Building Computational Grids with Apple's Xgrid Middleware

Baden Hughes

Department of Computer Science and Software Engineering
The University of Melbourne,
Parkville VIC 3010, Australia
Email: badenh@csse.unimelb.edu.au

Abstract

Apple's release of the Xgrid framework for distributed computing introduces a new technology solution for loosely coupled distributed computation. In this paper systematically describe, compare and evaluate the Apple native Xgrid solution and a range of third party components which can be substituted for the native versions. This description and evaluation is grounded in practical experience of deploying a small scale, internationally distributed, heterogenous computational infrastructure based on the Xgrid framework.

Keywords: Xgrid, Apple, computational grid, middleware

1 Introduction

The release in 2005 of Apple's Xgrid framework for grid introduces a new technology solution for loosely coupled, distributed computation. Xgrid has been widely promoted as an extremely usable solution for less technical user communities and challenges the systems management paradigm incumbent in many computational grid solutions currently deployed. As such, the uptake of grid computing by ad hoc groups of researchers with non-dedicated infrastructure using the Xgrid framework has been significant, in numerical terms and in terms of the visibility of the solution. A particular point to note is that the simple Xgrid framework has the potential to fundamentally change the delineation between grid users and grid maintainers, and as such to promote new types of research enabled by a self-sustaining model for managing computational grid infrastructure.

For researchers in the grid computing space and for systems manageers of production grid facilities, Xgrid has often been viewed as a toy solution. This paper seeks to counter this perception in two ways: first by describing the Xgrid architecture and its components in detail, and secondly by adopting an analysis model more prevalent in the grid computing domain. A notable point here is that this paper does not simply cover the Apple distributed native Xgrid components but also considers a range of third party components which can be used to extend the Xgrid framework in directions more amenable to the types of production environments currently occcupied by solutions such as the widely used Globus toolkit.

The structure of this paper is as follows. First we consider the overall positioning of the Apple Xgrid solution, and its high level systems architecture. Next we review in depth the Xgrid architecture and components officially distributed by Apple in Xgrid 1.0. Following this we will review a range of interoperable third party components that can be used to complement or replace the proprietary Apple components under certain circumstances. We report experience in using a heterogenous Xgrid to perform some experiments in natural language processing; and report on a range of other production uses of Xgrid based on published papers and user group surveys. Finally we conduct an evaluation of the overall strengths and weaknesses of the Xgrid solution and consider the niche(s) into which Xgrid based solutions may effectively be deployed and offer some concluding thoughts.

It is worth stating upfront that this paper specifically does not seek to either conduct empirical performance comparisons between Xgrid and alternative grid computing solutions nor to report the results of a specific scientific experiment which is enabled by Xgrid-based infrastructure. Rather the purpose of this paper is to describe, and where relevant compare and evaluate the overall Xgrid architecture and its components from a functional perspective.

2 A Brief History of Xgrid

Xgrid was first introduced by Appple in January 2004 as a Technology Preview (TP1). Xgrid TP1 was considered as a proof of concept, and was not designed for production applications owing to reliability, security and scalability issues. Rather it was designed to draw feedback from early adopters as to the viability of an Apple grid computing product.

Xgrid Technology preview (TP2) was released in November 2004, retaining most of the functionality of TP1, but with some underlying CLI and data format changes. This was a very widely adopted release, Xgrid based computational environments were deployed for production use, and third party components began to emerge.

Xgrid 1.0 was released with Mac OS X 10.4 'Tiger' in April 2005. 1.0 introduced a significant number of changes. Perhaps controversially, Xgrid 1.0 included a dependency on Mac OS X Server (TP1 and TP2 did not require a server grade operating system), which allowed Apple to leverage the significant investment it had made in Mac OS X Tiger Server in the areas of scalability, single sign on, job specific authorization, server local and remote administration, server grade documentation, and the inclusion of a GUI based interaction model (TP1 and TP2 only had a CLI).

3 Solution Architecture

The Apple Xgrid architecture is a standard three tier architecture consisting of a Client, Controller and Agent. We will review each of these tiers in turn. The Controller, Agent and Client can all exist on a single machine, although in practice, these are more typically distributed.

3.1 Client

An Xgrid client provides the user interface to an Xgrid system. The client is responsible for finding a suitable controller, submitting a jobs, and retrieving the results. Clients can rely on the controller to mediate all job submissions; they do not need to be aware of the actual job execution schedule across available computational agents. It is useful to note that an Xgrid client is detachable from the network even while jobs are being executed - completed jobs are retrieved by the client from the controller once network connectivity is re-established.

3.2 Controller

The controller, representing the middle tier, is the centre of the Xgrid framework. A controller typically runs on a dedicated system (like a cluster head node). The controller handles receiving jobs from clients, dividing them into tasks to execute on various agents, and collecting and returning the results.

3.3 Agent

The final tier in the Xgrid solution is the Agent. Typically there is one agent per compute node, not dissimilar to other grid computing frameworks, although natively Xgrid agents on dual-CPU nodes default to accepting one task per CPU (similar policies are often implemented in a cluster LRMS). In similarity with other computational grid solutions, Xgrid allows for a range of agent types ranging from full-time agents, part-time (cycle stealing) agents, and remote agents (for distributed computational grids).

4 Xgrid Systems Configuration and Management

The client, controller and agent software ships standard with Apple's Mac OS X 10.4 (Tiger) operating system. As such, the systems management task is largely configuration oriented rather than installation oriented. A simplified configuration can be setup in less than 5 minutes, significantly reducing the barrier to entry for less technical users.

Desktop systems or dedicated cluster nodes can be configured and enabled as Xgrid agents either locally through the Sharing pane in their System Preferences application, or remotely via SSH or one of Apples network-based desktop management tools (Apple Remote Desktop, NetBoot, or Network Install). For larger installations als Xgrid supports the Apple NetBoot service which allows for nodes to load a standard configured operating system image from a central server (similar to the types of systems typically used for cluster management).

Mac OS X Server systems are configured and enabled as Xgrid agents or controllers through the standard Server Admin application suite. Server installations provide the host infrastructure to manage authentication, which can include none, shared password, or Single Sign On (using a Kerberized service such as LDAP or Active Directory). Notably in this area Xgrid does not adopt the X.509 certificate based authentication prevalent in other computational grid middleware suites. Service control is also enabled via the Server Admin environment - in addition, there is a command line control interface for Xgrid, which allows service management (with the exception of authentication policies) from a shell environment.

The discovery mechanism in Xgrid is that both clients and agents natively search for a controller. The controller is the only Xgrid component that requires an open TCP port for such discovery requests. All communications between clients, controllers and agents are able to be strongly encrypted over the network ensuring in transit security across segregated administrative domains.

Xgrid can automatically discover available resources on a local network via a number of Apple services including ZeroConf, Rendezvous or Bonjour. Discovery is recursive within a domain or subdomain or Xgrid configurations can be created by manually entering IP addresses or hostnames.

5 Xgrid Job Management

Xgrid jobs are expressed in Apple's standard `plist` format (Apple, 2005a). Further details of the expression are provided in a later section in the context of the Xgrid Command Line Interface.

Jobs can be submitted to the Xgrid controller using a range of client tools (discussed below). In all cases, an accompanying job specification is used by the controller to determine whether (and how) to decompose a submitted the job, the code and data payloads for a given job, and whether the job submission is to be synchronous or asynchronous. Xgrid controllers schedule jobs in the order they are received, assigning each task to the fastest agent available at that time (determined by active probing of the current computational load of each agent). Alternatively, the job specification allows for dependencies among jobs and tasks to be expressed to ensure scheduling in the proper order. In most cases, if any job or task fails, the scheduler will automatically resubmit it to the next available agent. Xgrid tasks using password-based authentication will minimize possible interactions with the rest of the system by executing as unprivileged Xgrid users (by default the system user nobody in the system's /tmp directory). Tasks using single sign-on for both clients and agents will run as the submitting user, allowing appropriate access to local files or network services.

Because of the three-tier architecture of Xgrid, clients can submit jobs to the controller asynchronously, then disconnect from the network as mentioned earlier. The controller will cache all the required data, manage scheduling and task-level failover, then hold and return all the results, including standard output and standard error from each task. If requested, the controller will notify the user via email that the job has been completed. The user can then retrieve the results from any authenticated Xgrid client system using the relevant job ID.

6 Xgrid System Scalability

Apple's published scalability benchmarks are relatively modest: Xgrid 1.0 has been tested on configurations of up to 128 agents; 20,000 queued jobs (or 100,000 tasks per job); 2Gb submitted data per job; 1Gb results per task; 10Gb aggregate results per job. Most notably, this testing is in a cluster (rather than distributed) mode.

Some of these benchmarks have been exceeded, and are discussed further in the section entitled 'Other Use Cases' elsewhere in this paper.

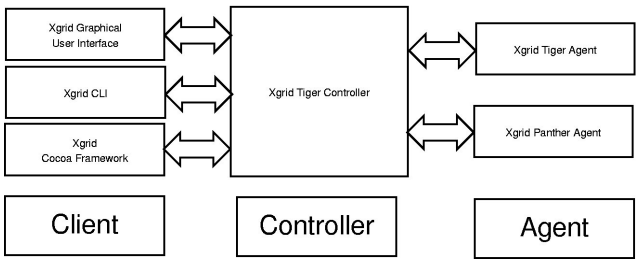

Figure 1: The Native Xgrid Architecture

7 Xgrid Native Components

Having described the high level architecture of Xgrid, in this section we review the official Xgrid architecture and components as distributed in Xgrid 1.0. A diagrammatic representation can be seen in Figure 1. Here we proceed to describe the Xgrid clients, controller and agents in that order.

7.1 Xgrid Graphical User Interface

Mac OS X ships with a simple GUI based interface to Xgrid called `Xgrid.app`, which supports only a few basic functions: view the Xgrid Agents and some basic properties for each Agent from a single Controller; submit single task jobs via a GUI; retrieve job status and results. This being said, `Xgrid.app` is a demonstrator application, and it is not promoted for tasks more complicated than basic testing. Other interfaces to be discussed next and in the third party section are much more fully featured.

7.2 Xgrid Command Line Interface

The Xgrid framework has a well documented Command Line Interface (CLI) The Xgrid command line client is simply called `xgrid`. `xgrid` takes a number of parameters vis: executable and the input file or input file directory and the output file or output file directory Executables do not have to be supplied as transfer objects, they can be instantiated from a locally installed version of the executable. The executable (if required) and the file or directory is copied to the Agent, after which it is executed. Std error and result streams are returned to the controller after execution. Naturally such a simple set of parameters make it easy to wrap the CLI directly in a language such as Unix shell.

The CLI can be instantiated in two different ways depending on whether syncrhonous or asynchronous execution is required.

For synchronous exection: `xgrid -h $HOSTNAME -job run -in $INPUTFILE -out $OUTPUT` where `$HOSTNAME` is the fully qualified host name and domain; `$INPUTFILE` is the input file or directory containing the payload data; and `$OUTPUTFILE` is the output file or directory for the results.

For asynchronous execution: `xgrid -h FQHOSTNAME -job submit -in $INPUTFILE -out $OUTPUT` where the parameters are the same as for synchronous, although the command is changed from `run` to `submit`.

The job submission process returns a numerical job identifier ($ID), which can then be used to query status of the job on the computational grid, vis: `xgrid -job attributes -id $ID`

The output of a status query includes a number of fields and corresponding values, vis: `activeCPUpower` (number of CPU cycles currently consumed by the job); `applicationIdentifier` (name of the application which instantiated the job); `dateNow` (current date and time); `dateStarted` (date and time job was started); `dateStopped` (date and time the job finished); `dateSubmitted` (date and time the job was submitted); `jobStatus` (either running or finished); `name` (the executable name); `percentDone` (a numerical indicator of progress); `taskCount` (the number of tasks done); and `undoneTaskCount` (the number of tasks pending).

The results of jobs can be retrieved again using the job identifier $ID, vis: `xgrid -job results -id $ID`.

In addition to singular task jobs, the Xgrid CLI also supports multitask jobs by virtue of a batch mode which supports job specification via a `plist` (property list file similar in type to a the more commonly used plan file in other grid environments) (Apple, 2005a).

An example `plist` is shown below, this executes the Unix calendar program to generate the March 2005 output.

```
{
jobSpecification = {
applicationIdentifier = "com.apple.xgrid.cli";
inputFiles = {};
name = "Calendar";
submissionIdentifier = calendar;
taskSpecifications = {
0 = {arguments = (3, 2005);
command = "/usr/bin/cal";};
};
};

}
```

The CLI syntax for instantiation is slightly different with the use of a plist: `xgrid -job batch $PLISTFILE`. Likewise, a different status output from the Xgrid CLI in response to a query; (`xgrid -job specification -id $ID`) is as follows: `applicationIdentifier` (name of the application which instantiated the job) `inputFiles` (input data files) `name` (the executable name) `submissionIdentifier` (name of the plist submission which instantiated the job) `taskSpecifications` (a task list consisting of arguments and commands) `arguments` (any relevant arguments for the executable) `command` (the executable)

The plist can be extended with multiple arguments or commands or both as necessary, each task with an arbitrary identifier.

A `plist` can also be expressed as XML which adds a degree of flexibility as to how these job specifications are created. A `plist` expressed in XML is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC
"-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
 <key>jobSpecification</key>
 <dict>
  <key>applicationIdentifier</key>
  <string>com.apple.xgrid.cli</string>
  <key>inputFiles</key>
  <dict/>
  <key>name</key>
  <string>Calendar</string>
  <key>submissionIdentifier</key>
  <string>calendar</string>
  <key>taskSpecifications</key>
```

```
  <dict>
   <key>0</key>
   <dict>
    <key>arguments</key>
    <array>
    <string>3</string>
    <string>2005</string>
    </array>
    <key>command</key>
    <string>/usr/bin/cal</string>
   </dict>
  </dict>
 </dict>
</plist>
```

7.3 Xgrid Cocoa Framework

In order to support developers building native Mac OS X GUI based applications, Apple provides `XGridFoundation`, a Mac OS X Cocoa framework based on ObjectiveC. `XGridFoundation` can be used by any Cocoa application by including the relevant header files eg.`#import <XGridFoundation/XGridFoundation.h>`.

In order to understand the underlying architecture of the Xgrid framework, and because such frameworks are not particularly common among grid computing middleware suites, we consider in detail the specific classes exposed by `XGridFoundation`.

First the classes themselves are described, in approximately the order they are encountered in a typical application.

- `XGConnection` is used to represent a connection to an Xgrid server. It can be initialized with a host name, or via Bonjour.
- `XGAuthenticator` is at connection instantiation, which requires some form of authentication with the Xgrid server. `XGAuthenticator` is an abstract class whose subclasses are used by an XGConnection to authenticate.
- `XGTwoWayRandomAuthenticator` is a subclass of `XGAuthenticator` which used to perform password based authentication.
- `XGGSSAuthenticator` is a subclass of `XGAuthenticator` which authenticates with Single Sign-On (eg LDAP).
- `XGController` Instances of this class are proxies for Xgrid controllers. They are initialized with an `XGConnection`, and are used to submit jobs.
- `XGActionMonitor` is a class used to monitor the activity of some asynchronous requests, such as submitting a job via an XGController.
- `XGResource` is an abstract class which remote-grid resources, like grids and jobs. Instances of subclasses of `XGResource` are proxies for entities on the Xgrid server.
- `XGGrid` is a subclass of XGResource which represents the grids on the Xgrid controller.
- `XGJob` is a subclass of `XGResource` which represents the jobs running on an Xgrid controller.
- `XGFile` is a file or stream that is stored on the Xgrid controller.
- `XGFileDownload` is a class used to retrieve files and streams from the Xgrid controller after a job is complete.

A complete description of all Cocoa classes and methods can be found in (Apple, 2005b). A significant point of differentiation between Xgrid and other grid computing middleware suites is that the programmatic framework is fully vendor supported by Apple.

7.4 Xgrid Controller

The Xgrid Controller has the primary task of managing communications between the various components of the Xgrid. It interfaces with the Clients (as described above) and the Agents (as described later).

The controller interprets the job submissions from the client, and decomposes them as appropriate, then instantiates execution by transferrring the job to the next available Agent. In addition, the Controller monitors each Agent directly, and determines availablity to be based on the current CPU consumption on a given Agent with the lowest CPU consumption indicating the next available Agent. There can only be one Controller per logical grid, although each controller can have an arbitrary number of Agents connected to it.

A Controller is typically hosted in a high availability network location, having a fixed IP address with access over TCP port 4111 required to be open for Agent and Client communication. Typically the Controller is installed on the same subnet as Agents and Clients, facilitiating discovery using one of the various Apple resolution protocols.

The Xgrid Controller has a command line administration tool `xgridctl` which can be used to start, stop and restart the Controller instance. This tool is very similar to the widely used `apachectl`, and can also be used to control a local agent via a simple command line switch.

In addition to the command line administration of the Controller, Apple distributes the Xgrid Admin tool, which is a GUI administration tool for Xgrid, and is usable on any Mac running an Xgrid Client, Agent or Controller. The Xgrid Admin tool allows a user to login to a Controller and monitor its activities, including measuring its CPU capacity, reviewing pending, active and completed jobs; querying agents for their status and job progress etc.

7.5 Xgrid Tiger Agent

The Xgrid Agent actually executes the computational tasks specified by a job. An Agent can belong to one virtual organisation at a time, by virtue of a relationship with a Controller. By default, an Agent will seek to bind to the first available Controller on a network, although this can be overwritten with a manual directive. The Xgrid Tiger Agent defaults to "part-time" mode, only accepting jobs if there has been no keyboard activity for 15 minutes on the host on which it is installed. Alternatively, an Agent can be configured to act as a dedicated node.

Agent behaviour on an individual machine is subject to the usual Unix based resource allocation mechanisms - they can be jailed or chrooted, have storage, CPU or memory quotas set, and be monitored via standard system utilities.

7.6 Xgrid Panther Agent

The Xgrid Panther Agent is identical to the Xgrid Tiger Agent above, except that it runs on machines installed with Mac OS X 10.3 (Panther). This solution is Apple's advocated solution for enabling legacy Mac OS X based systems to be utilized in computational grids.

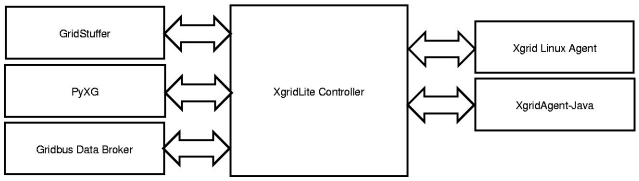

Figure 2: Xgrid Architecture with Third Party Components

7.7 Xgrid Network Communications

The communications protocol in Xgrid is built on BEEP, the Blocks Extensible Exchange Protocol (Rose, 2001). BEEP is an IETF standard similar to HTTP, but designed for two way multiplexed communication particularly for peer to peer environments. BEEP uses XML profiles to define multiple channels over a single socket, reducing negotiation overhead and thus network utilization.

8 Third Party Components

Next we turn to a discussion of a number of third party components which are interoperable with the Apple Xgrid framework. It should be noted that these third party components can be combined with the native components described earlier. Diagramatically, the third party components can be see in Figure 3. Again we proceed by discussing clients, controllers and agents in that order.

8.1 GridStuffer

GridStuffer (Parnot, 2005) is a Cocoa application which allows the submission of multi-task jobs in a simple manner which closely resembles the xgrid CLI format. The underlying idea of GridStuffer is emulation of the the batch `plist` format without the complexity of working with the `plist` itself, and is motivated by the need to to provide a simple GUI and easy result retrieval. The application uses Apple's Core Data framework to store information about job progress (allowing jobs to be restarted from a checkpoint). GridStuffer is being used by the Xgrid-Stanford project to submit large numbers of jobs each of several days duration, and is hence robustly engineered after significant text cycles.

A notable contribution of GridStuffer is the concept of a "metajob", a construct which may represent a number of smaller jobs. A metajob simply consists of a list of shell commands defined in an input file. One metajob can consist of thousands of commands and the status for each individual command can be monitored independently. Each command can be run more than once, is validated based on its output and will be automatically rescheduled in case of job failures. For each command, GridStuffer keeps track of the number of successes, the number of failures, and will consider a command 'Completed' or 'Dismissed' based on user-defined thresholds.

Technically, a metajob also consists of one or several Xgrid jobs, which themselves are made from one or several of the commands listed in the input file. GridStuffer automatically creates new Xgrid jobs as needed. The number of commands in one job is user definable. More Xgrid jobs are sent only as needed, on a regular basis. For instance, a metajob can consist of 1,000,000 commands, but only batches of 1000 are really submitted at a given time to the Xgrid controller. The Xgrid jobs are identified using the indices of the submitted commands The Xgrid jobs are automatically removed from the grid when they finish or fail.

The results of each command are automatically loaded to the local disk of the client instantiating GridStuffer. GridStuffer can handle this either directly using the standard Mac OS X Finder style directory structure; or by using flags in the input file, can use user preferences about output location.

Unlike the native Xgrid client components, GridStuffer allows connection to several Controllers at the same time. Any Controllers found on the local network (via Bonjour; restricted to the host subnet) will automatically be located and listed. Additionally, the address of a Controller can be manually provided (for cases where the Controller is on a different subnet). All the jobs are randomly sent to only one of the connected Controllers, and all jobs in a metajob are sent to the same Controller. (The author of GridStuffer advises that in the next version, jobs will be able to be submitted to several grids and several controllers simultaneously - a metajob will simply be divided in chunks and sent to the different servers, optionally based on the number of available agents.)

8.2 PyXG

PyXG (Granger, 2005) provides a Python interface to an Xgrid, allowing users to submit and manage Xgrid jobs on a cluster from a Python script or within an interactive Python session. PyXG only works with Mac OS X 10.4 (Tiger) Clients and requires PyObjC.

The main functions of PyXG are as follows. Xgrid executions can be instantiated from Python scripts and Python interactive sessions. Single task and batch Xgrid jobs can be submitted and managed from within Python; available grids and their corresponding status can be queried. Active Xgrid jobs can be listed, their status queried, and administrative actions such as delete, restart etc can be issued to jobs.

The Python to Xgrid communication in PyXG is implemented through a set of Python classes that wrap the Xgrid command line directly. Thus, all `xgrid` parameters are available to PyXG. The Cocoa API is not used in PyXG. (The author of PyXG advises that a version of PyXg with Cocoa support is currently under development).

8.3 Gridbus Data Broker Interface

Experimental support for Xgrid instantiation has been introduced in the Gridbus Data Broker (Venugopal, Buyya and Winton, 2004) since the release of version 2.2 (Assuncao et al, 2005). In essence this support equates to an XML job specification transformation from the internal Gridbus Data Broker's format to match the Apple `plist` format, and a wrapper around the Xgrid command line interface to facilitate execution. The interested reader is referred to Assuncao et al (2005) for a more detailed treatment of the implementation and experimental evaluation.

8.4 XgridLite

XgridLite (Baskerville, 2005) is an extension for Mac OS X 10.4 (Tiger) which emulates a full Controller on the standard version of Mac OS X Tiger. Hence, XgridLite is a drop in replacement for Mac OS X 10.4 (Tiger) Server's Xgrid Controller, albeit with some feature reduction.

The main features of XgridLite are the ability to manage the status of the Xgrid Controller directly; to set passwords for client and agent authentication; and administratively reset the Controller to default

settings. Unlike the other third party components, XgridLite is not free, but is nominally priced shareware.

8.5 Xgrid Linux Agent

In addition to third party components which provide alternative client interfaces to the Xgrid Controller, there is also an Xgrid Linux Agent (Cote, 2004) which extends the flexibility of an Xgrid-based computational grid to allow for alternative operating systems such as Linux. The Xgrid Linux Agent was one of the first third party components available for Xgrid. The Xgrid Linux Agent compiles on a range of Linux and Unix variants including Debian, RedHat, Solaris and OpenDarwin.

Using the Xgrid Linux Agent, an native Xgrid Controller is required. Additionally, the application instances need to be constructed in such a way that they are either aware of the multi-architecture nature of the computational grid, or so that they are architecture independent.

A notable point is also that the Xgrid Linux Agent only supports operation in the passwordless authentication mode.

8.6 XgridAgent-Java

XgridAgent-Java (Campbell, 2005) is a pure Java Agent for Xgrid written entirely in Java. The primary motivation of this project is to provide a platform independent Xgrid agent, allowing for heterogenous Xgrid clusters to be deployed. XgridAgent-Java utilises a number of open source components including JmDNS (van Hoff and Blair, 2005), BEEPCore-Java (Franklin, 2005) and Base64 (Brower, 2005). XgridAgent-Java supports dynamic resolution of an Xgrid cluster controller via a range of Apple supported resource discovery services (either ZeroConf or Rendezvous or Bonjour), all via JmDNS. BEEPCore-Java is used to handle the BEEP layer of the Xgrid protocol. Base64 is used to handle binary elements in the Apple XML plist layer of the protocol.

9 Experimental Experience

In evaluating the Xgrid framework, we have deployed three small scale, al grids. The specifications for each of the relevant grids is described in the tables below, along with the location information for each node.

9.1 Infrastructure

In Grid 1 (Figure 3), a small cluster configuration is the infrastructure model selected, with native Xgrid components being deployed.

In Grid 2 (Figure 4), we use the same cluster arrangment, hardware and operating system as in Grid 1, except we replace the native Xgrid components with relevant third-part components.

In Grid 3 (Figure 5), a range of processor architectures and operating systems are featured on the Xgrid, demonstrating the considerable flexibility in building heterogenous Xgrid's using third-party components. Grid 3 is also distributed geographically, with infrastructure in multiple Australian locations, in Europe and in the USA.

A point of considerable interest in the specifications of these test grids is that despite Xgrid being an Apple product, through the utilisation of 3rd party middleware bindings, the Xgrid environment can be deployed across multiple hardware and software platforms.

In essence the difference between Grid 1, Grid 2 and Grid 3 is that Grid 1 was deployed using the native Xgrid components; Grid 2 was deployed using third party components over the same hardware as Grid 1, whereas Grid 3 was deployed using a combination of native and third party Xgrid 1.0 components over variable hardware and operating systems. More specifically in Grid 1 we used the native Xgrid Command Line Interface as the Client; the native Xgrid Controller as the Controller; and the native Xgrid Agent for the Agents. Contrastively in Grid 2 we used GridStuffer as the Client, the XgridLite Controller, and XgridAgent-Java for the Agents. In Grid 3 PyXG was used as the Client; the native Xgrid Controller was used as the Controller; and both Xgrid Linux Agent and XgridAgent-Java were used for the Agents.

9.2 Experiments

The experimental grids were used to perform a natural language processing task. This paper does not intendto report detailed metrics for this particular task, but rather used it as a method of testing the Xgrid framework.

For reference the experimental task is to to annotate the English Gigaword Corpus (Graff, 2002) with Part of Speech (POS) tags. The analysis and annotation is performed using the Python-based Natural Language Toolkit (Bird and Loper, 2004). The task is embarassingly parallel in the dimension of the corpus segmentation: 314 individual files with an average size of 38 Mb per file; the total corpus is 12Gb in size. The Natural Language Toolkit is installed locally on each of the agent systems, with the input data and a processing script transferred from the client to the controller to the agents, and back at the end of the experiment.

On Grid 1 (the cluster configuration with native Xgrid components) processing the entire end to end task took approximately 380 minutes of wall clock time to complete (with network transfer overhead being very low, owing to the Client being on the same LAN as the Controller and Agents).

On Grid 2 (the cluster configuration with third-party components) processing the entire end to end task took approximately 402 minutes of wall clock time. Again the network transfer overhead being very low, owing to the Client being on the same LAN as the Controller and Agents.

On Grid 3 (the larger distributed heterogenous configuration) processing the entire end to end took approximately 670 minutes of wall clock time. The network transfer overhead in this context was considerably higher given the international transfers required to move data to the nodes in the USA and Europe). However, because the computational grid consisted of more nodes, naturally a greater degree of parallelism was achieved.

It is interesting to note that there is only a statistically insignificant difference between elapsed time between Grid 1 and Grid 2, despite the substitution of third party components in place of native Xgrid components. Detailed instrumentation was not implemented, but anecdotal evidence suggests that the overall performance of Grid 2 using third party components is very comparable to the native Xgrid alternative. The distributed nature of the task in Grid 3 increases the overall time required to complete the task, which is not to be unexpected given the network transfer time for 12Gb of data, some of which is transferred to locations in the USA and Europe.

While the purpose of this paper is specifically not to report experimental results, it is important to note that this scale of task can be robustly completed using either the native Xgrid components or the third-

Hardware	OS	Role	Software	Location
1 x G5 PPC	OSX 10.4 Server	Controller	Xgrid Controller	Melbourne - CSSE
4 x G4 PPC	OSX 10.4	Agent	Xgrid Tiger Agent	Melbourne - CSSE
1 x G3 PPC	OSX 10.4	Client	Xgrid Tiger Client	Melbourne - CSSE

Figure 3: Experimental Grid 1

Hardware	OS	Role	Location	
1 x G5 PPC	OSX 10.4 Server	Controller	XgridLite Controller	Melbourne - CSSE
4 x G4 PPC	OSX 10.4	Agent	XgridAgent-Java	Melbourne - CSSE
1 x G3 PPC	OSX 10.4	Client	GridStuffer	Melbourne - CSSE

Figure 4: Experimental Grid 2

party Xgrid components. Subsequent production experiments in natural language processing (see Hughes et al 2005) utilise Grids 2 and 3.

10 Other Use Cases

An interesting metric by which to assess the success or otherwise of the Xgrid framework is the degree to which it has been adopted. An informal survey on the xgrid-users list in July 2005 revealed a range of application domains and infrastructure models, summarised below. For reference there were 11 responses to the survey which were sent to the list directly, other private replies to the Xgrid product manager may have been received.

The majority of users who responded are using Xgrid 1.0 (63%) with the remainder using Technology Preview 2 with forward migration plans (37%). The size of computational grids ranged up to 300 part time (ie cycle stealing) nodes; for full time nodes the largest grid had 60 compute nodes. The usage domains included graphics rendering, spatial analysis, population genetics, natural language processing, bioinformatics, spatial biochemical modeling and sensitivity analysis, cryptography, and Monte Carlo simulations. Over two-thirds of respondents indicated that their grids were in production, rather than development. The majority of users were using the Apple provided CLI, although a significant number were evaluating the GridStuffer and PyXG applications.

A further sample of the types of projects using Xgrid for scientific computation includes the following: modeling of biochemical receptors (Stanford University); nonlinear-system computations for an epidemiological model (Center for Advanced Computation at Reed College); rendering POV-Ray animations of LDraw models (University of Utah Student Computing Labs) finding low autocorrelation binary sequences (Simon Fraser University); Monte Carlo simulations of biophotonic tissue analyses (Ontario Cancer Institute at the University of Toronto). Additional academic papers which involve the use of Xgrid in "real science" are also available from `http://www.apple.com/acg/`, and cover a range of scientific domains.

11 Discussion and Conclusion

This paper has offered a range of observations on specific issues with the Xgrid platform throughout. In conclusion, it is useful to generalise these observations into the relative strengths and weaknesses of the Xgrid platform.

On the positive side, Xgrid represents a very low barrier to entry for grid computing - with simple setup and adminstration - allowing for a new range of users to effectively access the power of distributed, loosely coupled computational environments. The fact that Xgrid is already shipped standard with the operating system is a distinct further advantage. Increasingly too, native Xgrid support is being offered by application vendors (with application domains ranging from digital image processing to mathematics to bioinformatics), which allows end users to rely on vendor support directly for embarassingly parallel computations.

This is not to say that the Xgrid framework does not have some (arguably considerable) disadvantages. Xgrid also introduces another authentication paradigm, while not incompatible with the well entrenched X.509 frameworks given appropriate middleware, are likely to represent problems for the connection of Xgrid based infrastructure with the broader grid communities. Another explicit weakness in the Xgrid framework to date is the lack of a cross platform client implementation - even clients provided by third parties currently just wrap the existing CLI. As such, Xgrid, while open at the Agent tier, is not open at the Client tier, binding users to a Mac OS X client. For extensibility of the Xgrid framework into the wider grid computing community, this issue must be addressed. Furthermore, the lack of any explicit agent capability specification, while grounded in Apple's intention that the Xgrid framework only be deployed over its own hardware is a significant shortcoming.

While clearly not a of the same level of maturity as other widely utilised grid middleware such as Globus, the Apple Xgrid product does provide a number of advantages (particularly in the area of ease of use), which we believe will lead to widespread ad-hoc adoption within research communities and as such represents a significant advance. Typical of Apple's engagement with a wide range of applications, Xgrid vastly simplifies the task of building a computational grid and using a computational grid to execute tasks.

References

Apple Computer Inc, 2005a. Property List Format. `http://developer.apple.com/documentation/Darwin/Reference/ManPages/man5/plist.5.html` Last visited 21 September 2005.

Apple Computer Inc, 2005b. XgridFoundation Reference. `http://developer.apple.com/documentation/Performance/Conceptual/XgridDeveloper/index.html` Last visited 21 September 2005.

Apple Computer Inc, 2005. Xgrid. `http://www.apple.com/macosx/features/xgrid/` Last visited 21 September 2005.

Apple Computer Inc, 2005. Mac OS X Server Xgrid Administration Guide. `http://images.apple.com/server/pdfs/Xgrid_Admin_v10.4.pdf` Last visited 21 September 2005.

Hardware	OS	Role	Location	
1 x G5 PPC	OSX 10.4	Controller	Xgrid Controller	Melbourne - CSSE
1 x G3 PPC	OSX 10.4	Client	PyXG	Melbourne - CSSE
4 x G4 PPC	OSX 10.4	Agent	Xgrid Tiger Agent	Melbourne - CSSE
1 x G5 Intel	OSX 10.4	Agent	Xgrid Tiger Agent	Melbourne - MARCC
1 x P4 i386	Linux	Agent	XgridAgent-Java	Melbourne - CSSE
1 x P4 i386	Linux	Agent	XgridAgent-Java	Melbourne - MARCC
2 x P4 i386	Windows XP SP2	Agent	XgridAgent-Java	Melbourne - CSSE
1 x P4 i386	Linux	Agent	XgridAgent-Java	Brisbane
1 x AMD i386	Linux	Agent	XgridAgent-Java	Houston, USA
1 x P4 i386	Linux	Agent	XgridAgent-Java	Edinburgh, UK
1 x P4 i386	Linux	Agent	XgridAgent-Java	Tuscson, USA
1 x P4 i386	Linux	Agent	XgridAgent-Java	Aarhus, Denmark

Figure 5: Experimental Grid 3

Marcos Dias de Assuncao, Krishna Nadiminti, Srikumar Venugopal, Tianchi Ma, and Rajkumar Buyya, 2005. An Integration of Global and Enterprise Grid Computing: Gridbus Broker and Xgrid Perspective. Proceedings of the 4th International Conference on Grid and Cooperative Computing. Lecture Notes on Computer Science XXXX. Springer-Verlag. pp. XX–YY.

Robert Brower, 2005. Base64 Encode/Decode Utility. `http://sourceforge.net/projects/base64` Last visited 21 September 2005.

Ed Baskerville, 2005. XgridLite. `http://edbaskerville.com/software/xgridlite/` Last visited 21 September 2005.

Huston Franklin, 2005. BeepCore-Java. `http://sourceforge.net/projects/beepcore-java` Last visited 21 September 2005.

Steven Bird and Edward Loper, 2004. NLTK: The Natural Language Toolkit. Proceedings of the ACL 2004 Demonstration Session. Association for Computational Linguistics. pp. 214–217.

Marshall Rose, 2001. The Blocks Extensible Exchange Protocol (BEEP). IETF RFC 3080. `http://www.faqs.org/rfcs/rfc3080.html`

Curtis Campbell, 2005. XgridAgent-Java. `http://www.apple.com/downloads/macosx/unix_open_source/xgridagentforjava.html` Last visited 21 September 2005.

Daniel Cote, 2004. XgridAgent for Unix Architectures. `http://www.novajo.ca/xgridagent/` Last Visited 21 September 2005.

David Graff, 2003. English Gigaword. Linguistic Data Consortium at the University of Pennsylvania. LDC2003T05.

Brian Granger, 2005. PyXG. `http://hammonds.scu.edu/~classes/pyxg.html` Last visited 21 September 2005.

Baden Hughes, James Curran, James Haggerty, Saritha Manickam and Joel Nothman, 2005. A Distributed Architecture for Interactive Parse Annotation Proceedings of the Australasian Language Technology Workshop 2005. Australasian Language Technology Association. pp. XX–YY.

Arthur van Hoff and Rick Blair, 2005. JmDNS. `http://sourceforge.net/projects/jmdns/` Last visited 21 September 2005.

Drew McCormack, 2005. Distributed Tiger: Xgrid comes of Age. `http://www.macdevcenter.com/pub/a/mac/2005/08/23/xgrid.html` Last visited 21 September 2005.

Drew McCormack, 2005. Sweetening Your Xgrid with Cocoa. `http://www.macdevcenter.com/pub/a/mac/2005/09/13/xgrid.html` Last visited 21 September 2005.

Charles Parnot, 2005. GridStuffer. `http://cmgm.stanford.edu/~cparnot/xgrid-stanford/html/goodies/GridStuffer-info.html` Last visited 21 September 2005.

Srikumar Venugopal and Rajkumar Buyya and Lyle Winton, 2004. A grid service broker for scheduling distributed data-oriented applications on global grids. Proceedings of the 2nd Workshop on Middleware for Grid Computing. ACM Press. pp. 75–80.

Acknowledgements

The research in this paper has been supported by Apple Computer Inc, through the Apple University Consortium and the Apple University Development Fund.

I am grateful to Ernest Prabakhar and Richard Crandall from Apple comments on an earlier version of this paper.

Additionally I wish to thank my colleagues who facilitated the availability of remote systems to use in testing the Xgrid framework: Ewan Klein at the University of Edinburgh; Terry Langendoen at the University of Arizona; Kaja Christiansen at the University of Aarhus; Paul Edwards at The University of Melbourne; and Andrew Smith at the University of Queensland.