Winter 5-6-2011

CAMPUS GRIDS: A FRAMEWORK TO FACILITATE RESOURCE SHARING

Derek J. Weitzel
University of Nebraska-Lincoln, dweitzel@cse.unl.edu

CAMPUS GRIDS: A FRAMEWORK TO FACILITATE RESOURCE SHARING

by

Derek Weitzel

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Master of Science

Major: Computer Science

Under the Supervision of Professor David Swanson

Lincoln, Nebraska

May, 2011

CAMPUS GRIDS: A FRAMEWORK TO FACILITATE RESOURCE SHARING

Derek Weitzel, M. S.

University of Nebraska, 2011

Adviser: David Swanson

It is common at research institutions to maintain multiple clusters. These might fulfill different needs and policies, or represent different owners or generations of hardware. Many of these clusters are under utilized while researchers at other departments may require these resources. This may be solved by linking clusters with grid middleware. This thesis describes a distributed high throughput computing framework to link clusters without changing security or execution environments. The framework initially keeps jobs local to the submitter, overflowing if necessary to the campus, and regional grid. The framework is implemented spanning two campuses at the Holland Computing Center. We evaluate the framework for five characteristics of campus grids. This framework is then further expanded to bridge campus grids into a regional grid, and overflow to national cyberinfrastructure.

ACKNOWLEDGMENTS

I would like to thank my advisor, David Swanson. I appreciated his guidance and patience while I implemented this thesis project. Also, I thank David for hiring me as an freshman, with little computing experience and giving me the support to learn and excel at scientific computing.

I want to thank Dan Fraser for bringing this interesting project to me and giving me the opportunity to create my own solution to this problem.

I want to thank Brian Bockelman for excellent technical advise during my time at HCC. He worked diligently with me on my thesis.

I also want to thank all the people at the Holland Computing Center. I have broken their systems, asked for advise, and caused an untold amount of extra work for them. I especially want to thank my colleagues for the assistance they've provided over the years: Garhan Attebury, Carl Lundstedt, Tom Harvill, Adam Caprez, Ashu Guru, and Chen He.

Contents

Contents **iv**

List of Figures **vii**

List of Tables **viii**

1 Introduction **1**

2 Background **6**

2.1 Characteristics of Campus Grids 6

2.1.1 Trust Relationships 6

2.1.2 Job Submission 8

2.1.3 Resource Independence 9

2.1.4 Accounting 10

2.1.5 Data Management 11

2.2 Background 12

2.2.1 High Throughput Computing 12

2.2.2 Condor 13

2.2.3 Open Science Grid 14

2.3 Thesis Overview 15

3 Related Work **16**

3.1 Technology to Create a Campus Grid 16

3.1.1 Globus 16

3.1.2 Condor Flocking 17

3.1.3 GlideinWMS 18

3.1.4 PanDA 19

3.2 Other Campus Grids 20

3.2.1 University of Virginia Campus Grid 20

3.2.2 University of Oxford Campus Grid 21

3.2.3 Purdue University 21

3.2.4 Grid Laboratory of Wisconsin 22

3.2.5 FermiGrid 23

3.2.6 Overview of Campus Grids 24

4 Design and Implementation of the HCC Campus Grid **25**

4.1 Campus Grid Factory 26

4.1.1 Flocking 30

4.1.2 Condor & BLAHP 31

4.1.3 Pilot Jobs 32

4.1.4 Pilot Submission Algorithms 34

4.1.5 OfflineAds 36

4.1.5.1 Influence OfflineAds Have on the CGF 37

4.1.5.2 Creating OfflineAds 38

4.1.5.3 Managing OfflineAds 38

4.2 Bridging Campus Grids 39

4.3 Full Campus Infrastructure 40

5 Evaluation **43**

5.1 University of Nebraska Holland Computing Center Campus Grid . . . 43

5.1.1 Prairiefire Cluster Configuration 46

5.1.2 Firefly Cluster Configuration 47

5.1.3 GlideinWMS OSG Interface Configuration 47

5.1.4 Flocking to Purdue Configuration 47

5.1.5 User Submission 48

5.2 Characteristics of HCC Campus Grid 48

5.2.1 Trust Relationships 49

5.2.2 Job Submission 51

5.2.3 Resource Independence 52

5.2.4 Accounting 54

5.2.5 Data Management 55

5.2.6 Updated table of Campus Grid Attributes 56

5.3 Usage 56

6 Conclusions and Future Work **59**

Bibliography **61**

List of Figures

1.1 Widening Circle of Resources 3

3.1 Overview of Flocking 17
3.2 Grid Laboratory of Wisconsin Campus Grid 23

4.1 Overview of the Campus Grid hardware 26
4.2 Overview of Campus Factory components 27
4.3 Overview of Campus Factory Function 28
4.4 Layered Diagram of the Worker Node 32
4.5 Widening Circle of Resources 40
4.6 The Full Campus Grid Architecture 41
4.7 Overview of the Campus Grid software 42

5.1 The HCC Campus Grid 44
5.2 Campus Grid Submission Scripts 49
5.3 Snapshot of Accounting of the HCC Campus Grid 54
5.4 Snapshot of Usage of the Extended HCC Campus Grid 58

List of Tables

2.1 Condor Daemon Functions 14

3.1 Campus Grid Attributes 24

4.1 Changes to ClassAds for Offline Function 38

5.1 Updated Campus Grid Attributes 57

Chapter 1

Introduction

A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities[18]. A campus grid is a specialized grid where resources are owned by the same organization, though they can be in multiple administrative domains. We further restrict our considerations to those campuses that have multiple computation resources.

A campus grid has become necessary to alleviate demand on newer parallel machines by moving single core jobs to other resources that have idle cycles, such as previous generations of parallel machines and idle workstations. By moving the single core jobs, it can free the newest hardware for large parallel jobs that can benefit from better interconnects, larger and faster storage, and increased core count that are on the newest hardware.

A campus grid requires a framework for distributing computation among independent clusters within a campus. A campus typically contains multiple compute clusters that are independently administered. A campus grid's purpose is to increase the processing power accessible to users by connecting compute resources. By offloading jobs

from clusters that are full to idle clusters, they can increase utilization. Additionally, the users can benefit by increased processing power.

The framework described in this thesis is used to create a production campus grid. The campus grid includes technology to allow participation in the campus grid by clusters that use several schedulers, and uses production quality software integrated into a solution that is deployed at several clusters in the U.S. The campus grid framework provides a method for users to run jobs transparently on available distributed resources on the campus grid. Further, it can expand beyond the campus and onto other campuses by simple configuration changes.

At the Holland Computing Center (HCC) [30] at the University of Nebraska – Lincoln, I created a campus grid, the HCC Campus Grid, that spans two clusters and can overflow onto the national grid infrastructure. The HCC Campus Grid borrowed concepts and techniques from earlier campus grids and a national grid, the Open Science Grid (OSG). The OSG focuses on High Throughput Computing (HTC), concentrating on tasks that require as much computing power (throughput) as possible over long periods of time [29]. The HCC campus grid also focuses on HTC tasks.

The HCC Campus Grid bridges clusters and the national infrastructure while running production processing jobs from on-campus researchers. Since the beginning of 2010 to March 2011, we ran 8,151,607 jobs for 9,022,655 hours on the HCC Campus Grid infrastructure.

The HCC Campus Grid differs from existing campus grids by building an expanding pool of resources for the users. Other campus grids are limited to available resources on the campus or in a regional grid. The HCC Campus Grid can transparently submit to ever increasing distant resources. Figure 1.1 describes the widening available resources to users. The HCC Campus Grid will first attempt to run at the

local cluster, then campus clusters, and finally out to other campus grids or a national grid.

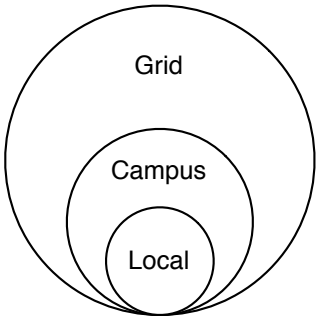

Figure 1.1: Widening Circle of Resources

Building a campus grid represents a significant commitment for both users and resource providers, which should be evaluated against the benefits. Other Campus grids may provide one or two of the listed benefit, but the HCC Campus Grid provides them all in a cohesive framework. The primary benefits of the HCC Campus Grid are:

- **Resource sharing**: An HTC-based approach focusses on using all resources effectively. Resources are typically bought for peak, not average, usage; utilizing the idle time across the entire campus improves the value of the investment.
- **Homogeneous interfaces to multiple resources**: Moving researchers from one resource to another results in a (possibly large) upfront cost in time and energy. By providing a homogeneous interface across the campus, researchers can quickly utilize new resources without the pain of migration.
- **Independence from any single computational resource**: It is expensive to provide highly available resources. If a researcher does not rely on a specific

single cluster, individual cluster downtimes have a smaller impact. This reduces the need for high levels of redundancy and stretches the campus computing budget further.

An obvious requirement for campus grids is having multiple resources on campus. However, this is not sufficient for resource providers–the resources should also be interchangeable. A grid composed of a single AIX cluster, Linux cluster, and Windows cluster, will likely never see any resource pooling or sharing. This does not necessarily imply the resources need to be identical–complete homogeneity is typically impossible due to individual resource requirements or ownership. Often it is undesirable to have homogeneity as resources can fulfill different resource requirements such as large memory or local scratch space.

A mistake in campus grids is to focus on the infrastructure for pooling resources without similarly engaging and supporting the activities of users. An analogy can be made to fluid: if there is a sink (resources) with no source (user jobs), the flow quickly stops, and the campus grid is forgotten. Personnel investment must be made to engage the user community. Even before this investment is made, a campus should identify whether the on-campus scientific computing has a significant portion of tasks that can be converted to HTC workflows. Prioritization should be applied so the users with the simplest workflow and the most to benefit are converted first. Tasks with the following characteristics should typically be avoided:

- **Large scale (multi-node) MPI:** Require specific tunings to the given resource.
- **Multi-day jobs:** Shared resources often need to be reallocated quickly back to the owner, meaning these jobs are unlikely to complete.
- **Sensitive data or software:** Tasks with sensitive (for example, HIPPA-

protected) data may have legal boundaries preventing distribution. Software with strict licenses may also be illegal to use across the grid.

There have been several methods to create a distributed campus grid. They all use some technology to distribute and schedule the jobs on the grid. Some methods for distribution are commercial products such as the meta scheduler Moab [24]. Others are translation layers between a generic description language and a scheduler, such as Globus [17], CREAM [3], and ARC [11]. Still others are entire resource managers that can span multiple clusters like Condor [46]. Each of these solutions can be utilized to create a campus grid, but they all have drawbacks that are detailed in Section 3.2.

Chapter 2

Background

2.1 Characteristics of Campus Grids

In this section, we explore five characteristics of HTC-centric campus grids. While the list is not exhaustive, these are foundational characteristics of campus grids.

It's worth noting that by the NIST's definition of Clouds [31], a grid is also a cloud. A campus grid also fits this definition. Running scientific jobs in the cloud has been done, but can be very expensive in commercial clouds.

We've found that campus grids can be characterized by how they approach trust relationships, job submission, resource independence, accounting, and data management.

2.1.1 Trust Relationships

A campus grid must have an acceptable trust model in order to succeed. A trust relationship enables a resource provider to grant campus users controlled access to the resource, and may be established through sociology and/or technology-based security methods.

In the OSG, the trust model used is designed to be homogeneous and to meet the most stringent requirements of all participating sites. The implementation involves using Globus's Grid Security Infrastructure (GSI) with Virtual Organization Membership Service (VOMS) attributes, a Public Key Infrastructure (PKI) extension [14]. The GSI model is widely accepted, allowing the OSG to participate in the Worldwide LHC Computing Grid (WLCG) [7]. FermiGrid, located at the Fermi National Accelerator Laboratory in Batavia, Illinois, uses GSI authentication on its campus grid. While it provides a highly secure, decentralized authorization model, it is more difficult for end users compared to traditional username/password authentication. Thus, campus grids may be motivated to use alternate trust models.

On-campus resource providers may have a higher implicit degree of trust than at the national level due to sociological reasons. This trust is partially based on locality–it is easier to establish a working relationship with a colleague locally on campus than 1000 miles away. Additionally, a national lab such as Fermilab is higher profile than most universities and therefore requires stronger trust relationships to maintain security and accountability.

Security requirements on some university campuses are simply less stringent than that of federal labs. A campus may not have strict policies governing user job separation or traceability requirements. Some campus clusters may be satisfied with running any job originating from elsewhere on the campus to an unprivileged account. When a job crosses domains (from local cluster to across campus, or from campus to the national grid), it must satisfy the security requirements for the destination domain. Thus, if a campus grid would like to bridge to the national grid, users must be able to associate GSI credentials with their jobs.

A technical reason for different trust relationships between campuses and larger grids is the location of user job submit hosts. Unlike the OSG, where users can submit

jobs from any worldwide host, campus users often submit from a few trusted campus resources. If limited to a few well-managed hosts, IP-based security may be sufficient for campuses, as the security such as username and password is applied to submit hosts rather than cluster entry points.

2.1.2 Job Submission

In order for a HTC-oriented campus grid to function, users need a usable job submission interface. The Globus Toolkit [17] provides the Globus Resource Allocation Manager (GRAM) interface for job submission and corresponding clients. The GRAM layer abstracts the batch system; the remote user interacts with the site's GRAM install and GRAM converts these actions into batch system commands at the destination. The GRAM interface is used by the OSG at the scale of over 100 million jobs a year. It abstracts many batch system constructs, and is also used on the TeraGrid to submit larger jobs running on hundreds or thousands of cores. While GRAM can be used directly, users almost exclusively prefer to interact with it via Condor-G [21] , which provides a batch system interface on top of GRAM. FermiGrid relies on Condor-G submission to GRAM for job submission.

An abstraction layer like GRAM introduces a new user experience (even if Condor-G is used), requiring new expertise. An alternate approach is to uniformly use batch system software that can interact with multiple instances of itself. By linking resources at the batch system level rather than adding an abstraction layer on top, we improve the user experience–users no longer need to learn additional tools. The client tools do not need to translate errors across different domains, easing a common source of frustration in the grid. When Condor-G is used, we have a batch-system interface abstracting an API which, in turn, abstracts remote batch systems; thus, error propa-

gation is extremely difficult. In some campus grids described in Section 3.2, resources are linked through use of a common batch system, Condor, through a mechanism Condor refers to as "flocking". A hybrid between Condor-only and GRAM is given by GlideinWMS [43].

In our observations, the closer the grid user experience is to the batch system user experience, the more likely a user will adopt the campus grid.

2.1.3 Resource Independence

Compared to a corporate IT environment, one unique aspect of universities is the diversity of management of computing resources. On a campus, several distinct teams may manage distinct clusters due to campus organization or ownership. Management of resources may be divided by college, department, or lab. One characteristic of campus grids is the independence of resources–the level of decision-making delegated out to the resource providers.

The simplest campus grids can be formed by requiring all clusters on campus to run the same batch system and linking batch system instances–Grid Laboratory of Wisconsin's (GLOW) use of Condor is an example. Every cluster in GLOW runs the Condor batch system, providing a common interface. System administrators are not free to choose their own batch systems if they want to participate in this grid (participation is voluntary, and participants obviously believe the benefits of GLOW membership outweighs this drawback). It may be desirable for a specialized cluster to have a distinct batch system from the rest of the campus; resource independence allows the cluster owners to best optimize their resource to suit their needs.

Resource independence comes at a cost to the end-user. Extremely heterogeneous resources can be difficult to integrate at the software level–an application compiled

for Linux will not be compatible with Windows. Some guarantees about the runtime environment or other interfaces need to be clearly articulated and agreed upon to prevent frustration. Differences that are unavoidable or are expected to be handled by the user should be clearly expressed to the user [39]. At the OSG level, we have found the users often frustrated by the amount of heterogeneity, especially unexpected heterogeneity, compared to using a single site or a grid with a smaller number of sites [48].

2.1.4 Accounting

Accounting may not seem to be an important grid characteristic–it certainly isn't required for users to successfully run a job. However, it is critical for the long-term health of the campus grid as it provides a quantitative measurement of the grid's value. One economic model for the grid is for resource providers and users to "barter" for computing hours as reported by accounting services.

Accounting systems do not need to be technically advanced. Most batch systems provide a local accounting system. The most basic method is for each cluster to parse these logs into a CSV file per cluster, and to build an Excel spreadsheet out of the aggregated files. This is functional, but painful when statistics are needed more than once a month. Most batch system vendors sell accounting systems usable for multiple clusters, provided all clusters involved use the same batch system.

Many research computing centers have written their own accounting systems at some point; most implementations are in the style of PHP-based web interface on top of a custom database, again fed by custom log-parsing scripts. Both the OSG and TeraGrid have spent effort on accounting software to suite their needs. The OSG's Gratia [20] is designed to be reusable by other organizations, and is in use at the

FermiGrid campus grid.

Any site-local accounting systems–homegrown, vendor provided, or designed for the national grids–can work at the campus grid level as long as they can answer the following questions for a given time period:

- How much computing resource was consumed overall?
- How much computing resource did a specific user/group consume?
- How much computing resource did a specific user/group consume on resources they did not own; i.e., how much did I get from resource sharing?
- How much computing resource did a specific cluster provide?
- How much computing resource did a specific cluster provide to groups that did not own it; i.e., how much did I give away due to resource sharing?

2.1.5 Data Management

Scientific data management presents two challenges for research computing centers: volume of data and archival requirements. The data volume is often larger than a single scientist can keep on their personal systems, and archiving requires expertise outside their field.

Distributed computing can present an additional challenge: managing data location. Data access costs may be variable between different resources on a grid, or required data may simply be unavailable at some locations. A simple solution is to export the same file system to all resources, hiding data locality from the user. Unfortunately, this solution breaks down outside the campus since the file system would need to be accessed from outside campus, which can significantly hurt performance

and increase administration cost on other campuses. A single file system solution may not work in highly-distributed campuses as well.

More complex solutions include declaring data dependencies for jobs explicitly inside the job submissions (gLite WMS [2], Condor), promoting data to be a top-level abstraction equal to jobs (Stork [27]), or promoting data to be the central concept above jobs (iRODS [26, 37]). The Compact Moun Solenoid (CMS) experiment's model separates the data management and job submissions systems, allowing the job submissions to simply assume all data is available (CMS PhEDEx [12]).

While any system can be used for campus grids, the examples we consider in Sections 3.2 and 5.1 either export a file system or utilize the tools from the job submission system. The addition of a separate data management system often presents complexity to the users. In some situations, it is easier for the user to not use distributed computing rather than deal with a complex data system. Distributed computing is the cheaper alternative. However, we note iRODS is an increasingly popular option and may have significantly decreased the operational cost of data management systems.

2.2 Background

2.2.1 High Throughput Computing

High Throughput Computing (HTC) is defined as tasks that require as much computing power (throughput) as possible over long periods of time [29]. This is in contrast to High Performance Computing (HPC), where users are concerned with maximizing instantaneous resources and response time. HTC workflows are usually ensembles of independent single processor jobs with no communication between them. This is not

to say there isn't coordination; many workflow managers can utilize HTC to solve complex problems with many steps.

As there is no communication between the jobs in a given task, they can be distributed across multiple resources. This increases throughput, the end-goal of HTC. HTC can use pooled resources mostly interchangeably and as such is well suited to distributed and grid computing models. The OSG has demonstrated its technologies are successful; in Q4 2010, the OSG averaged over 400,000 jobs and a million computational hours a day using HTC.

2.2.2 Condor

Condor was developed at the University of Wisconsin–Madison. An overview from [46]:

> Condor is a high-throughput distributed batch computing system. Like other batch systems, Condor provides a job management mechanism, scheduling policy, priority scheme, resource monitoring, and resource management. Users submit their jobs to Condor, and Condor subsequently chooses when and where to run them based upon a policy, monitors their progress, and ultimately informs the user upon completion.

An important technology used in Condor is the Classified Advertisement (ClassAd) mechanism. ClassAds are the language that Condor uses to communicate between daemons and for matchmaking [38]. A ClassAd is a list of keys and values, where the values can be strings, numbers, or expressions. All resources are described by ClassAds. Job ClassAds have attributes such as log file, output, input, and job requirements. Resource ClassAds have attributes such as requirements to run on the

Daemon	Function
`condor_master`	Maintains Condor daemons
`condor_collector`	Information Provider
`condor_schedd`	User Job Queue
`condor_negotiator`	Scheduler: Matches jobs with resources
`condor_startd`	Execution manager. Runs on the resource

Table 2.1: Condor Daemon Functions

resource, ownership, and policies. ClassAds are used for matching jobs to resources by evaluating requirements of both the jobs and the resources.

Another component of Condor is the grid computing agent Condor-G [21]. Condor-G communicates with Globus [17] sites. Condor provides job submission, error recovery, and creation of a minimal execution environment. Along with Condor-G, Condor can also submit jobs to other systems including Amazon EC2 [1] and PBS [25].

Condor categorizes jobs by `universe`. A `universe` in Condor specifies how the job should be handled. The simplest example is the `vanilla universe`. In this `universe`, the job is handled as a single executable, with input and output, that will exit when the job has completed. When the `universe` is `grid`, this means that the job will be translated to another submission method. The other submission methods could be a Globus GRAM submission, or as used in this thesis, as a PBS job submission.

Commonly used Condor daemons and their functions are described in Table 2.1.

2.2.3 Open Science Grid

The Open Science Grid (OSG) [36] is a national cyber infrastructure consortium that provides dedicated and opportunistic use of computation and storage resources consisting of nationally distributed universities and national laboratories. The OSG provides infrastructure, services, software, and engagement to the users.

The OSG Production Grid provides a common interface to computing and stor-

age resources: Globus and Storage Resource Manager (SRM)/GridFTP respectively. Access to resourcesis provided by an OSG client. Both Globus and SRM/ GridFTP were developed by consortium members and are included into the OSG packaging. Many software tools have been contributed to the OSG software stack such as Condor and the Berkeley Storage Manager [28].

A typical OSG site has a compute element that can run jobs and a storage element that can store data. The storage element is both physically close and tightly coupled to the compute element in order to minimize latency for data. Since most sites have their own storage element, they are used to stage data into and out of the site.

A user will install the OSG Client tools and submit to the sites. The OSG tools will also include applications that can be used to stage data to the storage elements. A user only needs a valid OSG certificate to access the grid resources.

The OSG is organized into groups of users called Virtual Organizations (VO's). These VO's help users to run on the grid, as well as provide organization to many thousands of researchers. Additionally, the VO can sign member's certificates to help sites identify users as belonging to the VO.

2.3 Thesis Overview

The rest of this thesis first discusses technology to create campus grids as well as existing campus grids in Chapter 3, and then describes our implementation in Chapter 4. Chapter 5 describes how we evaluate our system and presents the results. Chapter 6 presents our conclusions and describes future work.

Chapter 3

Related Work

3.1 Technology to Create a Campus Grid

3.1.1 Globus

Globus GRAM is a translation layer between the Globus Resource Specification Language and the local resource manager. It has been very successful and has a large install base. Globus also has deep integration with standard grid authentication methods such as PKI.

Placing Globus gatekeepers on each cluster allows jobs to be submitted to each cluster without modifying the underlying batch system. This requires a higher layer of abstraction over the Globus gatekeepers to optimally balance load between clusters. Globus GRAM implements the Grid Security Infrastructure (GSI) security that is inconsistent with most existing campus security architectures such as LDAP [22] authentication or Kerberos [45]. It does not provide a method for transparent execution on other clusters: each submission must target a specific execution resource. Therefore, there is no overflow capability in Globus, which experience on the OSG

has shown is important to users.

3.1.2 Condor Flocking

A single software solution is Condor. Each resource can run Condor on their clusters and 'flock' [13] to each other. In this solution, jobs may not be balanced on each resource due to Condor's greedy scheduler algorithm, but they will find any idle slots available on the resources.

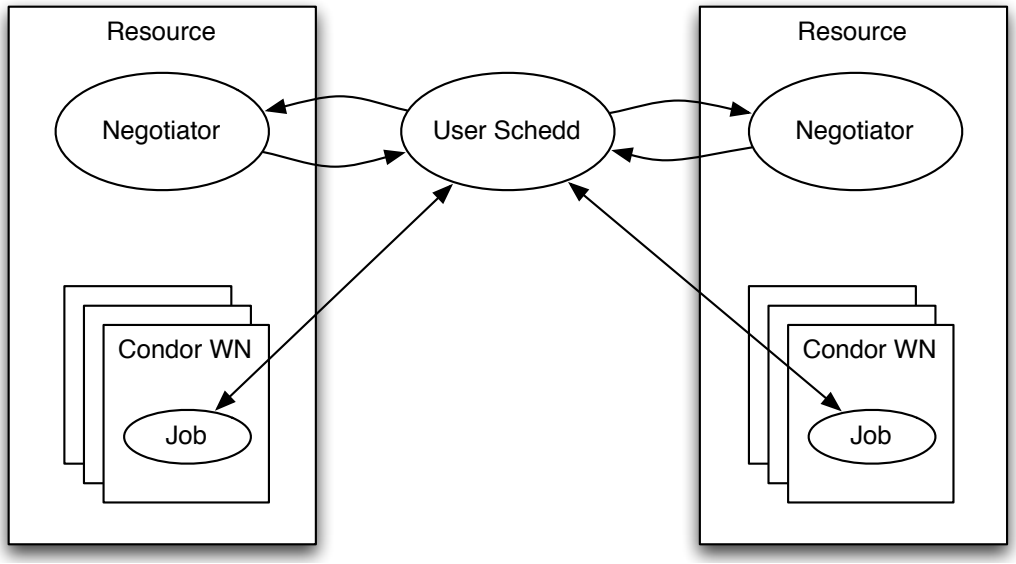

Figure 3.1: Overview of Flocking

Flocking jobs is accomplished by a multi-step process, displayed in Figure 3.1. First, the `condor_schedd` reports to the remote `condor_collector` that it has idle jobs available to run. During the next negotiation cycle, the remote `condor_negotiator` contacts the `condor_schedd` to match any available remote resources to the requested jobs. If there is a match, it is sent to the `condor_schedd` and it contacts the resource directly to claim it. After the claim is successful, the job starts on the remote resource. Flocking is further described in Section 4.1.1.

Condor flocking has many advantages. Since the job submitter (`condor_schedd`) directly contacts the executing resource, there is no central service that is relied on. Flocking handles failures gracefully. If an execution resource becomes disconnected, Condor will attempt to reconnect, or re-run the job elsewhere. Jobs will still execute on previously claimed resources if the central manager becomes disconnected. Condor treats flocked jobs just as it would a local job.

This solution requires each resource to run Condor as their scheduler and resource manager. Condor must be running on each worker node, increasing the administration requirements.

3.1.3 GlideinWMS

The Glidein Workflow Management System (GlideinWMS) [42] is a job submission method that abstracts the grid interface, leaving only a batch system interface. GlideinWMS accomplishes this abstraction by using pilot jobs. Pilot jobs are containers that once started, will request work from the user's queue. User jobs will not wait in remote queues; therefore, user jobs do not waste time in remote queues when idle resources are available. GlideinWMS separates the system into two general pieces, the frontend and the factory. The frontend monitors the local user queue and requests glideins from the factory. The factory serves requests from multiple frontends and submits pilots to the grid resources on their behalf. The factory only submits jobs to grid interfaces on multiple resources and can be optimized for that purpose. The frontend only deals with the local batch system, and can be optimized for the user's jobs.

The GlideinWMS system uses Condor throughout. It uses Condor-G [21] to submit to the grid resources, as well as manage user jobs on the frontend. The frontend

and factory are daemons that run on the user submit host and a central machine, respectively.

GlideinWMS is heavily used on the the Open Science Grid. A major user and developer of the software is high energy physics, specifically the Compact Muon Solenoid (CMS) experiment [5] and the Collider Detector at Fermilab (CDF) [48]. They have demonstrated recently that GlideinWMS can scale beyond 25,000 running jobs.

GlideinWMS does have a few drawbacks. GlideinWMS uses an external factory that acts as a single point of failure. If the factory quits submitting jobs to grid resources, then users cannot run jobs. Also, GlideinWMS is designed to only submit to GSI secured sites. GSI is typically used only on production grids, and is rarely used inside a campus where the trust relationship is implicitly stronger.

3.1.4 PanDA

PanDA is a distributed pilot-based submission system developed by US ATLAS for analysis of the ATLAS Experiment [15] data. PanDA is designed with tight integration with the ATLAS distributed data management system. It has integrated monitoring for production and analysis operations, user analysis interfaces, data access and site status.

PanDA is designed around a central server. All jobs are submitted to this single server that centrally manages all job information. The user submits jobs using a HTTP interface to the central server. The end-users are insulated from the grid by only accessing the central PanDA server.

PanDA has very strong data management as it is integrated with the ATLAS data management system. The client interface is generic enough that jobs can be submitted to multiple grids transparently. Since all submissions are from a central

server, accounting and monitoring of jobs are trivial and very accurate.

The PanDA system is very reliant on the uptime of the central global server. Though the resource independence is high when the resources are grid sites, the system still is reliant on the central PanDA server for any jobs to start. However, in practice, this has been very reliable.

3.2 Other Campus Grids

3.2.1 University of Virginia Campus Grid

The University of Virginia Campus Grid [23] designed a campus grid using the Web Services Resource Framework (WSRF) with Globus. The goal of the campus grid was to use as much existing infrastructure as possible. The grid utilized the existing authentication system by developing a new credential generator called CredEx [9] that interacts with the local LDAP servers to create PKI certificates. Globus version 4 (now deprecated) was used to interact with the Linux clusters on campus.

Another focus of the Virginia campus grid was policy expression and enforcement. This is a common theme for many grids since they span multiple administrative domains. In the Virginia grid, an enforcement service would enforce these rules by cutting off and redirecting users to and from resources. The load balancing would be enforced by the enforcement services, as well as policies regarding a resource's prioritization of jobs. Additionally, a broker was developed to distribute jobs.

The Virginia campus grid has many attributes shared with other Campus Grids. The Virginia grid approaches trust relationships by utilizing the existing campus authentication infrastructure. Job submission is handled by WSRF and distributed with a custom developed broker. There are many central services such as the enforcement

service and the broker which limits resource independence of the grid. A downtime in either of these services would limit usefulness of the grid. Accounting and authentication are handled by central IT. Data management is not addressed in the campus grid.

3.2.2 University of Oxford Campus Grid

The University of Oxford Campus Grid [47] built a comprehensive campus grid including both compute and data provisioning. The compute provisioning uses Condor-G [21] and a information server. The information server injected resource specific information into the Condor-G matchmaker, allowing Condor to match jobs to appropriate resources as well as to follow resource policies. For data provisioning, the Storage Resource Broker (SRB) [4] was used with a dedicated data node. Authentication is handled through on-campus Kerberos. Accounting is done by a custom daemon written at Oxford that keeps detailed statistics for every job.

The Oxford campus grid resembles the Open Science Grid model. Each resource has a gatekeeper, a central node that provides access to the underlying nodes. The information server and virtual organization management both have analogies in the OSG. The resource broker and data vault conflict with the design of the OSG. Both of these resources are single points of failure that can severely degrade the usability of the campus grid.

3.2.3 Purdue University

The Purdue University campus grid [44, 19] is part of a larger grid, DiaGrid, which serves a number of universities in Indiana and Wisconsin. This grid is based upon the Condor and Condor flocking technology. All jobs are submitted via Condor.

For security, Purdue manages a small number of submit hosts that are allowed to run jobs on their grid. External jobs can flock to Purdue and are mapped to an unprivileged user account on the execute host. In order to maximize the resources in its grid, Purdue also installs the Condor batch system next to the Portable Batch System (PBS) batch system on its clusters. In this side-by-side configuration of Condor and PBS, each batch system is independent, except any PBS job on a given node will preempt any Condor jobs. While idle resources are thus utilized, PBS may unnecessarily interrupt Condor jobs and all Condor jobs are inherently lower priority. The largest resources are centrally administered by a single organization, but there are large pools independently configured and managed. Usage accounting is done through Condor and a homegrown system. On large subsets of the grid, data is kept on a shared file system but no single file system is exported to all resources. Condor file transfer can be used throughout the grid.

3.2.4 Grid Laboratory of Wisconsin

The Grid Laboratory of Wisconsin (GLOW) [19, 35] is a grid at the University of Wisconsin at Madison. GLOW uses Condor to distribute jobs on their campus grid. A diagram of the campus grid is shown in Figure 3.2. Security is based on IP whitelisting. Since all resources are based on Condor, job submission and distribution is managed through the same Condor-only mechanisms as Purdue. While there is a central team available to assist with management, each resource is free to define its own policies and priorities for local and remote usage. Cluster ownership is distributed, although there's also a general-purpose cluster available. Software and data are managed by an Andrew File System (AFS) [32] installation and Condor file transfer. AFS is a global file system that every worker node will mount, therefore providing a global space to

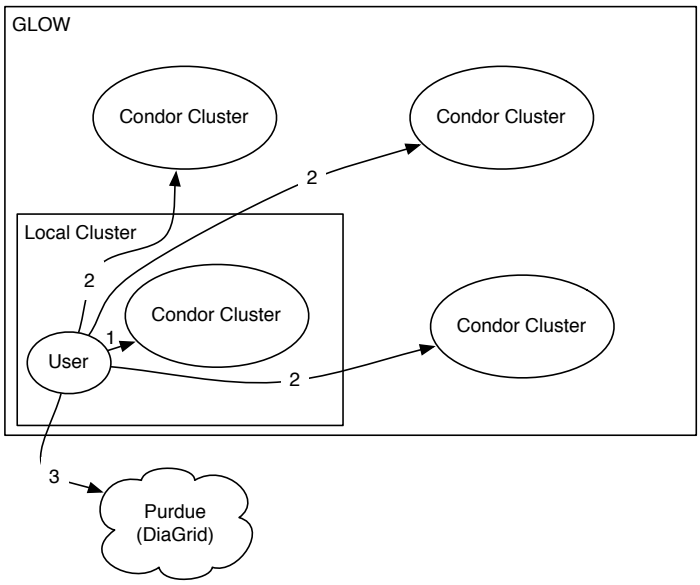

Figure 3.2: Grid Laboratory of Wisconsin Campus Grid

store data and applications. This simplifies data distribution by providing a staging area.

3.2.5 FermiGrid

FermiGrid [19, 8] is made up of resources located at the Fermi National Accelerator Laboratory in Batavia, Illinois. The FermiGrid campus grid is the closest example found of a "mini-OSG". Its uses the same Compute element software, information systems, and storage elements as the OSG. Trust relationships on FermiGrid are based on the Grid Security Infrastructure (GSI) [14], the same authentication method used by OSG. Job submission is managed by Condor-G through a Globus submission layer to the clusters. As this is the same method used to submit to the OSG, it provides one strategy to getting users from campus to the national grid. Most clusters are managed by a central team, while at least one is independently managed. Some of the grid services (authorization and information services, for example) are run

centrally. Accounting is done through Gratia [20], the same software that is used on the OSG. A central cluster file system is available to most clusters, but Globus-based GridFTP file transfer is also heavily used.

3.2.6 Overview of Campus Grids

Table 3.1 compares the campus grid architectures with that of the OSG.

Grid	Trust Relationship	Job Submission	Resource Independence	Accounting	Data Management
Virginia	LDAP/PKI	None Described	Strict	Central	None
Oxford	Kerberos/PKI	Central	Central Submission	Custom	SRB
Purdue	Host	Distributed	Strict	Custom	Condor Transfer
GLOW	Host	Distributed	Strict	None	Condor Transfer
FermiGrid	PKI	Central	Strict	OSG Gratia	Central File System
OSG	PKI	Distributed	Strict	OSG Gratia	Distributed

Table 3.1: Campus Grid Attributes

Chapter 4

Design and Implementation of the HCC Campus Grid

The design of the Campus Grid at HCC attempts to meet these three goals:

1. **Encompassing**: The campus grid should reach all clusters managed by HCC.
2. **Transparent**: There should be an identical user interface for all resources, whether running locally or remotely.
3. **Decentralized**: A user should be able to utilize his local resource even if it becomes disconnected from the rest of the campus. An error on a given cluster should only affect that cluster.

The campus grid framework is shown in Figure 4.1. The campus grid is made of user submission hosts, and resources. The components of the campus grid are described in the next sections.

Along with these goals, the technologies in this chapter can be characterized by the framework described in Section 2.1: trust relationships, job submission, resource independence, accounting, and data management.

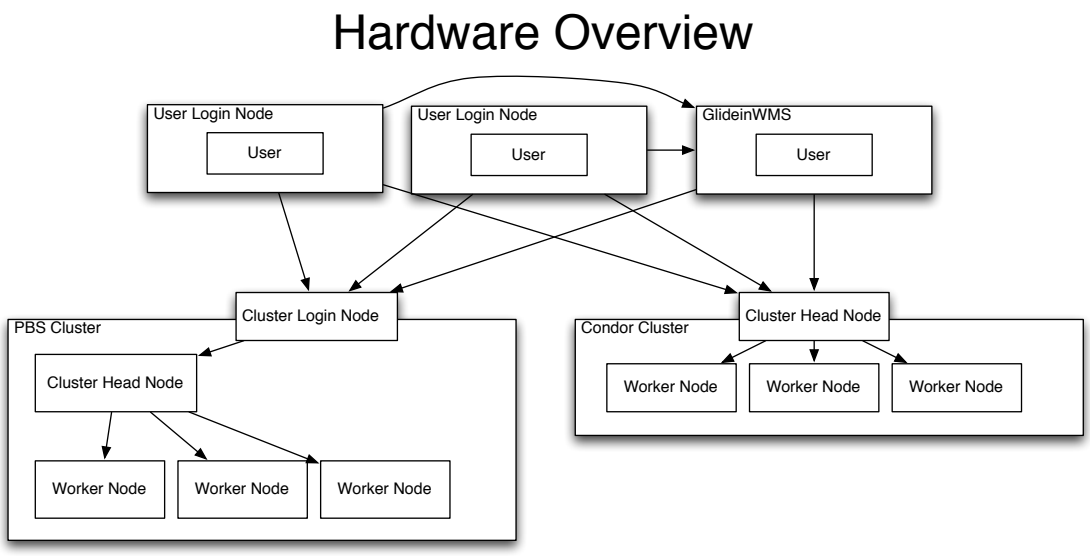

Figure 4.1: Overview of the Campus Grid hardware

4.1 Campus Grid Factory

To address the encompassing goal described above, the CGF was designed to bridge non-Condor clusters (Section 4.1.2) into the campus grid. The CGF fulfills the decentralization, and resource independence, goal by attaching directly to a single cluster that it is serving, eliminating a central service.

The Campus Grid Factory (CGF) is a collection of daemons that run on non-Condor clusters in order to submit pilots (Section 4.1.3) when additional resources are requested. The CGF includes a custom daemon, `campus_factory` that is a wrapper around Condor, using functionality in Condor to talk to user queues and submit to the local resource manager. The CGF instance must be on a 'gateway' node in order to flock (Section 4.1.1) with campus resources; the node must be able to talk to both the remote clusters and the local nodes. The `campus_factory` communicates with the `condor_collector` daemon in order to detect requests for resources, and the `condor_schedd` daemon to submit jobs to the Local Resource Manager (LRM).

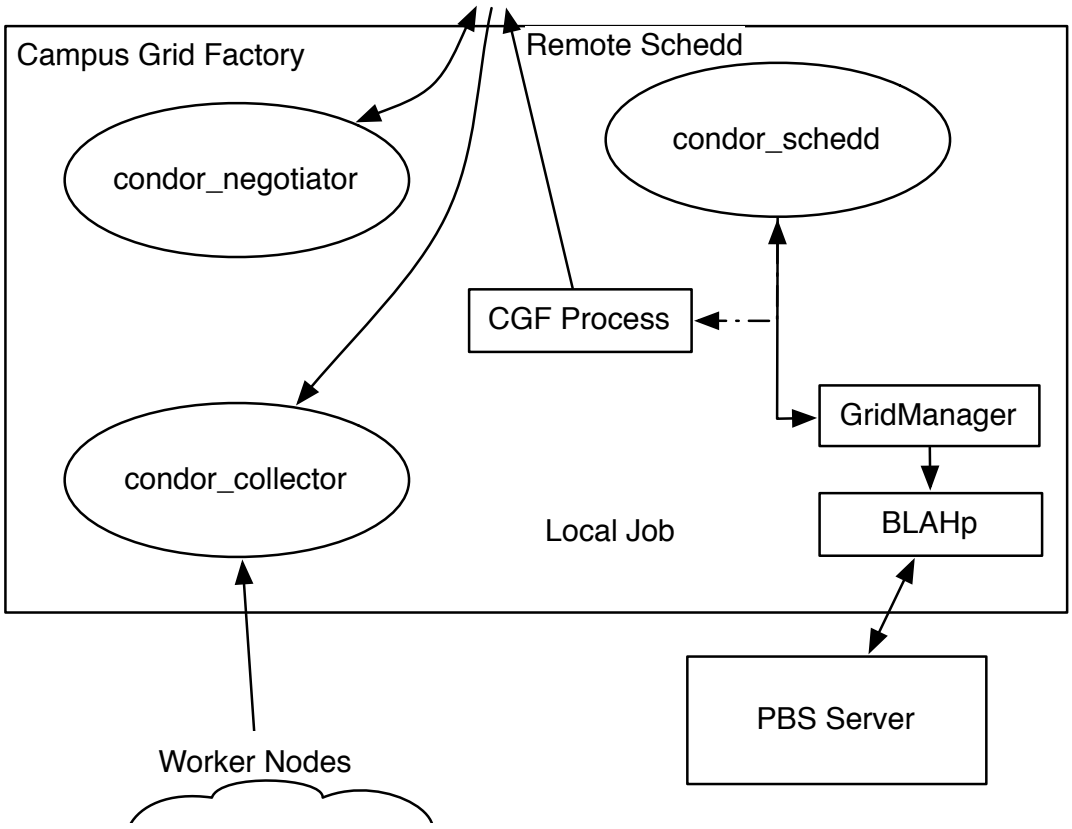

Figure 4.2: Overview of Campus Factory components

The components of the Campus Grid Factory are shown in Figure 4.2. The CGF runs as a condor job on the local submission machine. The CGF will communicate with remote queues, querying for idle jobs. When idle jobs are detected, the `campus_factory` will submit jobs as grid universe `PBS`. The GridManager will handle interaction between the CGF's `condor_schedd` and the BLAHP, which will translate the jobs to PBS submission syntax. The `condor_negotiator` and `condor_collector` will communicate with remote queues. The `condor_collector` will maintain a list of active pilot jobs inside the cluster.

The Campus Grid Factory functions are shown in Figure 4.3. The CGF software starts by querying all the Condor schedds listed in a configuration file to determine if

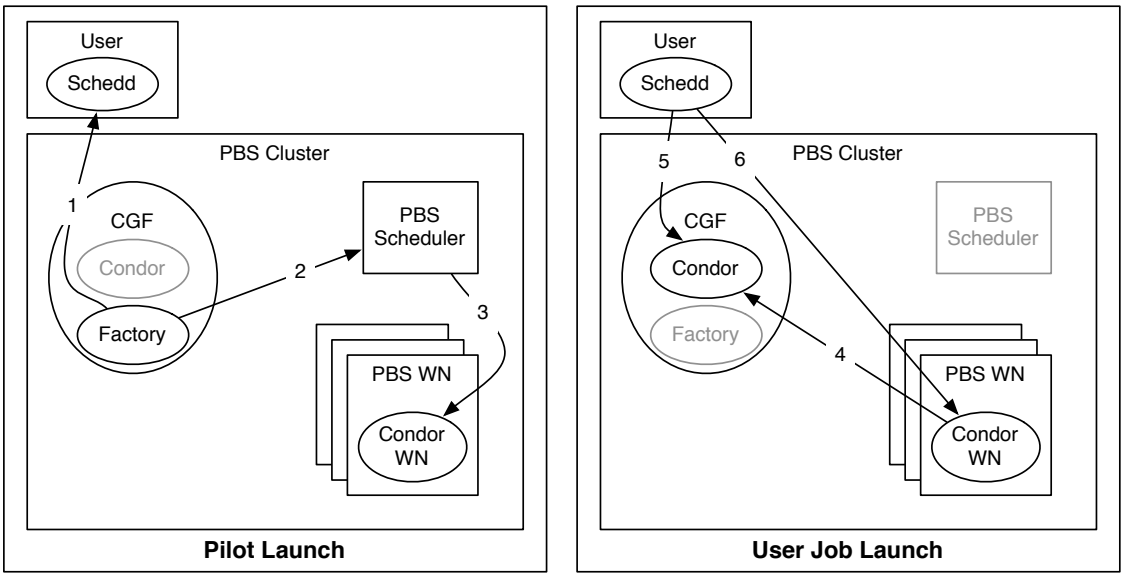

Figure 4.3: Overview of Campus Factory Function

they have jobs to run (1). If idle jobs are found, the `campus_factory` will submit (2) a pilot job for execution to the PBS scheduler. When PBS resources are available, PBS will start the pilot job (3) on an execute host, which becomes a Condor worker node. After starting, the Condor worker node will contact (4) the Condor installation at the CGF and list itself as a node available to run jobs. This is the "pilot launch" sequence.

To launch user jobs, the CGF uses Condor flocking mechanisms. The user schedd will first advertise (5) it is has idle jobs to run on the CGF's Condor collector. The CGF Condor negotiator matches the resources and orchestrates a direct connection (6) between the execute and submit hosts. Then the execute host will transfer files and begin running the user job.

The CGF is an integral part of the campus grid because it allows non-Condor clusters to participate in the grid. A Condor cluster would not need the CGF as it already can flock. The CGF obeys the priorities set in the LRM allowing resource

independence. The CGF collector then is able to route jobs from other clusters to these available Condor job slots just as it would for an all-Condor cluster.

There are many architectural similarities between the CGF and GlideinWMS software: Condor pilot jobs, submission using a translation layer, and querying user queues to detect idle jobs. However, GlideinWMS was not used since GlideinWMS is designed to have one central factory and a frontend on each submit host. If a non-Condor cluster becomes disconnected from the GlideinWMS factory, even jobs that are submitted to a local cluster will be unable to run. The GlideinWMS factory needs access to the cluster in order to start jobs, breaking the decentralization goal. The CGF merges the roles of the frontend and factory in the GlideinWMS architecture, removing configuration and maintenance of a separate GlideinWMS daemon on the submit host. GlideinWMS is designed around GSI for security; while HCC uses GSI security for its OSG work, it is preferable to avoid making it a requirement for users running on the campus grid.

The GlideinWMS system prepares and validates runtime environments via a VO-supplied script, an essential element for removing a common source of grid frustration. However, the runtime environment problem is lessened on campuses because of the smaller number of resources and the closer working relationships between system administrators. It is unsolved at the inter-campus level.

Since the `campus_factory` runs as a Condor job, the Condor daemons will ensure that it stays alive, eliminating the need to monitor an additional daemon.

The `campus_factory` depends on Condor daemons to carry out many tasks such as:

- Run and maintain the `campus_factory` daemon.
- Submit the pilot jobs to the LRM (See Section 4.1.2).

- Collect and advertise information on the pilot jobs.
- Negotiate with submitters in order to route jobs to the pilots.

4.1.1 Flocking

In the GLOW and Purdue campus grids, every resource runs the same scheduler, Condor. They use Condor's Flocking mechanisms to distribute jobs between clusters.

Flocking [13] is a method of linking Condor clusters into a larger grid. Flocking was illustrated in Figure 3.1. Condor daemons are described in Table 2.1. When a user submits jobs to Condor, they are stored and managed by the `condor_schedd`. The `condor_schedd` acts as an agent on the user's behalf to keep track of jobs, and place jobs efficiently. Flocking improves the job submission and transparent execution environment of campus grid jobs by eliminating the need to explicitly specify pools for execution. It improves data management by directly transferring files from the submitter to the execute host, bypassing the gatekeeper. Job file dependencies are described in the submission file. Typically in the OSG, job data is staged to the gatekeeper or a storage element before transferring to the worker node. Condor will handle faults in flocked jobs, such as resources going away and disconnections to remote resources, just as it would a local job, leading to better decentralization of the campus grid.

After the jobs have been submitted, the `condor_schedd` will maintain an integer `FlockLevel`. At first, the `FlockLevel` will be set to 0 and jobs will only run on the local resources. After a few minutes, if there are still idle jobs in the queue, the `FlockLevel` will increase by 1. Each time the `FlockLevel` increases, the `condor_schedd` will advertise to another Condor pool in the flocking list that it has idle jobs to run. When the remote Condor pools see idle jobs, the `condor_negotiator`

for that pool will contact the `condor_schedd` to attempt to match jobs to the remote resources. If a match is found, the schedd directly contacts the execute host to begin running the job.

Flocking does not delegate responsibility for a job. The original `condor_schedd` will maintain the job, transferring input and output and monitoring its status. This is analogous to hub-and-spokes: the job ownership never moves, but the jobs can execute at multiple resources.

The CGF uses flocking for job distribution. The CGF will flock jobs to and from other campus resources.

4.1.2 Condor & BLAHP

If the grid does not have the same scheduler on all clusters, there needs to be a translation layer from one scheduler to another. The Batch system Local ASCII Helper Protocol (BLAHP) was designed to offer a simple abstraction layer over different local resource manager services, providing uniform access to the underlying computing resources [40]. BLAHP is maintained by the gLite [16] collaboration at the European Organization for Nuclear Research (CERN). The BLAHP supplements the encompassing goal by allowing execution of Condor jobs to the underlying resource manager.

BLAHP is distributed with Condor as an additional library. Condor uses BLAHP to submit jobs when specifying the PBS or LSF `universe` in the submission file.

BLAHP is used by the CGF to submit pilot jobs to the underlying batch system. The `campus_factory` only needs to communicate with Condor in order to submit to the underlying PBS scheduler.

4.1.3 Pilot Jobs

Condor Glidein [21] is a pilot job–based grid submission that creates an overlay network on remote resources. Glidein uses the pilot method for job execution. When a Glidein starts, it advertises its availability to run jobs. Glidein is designed to use standard Condor mechanisms to advertise its availablity to a Condor Collector process, which is queried by the Scheduler to learn about available resources. Each user job running in a glidein is run in a sandbox on the local disk and is provided with a consistent execution environment across hosts and clusters.

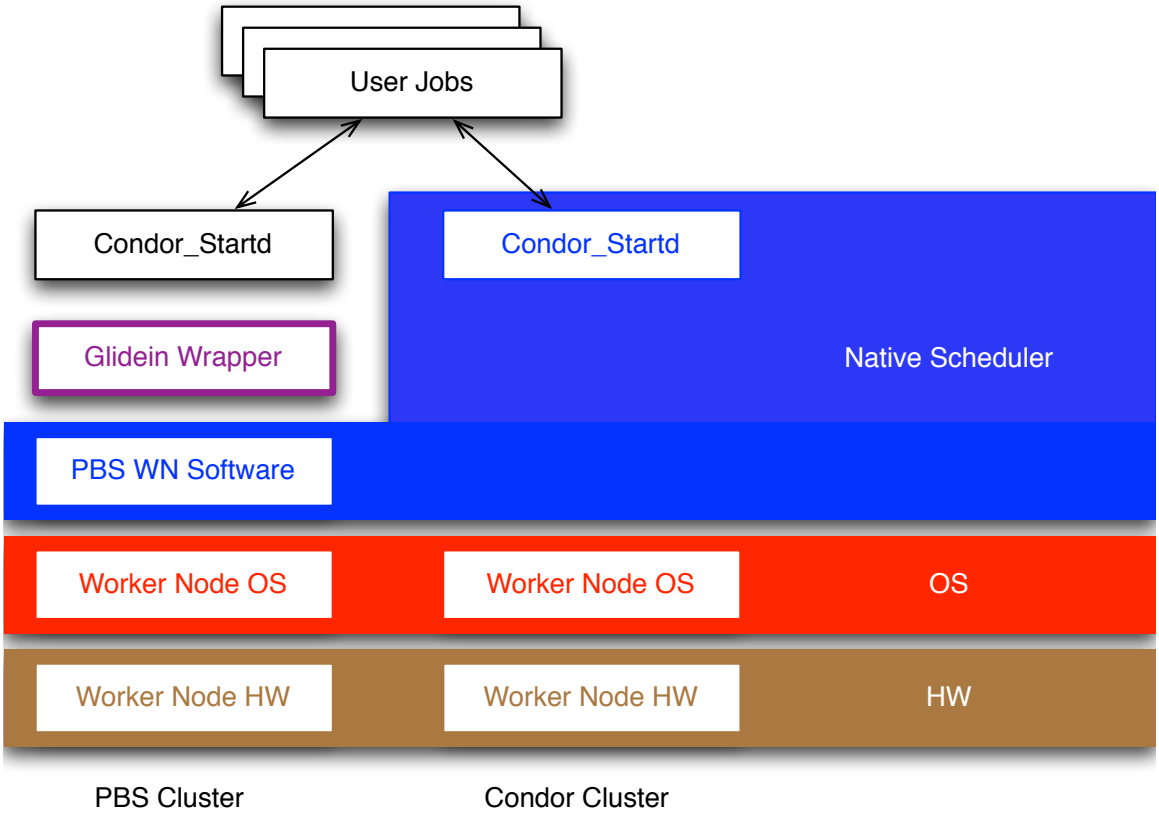

Figure 4.4: Layered Diagram of the Worker Node

The worker node layered diagram is shown in Figure 4.4. In the figure, you can see

that the hardware and OS are the same for both PBS and Condor clusters. But, since PBS is the native scheduler on a PBS worker node, abstractions must be built on top of it. The Glidein Wrapper is a script I developed to create the glidein sandbox, and start the glidein daemons. This creates a single interface to Condor_Startd's, regardless of which cluster the user is communicating with.

Pilot jobs are used by many physics experiments [34, 48, 5]. Physics experiments create pilot frameworks because they want a consistent execution environment across hundreds of clusters. Pilot workflow management systems have the following benefits.

- **Scheduling Optimization:** The pilot reports only when a CPU is immediately available to run a job. This is in contrast to direct submission to a grid resource, where after submission, you have committed to running on that resource. If idle cores become available on another cluster, you are unable to execute on them as long as your job is submitted elsewhere.

- **Input/Output Automation:** The pilot can transfer input and output for the user directly to the worker node. This bypasses a possible bottleneck at the grid gatekeeper. Additionally, the pilot can be customized to transfer input from third parties such as storage elements.

- **Monitoring:** The monitoring of a job is improved by the pilot infrastructure. The grid translation layers between the local batch system and the grid interface can hide many errors and incorrectly report usage.

- **Fault Tolerance:** A job failure can be more accurately detected and recovered inside the pilot. The pilot can detect that the payload job has failed and report back the error. Additionally, it can verify the environment is acceptable for jobs before allowing jobs to begin.

- **Multiple Runs:** Each pilot can run multiple jobs serially, reducing submissions through the site gatekeeper. This will increase throughput since only a single authentication is necessary between the pilot and the user.

Condor Glidein jobs require several condor daemons packaged with a wrapper script. When the job starts, the Glidein job will:

1. Create a temporary directory on the local disk. This will be used for the job sandboxes.
2. Unpack the glidein executables onto the local disk.
3. Set the late binding environment variables for Condor to point to the temporary directory.
4. Start the `condor_master` daemon included in the glidein executables.

The `condor_master` will start the `condor_startd` daemon which will advertise itself to the glidein collector, making the node available for remote jobs.

Glideins are being used in production in the Open Science Grid using the software GlideinWMS [42].

4.1.4 Pilot Submission Algorithms

The `campus_factory` process decides whether to submit pilots to the underlying non-condor cluster. The `campus_factory` has two configuration options relating to submission of pilots to the LRM: `MaxIdleGlideins` and `MaxQueuedJobs`.

`MaxIdleGlideins`

An integer representing the number of idle slots that will be allowed before the `campus_factory` stops submitting jobs.

`MaxQueuedJobs`

The maximum number of queued pilots that will be idle in the queue of the LRM.

The `campus_factory` will query user queues and record the number of idle jobs. When there are idle jobs at user queues, it will use the following logic to determine how many pilot jobs to submit. The variable `idleuserjobs` in Algorithm 1 are the recorded number of idle jobs at user queues.

The `campus_factory` submission logic is as follows:

---

idleuserjobs ← QueryUserQueues()
if idlejobs < MaxIdleGlideins && queuedglideins < MaxQueuedJobs **then**
  toSubmit ← min(MaxIdleGlideins - idlejobs, MaxQueuedJobs - queuedglideins, idleuserjobs)
else
  toSubmit ← 0
end if
return *toSubmit*

Algorithm 1 Algorithim for determining how many pilots to submit.

Note that the QueryUserQueues function requires the `campus_factory` to implement external communication with the user queues.

Additionally, the `campus_factory` has logic to detect pilots that are not reporting to the collector. Pilots that have been submitted to the CGF's Condor instance will show their status as reported by PBS and BLAHP. The `campus_factory` will look for the same number reporting as 'Running', and reporting to `condor_collector`. If these two numbers differ by more than 10 percent, the `campus_factory` will stop submitting jobs. This can happen when the LRM cannot transfer files, if the BLAHP is incorrectly reporting the status of a job, or if there is something wrong with the pilot jobs.

4.1.5 OfflineAds

OfflineAds are a Condor feature that were designed to be used for power management. When a node hasn't been matched for a configurable amount of time, the machine can be turned off to save power. When the machine is preparing to turn off, it sends an OfflineAd to the collector that describes the machine so that it can be restarted if needed. OfflineAds are in the HCC Campus Grid used to optimize the submission of pilot jobs to the underlying batch system. The OfflineAd is just a normal ClassAd that includes all of the features that can be used when matching against an online resource, such as memory, disk space, and installed software. When the OfflineAds describing the machine are matched to a job by the Condor negotiator, the negotiator inserts a new attribute into the OfflineAd called `MachineLastMatchTime`. When used for power management, the Condor Rooster daemon periodically queries the collector for the OfflineAds. If one has been recently matched, then it wakes the corresponding node.

In the `campus_factory`'s implementation, the OfflineAds are used to match possible pilot resources to idle jobs. Dead pilots are thought of as "offline machines." Again, the `condor_negotiator` will treat the OfflineAds just as it would a real ad and matches it to idle jobs. Since the OfflineAd is an exact copy of a running glidein, it is reasonably expected that one can get a similar glidein when you submit to the local scheduler. As the negotiator sees no difference between running and OfflineAds, the OfflineAds will be matched even when flocking from another Condor pool.

Since the OfflineAds are exact copies of previously live pilots, the OfflineAds increase the accuracy of matching with idle jobs by exactly resembling running glideins. In GlideinWMS and previous pilot implementations, filters were applied to the user queues to determine if a job was capable of running on the resource. The filters

were customized by administrators to route jobs to the pilots that matched their requirements.

4.1.5.1 Influence OfflineAds Have on the CGF

The OfflineAds do not completely replace all the logic described in Algorithm 1. The site must still meet the idle glideins and idle slots requirements that were originally used to throttle new pilot submissions. The OfflineAds replace the need for the `campus_factory` to query remote schedds for idle jobs. This improves the efficiency and simplicity of the `campus_factory` by eliminating communication. But the large benefit is the increased accuracy of the pilot descriptions. The `campus_factory` will only submit jobs when the OfflineAds detect a definite match to the cluster's resources. The negotiator and collector take care of all job matching with accurate glidein ClassAds.

The logic of the `campus_factory` is described in Algorithm 2.

---

idleuserjobs ← QueryOfflineAds()
if idlejobs < MaxIdleGlideins && queuedglideins < MaxQueuedJobs **then**
  toSubmit ← min(MaxIdleGlideins - idlejobs, MaxQueuedJobs - queuedglideins, idleuserjobs)
else
  toSubmit ← 0
end if
return *toSubmit*

Algorithm 2 Algorithim for determining how many glideins to submit with OfflineAds

---

In contrast to Algorithm 1, this does not have any external communication. The QueryOfflineAds queries the local `condor_collector`.

4.1.5.2 Creating OfflineAds

After the `campus_factory` submits pilot jobs, it detects the classads of running pilot jobs, copies the classads, and re-advertises them as OfflineAds.

The changes required to transform a glidein classad into an OfflineAd are listed in Table 4.1.

ClassAd	New Value	Comment
Offline	True	Enable the OfflineAd logic in Condor daemons.
Name	Unique name	Mandatory name for indexing in the Condor collector.
MyCurrentTime	LastHeardFrom - Time now	Used for offset time.
ClassAdLifetime	24 hours	To address a bug in the handling of OfflineAds by Condor. This is how many seconds the collector will keep this ad.
State	Unclaimed	Make sure it will match with idle jobs.
Activity	Idle	Again, for matching
PreviousName	Name	Value of 'Name' attribute of the original ad. Useful for debugging.

Table 4.1: Changes to ClassAds for Offline Function

4.1.5.3 Managing OfflineAds

By default, the factory will attempt to maintain a specific number of OfflineAds. By default, it maintains the newest 10 ads. One can sort by different 'types' of machines (big memory, big disk), and keep an assortment of unique ads. This method will better represent the heterogeneous nature of the resource. Ten was determined to be an appropriate sample size as it is large enough to represent multiple nodes (currently eight cores per node standard). Larger number of ads will cause a heavier load on the CGF as it matches the OfflineAds to idle jobs.

The factory will maintain the newest 10 OfflineAds. If it detects less than ten, the OfflineAd manager will list the site as delinquent in an internal structure and will recommend the factory submit more pilots to the LRM.

4.2 Bridging Campus Grids

There are two methods for expanding the campus grid: through GlideinWMS to the OSG, or by bridging campuses through flocking. Bridging provides a method for jobs to leave the boundaries of the campus. The benefits of bridging externally are obvious–increased throughput for the local user's jobs. HCC has been able to bridge to the GLOW, Purdue, and FermiGrid grids discussed in Section 3.2. We connect to FermiGrid and GLOW through the OSG and to Purdue via Condor flocking.

Unlike the OSG, where the trust relationship is defined by a central consortium and agreed upon by all sites, trust is established between campuses with flocking on a case-by-case basis. The current model for trust is based on limited trusted hosts (IP based authorization). Each site publishes a list of submit and negotiator hosts that are trusted to submit and accept jobs, respectively. This implicitly trusts an entire campus, while the OSG trust model is based on virtual organizations that may have no relationship to a physical campus or submit host.

The ever-widening circle of resources expands from the locality of the user (Figure 4.5). It goes from the resource the user knows best (and has the best support for), the local cluster, to the most foreign one, the national grid. This is a very natural progression. Each step described comes with more complexity and new failure modes. If the user is ever frustrated at one transition, he can just remain contented with the resources he has, as opposed to having to switch between "local mode" and "grid mode," as must be done with Condor vs Condor-G. Another usability advantage is

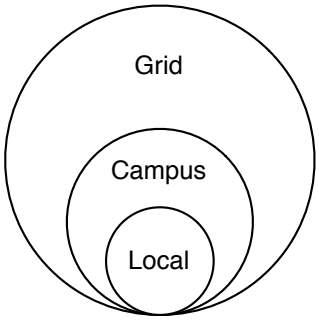

Figure 4.5: Widening Circle of Resources

that all end-user interfaces are Condor. The user never encounters errors translated between systems (a common user frustration); the user needs to develop expertise in Condor alone.

4.3 Full Campus Infrastructure

The full campus grid architecture with bridging is shown in Figure 4.6. This campus infrastructure includes all the on-campus resources, as well as meets the goals of the campus grid: Encompassing, Transparent, and Decentralized. The user first submits jobs to the local Condor cluster (1). If the local cluster can fulfill the user's needs, then all the jobs will remain there. If the local cluster is full or cannot meet the user's demand, Condor flocking will start jobs on other campus clusters (2), either with pure Condor or utilizing the CGF. If the on-campus resources are unable to meet the user's request, the local Condor schedd will expand its reach again (3) by looking outside the campus. The jobs can also be sent to the OSG via flocking to a GlideinWMS frontend, which creates an overlay pool of grid resources. In this architecture, every effort is given to find resources for the user (local, across campus, or across the nation), while maintaining the same Condor interface for the user.

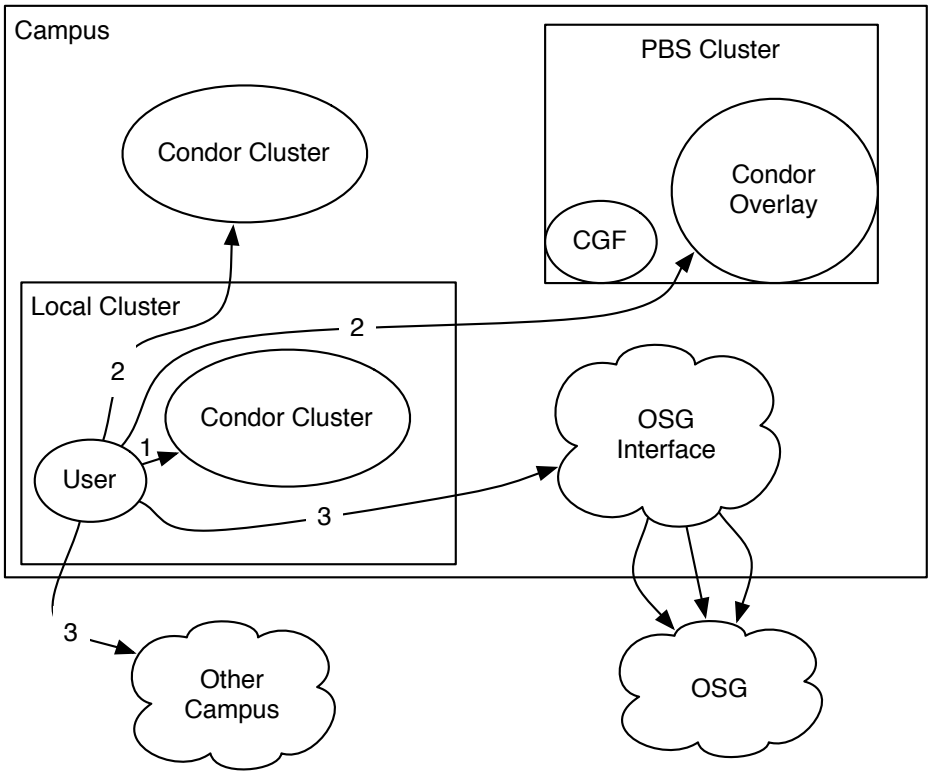

Figure 4.6: The Full Campus Grid Architecture

The software described in the previous sections are further shown in Figure 4.7.

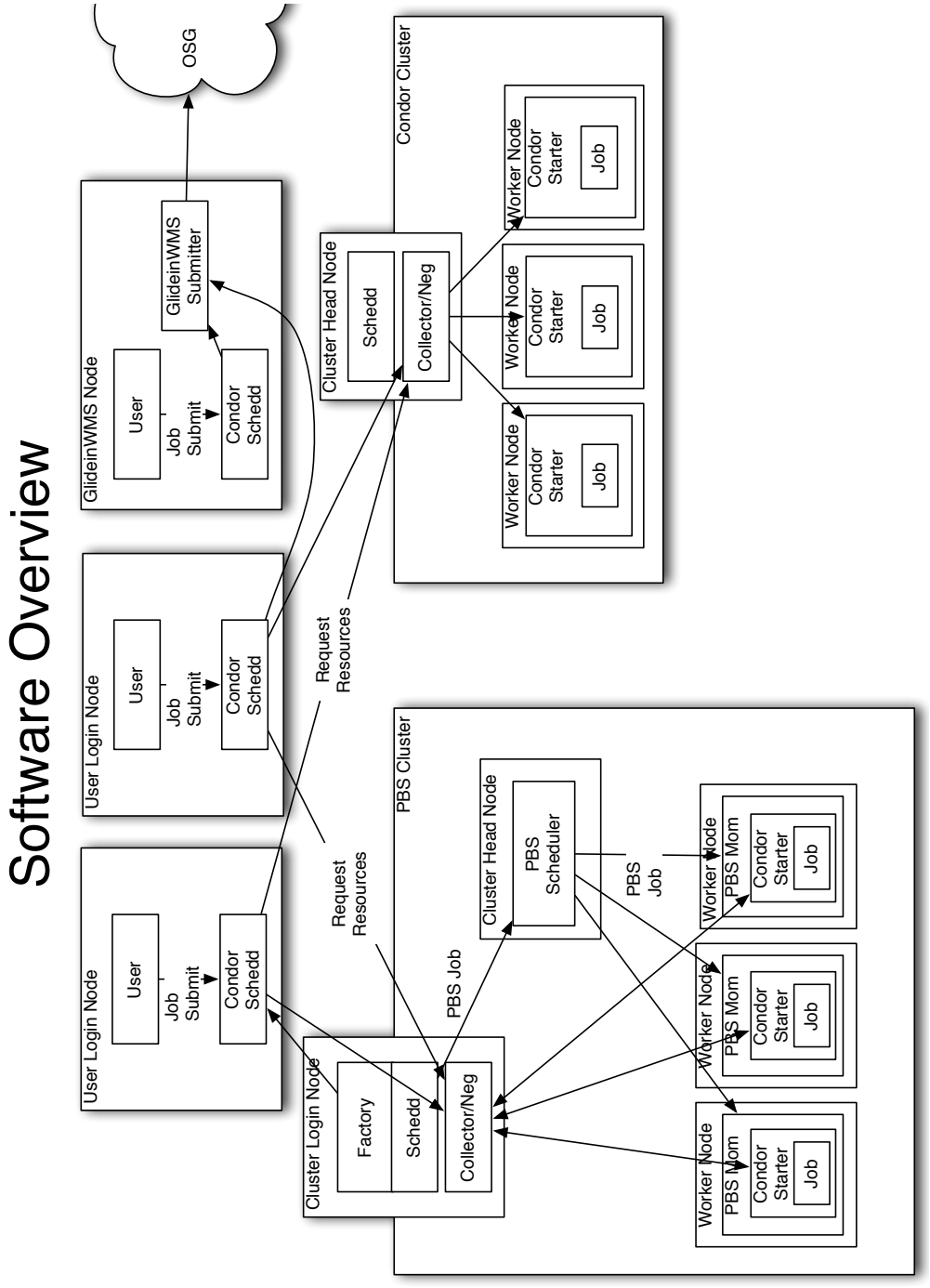

Figure 4.7: Overview of the Campus Grid software

Chapter 5

Evaluation

This chapter is divided into two sections, details of the HCC Campus Grid in Section 5.1, and evaluation and comparison of the HCC Campus Grid in Section 5.2.

5.1 University of Nebraska Holland Computing Center Campus Grid

In order to evaluate the framework described in this thesis, a campus grid was implemented at the Holland Computing Center (HCC). A diagram describing the HCC Campus Grid is shown in Figure 5.1. In this diagram the user submits jobs on a central machine. First, the jobs will attempt to run on campus resources at the clusters Prairiefire and Firefly. Next, it will branch out to the GlideinWMS interface and flocking to Purdue.

Like Purdue and GLOW, we have based the campus grid upon Condor and flocking. Each resource has a Condor-based interface, giving an identical user experience regardless of what the user considers his or her "local" cluster. While one of the local clusters run Condor as the primary batch system, the other is based upon PBS. PBS

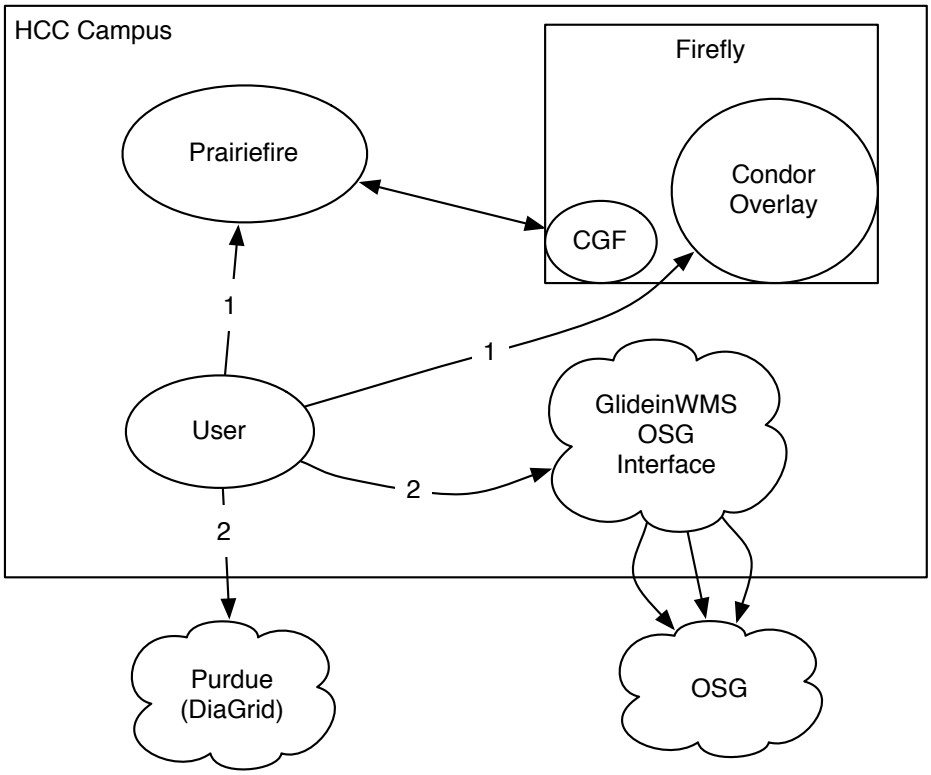

Figure 5.1: The HCC Campus Grid

was chosen because of its ability to plan large-scale parallel jobs run on the resource. GLOW's Condor-only approach did not fit our case due to the PBS requirement, and Purdue's model of running multiple schedulers was rejected because we wanted a less-invasive approach and because we wanted more efficient scheduling. The Campus Grid Factory (CGF) provides a Condor interface for our PBS cluster; as covered in Section 4.1.

Through Condor flocking and the CGF, we have successfully encompassed all local resources. To provide even more value to HCC, jobs can also bridge to the OSG and other campus grids. The interface to the OSG uses the GlideinWMS [42, 43] frontend software, while we link to other campuses using Condor flocking (the same method

Purdue uses to link the campuses of DiaGrid). Unlike Condor-G, which provides a Condor interface to GRAM, these two methods give the same user experience as using Condor as a batch system.

All clusters on the campus grid are managed by the Holland Computing Center; therefore, the trust relationship between the hosts are implicitly strong. A special account is set aside on the PBS cluster for the CGF to run campus grid jobs. The CGF daemon also runs as this user. On the Condor-managed cluster, campus grid jobs run as user `nobody` while locally-submitted jobs run as the submitting user.

Because resources are run by the same organization, we have the ability to provide distinct user priorities per resource through Condor. Further, because Condor runs *inside* PBS rather than alongside it, PBS can schedule its jobs without interrupting Condor ones. An administrator can prioritize jobs submitted directly from the local cluster over those from a remote submission, even for the same user.

Each submission host runs the Gratia accounting software to provide user accounting. Gratia was chosen because of:

- Integration into the larger OSG accounting.
- Separation between remote clusters and the central database (updates are done via HTTP).
- Ability to integrate new resource types easily.

. For integration in the OSG, we have extended the software to record both the submission host and the remote OSG cluster utilized.

HCC does not have a shared file system across all clusters, so the HCC Campus Grid data management is handled by Condor file transfer.

The environment includes two local clusters, as well as an interface to the OSG and another campus grid.

5.1.1 Prairiefire Cluster Configuration

Prairiefire is running both Condor and PBS in a side-by-side configuration. This is very similar to how Purdue uses Condor. When a PBS job arrives on a node, the Condor daemons will preempt the running job and move to another node. Therefore, PBS has priority access to nodes, while Condor is treated as an opportunistic user.

Condor runs on the head node of the cluster, which has both a public and private network interface. Each worker node connects to the outside world through a Network Address Translation (NAT) layer. The NAT is used to aggregate the outside connectivity of the cluster by funneling traffic through a single gateway. Since Condor must have direct communication between the submitter (possibly outside of Prairiefire) and the execute host (worker nodes), Prairiefire must run the Condor Connection Broker (CCB). The CCB runs on the head node and negotiates connections for nodes behind the NAT to connect with the submitter.

Prairiefire will schedule jobs from the Firefly CGF and the GlideWMS submit machine to run via Condor flocking. Prairiefire's head node can also flock jobs to Firefly. These interactions are shown in Figure 5.1. To do this, the configuration variable `FLOCK_FROM` was set to `ff-grid.unl.edu, glidein.unl.edu` and `FLOCK_TO` to `ff-grid.unl.edu`. Further, the security is set up such that jobs coming from outside of the Prairiefire pool will use the user `nobody` when running the job. Condor uses this unprivileged account so that a rogue user can cause minimal impact to the system, and to protect other users of the system. The security on the machine is IP based; it trusts all users from the `FLOCK_TO` and `FLOCK_FROM` hosts.

5.1.2 Firefly Cluster Configuration

Firefly is running the Campus Grid Factory (CGF) on a node straddling the border of the public and private networks. Unlike the interactive login node, the CGF host does not have a firewall installed as Condor uses many ports.

The CGF runs as a local unprivileged user. Condor is installed in this user's home directory and runs under this user's account. The CGF runs as a Condor job; it is maintained by the Condor schedd. Submissions to PBS use the default queue.

The Condor instance that runs on the gatekeeper is configured to allow flocked jobs from Prairiefire and the GlideinWMS submission node. Alternatively, users can submit jobs to the CGF's Condor instance at Firefly, allowing them to run on Firefly or flock to Prairiefire.

5.1.3 GlideinWMS OSG Interface Configuration

The GlideinWMS interface runs the GlideinWMS frontend software, as well as a Condor installation. The frontend periodically queries the queues of multiple campus machines to detect idle jobs. When idle jobs are present, the frontend sends a request for glideins to be submitted to the GlideinWMS factory. The HCC frontend requests glideins from the central OSG factory run at the University of California at San Diego.

Glideins are submitted to resources across the country on behalf of HCC. When the glidein jobs start on the remote resources, they pull Condor executables from the central factory, start them, and contact the HCC frontend to request jobs.

5.1.4 Flocking to Purdue Configuration

Flocking to Purdue is enabled by publishing a list of collectors and schedds at Purdue and HCC. This list is then manually inserted into the configurations of machines that

will send and receive jobs. The collector locations are placed in `FLOCK_TO` and the schedds are specified in `FLOCK_FROM`.

Security between the hosts are established with the `CLAIMTOBE` environment in Condor. In this environment, Condor trusts the daemons to give accurate information on job ownership and authentication. The security is further refined by limiting the authentication to only the collector and schedd hosts specified above. This is the same method used on the HCC campus grid described in Section 5.1.1.

In total, Purdue has seven collectors and six schedds participating in the flocking. HCC has two collectors and three schedds.

5.1.5 User Submission

User submission is by design very similar to submission on a dedicated Condor resource. The user will specify the executable and where to store the stdout and stderr. Input files are transferred per-job to the execute machine. Condor will automatically determine output files by scanning the sandbox directory for any new files created. The new files will be transferred back to the submitter.

Condor will only transfer files when the variables `should_transfer_files` and `when_to_transfer_output` are set. A typical submission file is shown in Figure 5.2. In the example, Condor will transfer the executable (`/bin/hostname`) to the execute host. Condor will return the stdout and stderr from the execute host back to the submitter.

5.2 Characteristics of HCC Campus Grid

This section evaluates the HCC Campus Grid on the characteristics defined in Section 2.1.

```
universe = vanilla
output = condor_out/ouptut
error = condor_out/error
executable = /bin/hostname
log = test.log


queue
```

(a) Non-Campus Grid submission

```
universe = vanilla
output = condor_out/ouptut
error = condor_out/error
executable = /bin/hostname
log = test.log
when_to_transfer_output = ON_EXIT
should_transfer_files = YES
queue
```

(b) Campus grid submission

Figure 5.2: Campus Grid Submission Scripts

5.2.1 Trust Relationships

On most campuses, trust relationships are very strong. On some campuses, a single group maintains the campus clusters. On others, the proximity of administrators has facilitated trust.

At HCC, one group administers the clusters on campus, therefore the trust relationship is strong. The execution gateways in the campus grid restrict access by IP address. Therefore, there is a set of trusted submission hosts. Each host trusts each other's claimed user authentication. Additionally, jobs running on the CGF use a valid user account rather than a unprivileged account such as user `nobody`.

When we compare this to other campus grids, we can see that the IP based filtering policy is consistent with some campuses, and less restrictive than others.

In the Virginia Campus Grid, the user uses LDAP and PKI for authentication. The user first authenticates with local LDAP servers, then creates a PKI certificate

to interact with the on–campus clusters. While the LDAP authentication is consistent with on-campus policies, PKI is not. The series of interactions complicates the authentication with the servers, and necessitates the creation of a separate daemon called CredEx. The HCC policy requires only one authentication with a submit host to have access to the campus grid.

In the Oxford Campus Grid, Kerberos is used for on campus submissions, while PKI is used for external access. This is consistent with on-campus policies.

The OSG focuses on the security of execution gateways (Compute Elements) as opposed to the security of submission hosts. Though a compromise of a gateway can lead to access of many short–lived, limited proxies, the compromise of a submission host can give access to a long–lived, unlimited proxy of a few users. Further, since the OSG allows unregistered submission hosts, and they are relatively easy to set up compared to the gateways, there are many submission hosts. OSG CE's are most often maintained by professional or knowledgable administrators; in contrast, submission hosts may be maintained by users with limited knowledge of security. Submission hosts could be compromised without knowledge of their owners, and could give access to unrestricted certificates. Thus, we argue the whitelisting of submit hosts under the care of known, dependable system administrators outweighs the lack of "strong security" between submit hosts and resources.

The GLOW and Purdue campus grids are similarly IP security based grids. The IP based security simplifies the setup of a grid based on Condor flocking that GLOW, Purdue, and HCC use. The submission host is additionally protected by the fact it only submits jobs to known good hosts.

The trust relationships inside campuses are simple compared to those outside of campus. When jobs begin flowing outside the local domain, the authentication must match that of the external entity. Jobs flow off campus through flocking to external

campuses and to the OSG through the GlideinWMS interface.

When flocking to external campuses, the HCC Campus Grid again uses IP based security negotiated with the campuses. In practice, this involves publishing a list of trusted hosts on each campus. Since campuses usually use a single authentication method for all of their machines, this creates a scenario where either a campus will trust the entirety of another campus, or not at all. For example, HCC trusts all of Purdue's submit hosts, and therefore all of the Purdue users. More accurately, we trust the Purdue administrators to monitor their users' usage, and to contain and contact us about possible security threats.

When jobs move to the OSG, we must match the authentication methods used on the OSG, which is PKI. GlideinWMS simplifies this as we use a single certificate to authenticate the GlideinWMS pilot, and user jobs may not require further credentials when running.

The HCC Campus Grid creates a web of trust inside campus composed of IP based security. Outside of campus, it conforms to the external requirements for authentication with either a published list of trusted hosts (flocking with Purdue) or a PKI certificate (OSG).

5.2.2 Job Submission

Users have the most interaction with the job submission mechanism; therefore, it must be simple and intuitive.

The Virginia campus grid and the Oxford grid use Globus to submit and receive jobs. Globus does not have load balancing or job distribution built in; therefore, Oxford created a resource broker to balance the load among campus resources.

In the HCC, GLOW, and Purdue campuses, jobs are flocked to execute hosts inside

the campus, automatically spreading out the load to available resources. Condor takes a greedy approach to scheduling jobs; if there is an empty slot, it will fill it without thinking of future submission or other resources. Therefore, if a user submits many jobs, and the first resource that it contacts has many idle slots, it will fill those slots without looking at other resources. Condor will evenly distribute resources across all users of the cluster.

Submission on the HCC, GLOW, and Purdue grid requires only two more lines to the regular Condor submit file as shown in Figure 5.2. Additionally, all negotiation and load balancing are handled by Condor internally; therefore, there are less dependencies in the campus grid infrastructure.

5.2.3 Resource Independence

In the OSG, resource independence is guaranteed by strict separation of resources; each resource has no dependencies on any other resource. This is accomplished by independent clusters having all necessary infrastructure installed locally, while only sending information to a distributed set of central services. The HCC campus grid attempts to mimic these design patterns.

While resources on the OSG are independent of each other, user job submission frameworks are not. For example, when using GlideinWMS, if the factory is disconnected or is terminated, no new jobs will start. More importantly, jobs will not start on local resources either. PanDA similarly cannot start jobs when disconnected from the central PanDA server at CERN.

The HCC Campus Grid installs all infrastructure to submit jobs on the local resource and distribute to remote resources. For a Condor cluster, this is simply the existing Condor install. For the PBS cluster, the CGF is installed locally. Therefore,

possible failures are:

- **Submitter failure**: If the user's submission machine is taken offline, only jobs submitted on this resource will be affected. Any currently running jobs will be recoverable for 15 minutes while the job lease is active.

- **Condor cluster network failure**: If the cluster becomes disconnected from the network, jobs running on the cluster from remote submitters will terminate after their 15 minute job leases have expired. Jobs submitted locally will continue to run on local resources, but will be unable to run on remote resources. If the Condor instance is terminated, the locally submitted jobs will be recoverable for the lease time, then terminate.

- **CGF installed cluster failure**: This could happen if the CGF crashes, the node running the CGF crashes, or any other scenario where the CGF becomes unavailable. Remotely submitted jobs will terminate after their lease has expired. Locally submitted jobs will continue to run on local resources, including pilots previously running under PBS. If the CGF is terminated, no more pilots will be submitted to PBS, but jobs will continue to execute on existing pilots. Condor will attempt to restart the CGF if it terminates abnormally. If Condor is terminated, local jobs will execute until their job leases have expired, and pilots will shut down after receiving no new jobs.

The HCC Campus Grid is resistant to failure due to design decisions regarding the placement of the Campus Factory. Compared to the centralized GlideinWMS factory, this provides more reliability.

5.2.4 Accounting

Accounting has two perspectives:

- **User**: How many resources have I consumed?
- **Resource Owner**: By whom and to what extend have my resources been used?

In the first perspective, the submitter or group of submitters want to know their usage (time). Installing accounting software on the submitter machines will account for usage of the submitters. Since submitter machines are tightly controlled in the HCC Campus Grid, we mandate the usage of accounting software. We use the accounting software from the OSG, Gratia, for this task. Gratia uploads records for each job from the `condor_schedd` to a HCC Gratia collector. The collector can then make usage graphs from its DB, such as Figure 5.3.

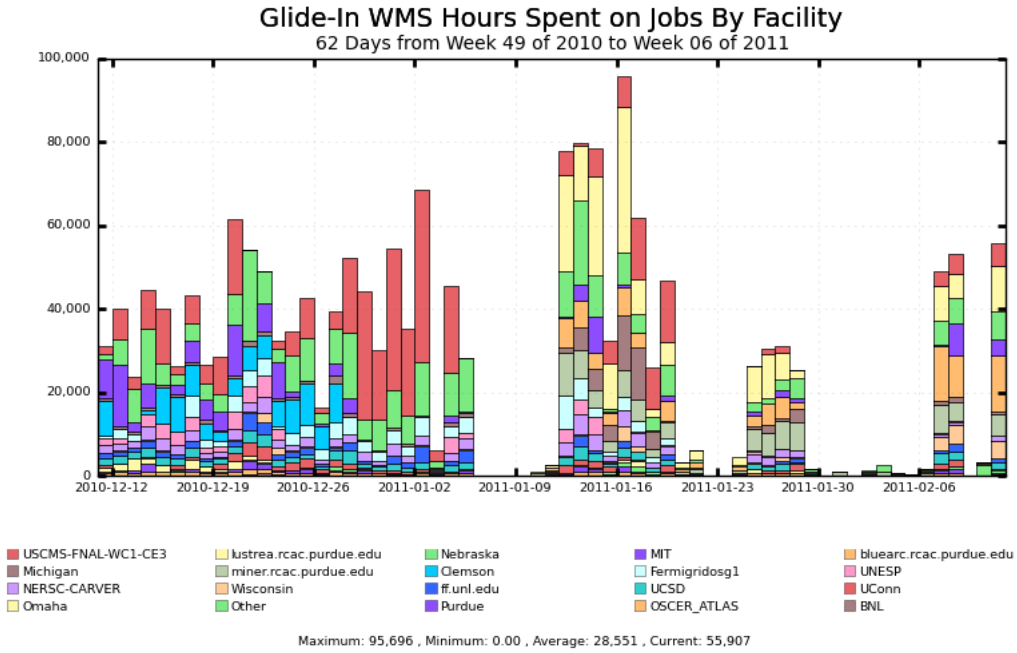

Figure 5.3: Snapshot of Accounting of the HCC Campus Grid

Accounting on the execute resource side is very useful and is the current model in the OSG. When a job is submitted in the OSG, it always passes through a gatekeeper on the local resource. Accounting is done on the gatekeeper since it will see every OSG job running at the resource.

Unfortunately, due to the nature of flocking, accounting on the execute resources is more difficult on the HCC Campus Grid. When flocking occurs, the submitter and the execute resource communicate directly, bypassing any gatekeeper. The only way to keep accurate accounting data is to collect it from the worker nodes. In the CGF installs, the accounting would need to run on the pilot submitted to PBS. Condor does not support creating Gratia compatible records on the execute site. This will be left for future work.

5.2.5 Data Management

Data management has been done differently by each campus. GLOW maintains a global AFS that is available on every worker node. Purdue has a few large file systems that worker nodes can access. FermiGrid has a global Network File System (NFS) space for data. Oxford grid uses the Storage Resource Broker (SRB) and a single vault.

The OSG promotes staging data to a nearby storage element which is difficult for individual users to implement. Normally a user will stage data to the gatekeeper, then transfer it to the execute host. This can lead to several bottlenecks when transferring large amounts of data to the gatekeeper.

HCC does not have a central file system, and instead uses Condor file transfer. This method has several benefits:

- Removes the gateway as a bottleneck by transferring files directly from submit-

ter to execute host.

- Reduces dependence on the gateway as a failure point. The gateway could terminate or become unavailable while a job is running and Condor would still be able to transfer back the output, and even start another job.
- Reduces the work that the gateway needs to perform. The authentication and authorization are done by the execute and submission hosts.
- Not dependent on the reliability or bandwidth of the shared file system. The submitter is relatively unaffected by other users that are not running on the same submission host.

There are some downsides to this approach. The largest is the reliance on the submitter machine. If the submitter becomes unavailable, all data transfers will fail. This is an acceptable risk; if the machine is unavailable for more than the 15 minute job lease, all jobs from the submitter will stop.

5.2.6 Updated table of Campus Grid Attributes

The Campus Grid attributes with HCC are shown in Table 5.1.

5.3 Usage

In Figure 5.4 one can see a snapshot of production jobs submitted from the GlideinWMS interface host running on the HCC Campus Grid. Note the total number of jobs running, (8612): this is larger than the total number of cores in any single Nebraska cluster and larger than the sum total of all cores managed by HCC (8000).

A description of the resources shown in the picture follows.

Grid	Trust Relationship	Job Submission	Resource Independence	Accounting	Data Management
Virginia	LDAP/PKI	None Described	Strict	Central	None
Oxford	Kerberos/PKI	Central	Central Submission	Custom	SRB
Purdue	Host	Distributed	Strict	Custom	Condor Transfer
GLOW	Host	Distributed	Strict	None	Condor Transfer
FermiGrid	PKI	Central	Strict	OSG Gratia	Central File System
OSG	PKI	Distributed	Strict	OSG Gratia	Distributed
HCC	Campus Defined	Distributed	Strict	OSG Gratia	Condor Transfer

Table 5.1: Updated Campus Grid Attributes

- **Local Resources**: ff.unl.edu, prairiefire.unl.edu
- **Peered campus resources**: bluearc.rcac.purdue.edu, lustrea.rcac.purdue.edu, miner.rcac.purdue.edu
- **OSG through GlideinWMS**: USCMS-FNAL-WC1-CE3 (Fermilab), Omaha (Globus submission to Firefly), Michigan, Caltech, NERSC-CARVER, Nebraska (Nebraska Tier 2), UCSD, UConn, BNL (Brookhaven), Wisconsin, MIT, OSCER_ATLAS (OU).

Another observation is the number of jobs running at the peered campus, Purdue. Purdue has large resources and peering with them has significantly increased the available resources to Nebraska researchers. As described in Figure 4.6, the submitter first looks at local resources `ff.unl.edu` (CGF) and `prairiefire.unl.edu` (Condor). Both had few resources available, so the submitter moved onto the OSG and peered campuses. The peered campus had many resources available (especially

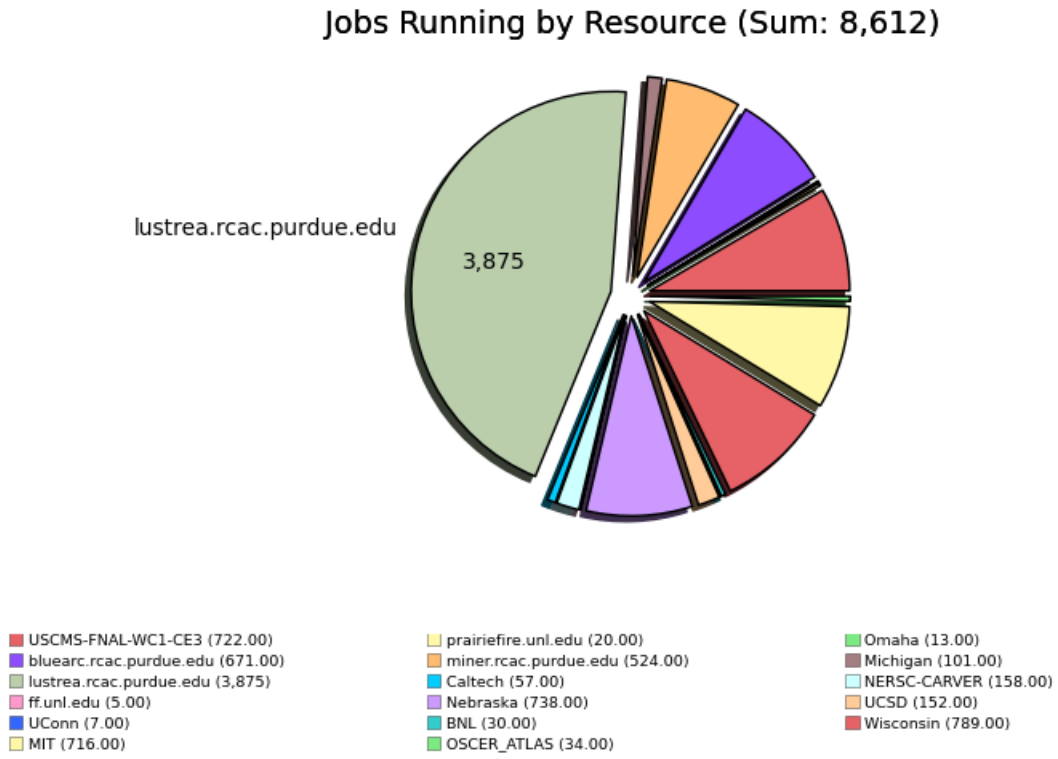

Figure 5.4: Snapshot of Usage of the Extended HCC Campus Grid

`lustrea.rcac.purdue.edu`). Sites in the OSG were able to start jobs (see Fermilab, Wisconsin, MIT).

Currently, the campus grid factory is used in production only on Firefly at Nebraska. It is currently being run in test environments at the National Center for Supercomputing Applications (NCSA) [33]. At NCSA, researchers from the Renaissance Computing Institute [41] have installed the CGF to flock from their submission host at their institution. Campus grids such as Louisiana Tech, Virginia Tech, and the Sunshine Grid in Florida are experimenting with the framework.

Chapter 6

Conclusions and Future Work

The framework described in this thesis creates a grid of clusters that transparently overflow to each other. The campus grid can overflow to national cyberinfrastructure and peered campuses through production interfaces. The CGF was developed to connect a PBS cluster into the Condor campus grid. The HCC Campus Grid is a production grid, running many users' jobs.

The framework includes many components developed by external organizations such as BLAHP (gLite), Condor (University of Wisconsin–Madison), and GlideinWMS (Fermilab). These components are glued together by Condor and the CGF to create a uniform grid. Additionally, I repurposed OfflineAds for efficient pilot job matching.

By using production components from other grids, I was able to build a framework that integrates all clusters on the campus into the HCC Campus Grid, as well as overflow to the regional and national cyberinfrastructure. Further it develops the idea of a tiered model: the framework prefers to run jobs locally before expanding to the campus, and finally overflowing to regional and national grids.

Further refinement of data management and accounting are left for future work.

Data management has proven difficult in the OSG, and it is no different in a campus grid. Transparent access to storage has been a goal for the OSG and major collaborators for some time. CMS and ATLAS are moving towards cache–based data distribution methods [6]. The campus grids should follow this model as well, whether this is as simple as using web caching, or more complicated such as utilizing Xrootd [10] for data distribution. Similar to the BLAHP, Condor, and GlideinWMS, the components developed by the large experiments CMS and ATLAS should be adapted for use on the campus grid. The effort available by the large experiments to adapt these components to their grids is much larger than the effort available for campus grids.

Accounting also will be improved in the future. It is important for resource owners to determine the hours given to external entities. This can be accomplished with the execution side reporting usage. This model is not used in the OSG and will need to be developed separately.

The campus grid ideas outlined in this thesis are being used in other states for their campus grids. Institutions such as Florida State and Louisiana Tech are installing the software and evaluating it for their campus grids. The Campus Grid area of the OSG will include the software and ideas developed here as the HCC Campus Grid.

Bibliography

[1] E.C. Amazon. Amazon elastic compute cloud, Jaunary 2011. `http://aws.amazon.com/ec2/`.

[2] P. Andreetto, S. Andreozzi, G. Avellino, S. Beco, A. Cavallini, M. Cecchi, V. Ciaschini, A. Dorise, F. Giacomini, A. Gianelle, et al. The gLite workload management system. In *Journal of Physics: Conference Series*, volume 119, page 062007. IOP Publishing, 2008.

[3] P. Andreetto, SA Borgia, A. Dorigo, A. Gianelle, M. Marzolla, M. Mordacchini, M. Sgaravatto, F. Dvorák, D. Kouril, A. Krenek, et al. CREAM: a simple, Grid-accessible, job management system for local computational resources. *CHEP 2006, Mumbay, India*, 2006.

[4] C. Baru, R. Moore, A. Rajasekar, and M. Wan. The SDSC storage resource broker. In *Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative research*, page 5. IBM Press, 1998.

[5] D. Bradley, O. Gutsche, K. Hahn, B. Holzman, S. Padhi, H. Pi, D. Spiga, I. Sfiligoi, E. Vaandering, et al. Use of glide-ins in CMS for production and analysis. In *Journal of Physics: Conference Series*, volume 219, page 072013. IOP Publishing, 2010.

[6] S. Campana, D. van der Ster, A. Di Girolamo, A. Peters, D. Duellmann, M. Coelho Dos Santos, J. Iven, and T. Bell. Commissioning of a CERN Production and Analysis Facility Based on xrootd. 2011.

[7] CERN. WLCG, Jaunary 2011. `http://lcg.web.cern.ch/lcg/`.

[8] K. Chadwick, E. Berman, P. Canal, T. Hesselroth, G. Garzoglio, T. Levshina, V. Sergeev, I. Sfiligoi, N. Sharma, S. Timm, et al. FermiGrid–experience and future plans. In *Journal of Physics: Conference Series*, volume 119, page 052010. IOP Publishing, 2008.

[9] D. Del Vecchio, M. Humphrey, J. Basney, and N. Nagaratnam. Credex: User-centric credential management for grid and web services. In *2005 IEEE International Conference on Web Services, 2005. ICWS 2005. Proceedings*, pages 149–156, 2005.

[10] A. Dorigo, P. Elmer, F. Furano, and A. Hanushevsky. XROOTD-A Highly scalable architecture for data access. *WSEAS Transactions on Computers*, 1(4.3), 2005.

[11] P. Eerola, T. Ekel
"of, M. Ellert, M. Grønager, J.R. Hansen, S. Haug, J. Kleist, A. Konstantinov, B. Kónya, F. Ould-Saada, et al. Roadmap for the ARC Grid middleware. In *Proceedings of the 8th international conference on Applied parallel computing: state of the art in scientific computing*, pages 471–479. Springer-Verlag, 2006.

[12] R. Egeland, T. Wildish, and S. Metson. Data transfer infrastructure for CMS data taking. In *XII Advanced Computing and Analysis Techniques in Physics Research*. Proceedings of Science, 2008.

[13] D.H.J. Epema, M. Livny, R. van Dantzig, X. Evers, and J. Pruyne. A worldwide flock of Condors: Load sharing among workstation clusters. *Future Generation Computer Systems*, 12(1):53–65, 1996.

[14] S. Farrell and R. Housley. Rfc3281: An internet attribute certificate profile for authorization. *RFC Editor United States*, 2002.

[15] European Organization for Nuclear Research. Atlas experiment, January 2011. `http://atlas.web.cern.ch/Atlas/Collaboration/`.

[16] European Organization for Nuclear Research. glite - lightweight middleware for grid computing, January 2011. `http://glite.cern.ch/`.

[17] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of High Performance Computing Applications*, 11(2):115, 1997.

[18] I. Foster and C. Kesselman. *The grid: blueprint for a new computing infrastructure*. Morgan Kaufmann, 2004.

[19] D. Fraser. OSG campus grids meeting, October 2010. `http://indico.fnal.gov/conferenceDisplay.py?confId=3674`.

[20] D. Fraser. Gratia, January 2011. `https://twiki.grid.iu.edu/bin/view/Accounting/WebHome`.

[21] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-G: A computation management agent for multi-institutional grids. *Cluster Computing*, 5(3):237–246, 2002.

[22] T. Howes and M. Smith. *LDAP: programming directory-enabled applications with lightweight directory access protocol*. Sams Publishing, 1997.

[23] M. Humphrey and G. Wasson. The University of Virginia campus Grid: Integrating Grid technologies with the campus information infrastructure. *Advances in Grid Computing-EGC 2005*, pages 50–58, 2005.

[24] Cluster Resources Inc. Cluster resources :: Products - moab grid-suite:. `http://www.clusterresources.com/products/moab-grid-suite.php`, December 2010.

[25] Cluster Resources Inc. Cluster resources :: Products - TORQUE Resource Manager:, Jaunary 2011. `http://www.clusterresources.com/pages/products/torque-resource-manager.php`.

[26] iRods. Irods:data grids, digital libraries, persistent archives, and real-time data systems, January 2011. `https://www.irods.org/index.php`.

[27] T. Kosar and M. Livny. Stork: Making data placement a first class citizen in the grid. In *Distributed Computing Systems, 2004. Proceedings. 24th International Conference on*, pages 342–349. IEEE, 2005.

[28] Lawrence Berkeley National Laboratory. Berkeley Storage Manager (BeStMan):, Jaunary 2011. `https://sdm.lbl.gov/bestman/`.

[29] Miron Livny and Rajesh Raman. High-throughput resource management. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.

[30] Holland Computing Center. Holland computing center, January 2011. `http://hcc.unl.edu/`.

[31] Peter Mell and Tim Grance. The NIST Definition of Cloud Computing. 2009.

[32] J.H. Morris, M. Satyanarayanan, M.H. Conner, J.H. Howard, D.S. Rosenthal, and F.D. Smith. Andrew: A distributed personal computing environment. *Communications of the ACM*, 29(3):201, 1986.

[33] NCSA. National Center for Supercomputing Applications at the University of Illinois, Jaunary 2011. `http://www.ncsa.illinois.edu/`.

[34] P. Nilsson. Experience from a pilot based system for ATLAS. In *Journal of Physics: Conference Series*, volume 119, page 062038. IOP Publishing, 2008.

[35] University of Wisconsin. Glow, January 2003. `http://www.cs.wisc.edu/condor/glow/`.

[36] R. Pordes, D. Petravick, B. Kramer, D. Olson, M. Livny, A. Roy, P. Avery, K. Blackburn, T. Wenaus, et al. The Open Science Grid. In *Journal of Physics: Conference Series*, volume 78, page 012057. IOP Publishing, 2007.

[37] A. Rajasekar, R. Moore, and F. Vernon. iRODS: A Distributed Data Management Cyberinfrastructure for Observatories. In *AGU Fall Meeting Abstracts*, volume 1, page 1214, 2007.

[38] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *High Performance Distributed Computing, 1998. Proceedings. The Seventh International Symposium on*, pages 140–146. IEEE, 1998.

[39] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *High Performance Distributed Computing, 1998. Proceedings. The Seventh International Symposium on*, pages 140–146. IEEE, 2002.

[40] D. Rebatto, F. Prelz, G. Fiorentino, M. Mezzadri, E. Martelli, and E. Molinari. Blahp: A local batch system abstraction layer for global use. Poster, 2006.

[41] Renci. Renaissance Computing Institute, Jaunary 2011. `http://www.renci.org/`.

[42] I. Sfiligoi. glideinWMS–a generic pilot-based workload management system. In *Journal of Physics: Conference Series*, volume 119, page 062044. IOP Publishing, 2008.

[43] I. Sfiligoi. Making science in the Grid world: using glideins to maximize scientific output. In *Nuclear Science Symposium Conference Record, 2007. NSS'07. IEEE*, volume 2, pages 1107–1109. IEEE, 2008.

[44] P.M. Smith, T.J. Hacker, and C.X. Song. Implementing an industrial-strength academic cyberinfrastructure at Purdue University. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–7. IEEE, 2008.

[45] J.G. Steiner, C. Neuman, and J.I. Schiller. Kerberos: An authentication service for open network systems. In *Proc. Winter USENIX Conference*, pages 191–201. Citeseer, 1988.

[46] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: The Condor experience. *Concurrency and Computation: Practice and Experience*, 17(2-4):323–356, 2005.

[47] D.C.H. Wallom and A.E. Trefethen. Oxgrid, a campus grid for the University of Oxford. In *Proceedings of the UK e-Science All Hands Meeting*, 2006.

[48] M. Zvada, D. Benjamin, and I. Sfiligoi. CDF GlideinWMS usage in Grid computing of high energy physics. In *Journal of Physics: Conference Series*, volume 219, page 062031. IOP Publishing, 2010.