CAMPUS GRIDS

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 Engineering

Under the Supervision of Dr. David Swanson

Lincoln, Nebraska

May, 2011

CAMPUS GRIDS

Derek Weitzel, M. S.

University of Nebraska, 2011

Adviser: Dr. David Swanson

It is common at research institutions to maintain multiple clusters. These might fullfill different needs, different policies, or represent generations of hardware. Many of these clusters are under utilized while researchers at other departments may require these resources. Linking these clusters with traditional grid software would require changes in security and execution environments. In this paper we will describe a framework and technology that are used to link departmental clusters such that submission at one cluster could lead to execution on another. We evaluate the framework on five important attributes found in campus grids. This framework is then further expanded to bridge campus grids into a regional grid.

Contents

Chapter 1

Introduction

A campus grid is a framework for distributing computation among independent clusters within a campus. A campus typically consists of 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. The compute clusters can benefit from increased utilization, while the users can benefit by increased processing power.

The Holland Computing Center (HCC) created a campus grid that spans two clusters and can overflow onto national grid infrastructure. The HCC grid borrowed concepts and techniques from earlier campus grids and a national grid. The campus grid is quickly becoming the model of a homogeneous campus grid at other sites.

The HCC campus grid bridges clusters and the national infrastructure while running production processing jobs from on-campus researchers. Over the last X (TODO find stats on number of hours) we have ran Y hours on the campus grid, along with Z hours bridging to other campus grids.

In many campus grids, the user manually submits to specific clusters. In others, a layer of software distributes the computation among clusters in the grid. Generally,

users are provided a uniform interface to submit jobs to remote clusters inside their campus.

Building a campus grid represents a significant commitment for both users and resource providers, which should be evaluated against the benefits. The primary benefits are:

- **Resource sharing**: An HTC-based approach focusses on using all resources effectively. Resources are typically bought for peak, not average, usage; integrating the idle time across the entire campus and using it 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 do 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.

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 flows: 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, as these 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.

A common advantage for using campus grids is to alleviate demand on newer parallel machines by moving single core jobs to other clusters that have idle cycles. Recently, the single core performance has been relatively stagnant, therefore moving the single core jobs to the older hardware will not significantly decrease performance. Also, by moving the single core jobs, it can free the newest hardware for large parallel jobs which can benefit from better interconnects, larger and faster storage, and increased core count that are on the newest hardware.

There have been several attempts 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 Moab [19]. Others are translation layers between a generic description language and a scheduler, such as Globus [14]. Still others are entire resource managers that can span multiple clusters like Condor [36]. Each of these solutions can be built to create a campus grid, but they all have drawbacks that are highlighted in Section 2.2.

The framework described in this paper creates a production campus grid. The campus grid includes technology to allow clusters to participate in the campus grid. It integrates 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 distributed resources. The jobs are sent to available resources on distributed clusters on the campus grid. Further, it can expand beyond the campus and onto other campuses by simple configuration changes.

The campus grid is used in production at the University of Nebraska, connecting two geographically separate clusters. Users can utilize the campus grid by submitting to one of the connected clusters, while their jobs run on either. The jobs are transported with their input files to a execution sandbox on the worker node. Further, the campus grid expands beyond Nebraska, peering with the Purdue campus grid.

1.1 Attributes of Campus Grids

In this section, we explore five characteristics of HTC-centric campus grids. While the list isn't exhaustive, we've found campus grids can be characterized by how they approach trust relationships, job submission, resource independence, accounting, and

data management.

1.1.1 Trust relationships

A successful 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 VOMS attributes a PKI extension [11]. The GSI model is widely accepted, allowing the OSG to participate in the Worldwide LHC Computing Grid (WLCG) [5]. Fermigrid's campus grid user authentication is based on the GSI model. While it provides a highly secure, decentralized authorization model and proven at the worldwide scale, 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 degree of trust than at the national level due to sociological reasons. This trust may just be based on locality – it is easier to establish a working relationship with a colleague locally on campus than 1000 miles away.

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 is applied to submit hosts rather than cluster entry

points.

Security requirements on some university campuses are simply less stringent than that of federal labs, explaining Purdue and Wisconsin's preference for IP security compared to FermiGrid's GSI. 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 most be able to associate GSI credentials with their jobs.

1.1.2 Job submission

In order for a HTC-oriented campus grid to function, users need a usable job submission interface. The Globus Toolkit [14] provides the GRAM interface for job submission and corresponding clients. GRAM layer abstracts above the batch system; the user interacts with GRAM, and GRAM converts these actions into batch system commands at the destination. The GRAM interface is used by the OSG, and is being used at the scale of over 100 million jobs a year. The GRAM interface 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 [17] , 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 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 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 - error propagation is extremely difficult. In the Purdue, GLOW, and HCC campus grids, 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 [34].

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.

1.1.3 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 or departmental level. One characteristic of campus grids is thus 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 - GLOW's 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 chose 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 - a binary compiled for Linux will not be compatible with Windows. Some guarantees about the runtime environment or other interfaces need to be clearly articulated to prevent frustration. Differences that are unavoidable or are expected to be handled by the user should be clearly expressed to the user [31]. At the OSG level, we have found the users often frustrated by the amount of heterogeneity, especially compared to using a single site or a grid with a smaller number of sites [38].

1.1.4 Accounting

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

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 OSGs, Gratia [16], is designed to be reusable by other organizations, and is in use by all four campus gridsdiscussed in this paper.

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?

1.1.5 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 his personal systems, and archiving requires expertise outside his 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 and may break down in highly-distributed campuses.

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 like jobs (Stork [22]), or promoting data to be the central concept instead of jobs (iRODS [21, 30]). The CMS model separates the data management and job submissions systems, allowing the job submissions to simply assume all data is available (CMS PhEDEx [9]).

While any system can be used for campus grids, the examples we consider in Sections 2.2 and 4.1 either export a file system or utilize the tools from the job submission system. The addition of a separate data management system often presents such complexity to the users that not buying more hardware and not using 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.

1.2 Background

1.2.1 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 [24]. This is in contrast to High Performance Computing (HPC), where users are concerned with 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 coordina-

tion, 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 a 400,000 jobs and a million computational hours a day using HTC.

1.2.2 Condor

Condor is a high–throughput batch computing system developed at the University of Wisconsin - Madison. An overview from [36]:

> 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 ClassAd mechanism. ClassAds are the language that Condor uses to communicate between daemons. 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 requirements to run on the 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 [17]. Condor-G communicates with Globus [14] 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 [20].

1.2.3 Open Science Grid

The Open Science Grid (OSG) [29] is a national cyber infrastructure consortium that provides dedicated and opportunistic use of computation and storage resources. The OSG consists of nationally distributed universities and national laboratories that provide access to shared computing and storage. The OSG provides infrastructure, software, and engagement to the users.

The OSG provides a common interface to computing and storage resources: Globus and SRM/GridFTP respectively. Clients can access the resources by a OSG provided client. Both Globus and SRM/GridFTP where developed by consortium members and are included into the OSG packaging. Many software modules have been contributed to the OSG software stack such as Condor and the Berkeley Storage Manager [23].

1.3 Paper Overview

The rest of this paper first discusses technology to create campus grids as well as existing campus grids in Section 2, and then describes our implementation in Section 3. Section 4 describes how we evaluated our system and presents the results. Section 5 presents our conclusions and describes future work.

Chapter 2

Related Work

2.1 Technology to create a campus grid

2.1.1 Globus

Globus is a translation layer between the (globus specific?) Resource Specification Language, and the local resource manager. It has been very successful in that is has a large install base. Globus also has deep integration with standard grid credentials such as PKI.

Placing Globus gatekeepers on each cluster would allow jobs to be submitted to each cluster without modifying the underlying batch system. But, this would require a higher layer of abstraction over the Globus gatekeepers to optimally balance load between clusters. Additionally, Globus has well known limitations such as job a low submission rate, and high resource usage. Additionally, Globus implements GSI (certificates) security that is inconstant with most existing campus security architectures. Also, it does not provide a method for transparent execution on other clusters, which experience on the OSG has shown is important to users.

2.1.2 Condor Flocking

Another single vendor solution is Condor. Each resource can run Condor on their clusters and 'flock' [10] to each other. In this solution, jobs would be balanced on each resource due to Condor's greedy scheduler algorithm.

Flocking jobs is accomplished by a multi-step process. First, the `condor_schedd` reports to the remote `condor_collector` that it has idle jobs available to run. During it's next iteration, the remote `condor_negotiator` contacts the `condor_schedd` to match available resources to the requested jobs. When the `condor_schedd` receives a match, it contacts the resource directly to claim it. After the claim is successful, the job starts on the remote resource.

Condor flocking has many advantages. Since the job submitter (`condor_schedd`) directly contacts the executing resource, there is no additional load on any central node. 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.

But, this solution again requires each resource to run Condor as their scheduler and resource manager. Additionally, Condor must be running on each worker node, increasing the administration requirements. This suffers from the same problem as Moab, restricting further innovation to the confines of a single solution.

2.1.3 GlideinWMS

The Glidein Workflow Management System (GlideinWMS) [33] is a job submission method that abstracts the grid interface, leaving only a batch system interface. GlideinWMS separates the system into two components, the frontend and the factory. The frontend monitors the local user queue and requests glideins from the factory.

The factory submits the pilots to the grid resources, while serving requests from multiple frontends. The factory only submits jobs to grid interfaces on multiple resources and can be optimized for that purpose. While the frontend only deals with the local batch system, and can be optimized for the user's jobs.

The GlideinWMS system is managed by Condor. It uses Condor to submit to the grid resources, as well as mange 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 CMS [4]. They have shown recently that GlideinWMS can scale beyond 25,000 running jobs.

GlideinWMS does have a few drawbacks. GlideinWMS uses a 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.

2.1.4 Panda

Panda [25] is the distributed production and distributed analysis system for the ATLAS Experiment [12]. Panda is a pilot based submission system developed by US ATLAS. 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 to be based around a central server. All jobs are submitted to this single server that centrally manages all job information. The user submits jobs using 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 since it is integrated with the ATLAS data management system. The client interface is generic enough that jobs can be submitted to multiple grids transparently. Accounting and monitoring are tightly integrated into the system and can be easily monitored since all jobs are run from a central server.

The PANDA system is very reliant on the uptime of the central 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. Though, in practice, this has been very reliable.

2.1.5 Moab

One solution to build a grid is to use a single vendor/software solution. For example, Cluster Resources offer a solution Moab Grid Suite[19]. This solution requires each resource to run a single piece of proprietary software, Moab. Moab is a meta-scheduler, using PBS to manage the underlying resources. By using Moab, the development of new grid technologies are limited to what can be done in Moab.

2.2 Other Campus Grids

2.2.1 Virginia Campus Grid

The Virginia Campus Grid [18] designed a campus grid using the Web Services Resource Framework (WSRF) with Globus. The goal of the campus grid was use as much existing infrastructure as possible. The grid utilized the existing authentication system by developing a new credential generator called CredEx [7] that interacts

with the local LDAP servers to create PKI certificates. Globus version 4 (now depreciated) 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, a 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 resources preference for 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. Resource independence is not discussed in the paper, but there are many central services such as the enforcement service and the broker. A downtime in either of these services would limit usefulness the grid. Accounting is handled with the central IT along with authentication. Data management is not addressed in the campus grid. Data management is very important to any campus grid given the rising demands of data intensive computing for scientific applications.

2.2.2 Oxford Campus Grid

The Oxford Campus Grid [37] built a comprehensive campus grid including both compute and data provisioning. The compute provisioning uses Condor-G [17] 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 follow resource policies. For data provisioning, the Storage Resource Broker (SRB) [3] was used with a dedicated data vault. 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 very closely 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.

2.2.3 GLOW

GLOW [15, 28] is an University of Wisconsin grid used at the Madison campus to distribute jobs on their all-Condor grid. 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 is managed by an AFS [26] install and Condor file transfer.

2.2.4 Purdue

The Purdue campus grid [35, 15] is part of a larger grid, Diagrid, which serves a number of smaller 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 on the execute host. In order to maximize the resources in its grid, Purdue also installs Condor on its PBS-based clusters. Each batch system makes decisions independently, 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.

2.2.5 FermiGrid

Fermigrid [15, 6] is made up of resources located at the Fermi National Accelerator Laboratory in Batavia, IL. The Fermigrid campus grid is the closest example found of a "mini-OSG". Its uses the same CE software, information systems, and storage elements as the OSG. Trust relationships on Fermigrid are based on the Grid Security Infrastructure (GSI) [11], the same authentication method used by OSG. Job submission is managed by Condor-G through a Globus submission layer to the clusters. This same method can be used to submit to OSG, providing one strategy to getting users from campus to the national grid. Some clusters are managed by a central team, while others are done independently. Some of the grid services (authorization and information services, for example) are run centrally. Accounting is done through Gratia [16], the same software that is used on the OSG. A central cluster file system is available to most clusters, but Globus-based file transfer is also heavily used.

Chapter 3

Implementation

In designing a campus grid for the Holland Computing Center, we attempted to meet three goals:

- **Encompass**: The campus grid should reach all clusters managed by HCC.
- **Transparent execution environment**: There should be an identical user interface for all resources, whether running locally or remotely.
- **Decentralization**: 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.

Along with these goals, the technologies in this section will address the characteristics of a campus grid described in Section 1.1: trust relationships, job submission, resource independence, accounting, and data management.

3.1 Campus Grid Factory

To address the encompass goal, the Campus Grid Factory (CGF) was designed to bridge non-condor clusters into the campus grid. Additionally, 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 Factory is a daemon that runs on non-condor clusters in order to submit glideins when additional resources are requested. The Condor instance that the factory communicates with must be on a 'gateway' node: Able to talk to both the remote clusters and the local nodes. The Factory communicates with the `condor_collector` daemon in order to detect requests for resources, and the `condor_schedd` daemon to submit jobs to the LRM.

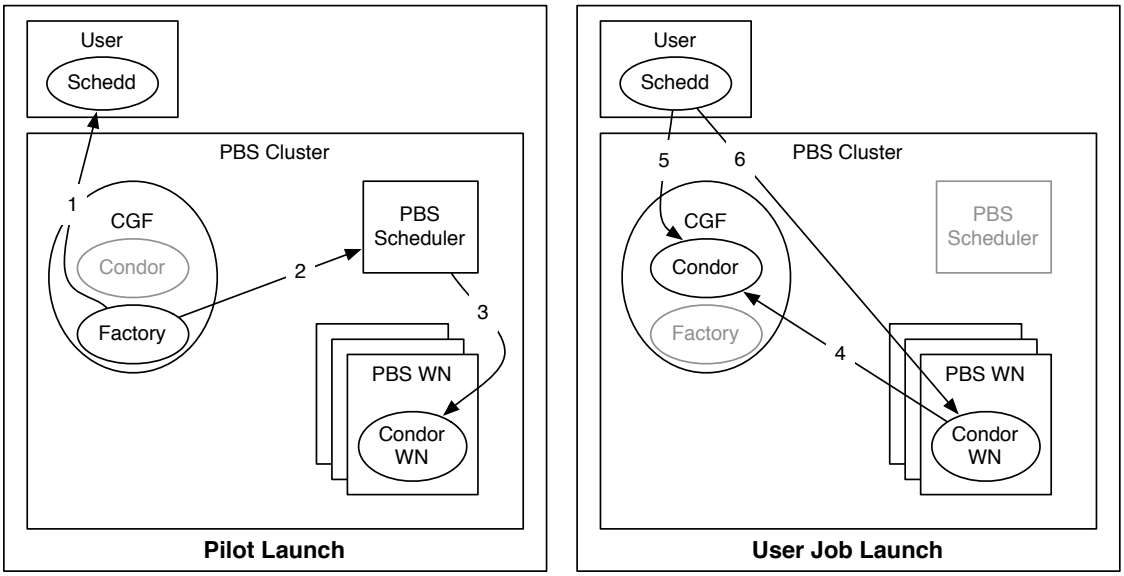

Figure 3.1: Overview of Campus Factory function

The Campus Grid Factory overview is shown in Figure 3.1. The factory software starts by querying all the Condor schedd's on the campus to determine if they have

jobs to run (1). If idle jobs are found, the 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 is 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 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, running the user job.

The factory 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 factory. The factory submits glideins to the Local Resource Manager (LRM), allowing them to run with the priorities set in the LRM. The jobs, once started, report back to the factory collector as a regular condor pool node. The factory collector then is able to route jobs from other clusters to these available glideins just as it would for an all-condor cluster.

There are many architectural similarities between the CGF and GlideinWMS teams. However, GlideinWMS was not chosen to run the CGF as the GlideinWMS is designed to have one central factory and a frontend on each submit host. Therefore, if a non-Condor cluster becomes disconnected from the factory, even jobs that are submitted on the local cluster will be unable to run on the cluster since the factory cannot reach it, violating the decentralization goal. The current CGF merges the roles of the frontend and factory in the GlideinWMS architecture, removing some complexity from the submit host. At the core of GlideinWMS is the use of GSI for security; while HCC uses GSI security for its OSG work, we want to avoid making it a requirement for the campus.

The GlideinWMS system does a superior job of preparing and validating runtime environments compared to the CGF, an essential element for removing a common source of grid frustration. However, we believe the runtime environment problem is mitigated on campuses because of the smaller number of resources and the closer working relationships between system administrators.

The Campus Factory is a python daemon that runs as a persistent condor job. Since it runs as a condor job, the condor daemons will ensure that it stays alive, eliminating the need to monitoring an additional daemon.

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

- Run and maintain the factory daemon.
- Submission of the glidein jobs to the LRM (See Section 3.4).
- Collect and advertise information on the glidein jobs.
- Negotiate with submitters in order to route jobs to glideins.

3.1.1 Determining when to submit glideins

The factory decides whether submit glideins to the underlying non-condor cluster. The factory has 2 configuration options relating to submission of glideins to the LRM: `MaxIdleGlideins` and `MaxQueuedJobs`.

`MaxIdleGlideins`

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

`MaxQueuedJobs`

: An integer number of queued glideins that will be allowed to be idle in the LRM before submitting more.

The factory submission logic is as follows:

---

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 glideins to submit.

Additionally, the factory has logic to detect glideins that are not reporting to the collector. This can happen when the the LRM cannot transfer files, if the BLAHP is incorrectly reporting the status of a job, or if there is something wrong with the glideins jobs.

3.1.2 OfflineAds

OfflineAds are used to optimize the submission of glideins to the underlying batch system. Offline ads are 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 offline ad to the collector that describes the machine. The OfflineAd 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 offline ad describing the machine is matched to a job by the Condor negotiator, the negoiator inserts a new attributed into the offline ad called `MachineLastMatchTime`. When using power management, the Condor Rooster periodically queries the collector for the offline ads, and wakes the powered off matched nodes to run the job.

In the factory's implementation, the OfflineAds are used to match possible glidein resources to idle jobs. When the OfflineAds are collected, the The condor negotiator will treat the offline ads just as it would a real ad and match it to idle jobs. Since the Negotiator sees no difference between running and offline ads, the offline ads will be matched even when flocking from another condor pool.

Since the OfflineAds are exact copies of active glideins, the OfflineAds will increase the accuracy of matching with idle jobs by resembling running glideins. Descriptions of the glidein host and installed software will be represented in the OfflineAds just as it would a regular glidein ad.

3.1.2.1 Influence OfflineAds have on the Factory

The OfflineAds do not completely replace all logic described in Algorithm 1. The site must still meet the idle glideins and idle slots requirements that were originally used to throttle new glideins. The OfflineAds do replace the need for the factory to query remote schedd's to for idle jobs. This improves the efficiency of the factory by eliminating communication. The negotiator and collector take care of all job matching with accurate glidein classads.

The logic of the factory follows:

1. Are there idle startd's?
2. Are there idle glideins in queue?
3. Have any of the idle offline ads matched in the last x seconds?
4. If we get this far, submit some glideins.

3.1.2.2 Creating OfflineAds

During operation of the factory, it will submit glideins to the underlying batch system. The factory detects the classads of running glideins, copies the classads, and re-advertises them as offline ads. Since the offline ad is an exact copy of a running glidein, it is reasonably expected that you can get a similar glidein when you submit to the local scheduler.

The changes required to transform a glidein classad into an offline ad is in Table 3.1.

ClassAd	New Value	Comment
Offline	True	Enable the offline ad 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 offline ads 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 3.1: Changes to ClassAds for Offline function

3.1.2.3 Managing OfflineAds

By default, the factory will attempt to maintain a specific number of offline ads. This can be done in a number of ways, such as always maintaining the newest 10 ads. Or you can sort by different 'types' of machines (big memory, big disk), and keep an assortment of unique ads. Currently, the factory only keeps the latest 10 ads.

To maintain the classads, the factory queries the collector for offline ads. If it detects more than 10, it will only keep the newest 10. If less then ten, the offline ad manager will list the site as Delinquent, and will recommend submitting more glideins.

3.2 Glidein Jobs

Glidein jobs compliment the job submission method and are used to present a transparent execution environment to the submitter. Additionally, it allows for better data management since the submitter directly connects to the execute host to transfer files and job information.

Glidein [17] is a pilot job based grid submission that creates an overlay network. Glidein is designed to used standard Condor mechanisms to advertise it's availablity to a Condor Collector process, which is queried by the Scheduler to learn about available resources. Each user job is run in a sandbox on the local disk and is provided with a consistent execution environment across hosts and clusters.

Pilot jobs are used by many physics experiments [27, 38, 4]. Physics experiments create pilot frameworks because they have need for very large distributed workflow management systems. Pilot workflow management systems are commonly used for multiple reasons:

- **Scheduling Optimization:** The pilot reports only when a cpu is immediately available to run a job.
- **Input/Output Automation:** The pilot can transfer input and output seamlessly for the user.

- **Monitoring:** The monitoring of a job can be more accurate while the job is running.
- **Fault Tolerance:** A job failure can be more accurately detected and recovered.
- **Multiple Runs:** Each pilot can run multiple jobs serially, reducing submissions through the site gatekeeper.

The factory uses glideins on non-condor resources in order to create a overlay condor cluster that can execute remote jobs via flocking. The glideins report to a Collector unique to each Cluster and are configured to exit after a configurable amount of time.

The glidein job requires six 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. Unpackage the glidein executables into 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` which will advertise itself to the glidein collector, therefore making the node available for remote jobs.

Glideins are being used in production in the Open Science Grid using the software GlideinWMS [33]. They provide several advantages over regular job submission. Each glidein sandboxes and monitors the user jobs it runs. The glidein can run multiple user jobs inside a single Local Resource Management (LRM) job, and will continue

to run until the configured time to stop. Glidein host connects directly with the submitter host, removing the need for staging data.

3.3 Flocking

TODO: Picture of Flocking

Flocking [10] is a method of linking Condor clusters into a larger grid. When a user submit 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 specifying pools for execution. It improves data management by directly transferring files from the submitter to the execute host, bypassing the gatekeeper (typically done in the OSG). It also allows for decentralization since pools can leave and appear in the campus grid independent of the submitters.

After the jobs have been submitted, the `condor_schedd` will maintain a `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, then the `FlockLevel` will increase by 1. Each time the `FlockLevel` increases, the `condor_schedd` will advertise to another Condor pool 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.

Flocking does not delegate responsibility for a job. The original `condor_schedd` will maintain the job, transferring input and output, and monitoring it's status. When flocking a job, first the negotiator for the remote pool contacts the local `condor_schedd`. If a match is found, the schedd directly contacts the execute host.

After transferring files, the schedd will monitor the status of the job. This is analogous to a hub and spokes, the job never moves, it just is executed elsewhere.

3.4 Condor & BLAHP

The Batch?system Local ASCII Helper Protocol (BLAHP) was designed to offer a simple abstraction layer over the different LRMS, providing uniform access to the underlying computing resources [32]. BLAHP is maintained by the gLite [13] collaboration at CERN. The BLAHP supplements the encompass 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. The Campus Grid Factory submits glidein jobs to the underlying resource manager through the BLAHP. Submitting glideins to Condor and allowing the BLAHP to manage the jobs simplifies the factory.

3.5 Full Campus Infrastructure

The full campus grid architecture with bridging is shown in Figure 3.2. In this campus grid, the user first submits jobs to the local Condor cluster (1). If the local cluster can fulfill the user's needs, then the jobs will remain there. If the local cluster is full or cannot meet the user's demand, Condor flocking will start submitting jobs to other campus clusters (2), either Condor or CGF-based. 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.

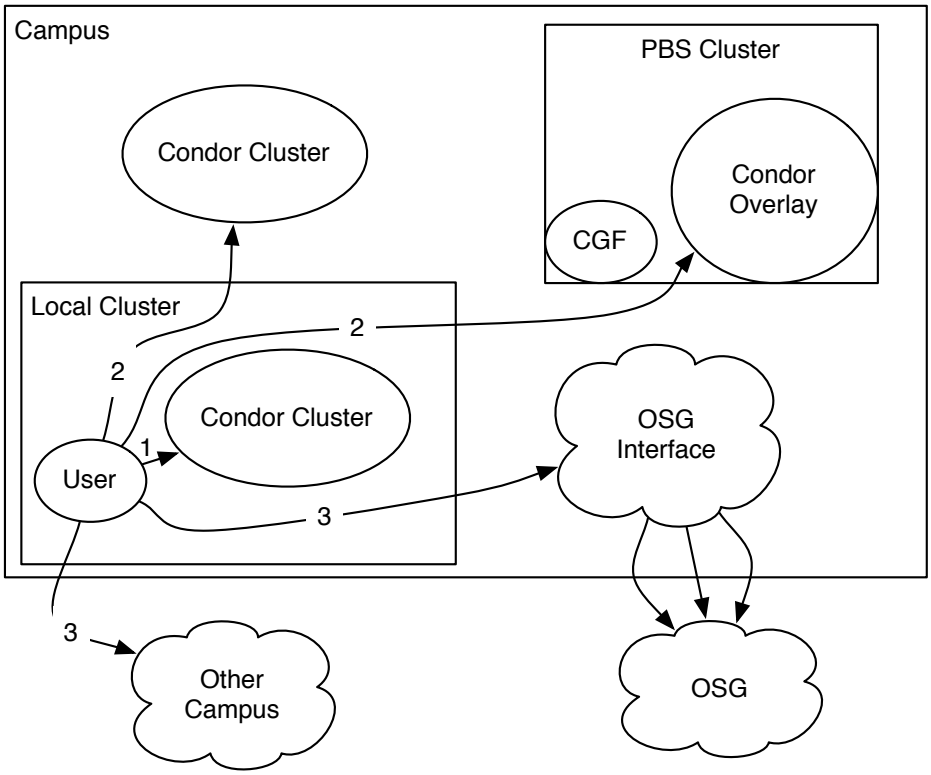

Figure 3.2: The HCC Campus Grid architecture, complete with connections to other campuses and the OSG. As each resource becomes fully utilized, the user's jobs will extend out to increasingly remote clusters.

In this architecture, every effort is given to find resources for the user (local, across campus, or externally), while maintaining the same vanilla Condor interface.

3.6 Bridging Campus Grids

There are two methods for expanding the campus grid described in Section 4.1: through GlideinWMS to the OSG, or by linking campus grids through flocking. The benefits of bridging externally are obvious - increased throughput for the local user's jobs. HCC has been able to utilize over 7,000 remote job slots at a time, and has been able to bridge to all the campus grids discussed in Section 2.2. We connect to

FermiGrid and GLOW through the OSG and to Purdue via Condor flocking.

Unlike the OSG, where the trust relationship is well-defined and common between 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. 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.

It is important to note the ever-widening circle of resources expands from the locality of the user. It goes from the resource the user knows best (and has the best support for) to the most foreign one. This is a very natural progression, and 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". Another usability boost is that all end-user interfaces are Condor vanilla universe. The user never encounters errors translated between systems (a common user frustration) and the user needs to develop expertise in Condor alone.

Chapter 4

Evaluation

4.1 Holland Computing Center campus grid

In order to test the framework described in this paper, we created a campus grid at Nebraska. A diagram describing the HCC campus grid is shown in Figure 4.1. In this diagram the user submits jobs on a central machine. First, the jobs will attempt to run on local resources at Prairiefire and Firefly. Next, it will branch out to the GlideinWMS interface and Purdue.

Like Purdue and GLOW, we have based the campus grid upon Condor. Each resource has a Condor-based interface, giving an identical experience regardless of what the user considers his or her "local" cluster. While two of the local clusters run Condor as the primary batch system, one is based upon PBS. PBS was chosen because of its superior scheduling of large-scale MPI jobs required for that resource. GLOW's Condor-only approach did not fit our case, and Purdue's model of running multiple schedulers was rejected because we wanted a less-invasive approach and because we wanted more efficient scheduling. So, for integrating our PBS cluster, we developed the Campus Grid Factory (CGF) to provide a Condor interface for the non-Condor

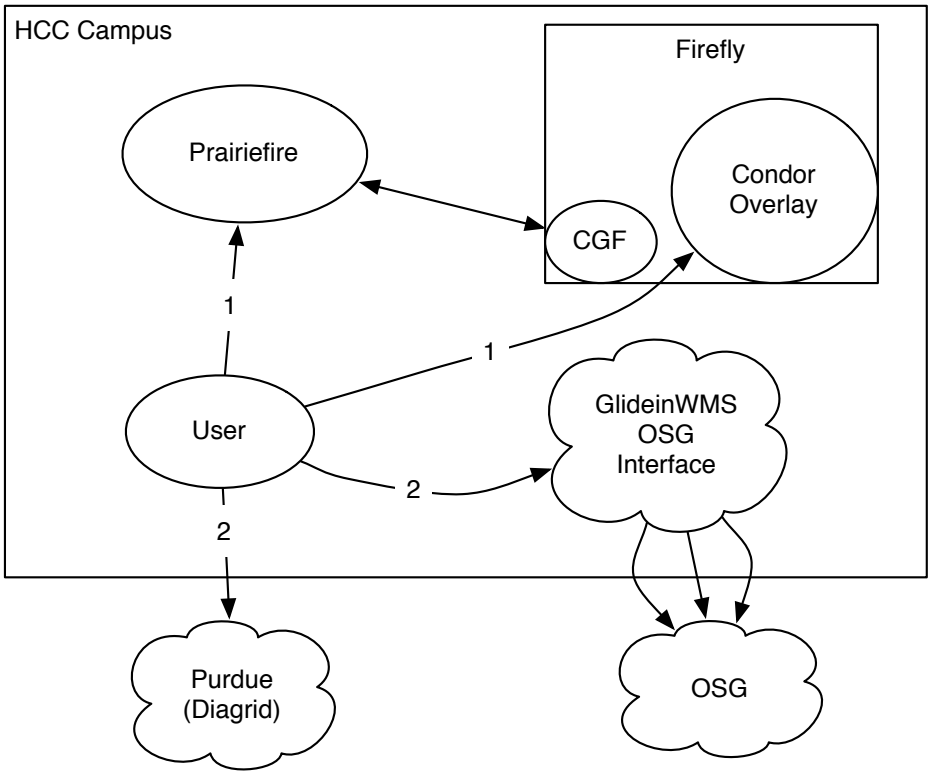

Figure 4.1: The HCC Campus Grid

clusters; this is covered in Section 3.1.

To enable decentralized operation, we utilized Condor flocking [10] between clusters. Condor flocking enables a transparent execution environment by imitating the interaction between submit and execute hosts when talking to remote resources. The jobs will continue to be managed by the original user's submit host, but the execute hosts can be outside what is managed by the local Condor pool. Furthermore, flocking can handle communication errors between remote hosts and can recover, providing disconnected operation when resources are unreachable.

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 [33, 34] 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. On the Condor-managed cluster, campus grid jobs run as user `nobody` while locally-submitted jobs run as the submitting user.

While all resources are run by the same team, 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 it has a clean separation between remote clusters and the central database (updates are done via HTTP), ability to integrate new resource types easily, and its ability to integrate into the larger OSG accounting. 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 filesystem across all clusters, so 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.

4.1.1 Prairiefire Cluster Setup

Prairiefire is setup as running both Condor and PBS. This is very similar to how Purdue uses Condor. When a PBS 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. The head node has both a public and private interface. Each worker node connects to the outside world through a NAT. 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 is setup to allow jobs from Firefly and the GlideWMS submit machine to run. Prairiefire's head node can also flock jobs to Firefly. To do this, the configuration variable `FLOCK_FROM` was set to `ff-grid.unl.edu, glidein.unl.edu` and `FLOCK_FROM` to `ff-grid.unl.edu`. Further, the security is setup such that jobs coming from outside of the Prairiefire pool will use the user `nobody` when running the job. Condor uses this underprivileged 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 certain hosts.

4.1.2 Firefly Cluster Setup

Firefly is running the Campus Grid Factory (CGF). It is running on the grid gatekeeper since it needs access to both the public ip and PBS submission. Additionally, unlike the login node, the grid gatekeeper does not have a firewall.

The CGF runs as a local unprivileged user on the gatekeeper. Condor is installed

in the user's home directory and runs as the user as well. The CGF runs as a condor job, therefore it is maintained and monitored by the Condor daemons. Submissions to PBS are to the default user queue.

The Condor instance that runs on the gatekeeper is configured to allow flocked jobs from Prairiefire and the GlideinWMS submission node. Also, users can submit jobs to the Condor instance, allowing them to run on Firefly or by being flocked to Prairiefire.

4.1.3 GlideinWMS OSG Interface Setup

The GldieinWMS interface is running the GlideinWMS Frontend software, as well as a Condor installation. The Frontend periodically queries the queues of multiple machines to detect idle jobs. When idle jobs are present, the Frontend sends a request for glideins to be submitted to the Factory. The HCC Frontend requests glideins from the central OSG factory run at UCSD.

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, starts them, and contacts the HCC Frontend to request jobs.

4.1.4 Flocking to Purdue Setup

Flocking to Purdue is enabled by publishing a list of collectors and schedds at Purdue and Nebraska. This list is then 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.

In total, Purdue has 7 collectors and 6 schedds participating in the flocking. Nebraska has 2 collectors and 3 Schedds.

4.1.5 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 4.2. 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.

```
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
```

Figure 4.2: Test test submission script

4.2 Characteristics of Campus Grid

Here I will evaluate the HCC campus grid on the characteristics defined in Section 1.1.

4.2.1 Trust relationships

On most campuses, trust relationships are very strong. On some campuses, a single group maintains the campus clusters. On others, proximity of administrators have 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 un-privileged account.

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

In the Virginia Campus Grid the user uses an authentication method consistent with on-campus policies to create a token that is inconsistent with on-campus policies. The user first authenticates with local LDAP servers, then creates a PKI certificate to interact with the on campus clusters. The series of interactions complicates the authentication with the servers, and necessitated 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. I believe this to be a mistake. 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 multiple submission hosts, and they are relatively easy to setup, there are many submission hosts. Also, gateways are traditionally mantained by professional or knowledgable administrators, where submission hosts are usually maintained by scientists with limited knowledge of security. Therefore, submission hosts could be compromised without knowledge of their owners, and could give access to unlimited certificates.

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 IP based security protects gateway hosts by only allowing very specific submission hosts. Further, the submission host is protected since 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. In the case of HCC, jobs flow out of the grid through flocking to external campuses and to the OSG through the GlideinWMS interface.

When flocking to external campuses, the HCC grid again uses IP based security as 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 admins to monitor the usage of their users, and to contain and contact us about possible security threats.

When jobs move the OSG, we must match the authentication methods used on the grid, 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 trust web 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, or an PKI certificate.

4.2.2 Job Submission

Users have the most interaction with the job submission mechanism. Therefore, it must be simple and intuitive. Most campuses further refine their submissions to distribute work across multiple resources.

The Virginia campus grid and the Oxford grid use globus to submit and receive jobs. Oxford further created a resource broker to balance 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 an empty slot, it will fill it without thinking of future submission or other resources. Therefore, 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. Though, Condor will fair share across all users of the pool.

Submission on the HCC, GLOW and Purdue grid requires only 2 more lines to the regular Condor submit file. Additionally, all negotiation and load balancing are handled by Condor internally, therefore there is less dependency on outside sources.

4.2.3 Resource Independence

In the OSG, resource independence is guaranteed by strict separation of resources. This is accomplished by independent clusters having all necessary infrastructure installed locally, while only sending non-blocking information to a distributed set of central services. This is very similar to the design of the HCC campus grid.

While resources on the OSG are independent, user job submission methods are not. When using GlideinWMS, if the Factory is disconnected or is terminated, then you cannot run on even local resources. When using Panda, all jobs require access to the central Panda server located at CERN.

The HCC Campus grid installs all infrastructure to distribute jobs on the local resource. For a Condor cluster, this is simply the existing Condor instance. While on the PBS cluster, the CGF install is installed locally on the gatekeeper node. Therefore, possible failures are:

- **Submitter failure**: The user's submission machine is taken offline only jobs submitted on this resource will be effected. Any jobs that where currently running will be recoverable for a short time while the job lease is active.

- **Condor cluster failure**: If the cluster becomes disconnected from the network, obs running on the cluster with remote owners will terminate after their 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 live for the lease time, then terminate.

- **CGF installed cluster failure**: Remotely submitted jobs will terminate after their lease has expired. Locally submitted jobs will continue to run on local

resources, including glideins previously running under PBS. If the CGF is terminated, no more glideins will be submitted to PBS, but jobs will continue to execute on existing glideins. Additionally, 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 glideins will shut down after receiving no new jobs.

The HCC campus grid is resistant to failure due design decisions regarding the placement of the Campus Factory, especially when compared to the GlideinWMS Factory.

4.2.4 Accounting

Accounting has two perspectives: How many resources have I used? Who and has used my resources?

The first perspective is easier than the second. 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, it is simple to install accounting software on each. Therefore, we used the accounting software from the OSG, Gratia, to do accounting on the submitter machines. Gratia uploads records for each job to a HCC Gratia collector. The collector can then make usage graphs such as Figure 4.3. The Gratia collector is a MySQL database.

Accounting on the resource's 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 job running at the resource.

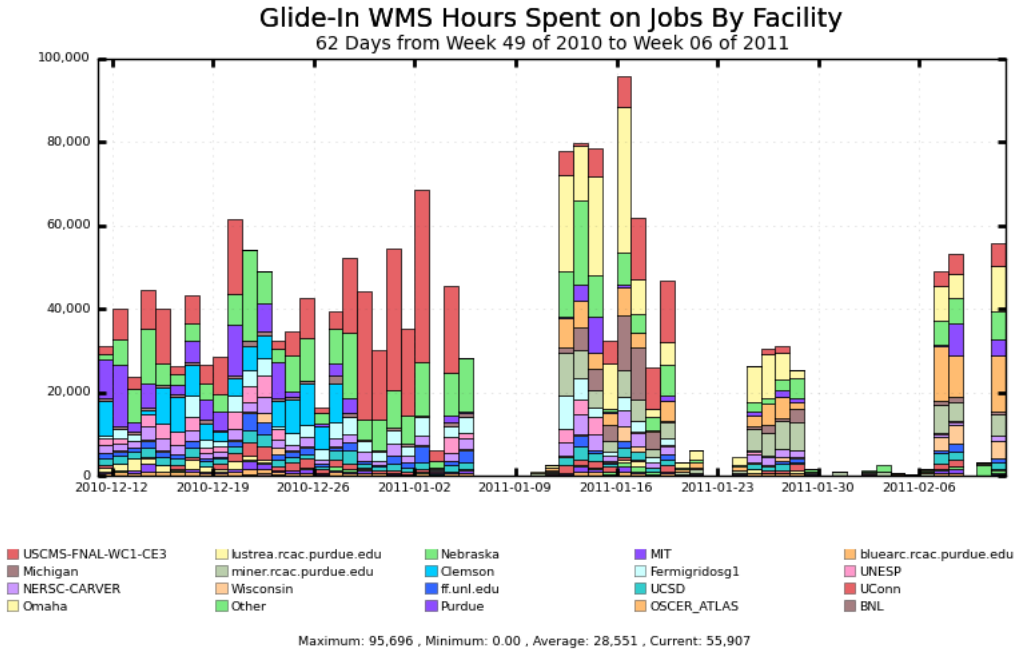

Figure 4.3: Snapshot of accounting of the HCC Campus Grid

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

4.2.5 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 filesystems that worker nodes can access. FermiGrid has a global NFS space for data. Oxford grid uses the SRB (storage resource broker) and a single vault.

The OSG promotes staging data to a nearby storage element which is difficult for individual scientists to do. Normally a scientist 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 filesystem, and is therefore forced to transfer files differently. In practice, each job transfers input files and executables directly from the submission to the execute host. This method has several benefits over the others.

- Removes the gateway as a bottleneck by transferring files directly from submitter to execute host.
- Reduces dependence on the gateway as a failure point. The gateway could terminate while a job is running and Condor would still be able to transfer back the output, and even start another job. The only major failure point is the submitter, which is an acceptable risk.
- Greatly reduces the work that the gateway needs to perform. The authentication and authorization is all done by the execute and submission host.
- Not dependent on the reliability or bandwidth of the shared filesystem. Further, the submitter is relatively unaffected by other users that are not running on the same submission host.

4.3 Usage

In Figure 4.4 you can see a snapshot of production jobs running on the HCC campus grid. The first thing to notice is the total number of jobs running, (8612). This is larger than the total number of cores in any single Nebraska cluster, and even larger

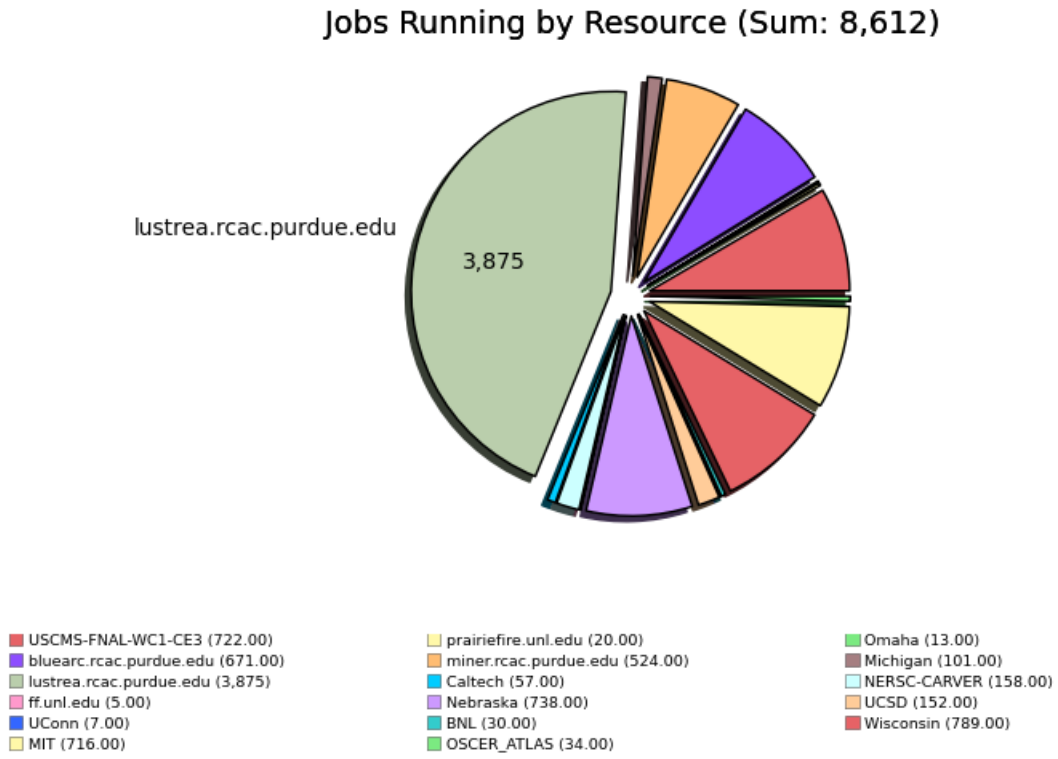

Figure 4.4: Snapshot of usage of the HCC Campus Grid

than the sum total of all cores managed by HCC (8000). This particular picture, Derrick Stolee is running a graph search workflow.

- **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 interesting observation is that number of jobs running at the peered campus Purdue. Purdue has large resources and peering with them has been very valuable to UNL researchers. As described in Figure 3.2, 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 lustrea.rcac.purdue.edu). Also, many sites in the OSG where running jobs (see Fermilab, Wisconsin, MIT).

Another usage metric is installation of the Campus Grid Factory. Currently, the campus grid factory is used in production only on Firefly at Nebraska. But, it is currently being run in test environments at the National Energy Research Scientific Computing Center, National Center for Supercomputing Applications, Louisiana Tech, Florida State as part of the Sunshine Grid.

Chapter 5

Conclusions and Future Work

The framework described in this paper creates a grid of clusters that transparently overflow to each other. Also, the grid can overflow to national cyberinfrastructure and peered campuses through interfaces that have been developed by external entities. The CGF was developed to connect a PBS cluster into the transparent execution grid.

The framework includes many components developed by external organizations such as BLAHp by gLite, Condor by University of Wisconsin – Madison, and GlideinWMS by the CMS collaboration. These components are glued together by Condor and the CGF to create a uniform grid.

The HCC campus grid is a production grid, running many users' jobs.

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. 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 [8] for data distribution.

Accounting also will be improved in the future. It is important for resource owners

to determine the hours given to external entities. This can only be accomplished with the execution side reporting usage. This model is not used in the OSG, and will need to be developed.

The campus grid ideas outlined in this paper 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 in their campus grids. The OSG Campus Grid Initiative is a priority goal inside the OSG that will include the software and ideas developed here.

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] 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.

[4] 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.

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

[6] K. Chadwick, E. Berman, P. Canal, T. Hesselroth, G. Garzoglio, T. Levshina, V. Sergeev, I. Sfiligoi, N. Sharma, S. Timm, et al. FermiGridexperience and

future plans. In *Journal of Physics: Conference Series*, volume 119, page 052010. IOP Publishing, 2008.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

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

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

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

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

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

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

[17] 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.

[18] 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.

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

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

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

[22] 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.

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

[24] 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.

[25] T. Maeno. PanDA: distributed production and distributed analysis system for ATLAS. In *Journal of Physics: Conference Series*, volume 119, page 062036. IOP Publishing, 2008.

[26] 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.

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

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

[29] 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.

[30] 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.

[31] 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.

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

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

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.