grid may change as the number of idle computers changes. How many licenses will be required? Our resolution of this issue will be discussed in a later section.

2.2 Integration into the Existing Infrastructure

The grid requires a controller that recognizes when grid members are available, and parses out job to available members. The controller must be able to see members on the network. This does not require that members be on the same subnet as the controller, but if they are not, any intervening firewalls and routers must be configured to allow grid traffic.

3. XGRID

Xgrid is Apple’s grid implementation. It was inspired by Zilla, a desktop clustering application developed by NeXT and acquired by Apple. In this report we describe the Xgrid Technology Preview 2, a free download that requires Mac OS X 10.2.8 or later and a minimum 128 MB RAM [5].

Xgrid, leverages Apple’s traditional ease of use and configuration. If the grid members are on the same subnet, by default Xgrid automatically discovers available resources through Rendezvous [6]. Tasks are submitted to the grid through a GUI interface or by the command line. A System Preference Pane controls when each computer is available to the grid.

It may be best to view Xgrid as a facilitator. The Xgrid architecture handles software and data distribution, job execution, and result aggregation. However, Xgrid does not perform the actual calculations.

3.1 Xgrid Components

Xgrid has three major components: the client, controller, and the agent. Each component is included in the default installation, and any computer can easily be configured to assume any role. In

fact, for testing purposes, a computer can simultaneously assume all roles in “local mode.” The more typical production use is called “cluster mode.”

The client submits jobs to the controller through the Xgrid GUI or command line. The client defines how the job will be broken into tasks for the grid. If any files or executables must be sent as part of a job, they must reside on the client or at a location accessible to the client. When a job is complete, the client can retrieve the results from the controller. A client can only connect to a single controller at a time.

The controller runs the GridServer process. It queues tasks received from clients, distributes those tasks to the agents, and handles failover if an agent cannot complete a task. In Xgrid Technology Preview 2, a controller can handle a maximum of 10,000 agent connections. Only one controller can exist per logical grid.

The agents run the GridAgent process. When the GridAgent process starts, it registers with a controller; an agent can only be connected to one controller at a time. Agents receive tasks from their controller, perform the specified computations, and then send the results back to the controller. An agent can be configured to always accept tasks, or to just accept them when the computer is not otherwise busy.

3.2 Security and Authentication

By default, Xgrid requires two passwords. First, a client needs a password to access a controller. Second, the controller needs a password to access an agent. Either password requirement can be disabled. Xgrid uses two-way-random mutual authentication protocol with MD5 hashes. At this time, data encryption is only used for passwords.

As mentioned earlier, an agent registers with a controller when the GridAgent process starts. There is no native method for the controller to reject agents, and so it must accept any agent that registers. This means that any agent could submit a job that consumes excessive processor and disk space on the agents. Of course, since Mac OS X is a BSD-based operating system, the controller could employ Unix methods of restricting network connections from agents.

The Xgrid daemons run as the user “nobody,” which means the daemons can read, write, or execute any file according to world permissions. Thus, Xgrid jobs can execute many commands and write to /tmp and /Volumes. In general, this is not a major security risk, but it does require a level of trust between all members of the grid.

3.3 Using Xgrid

3.3.1 Installation

Basic Xgrid installation and configuration is described both in Apple documentation [5] and online at the University of Utah web site [8]. The installation is straightforward and offers no options for customization. This means that every computer on which Xgrid is installed has the potential to be a client, controller, or agent.

3.3.2 Agent and Controller Configuration

The agents and controllers can be configured through the Xgrid Preference Pane in the System Preferences or XML files in /Library/Preferences. Here the GridServer and GridAgent processes are started, passwords set, and the controller discovery method used by agents is selected. By default, agents use Rendezvous to find a controller, although the agents can also be configured to look for a specific host.

The Xgrid Preference Pane also sets whether the Agents will always accept jobs, or only accept jobs when idle. In Xgrid terms, idle either means that the Xgrid screen saver has activated, or the mouse and keyboard have not been used for more than 15 minutes. Even if the agent is configured to always accept tasks, if the computer is being used these tasks will run in the background at a low priority.

However, if an agent only accepts jobs when idle, any unfinished task being performed when the computer ceases being idle are immediately stopped and any intermediary results lost. Then the controller assigns the task to another available member of the grid.

Advertising the controller via Rendezvous can be disabled by editing /Library/Preferences/com.apple.xgrid.controller.plist. This, however, will not prevent an agent from connecting to the controller by hostname.

3.3.3 Sending Jobs from an Xgrid Client

The client sends jobs to the controller either through the Xgrid GUI or the command line. The Xgrid GUI submits jobs via small applications called plug-ins. Sample plug-ins are provided by Apple, but they are only useful as simple testing or as examples of how to create a custom plug-in. If we are to employ Xgrid for useful work, we will require a custom plug-in.

James Reynolds details the creation of custom plug-ins on the University of Utah Mac OS web site [8]. Xgrid stores plug-ins in /Library/Xgrid/Plug-ins or ~/Library/Xgrid/Plug-ins, depending on whether the plug-in was installed with Xgrid or created by a user.

The core plug-in parameter is the “command,” which includes the executable the agents will run. Another important parameter is the “working directory.” This directory contains necessary files that are not installed on the agents or available to them over a network. The working directory will always be copied to each agent, so it is best to keep this directory small. If the files are installed on the agents or available over a network, the working directory parameter is not needed.

The command line allows the options available with the GUI plug-in, but it can be slightly more cumbersome. However, the command line probably will be the method of choice for serious work. The command arguments must be included in a script unless they are very basic. This can be a shell, perl, or python script, as long as the agent can interpret it.

3.3.4 Running the Xgrid Job

When the Xgrid job is started, the command tells the controller how to break the job into tasks for the agents. Then the command is tarred and gzipped and sent to each agent; if there is a working directory, this is also tarred and gzipped and sent to the agents.

The agents extract these files into /tmp and run the task. Recall that since the GridAgent process runs as the user nobody, everything associated with the command must be available to nobody.

Executables called by the command should be installed on the agents unless they are very simple. If the executable depends on libraries or other files, it may not function properly if transferred, even if the dependent files are referenced in the working directory.

When the task is complete, the results are available to the client. In principle, the results are sent to the client, but whether this actually happens depends on the command. If the results are not sent to the client, they will be in /tmp on each agent. When available, a better solution is to direct the results to a network volume accessible to the client.

3.4 Limitations and Idiosyncrasies

Since Xgrid is only in its second preview release, there are some rough edges and limitations. Apple acknowledges some limitations [7]. For example, the controller cannot determine whether an agent is trustworthy and the controller always copies the command and working directory to the agent without checking to see if these exist on the agent.

Other limitations are likely just a by-product of an unfinished work. Neither the client nor controller can specify which agents will receive the tasks, which is particularly important if the agents contain a variety of processor types and speeds and the user wants to optimize the calculations. At this time, the best solution to this problem may be to divide the computers into multiple logical grids. There is also no standard way to monitor the progress of a running job on each agent. The Xgrid GUI and command line indicate which agents are working on tasks, but gives no indication of progress.

Finally, at this time only Mac OS X clients can submit jobs to the grid. The framework exists to allow third parties to write plug-ins for other Unix flavors, but Apple has not created them.

4. XGRID IMPLEMENTATION

Our goal is an Xgrid render farm for Alias Maya. The Ringling School has about 400 Apple Power Mac G4's and G5's in 13 computer labs. The computers range from 733 MHz single-processor G4's and 500 MHz and 1 GHz dual-processor G4's to 1.8 GHz dual-processor G5's. All of these computers are lightly used in the evening and on weekends and represent an enormous processing resource for our student rendering projects.

4.1 Software Installation

During our Xgrid testing, we loaded software on each computer multiple times, including the operating systems. We saved time by facilitating our installations with the remote administration daemon (radmind) software developed at the University of Michigan [9], [10].

Everything we installed for testing was first created as a radmind base load or overload. Thus, Mac OS X, Mac OS X Developer Tools, Xgrid, POV-Ray [11], and Alias Maya were stored on a radmind server and then installed on our test computers when needed.

4.2 Initial Testing

We used six 1.8 GHz dual-processor Apple Power Mac G5's for our Xgrid tests. Each computer ran Mac OS X 10.3.3 and contained 1 GB RAM. As shown in Figure 1, one computer served as both client and controller, while the other five acted as agents.

Before attempting Maya rendering with Xgrid, we performed basic calculations to cement our understanding of Xgrid. Apple's Xgrid documentation is sparse, so finding helpful web sites facilitated our learning.

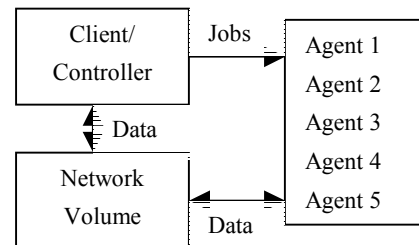


Figure 1. Xgrid test grid.

We first ran the Mandelbrot set plug-in provided by Apple, which allowed us to test the basic functionality of our grid. Then we performed benchmark rendering with the Open Source Application POV-Ray, as described by Daniel Côté [12] and James Reynolds [8]. Our results showed that one dual-processor G5 rendering the benchmark POV-Ray image took 104 minutes. Breaking the image into three equal parts and using Xgrid to send the parts to three agents required 47 minutes. However, two agents finished their rendering in 30 minutes, while the third agent used 47 minutes; the entire render was only as fast as the slowest agent.

These results gave us two important pieces of information. First, the much longer rendering time for one of the tasks indicated that we should be careful how we split jobs into tasks for the agents. All portions of the rendering will not take equal amounts of time, even if the pixel size is the same. Second, since POV-Ray cannot take advantage of both processors in a G5, neither can an Xgrid task running POV-Ray. Alias Maya does not have this limitation.

4.3 Rendering with Alias Maya 6

We first installed Alias Maya 6 for Mac OS X on the client/controller and each agent. Maya 6 requires licenses for use as a workstation application. However, if it is just used for rendering from the command line or a script, no license is needed. We thus created a minimal installation of Maya as a radmind overload. The application was installed in a "hidden" directory inside /Applications. This was done so that normal users of the workstations would not find and attempt to run Maya, which would fail because these installations are not licensed for such use.

In addition, Maya requires the existence of a directory ending in the path /maya. The directory must be readable and writable by the Maya user. For a user running Maya on a Mac OS X workstation, the path would usually be ~/Documents/maya. Unless otherwise specified, this directory will be the default location for Maya data and output files. If the directory does not

exist, Maya will try to create it, even if the user specifies that the data and output files exist in other locations.

However, Xgrid runs as the user nobody, which does not have a home directory. Maya is unable to create the needed directory, and looks instead for /Alias/maya. This directory also does not exist, and the user nobody has insufficient rights to create it. Our solution was to manually create /Alias/maya and give the user nobody read and write permissions.

We also created a network volume for storage of both the rendering data and the resulting rendered frames. This avoided sending the Maya files and associated textures to each agent as part of a working directory. Such a solution worked well for us because our computers are geographically close on a LAN; if greater distance had separated the agents from the client/controller, specifying a working directory may have been a better solution.

Finally, we created a custom GUI plug-in for Xgrid. The plug-in command calls a Perl script with three arguments. Two arguments specify the beginning and end frames of the render and the third argument the number of frames in each job (which we call the “cluster size”). The script then calculates the total number of jobs and parses them out to the agents. For example, if we begin at frame 201 and end at frame 225, with 5 frames for each job, the plug-in will create 5 jobs and send them out to the agents.

Once the jobs are sent to the agents, the script executes the /usr/sbin/Render command on each agent with the parameters appropriate for the particular job. The results are sent to the network volume.

With the setup described, we were able to render with Alias Maya 6 on our test grid. Rendering speed was not important at this time; our first goal was to implement the grid, and in that we succeeded.

4.3.1 Pseudo Code for Perl Script in Custom Xgrid Plug-in

In this section we summarize in simplified pseudo code format the Perl script used in our Xgrid plug-in.

```
agent_jobs{
  • Read beginning frame, end frame, and cluster size of
  render.
  • Check whether the render can be divided into an integer
  number of jobs based on the cluster size.
  • If there are not an integer number of jobs, reduce the cluster
  size of the last job and set its last frame to the end frame of
  the render.
  • Determine the start frame and end frame for each job.
  • Execute the Render command.
}
```

4.4 Lessons Learned

Rendering with Maya from the Xgrid GUI was not trivial. The lack of Xgrid documentation and the requirements of Maya combined into a confusing picture, where it was difficult to decide the true cause of the problems we encountered. Trial and error was required to determine the best way to set up our grid.

The first hurdle was creating the directory /Alias/maya with read and write permissions for the user nobody. The second hurdle was learning that we got the best performance by storing the rendering data on a network volume.

The last major hurdle was retrieving our results from the agents. Unlike the POV-Ray rendering tests, our initial Maya results were never returned to the client; instead, Maya stored the results in /tmp on each agent. Specifying in the plug-in where to send the results would not change this behavior. We decided this was likely a Maya issue rather than an Xgrid issue, and the solution was to send the results to the network volume via the Perl script.

5. FUTURE PLANS

Maya on Xgrid is not yet ready to be used by the students of Ringling School. In order to do this, we must address at least the following concerns.

- Continue our rendering tests through the command line rather than the GUI plug-in. This will be essential for the following step.
- Develop an appropriate interface for users to send jobs to the Xgrid controller. This will probably be an extension to the web interface of our existing render farm, where the student specifies parameters that are placed in a script that issues the Render command.
- Perform timed Maya rendering tests with Xgrid. Part of this should compare the rendering times for Power Mac G4's and G5's.

6. CONCLUSION

Grid computing continues to advance. Recently, the IT industry has witnessed the emergence of numerous types of contemporary grid applications in addition to the traditional grid framework for compute intensive applications. For instance, peer-to-peer applications such as Kazaa, are based on storage grids that do not share processing power but instead an elegant protocol to swap files between systems. Although in our campuses we discourage students from utilizing peer-to-peer applications from music sharing, the same protocol can be utilized on applications such as decision support and data mining. The National Virtual Collaboratory grid project [13] will link earthquake researchers across the U.S. with computing resources, allowing them to share extremely large data sets, research equipment, and work together as virtual teams over the Internet.

There is an assortment of new grid players in the IT world expanding the grid computing model and advancing the grid technology to the next level. SAP [14] is piloting a project to grid-enable SAP ERP applications, Dell [15] has partnered with Platform Computing to consolidate computing resources and provide grid-enabled systems for compute intensive applications, Oracle has integrated support for grid computing in their 10g release [16], United Devices [17] offers hosting service for grid-on-demand, and Sun Microsystems continues their research and development of Sun's N1 Grid engine [18] which combines grid and clustering platforms.

Simply, the grid computing is up and coming. The potential benefits of grid computing are colossal in higher education learning while the implementation costs are low. Today, it would be difficult to identify an application with as high a return on investment as grid computing in information technology divisions in higher education institutions. It is a mistake to overlook this technology with such a high payback.

7. ACKNOWLEDGMENTS

The authors would like to thank Scott Hanselman of the IT team at the Ringling School of Art and Design for providing valuable input in the planning of our Xgrid testing. We would also like to thank the posters of the Xgrid Mailing List [13] for providing insight into many areas of Xgrid.

8. REFERENCES

- [1] Apple Academic Research, <http://www.apple.com/education/science/profiles/vatech/>.
- [2] SETI@home: Search for Extraterrestrial Intelligence at home. <http://setiathome.ssl.berkeley.edu/>.
- [3] Alias, <http://www.alias.com/>.
- [4] Apple Computer, *Xgrid*, <http://www.apple.com/acg/xgrid/>.
- [5] *Xgrid Guide*, <http://www.apple.com/acg/xgrid/>, 2004.
- [6] Apple Mac OS X Features, <http://www.apple.com/macosx/features/rendezvous/>.
- [7] Xgrid Manual Page, 2004.
- [8] James Reynolds, *Xgrid Presentation*, University of Utah, <http://www.macos.utah.edu:16080/xgrid/>, 2004.
- [9] Research Systems Unix Group, *Radmind*, University of Michigan, <http://rsug.itd.umich.edu/software/radmind>.
- [10] *Using the Radmind Command Line Tools to Maintain Multiple Mac OS X Machines*, <http://rsug.itd.umich.edu/software/radmind/files/radmind-tutorial-0.8.1.pdf>.
- [11] POV-Ray, <http://www.povray.org/>.
- [12] Daniel Côté, *Xgrid example: Parallel graphics rendering in POVray*, <http://unu.novajo.ca/simple/>, 2004.
- [13] NEESgrid, <http://www.neesgrid.org/>.
- [14] SAP, <http://www.sap.com/>.
- [15] Platform Computing, <http://platform.com/>.
- [16] Grid, <http://www.oracle.com/technologies/grid/>.
- [17] United Devices, Inc., <http://ud.com/>.
- [18] N1 Grid Engine 6, <http://www.sun.com/software/gridware/index.html>.
- [19] Xgrid Users Mailing List, <http://www.lists.apple.com/mailman/listinfo/xgrid-users/>.