From Proposal to Production: Lessons Learned Developing the Computational Chemistry Grid Cyberinfrastructure

Rion Dooley
Center for Computation & Technology at LSU

Kent Milfeld
Texas Advanced Computing Center, UTA

Chona Guiang
Texas Advanced Computing Center, UTA

Sudhakar Pamidighantam
National Center for Supercomputing Applications, UIUC

Gabrielle Allen
Center for Computation & Technology, Department of Computer Science, LSU

September 1, 2005

Abstract. The Computational Chemistry Grid (CCG) is a 3-year, National Middleware Initiative (NMI) program to develop *cyberinfrastructure* for the chemistry community. CCG is led by the University of Kentucky, and involves collaborating sites at Louisiana State University, Ohio Supercomputing Center, Texas Advanced Computing Center, and the National Center for Supercomputing Applications. This paper discusses experiences developing the CCG cyberinfrastructure in the first year of the project. Special attention is paid to technological issues faced as well as issues raised running the CCG in production. The final section of the paper looks forward to challenges foreseen in the remaining two years.

Keywords: Grid, Chemistry, GridChem, CCG, Cyberinfrastructure, Gaussian, MolPro, NWchem, GAMESS

1. Introduction

The term *cyberinfrastructure*, coined by an "NSF Blue Ribbon Panel", refers to software and hardware which enable scientists to exploit cutting edge technology resources, including compute and data servers, visualization devices, instruments and networks, for advancing research in science and engineering. The need for cyberinfrastructure in the basic sciences is evident in the growing number of similar active projects today. The Asian Pacific BioGrid (APBioGrid, 2004), the EGEE Project funded by the EU (European Commission, 2005), the Singapore National Grid Life Science Virtual Community (LSVC, 2005), and Korea's national grid infrastructure initiative to support, in part, computational chemistry (KISTI, 2005) all exist today to provide scientists with tools and resources at a level never before seen.

The Computational Chemistry Grid (CCG) (GridChem, 2005) is a three year NSF funded project to develop cyberinfrastructure to serve scientists engaged in studying molecular structure and function. Computational chemistry algorithms and software are now widely used across the life sciences and other disciplines. Examples of their application are found in nanotechnology, biotechnology, medicine, pharmacology, biology, physics, materials science, structural mechanics, electrical engineering, chemical engineering and environmental quality modeling.

Users of both commercial and academic chemistry software packages, such as Gaussian, GAMESS, MolPro, NWChem, and Amber, are major users of supercomputer resources across the US and worldwide. The CCG leverages existing, established Grid middleware to provide an easy-to-use integrated computing environment for these and other chemistry applications for use on supercomputer resources across CCG member sites.

The CCG is led by the University of Kentucky (UKy), and involves collaborating sites at Louisiana State University (LSU), Ohio Supercomputing Center (OSC), Texas Advanced Computing Center (TACC), and the National Center for Supercomputing Applications (NCSA). This paper discusses initial experiences developing and running the CCG cyberinfrastructure through the first year of the project and looks ahead to challenges in the remaining two years. The discussion touches on important technological issues faced as well as the chosen solutions.

The format of this paper is as follows. Section 2 gives an overview of the current and future CCG architecture and the GridChem client application. Section 3 discusses the challenges of implementing the technological infrastructure and the roadmaps developed at the start of the project. Finally, Section 4 closes with concluding remarks.

2. Overview

The current design of the CCG, as shown in Figure 1, is a 3-tier architecture comprised of a client side graphical user interface (GUI) application, a middleware service, and a resource layer. The client application, called GridChem, is an open source Java application that remotely launches and monitors computational chemistry calculations on CCG supercomputers at remote sites. GridChem also provides several useful, tightly-integrated features such as application-specific molecular editors, output file parsing, and interfaces for pluggable visualization tools. GridChem is distributed as a self-installing Java Web Start application available from the project website (http://www.gridchem.org).

More information on the GridChem client can be found at (GridChem, 2005).

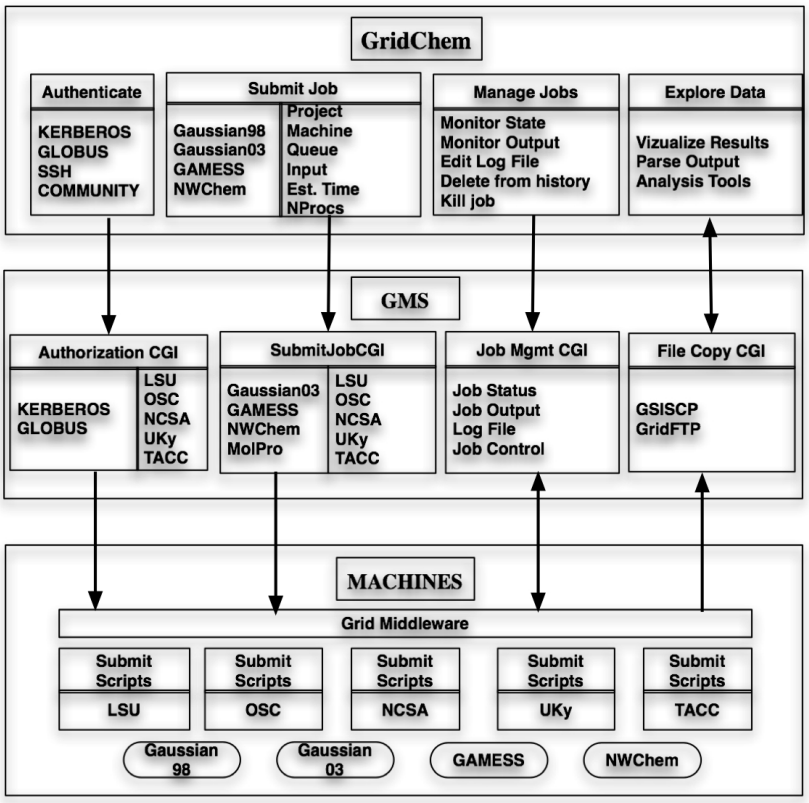

Figure 1. The current architecture for the Computational Chemistry Grid. All GridChem functionality is tied to one or more corresponding CGI scripts on the CCG middleware server. The CGI scripts in turn communicate with a database to persist information on users, jobs, files, and resources.

The remainder of this section examines the two remaining layers of our architecture: middleware and resource. It first looks at each layer as they it exists in the current CCG architecture shown in Figure 1, then concludes by discussing their composition within the framework of our planned architecture shown in Figure 2.

2.1. Current Architecture

Although the long-term goal of this project is to create a dynamic grid service architecture, one of the primary deliverables in the first year was to provide a production environment for users to submit, monitor, and retrieve output from jobs. Such an immediate user base provides feed-

back and experience necessary to produce a more useful and responsive grid for the community in the latter stages of the project.

To meet the short term goal of usability and to facilitate a long-term goal of implementing a robust grid architecture, it was decided to first implement server-side functionality (ie. the middle layer of the architecture in Figure 1) in CGI scripts. Thus, the current middleware layer consists of basic grid middleware (such as the Globus Toolkit, NMI Distribution, etc.) and the CGI scripts providing core functionality to the client application. The CGI scripts are responsible for enforcing such tasks as security, job submission, file tracking, job monitoring, and information provisioning. It is the CGI scripts which convert job requests made through GridChem into valid job descriptions for the system scripts in the resource layer. The CGI scripts are also responsible for all accounting in the CCG. Usage information, resource status, historical job information, etc. must all be recorded and updated in a database. Section 3.2 discusses this topic in greater detail.

The lowest level of the CCG Architecture is the resource layer which appears at the bottom of Figure 1. The resource layer consists of the physical resources, local schedulers, resource-specific low level information providers, and the software and middleware needed to run the computational chemistry applications on each machine. The CCG currently provides support for Gaussian, GAMESS, and NWchem. Not every application is provided on every CCG machine, however, community users have access to every application through GridChem. In the future, the list of supported applications will expand to include MolPro, Aces, and many other applications currently used by the computational chemistry community.

Included in the resource layer is a set of system scripts that take the CGI job descriptions and generate input files for the local queueing systems. These scripts are system-specific, meaning that they are each optimized to run on their respective system. The benefit of this approach is that, from the middleware perspective, little work is needed to submit a job on a particular machine. A single, common interface is exposed for each application, and no separate decision making is needed to successfully run an application at LSU rather than NCSA.

2.2. Future Architecture

In the longer term, CCG decided to pursue a different CCG architecture. Problems of scalability, distributed resource management, and the fluctuating nature of the Quality of Service (QoS) provided by each resource are inherent in any grid implementation. Thus, mechanisms to handle such characteristics are necessary to provide a functional,

long–lasting grid environment. In the CCG, a Service-oriented Architecture (SoA) was embraced to provide the mechanisms necessary for such a task. The SoA paradigm is increasingly being adopted as the basis for middleware design. This is evidenced by industry's adoption of web services and the movement of grid researchers and standards organizations such as the Global Grid Forum to grid services (IBM, 2005) (Web Services Interoperability Organization, 2004) (WS-RF, 2005). In a SoA, services may be composed hierarchically, allowing adopters to focus on developing the necessary meta-services needed for application specific grid implementations, such as those in Section 3, rather than primary grid services already being developed by others in the community. Using this approach, focus remains on the integration aspects of the project and the CCG benefits knowing that, as the quality of the underlying services improves, so too will the quality of the middleware meta-services.

A SoA architecture was chosen to implement the future CCG middleware layer rather than servlets or the existing CGI for several reasons. Using web services allows a high degree of portability and accessibility through well defined mechanisms such as RPC and SOAP. Web service interfaces are published through a common registry, thus making the service easily accessible using multiple technologies such as portals, client applications, and web pages. Web services also allow integration with existing grid technologies, such as the GSOAP plugin for secure communication, rather than relying on command line utilities.

A SoA architecture was also chosen due to the complex nature of the desired CCG. As can be seen in Section 3, the middleware must provide several features that would be difficult to achieve without heavy integration at the highest level. A web services implementation allows integration of the accounting, job submission, GSI security, and monitoring components in a way that is not possible in any of the individual underlying services. Web services allow cleaner implementation through a common programming environment (rather than a mix of perl, CGI, and system scripts), a well-defined API, and the added benefit of inheriting useful functionality from a stable container environment (Globus, Apache Tomcat, etc.). Further, this approach allows us to take advantage of several desirable features of the implementation language of choice such as client and server notification and dynamic service discovery. Finally, a SoA architecture was chosen to allow the CCG to grow in the coming years. The current approach of tailoring site-specific scripts and software stacks to meet the middleware needs is difficult to maintain and does not scale well. With a SoA, decisions can be made based on information from a common repository, and that repository can add and remove resources at will. This gives greater flexibility and

stability to the CCG and allows other sites to join and leave the CCG much easier than with the current approach.

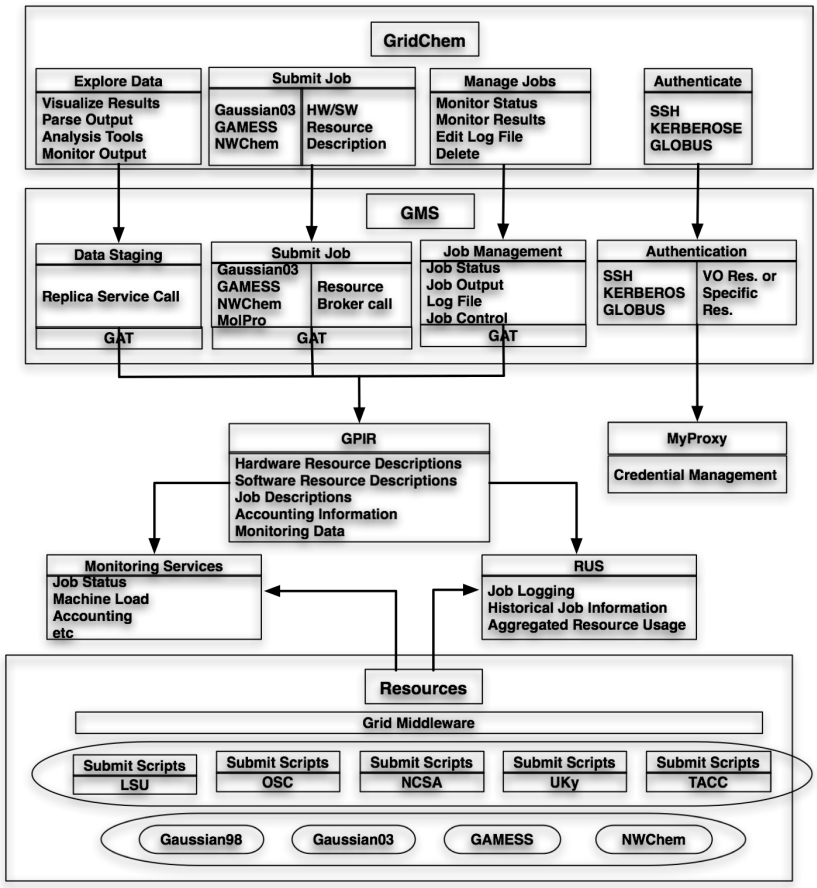

Figure 2. The planned architecture for the Computational Chemistry Grid. CCG is in the process of implementing a Service-oriented Architecture where the client utilizes the GridChem Middleware Service (GMS) for core functionality and the GMS in turn relies upon a series of grid and web services to provide functionality to the client. The current CCG architecture of Figure 1 is very similar to the planned architecture. The move from Figure 1 to Figure 2 is underway and consists of a one-for-one replacement of existing CGI scripts with GMS web services.

The CCG SoA architecture is shown in Figure 2. Notice that this figure is very similar to the current CCG architecture in Figure 1. The major difference being that in the future architecture, CGI scripts are replaced one-for-one with corresponding GridChem Middleware Service (GMS) implementations. Figure 3 shows how a typical job submission use case will occur under the new architecture.

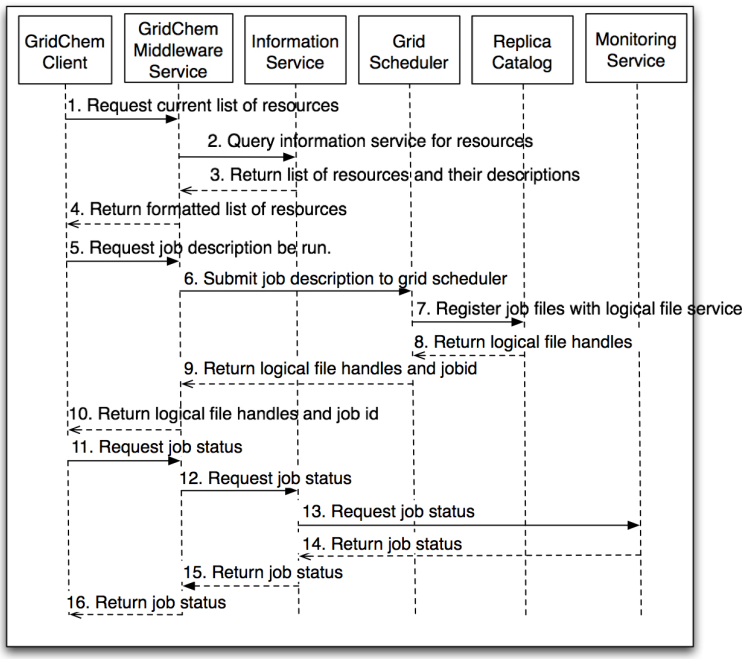

Figure 3. Proposed sequence diagram of GridChem interaction with the future GMS.

Notice that each action in the client is reflected by a call to the corresponding GridChem Middleware Service. The GMS in turn, will leverage one or more underlying third-party services to perform the requested action. In some cases, these third-party services will be in full production. In other cases, these services will be under development. In order to hide the details of interacting with these services, nearly all interaction with the underlying services is done through the Grid Application Toolkit (GAT) (Allen et al., 2002).

The GAT is a generic API for grid computing. There are currently four implementations of the GAT: C, C++, Python, and Java. Each implementation utilizes the adaptor design pattern to allow multiple implementations of common actions at run time. This means that, if one of the underlying services in the CCG, such as the information service, changes in the future, the GMS code does not have to change, the GAT will internally recognize this change and select the appropriate adaptor.

In relation to figure 2, information queries on available software, hardware, and job history will be forwarded by the GAT to the information service, GPIR (discussed in greater detail in section 3.3). Job submissions and queries on job status will be forwarded to the

grid scheduler service (discussed in greater detail in section 3.4). Data queries and file movement requests will be forwarded to the replica service, Globus RLS (discussed in section 3.5). Usage history requests will be forwarded to the resource usage service (RUS) (discussed in greater detail in section 3.2.1).

3. Technological Issues

Creating a production grid environment poses several significant technological problems related to security, accounting, information provisioning, resource brokering, and data management. In this section each issue is addressed in turn.

3.1. Security

One requirement of the CCG is to provide users with single sign-on access to all CCG resources. An ideal use case for the CCG is as follows. The user starts GridChem and opens the "Authenticate" panel. There, they enter their CCG username and password and click the "Login" button. GridChem then encrypts the username and password and sends them as arguments to the GMS Authentication Service. The service checks that the user's information is correct, then pulls a valid community credential from a MyProxy (MyProxy, 2005) server. Upon completion of this step, the user is authenticated and has full GridChem functionality via a community allocation at each site independently of the underlying security mechanisms and without need to authenticate again for the life of their session. The community allocation, however, is the ideal case. The average case is not this simple.

Full users of the CCG will have existing allocations on many, if not all participating systems. Other users will have allocations on at least one of the machines in the CCG. Users will all be accustomed to authenticating manually using Secure Shell (SSH), Kerberos, and Grid Security Infrastructure (GSI) mechanisms and performing their work via the command line using their own well-defined, and often home-grown methods. The philosophy taken is that potential users should not be forced to leave familiar methodologies behind if they wish to use the CCG. As well, users should not be required to install a large, complicated suite of grid middleware simply to use GridChem. The job of the CCG is to provide them with tools to enhance their experience while removing as many existing obstacles as possible.

Providing such tools is difficult, if not impossible, without detailed information about the systems on which CCG users perform their science. The CCG middleware relies on tailored information providers

as well as static information collected manually from CCG resources to make informed decisions and take appropriate action on the user's behalf. Thus, there is a tradeoff between community acceptance and a full grid architecture. The solution adopted in the CCG is to support multiple authentication methods through the GridChem client. GridChem supports SSH, Kerberos, and GSI security mechanisms as well as the notion of a "community user". The SSH and Kerberos interfaces provide more general interfaces to submit and manage jobs at the expense of sacrificed job monitoring ability. The MyProxy interface provides the complete set of GridChem features, but requires the user to keep track of their grid credentials. The CCG community user is the realization of the use case above. After a single authentication, a user's jobs are submitted and tracked, their data managed, and the resulting output can be parsed for meaningful data. Just as security and ease of use progressively increased with the introduction of SSH, Kerberos, and GSI, so too is this pattern reflected in GridChem. As users move from SSH and Kerberos to GSI authentication, functionality increases and users reap the benefit of better job management.

This is not by design, but is rather a result of the additional infrastructure needed to perform identical functionality with the other mechanisms. Because of the time and effort needed to put such infrastructure in place, and the desire to move users to the grid, the CCG does not fully supporting every authentication method. This, then, becomes a leveraging point to shift users from their existing mechanisms to use of MyProxy and community authentications.

3.2. Accounting

Historical job history and individual user tracking are both areas that must be addressed to provide adequate accounting for the CCG. The remainder of this section, looks at each of these areas in turn by first explaining why support of a specific feature is needed, then highlighting the challenges faced implementing that functionality needed for such a feature within the confines of the CCG architecture, and finally discussing the short and long term solutions to provide such functionality.

3.2.1. *Historical Job Information*

The first step in accurate accounting is collecting and aggregating historical job information. This requirement is driven from three sources: the NSF, local site administrators, and the CCG middleware architecture. The NSF mandates usage statistics be included as a part of the project reporting requirements. Individual site administrators require

us to show a degree of supervision over CCG users in exchange for the privilege of receiving a community allocation. To satisfy the NSF, the identity of CCG users and the resources they use must diligently be recorded . This information must also be validated against the local scheduler records on each resource. To satisfy local site administrators, every job run under the community allocation must be associated with a physical CCG user. This is primarily so that, in the event of an emergency, the appropriate people can be notified and the problem resolved. Without such mechanisms, user support is virtually impossible.

The accounting needs of the GMS must be met as well. The GMS needs historical information to implement key features such as quotas, meta-scheduling, and job status notification. Quotas in the CCG extend beyond simple sanity checks to verify that an allocation on a particular site has not expired. They ensure that a few users do not monopolize the entire community allocation. While the CCG advertises free time for community users on the CCG, users are not allowed free reign. Meta-scheduling, as discussed in Section 3.4 needs historical information to increase the accuracy of it's job predictions and quality of service estimates based on lessons learned from the past. Job status notification requires information about the start and stop time of jobs.

It is important to note that a long running job monitoring service can provide the information needed for job status notification. From a production standpoint, however, this approach is unfavorable. If the monitoring service fails at any point, all information on jobs started or stopped during the service's down time is lost. It is reasonable to foresee a use case where a user submits multiple long-running jobs and checks back days, or even weeks later to find their status. Without dependable, consistent, historical information, users cannot be given answers to simple questions such as, "Did my job complete?" "Was my job successful?" "How long did my job take to run?" "When did it complete?" and, "How long did my job wait in the queue before starting?"

Collecting historical information is difficult in a heterogeneous setting due to the diversity of schedulers across the CCG. As of the writing of this article, LSF, LoadLeveler, PBSPro, and OpenPBS can all be found on the CCG. To further complicate the task, each site has its own policies over what information should be available to the user. At NCSA, users are provided full access to their job history using a command line tool, called `qhist`. At OSC, users are not given historical job information other than an email informing the user of job commencement and termination. The challenge, then, is to find a unified way to access the historical information present on each site in a way

that does not violate each site's local policy, and persist it in a common format.

There has been significant work done in the GGF addressing this topic (Ainsworth et al., 2005). The Resource Usage Service Working Group (RUS-WG) looks to be one promising solution. The RUS is a grid service that aggregates usage records pushed to it from accounting information providers on each resource. The providers exist as a layer between the Globus job manager and batch scheduler on each resource. Their main function is to monitor all incoming job requests and update the local accounting records. The information gained from these two actions produces a usage record that is spooled and periodically pushed to the RUS. Access to these usage records is available through calls to a WS-Secure (Ainsworth et al., 2005) grid service.

The current architecture does not provide complete historical information as described above. Rather, the middleware server logs each job successfully submitted to a resource, then relies on email notification from individual resources to record the completion time of each job. While effective, this method lacks several of the features described above, and, as described in Section 3.2.2, is inadequate for user tracking. The main reason for pursing the current implementation was due to the lack of any standards-based grid resource usage tools. Now that such tools are available, it is foreseeable that the CCG will move towards a grid service, like RUS, in the near future.

3.2.2. *User Tracking*

In addition to information about what jobs a user submits, the accounting infrastructure must perform the equally important task of tracking user activity across multiple domains. This requirement is tied to the above discussion on usage history. If a user's identity cannot be tracked, their history cannot be found. GridChem is tagged with the responsibility of providing the user with specific information on their jobs, their accounts, and their data without divulging this information to others. This approach was chosen for practical as well as political reasons. For users submitting a handful of jobs every month, it is prohibitively tedious to ask them to sift through thousands of history records just to find the few jobs they ran. In addition, some users may be performing sensitive experiments which necessitate security mechanisms. For these reasons, as well as to avoid the potential namespace collisions encountered when running across multiple administrative domains, the CCG must accurately track its users.

Section 3.1 discussed how users can access the CCG using multiple authentication techniques. Thus, a user may have multiple usernames, passwords, and grid certificates. In order to manage user identities

across numerous resources, it is essential to know all possible user aliases and associate them with a common user identity within the CCG infrastructure. This is done using several mechanisms, the first of which takes place when a user requests an allocation to the CCG.

When a potential user wishes to join the CCG, they must first fill out an allocation request form stating their desire to be either a community or external user. External users are users wishing to utilize authentication methods other than the community account (i.e. MyProxy, Kerberos, or Secure Shell). The external user form requires the user to specify information necessary to track their activity such as machines on which they have existing accounts, usernames on those machines, and project memberships. The form also solicits a unique CCG handle for the user used in the next step for internal bookkeeping.

Section 2 described how GridChem interacts with the middleware server for the bulk of its functionality. As a result, when a user starts the client, they must authenticate with the middleware using their unique CCG username. In doing so, the context and permissions the user is employing to perform their work is always known. The process is as follows. The user starts the GridChem client and authenticates with the middleware server. The middleware server checks that the user has a valid username and password and returns the allocation classification of the user - community, external, or both. If the user has both an external and community allocation, they are prompted to specify which method to use. Community users are successfully authenticated at this point. GridChem then takes care of all the credential management and setup needed for a community user. External users will be taken to a second login screen which allows them to provide the username and password or grid credential needed to authenticate using their preferred method. Once successfully authenticated, GridChem and GMS know the user's CCG username, the authentication method they are using, and the remote username under which they are operating. Using this information, GMS can fully act on behalf of the user, tracking their usage on each system.

3.3. Information Provisioning

Accurate and dependable information provisioning is the largest single challenge of this project. Without reliable information from all aspects of the system, necessary and intelligent decisions on the user's behalf cannot be made. This information comes from several sources: historical job records, monitoring output, static and dynamic resource descriptions, and file metadata. A complete discussion on how information is aggregated to provide functionality to GridChem and fulfill user

requirements is a lengthy topic, and beyond the scope of this article. Instead, the remainder of this section focuses on describing the type of information provided by each source and how it will be provided through the CCG architecture.

3.3.1. *Historical Job Records*
Section 3.2, discussed the need for historical job information. Current plans involve the use of a third party grid service such as RUS (Ainsworth et al., 2005) to provide this information.

3.3.2. *Static and Dynamic Resource Descriptions*
Data can be placed in one of two categories: static or dynamic. Static data is data that changes very rarely, or not at all. A machine's name is an example of static data. Dynamic data is data that requires frequent updates. The load on a machine, the available bandwidth on a network, the number of available licenses for a piece of software, are all examples of dynamic data. Static information is relatively easy to acquire. It can be read from a text file or hard coded into an application. Dynamic information acquisition requires more creativity.

Dynamic information involves monitoring. In the context of this paper, monitoring is defined as consistently checking and recording the status of a particular property of interest. That property may be derived from reading a file, querying a web service, or explicitly measuring a quantity of interest. Whatever the actual mechanism used to collect the data, monitoring requires its repeated application to ensure accurate, up to date information.

The act of monitoring is useless if the data produced does not become information. A grid information service (GIS) is a means of aggregating large amounts of data into meaningful information. Briefly, a GIS provides a public schema for representing data. It accepts information formatted for this schema from a set of providers (or monitoring applications) and stores it for future reference. In the CCG, two information services, iGrid (iGrid, 2005) and GPIR (GPIR, 2005), were considered for use suppling resource information to the CCG.

iGrid is a hierarchical information service shown to perform upwards of an order of magnitude faster than the Globus MDS (Aloisio et al., 2005). The iGrid hierarchy is comprised of multiple layers of iServe and iStore instances. Each node in the iGrid tree has both an iServe and an iStore. iServes pull information from local resource. iStores make that information available for direct query. After careful review, it was found that the basic installation of iGrid does not provide resource descriptions robust enough to meet all the needs of the CCG grid architecture. Specifically, support for administrative information, software resource

descriptions, and individual node information were lacking. This was enough for us to recommend against using iGrid for this particular project.

The GridPort Information Repository (GPIR), in contrast to iGrid, is an aggregated information service. GPIR is designed for performance and employs a portal-oriented view of data (GPIR, 2005). In addition to providing traditional job and resource information, GPIR supports both, "dynamic data and 'human-centric' data (such as where a resource is located or whom to call for support)." (GPIR, 2005). An example of such information is listed in Figure 4.

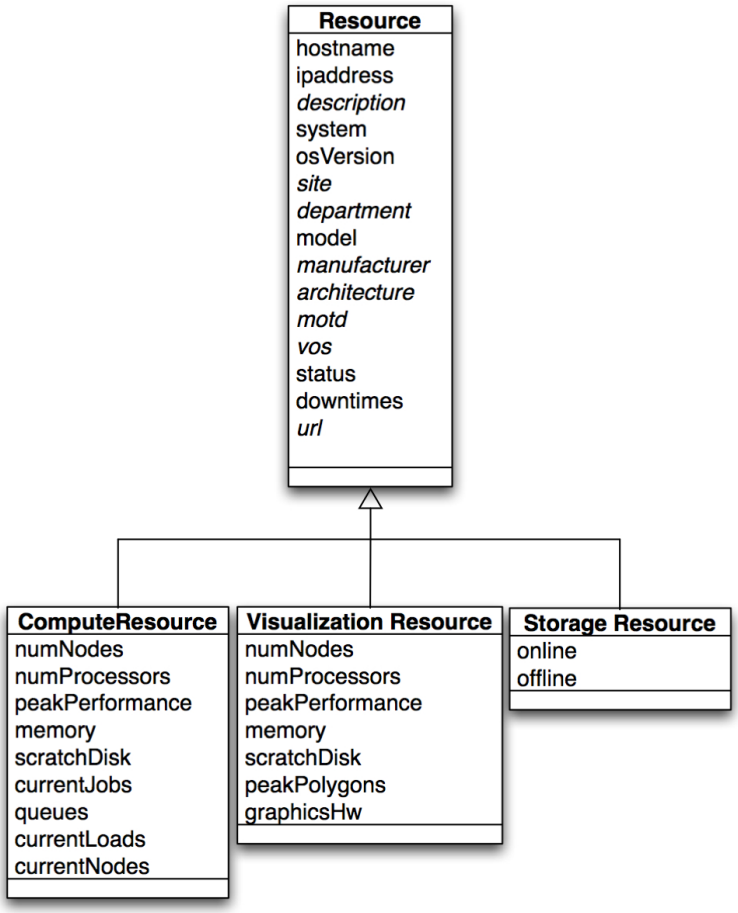

Figure 4. Description of the GPIR compute description. Notice the non-technical information (listed in italics) available in the GPIR information schema.

Rather than building up an information hierarchy, GPIR exists at the very highest level in the information chain, acting as a consumer of

several other information services. Thus, iGrid could potentially serve as an information provider to GPIR.

One advantage of designing GPIR with portal support in mind is that it includes non-technical, administrative information. This is one immediate and attractive argument for GPIR over iGrid. A second advantage is the fact that GPIR is in it's third full release and currently used by several projects including Open Grid Computing Environments (OGCE) (OGCE, 2005), TeraGrid (TeraGrid, 2005), Fleet Numerical Meteorology and Oceanography Center (FNMOC, 2004), the University of Tennessee Grid project (UTGrid, 2005), Southeastern Universities Research Association (SURA, 2005), and Texas Advanced Computing Center (TACC, 2005).

As with iGrid, initial examinations showed some drawbacks to using GPIR. As before, home-grown providers will have to be provided to fill in the missing pieces required from their information schema. To address the second problem, researchers in the CCG middleware group developed a new information provider, the Job And Machine Monitor Service (JAMMS) (Milfeld et al., 2005), discussed in the next section. It was also observed that data acquisition is done using unsecure remote database calls. While faster and more efficient than traditional web service calls, this may be an undesirable technique when transmitting user information.

3.3.3. *Monitoring Output*

Monitoring data will come from several sources. It is anticipated that the majority of these sources will be information providers distributed with the chosen information service. As stated in Section 3.3.2, there will be some discrepancy between the information needed and the information provided by these monitoring tools. To fill this gap, researchers in the CCG middleware group developed a new information provider, the Job And Machine Monitor Service (JAMMS). JAMMS is a Perl script that pulls information on queues, jobs, CPU's, machine utilization, and overall system status. This script is run as a cron job at each site. By default, JAMMS is set to run every 5 minutes as a local user (e.g. under the community account on each site). It uses the Perl Database Interface (DBI) module for sending information to a MySQL data base located on the CCG middleware server. A PHP program is used to extract information from the database and present it to the user, through their browser.

JAMMS programs, called filters, execute batch utilities (for either PBS, LSF, or LoadLeveler), and extract (filter out) needed information. Two or three batch utilities might be invoked from within the Perl script

to obtain the relevant information. For LSF, a single API program was developed to quickly extract all relevant information.

After several months of use, JAMMS has shown the potential to be an acceptable complement to existing information providers. Current plans are to augment JAMMS data with other system-level tools to ensure adequate total system information is collected in whatever GIS is employed. Work is currently underway to modify the JAMMS output format so it can serve as a provider to the iGrid information service. Plans are also in place to examining how JAMMS can be integrated into the GPIR framework. As of the writing of this paper, the final solution to the information needs of the CCG remains an open question.

3.4. Resource Brokering

It was stated in Section 2, that the problem addressed with the CCG is to provide cyberinfrastructure to enable the computational chemistry community to submit and manage jobs using a select set of well-known applications. Narrowing the focus from the general case of enabling complex workflows and scheduling for any given application, to the specific case of only supporting a few known software packages, simplifies the task of resource brokering. The applications the CCG user community will employ are known. The finite list of dedicated machines on which the user will run these applications are known. The means in which this introduction will happen are also known. Using such concrete information greatly reduces the complexity of a task and allows the SoA model to again be leveraged to perform resource brokering at two distinct levels.

At the lowest level, grid schedulers are used to submit and manipulate the user's job on every resource. Existing tools such as Condor (Thain et al., 2003), the Grid Resource Management System (GRMS) (GridLab, 2005), and GRAM (Globus, 2005)(Czajkowski et al., 1998) fit this description. In order to avoid dependence on any one scheduling service, the Grid Application Toolkit (GAT) (Allen et al., 2002) is employed. The GAT enables interchanging grid schedulers without altering the code base. It also allows the best features of each technology to be used to provide an overall service that is more sophisticated than it's individual parts. For example, GMS could use GRMS for resource selection based on predicted run time, and Condor for job submission.

A good example of leverage existing technology to solve low-level problems is proxy certificate management. Computational chemistry jobs can vary in length from a couple hours to many months. As job run time increases, so to does the possibility that a user's proxy will expire long before their job finishes. If the user's proxy expires, all grid-

based output file transfer will fail due to an expired credential. Condor-G is a grid scheduler that performs credential management on behalf of the user. Thus in the case of long running jobs, as an alternative to generating a credential with an extremely long life, Condor-G can be used as the underlying grid scheduler to renew the user's proxy credential on their behalf.

At the highest level, sophisticated services will be provided to the user such as throughput scheduling, economic scheduling, job monitoring, and notification. Intelligent scheduling of jobs, using different criteria for optimality, is one of the second year goals of the project and is crucial in ensuring efficient use of grid resources. By definition, throughput scheduling seeks to maximize job throughput by minimizing job turnaround time. Aside from requiring dynamic information that reflects current resource utilization, throughput scheduling necessitates reasonable values of three parameters that determine total job execution time, namely queue wait, data transfer time, and application run time. To obtain estimates of these parameters (within some specified error bounds) the CCG metascheduler will utilize a web services-based prediction toolkit that implements the instance-based learning (IBL) method pioneered by Smith (Smith, 2003).

Job scheduling, monitoring and notification services will be made available through the rich resource descriptions pushed to the information service described in Section 3.3. Detailed resource descriptions allow accounting to be integrated into the decision making process, which in turn, enables better decisions than could be made by a third-party broker.

With better information comes better accounting, with better accounting comes better brokering. As mentioned in Section 3.2, the CCG development team is working to provide more mature information providers and an advanced accounting system. As of this writing, these systems are not in place, thus the current CCG resource brokering capabilities are dependent on the strength of the underlying grid schedulers employed: currently Condor-G and GRAM.

3.5. Data Management

Data management is never trivial in a grid setting. Different directory structures, overlapping file and user namespaces, and heterogeneous site policies on how data should be stored, make the process of ensuring that the right data is placed in the right location, using the right mechanisms difficult. Within the context of this paper, data management is examined first from a user perspective and then from the perspective of the CCG architecture.

From the user's perspective data management should be taken for granted. If their job started, the user should have access to the associated data through GridChem. In order to achieve this level of fluidity, two things have been done. First, within the GridChem job creation editor, the user is allowed to specify a location to stage the output of their job. Second, the user's output data is internally mapped to the record of the job, so the user can simply select a job and use GridChem's grid file browser to retrieve that data. Figure 5 shows a screenshot of this tool.

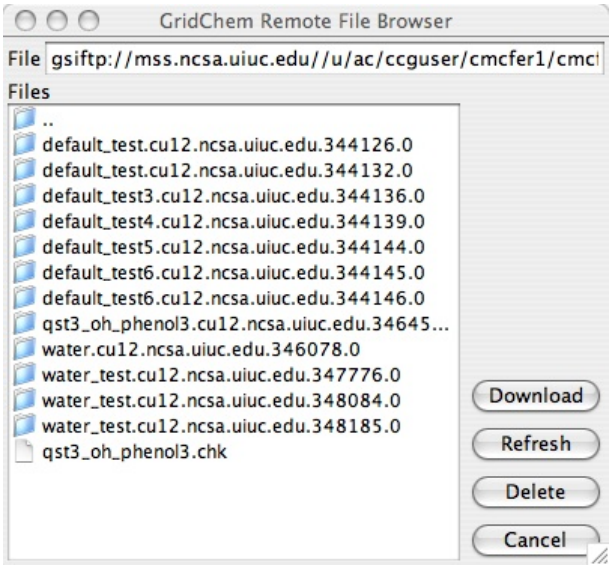

Figure 5. Screenshot of grid file browser included in the GridChem software.

The grid file browser uses the GAT to provide remote file access. Specifically, the Java GAT API is used to seamlessly access a user's remote files using whatever authorization mechanism the user employs. Security again, plays a large role in the design of the grid file browser. Users are pointed directly to the remote directory associated with their job and prohibited from accessing other areas of the remote resource. This helps enforce CCG's internal user tracking and prevents community users from treating other scientists results as community data.

From the perspective of the CCG architecture, data management is simplified by enforcing strict naming and storage policies within the middleware. When a user submits a job through GridChem, their request is parsed, validated, and forwarded on to the remote resource. On that resource, their job request is translated by application-specific job submission scripts into a file appropriate for input to the remote

batch scheduler. Part of the script logic deals with how to handle output data after the job executes. Currently, a directory structure is created for the user's job based on their username, job id, application type, and project name. Depending on the user's authentication mechanism, several options are available for storing data. Community user data, by default, is pushed into mass storage for permanent archiving. External users, as mentioned above, have the option of staging their data to a storage facility, their home directory, or a location specified in the job creation editor in GridChem. By enforcing this policy on data management, job output can be tracked from the middleware and forwarded back to the user through the GridChem client.

An alternative approach still under consideration is the use of a logical file service (LFS) such as Globus Replica Location Service (Chervenak et al., 2002) or Storage Resource Broker (Rajasekar et al., 2003), or the use of an advert service such as StorageBox (Hupfeld, 2004), which can easily double as an LFS. This approach would be more inline with the SoA discussed up to now. Using this approach, GridChem and the GMS would defer the responsibility of file management to a third party rather than splitting the responsibility between the GMS and client as is done now.

Several other benefits of an LFS are advanced tracking and versioning control, a globally unique namespace, and third party file transfers. Such features are attractive given the added functionality they enable. Through replica tracking, it would be possible to backup all user data in the CCG mass storage facility. Another advantage for larger files would be that the user is always assured that the nearest available copy of the data is download . At present, however, a working implementation is in place to handle data for every supported CCG application. Until new requirements are requested by the user community, there is no plan to alter the architecture.

4. Conclusion

Having already released the alpha version of GridChem in August, the project is now benefiting from user feedback during a "friendly user" period. One of the largest challenges encountered thus far is basic account administration. Section 3.2 discussed briefly the user allocation form. This form has highlighted several problems in the way users were foreseen using the CCG. Originally it was thought that users would flock to GridChem at the promise of free resources. What has been observed thus far is that the ability to use existing allocations on CCG resources is equally important to early adopters. This may be

due to the large percentage of the user community already possessing CCG allocations, or it may be due to misunderstanding as to how the community allocation works. To address the latter possibility, the CCG Education, Outreach, Teaching, and Support (EOTS) committee is currently working on updated documentation and online tutorials. As well, a training workshop will be given in November in Seattle, Washington, at Supercomputing 2005.

One recurring request from many users is for the incorporation of workflow support into the GridChem client. Several researchers are currently performing task farming and/or more complex jobs that GridChem could potentially support. Such a feature request requires significant adaptation at both the client and server levels. However, because workflow support is such a powerful tool that can, in the future, dramatically expand the CCG user community, it will be incorporated into GridChem early next year.

One goal put forth at the beginning of the project was to try and move people towards grid technology. Specifically, the desire was to make people feel comfortable learning about and using their grid certificates. At the first CCG Workshop in April, 2005, many users were enthusiastic about the notion of a community account for running their jobs. The SSH authentication interface was extremely popular and users expressed willingness to begin using grid certificates through a similar interface - especially if it meant gaining access to additional compute time.

Since that workshop, the CCG infrastructure has grown to enable such functionality. Users can move seamlessly from one authentication mechanism to another with nearly no effect on their overall experience. Now that the tools are in place to make a case to the community, the process of moving users towards grid technology and addressing the needs they are sure to raise has begun.

Acknowledgements

Special thanks Ian Kelley and Jon MacLaren for thoughtful review, as well as Michael Sheetz and the UKy development team for their contribution to the GridChem GUI. This work was funded in part by the National Science Foundation, Award #0438312 and the Center for Computation & Technology at LSU.

References

WS-I. Web Services Interoperability Organization Basic Profile Version 1.1. Final Material, August 2004. http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html

K. Milfeld, C. Guiang, S. Pamidighantam, J. Giuliani. Cluster Computing through an Application-oriented Computational Chemistry Grid. Proceedings of the 2005 Linux Clusters: The HPC Revolution.

D. Thain, T. Tannenbaum, and M. Livny. "*Condor and the Grid*", in F. Berman, A. J. G. Hey, G. Fox, editors, *Grid Computing: Making The Global Infrastructure a Reality*, John Wiley, 2003.

K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A Resource Management Architecture for Metacomputing Systems. In D. Feitelson and L. Rudolph, editors, Job Scheduling Strategies for Parallel Processing (Proceedings of the Fourth International JSSPP Workshop; LNCS #1459), pages 6282. Springer-Verlag, 1998.

G. Allen, K. Davis, T. Dramlitsch, T. Goodale, I. Kelley, G. Lanfermann, J. Novotny, T. Radke, K. Rasul, M. Russell, E. Seidel, O. Wehrens. "*The GridLab Grid Application Toolkit.*" HPDC 2002: 411.

W. Smith. "Improving Resource Selection and Scheduling using Predictions," in Grid Resource Management. J. Nabrzyski, J.M. Schopf, J. Weglarz (Eds). Kluwer Publishing, Fall 2003.

S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, S. Tuecke. "*A Directory Service for Configuring High-Performance Distributed Computations*". Proceedings of the 6th IEEE Symposium on High-Performance Distributed Computing, pp. 365-375, 1997.

J. Ainsworth, J. MacLaren, J. Brooke. "*Implementing a Secure, Service Oriented Accounting System for Computational Economies.*" Proceedings of the 5th IEEE International Symposium on Cluster Computing and the Grid. Cardiff, Wales. May, 2005.

G. Aloisio, M. Cafaro, I. Epicoco, S. Fiore, D. Lezzi, M. Mirto and S. Mocavero, "iGrid, a Novel Grid Information Service", to appear in Proceedings of Advances in Grid Computing - EGC 2005 (European Grid Conference, Amsterdam, The Netherlands, February 14-16, 2005, Revised Selected Papers), Lecture Notes in Computer Science, Springer-Verlag, Volume 3470, pp. 506-515, 2005.

A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunszt, M. Ripenu, B. Schwartzkopf, H. Stocking, K. Stockinger, B. Tierney. "Giggle: A Framework for Constructing Scalable Replica Location Services", Proceedings of the IEEE Supercomputing 2002.

F. Hupfeld. "Log-Structured Storage for Efficient Weakly-Connected Replication". Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS) Workshops 2004.

A. Rajasekar, M.Wan, R. Moore, W. Schroeder, G. Kremenek, A. Jaheesan, C. Cowart, B. Zhu, S. Chen, R. Olschanowsky. Storage Resource Broker - Managing Distributed Data in a Grid. Computer Society of India Journal, Special Issue on SAN, Vol. 33, No. 4, pp. 42-54 Oct 2003.

S. See and T. W. Tan. APBioBox and BioClusterGrid: computational infrastructure for life sciences. First International Workshop on Life Science Grid (LSGRID2004), May 31st-June 1st, 2004.

F. Raih, Y. Sharum, R. M. R. Moktar, N. M. Isa, N. L. Kian, N. M. Mahadi, R. Mohamed. EMASGRID: An NBBnet Grid Initiative for a Bioinformatics and

Computational Biology Services Infrastructure in Malaysia. First International Workshop on Life Science Grid (LSGRID2004), May 31st-June 1st, 2004. 117-124.

J. Basney, M. Humphrey, and V. Welch. The MyProxy Online Credential Repository. Software: Practice and Experience, Volume 35, Issue 9, July 2005, pages 801-816.

Aloisio G. , Cafaro M. , Epicoco I. , Fiore S. , Lezzi D. , Mirto M. , Mocavero S. Resource and Service Discovery in the iGrid Information Service Proceedings of International Conference on Computational Science and its Applications (ICCSA 2005), Springer-Verlag, Volume 3482, pp. 1-9, 2005.

M. Thomas, J. Boisseau. Grid Computing: Making the Global Infrastructure a Reality, Ch 28 F. Berman, G. Fox and T. Hey, eds. John Wiley and Sons, Ltd, Chichester (2003).

http://gridport.net/main/pubs/GridPort_Grids02.doc.

Globus WebPage–GRAM Overview.
http://www-unix.globus.org/toolkit/docs/3.2/gram/key/index.html.

Southeastern University Research Association WebPage.
http://www.sura.org/.

Fleet Numerical Meteorology and Oceanography Center WebPage.
http://www.fnmoc.gov.

Texas Advanced Computing Center WebPage.
https://portal.tacc.utexas.edu/portal.html.

Singapore National Grid Life Science Virtual Community Portal
http://www.ngp.org.sg/lsvgc/index.html.

The KISTI Supercomputing Center.
http://www.ksc.re.kr/english/5-1-2_chemistry.htm.

Chemistry at KISTI.
http://www.kisti.re.kr/kisti/english/english_main.jsp?content=8.

European Commission Information Technology Society WebPage.
.

IBM Grid Toolbox WebPage.
http://www-1.ibm.com/grid/solutions/grid_toolbox.shtml.

Access Grid WebPage. http://www.accessgrid.org/agdp/.

Wiki WebPage. http://wiki.org/wiki.cgi?WhatIsWiki.

OGCD WebPage. http://www.collab-ogce.org/nmi/index.jsp.

GridLab Project WebPage. http://www.gridlab.org/.

The Web Services Resource Framework. http://www.globus.org/wsrf/.

GridChem Project WebPage. http://www.gridchem.org/.

Asia Pacific BioGrid WebPage. http://www.apbionet.org/apbiogrid/.

UTGrid Project WebPage. http://www.ut.edu/grid.

TeraGrid WebPage. http://www.teragrid.org.