

Cluster Computing through an Application-oriented Computational Chemistry Grid

Kent Milfeld and Chona Guiang, Sudhakar Pamidighantam, Jim Giuliani

Texas Advanced Computing Center (TACC)
National Center for Supercomputing Applications (NCSA)
Ohio Supercomputer Center (OSC)

Abstract

Over the last 20 years, personal computers and networking infrastructures have greatly enhanced the working environment and communication of researchers. Also, Linux clusters now flourish, providing significant resources for executing parallel applications. But there is still a gap between desktop environments of individuals and the wide assortment of Unix-flavored high performance computing (HPC) system environments. Grid technologies are delivering tools to bridge this gap, especially between HPC systems; but the difficulty of implementing the infrastructure software (installation, configuration, etc.) has discouraged adaptation of grid software at the desktop level. Hence, users who employ long-running parallel applications in their research still log into a grid-enabled machine to submit batch jobs and manipulate data within a grid. An infrastructure model adapted by the Computational Chemistry Grid (CCG) [1] eliminates dependence on grid software at the desktop, is based on the need to run chemistry applications on HPC systems, and uses a “client” interface for job submission. A middleware server with grid software components is employed to schedule and deploy jobs, and manage resources transparently. This same infrastructure can be used to implement other client/server paradigms requiring pre- and post-processing of application data on the desktop, and application execution on large (HPC) systems, as well as small departmental (Linux) clusters. This paper describes the structure and implementation of the CCG infrastructure and discusses its adaptation to other client/server application needs.

Introduction

The Cluster Revolution has significantly increased the number and availability of small and midrange Linux clusters for science and engineering researchers. The attraction of these systems can be attributed to advances in commodity 32/64-bit proc-

essors, interprocessor communication, compilers, and libraries. In the scientific and engineering communities, these clusters are often configured as batch systems for running scalable applications. Hence, a user logs into a frontend, devoid of a desktop environment, creates inputs and job scripts, and submits and runs jobs. Depending upon the application, further manipulation of the results may be performed on the cluster or the results may be copied back to the “comfort” of the user’s desktop environment for further analysis.

In certain fields such as computational chemistry, defined narrowly as electronic structure and molecular modeling for the scope of this paper, research community users invoke the execution of modules within large packages through inputs that specify the chemistry and physical model for determining structure, dynamics, and properties. These computationally intense parallel applications often require a large number of runs; and a high-performance computing (HPC) cluster with batch scheduling is an ideal platform for running these jobs. In reality, there is really very little need to log into a system if an application is to be executed under batch control. Ideally, it would be less cumbersome to submit batch requests directly from the user’s workstation or PC. There would be no need for a user to become familiar with different Unix variants (Linux, HP-UP, IRIX, UNICOS, Solaris, etc.), local environments, and batch resource specifications of various HPC systems.

While a single interface for remote batch submission mechanism can simplify submission for the user, there is a significant cost for providing this service. At the lowest level, a secure and reliable mechanism must be developed for the common desktop environments: Mac OS X, Windows, and Linux desktops. A common GUI must be created and each job request must be translated into the specific resource format for the target batch system.

Grid technologies have been developed to address security, data transport, and even job submission from desktop environments; but “heavyweight” approaches, such as the installation of Globus [2] on PCs and workstations, are often challenging to set up, install, maintain and use. An alternative approach is to push the heavyweight grid technologies to a server and provide a light-weight client on the desktop. The client interacts with the server, which in turn communicates with a remote HPC system running the applications. This “lightweight” desktop model is implemented in the Computational Chemistry Grid as a three-tier system (client/grid-server/HPC-resource), as illustrated in Figure 1; but the end user only sees a two-tier system (client/HPC-systems).

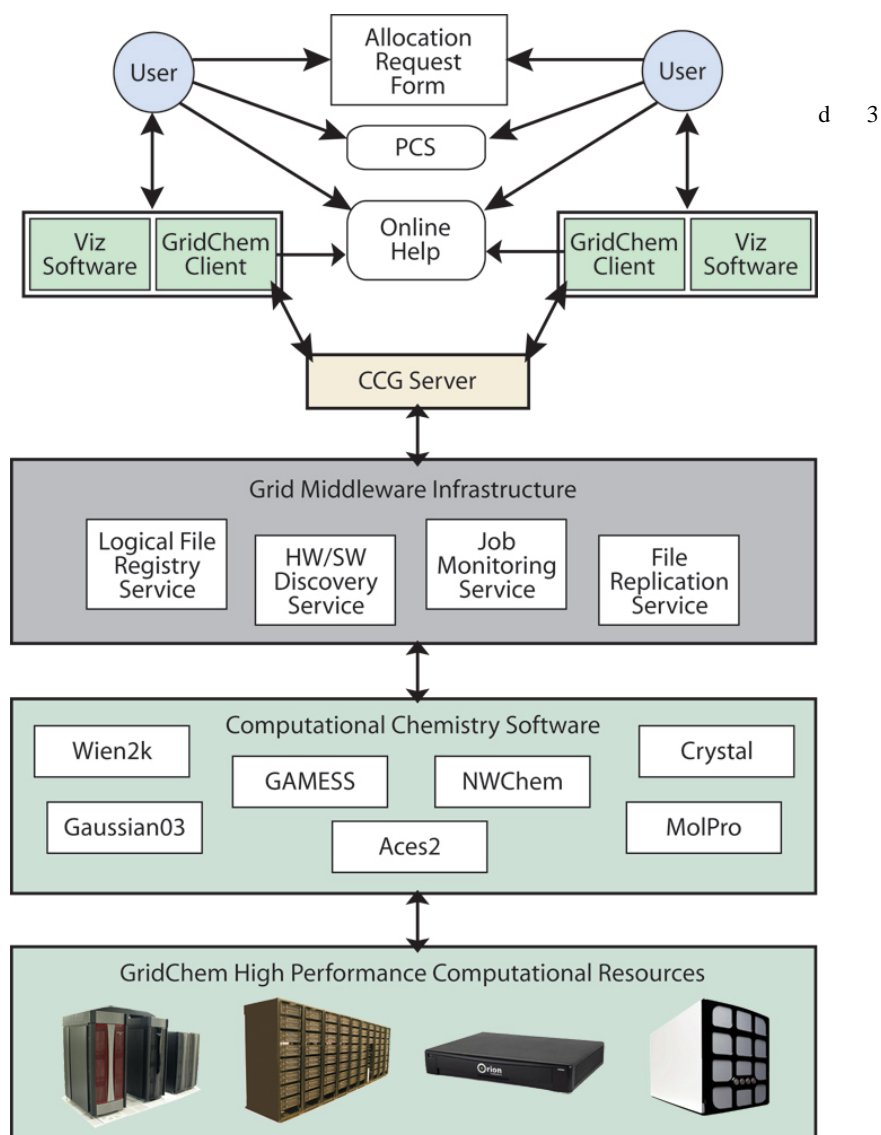


Fig. 1. CCG Three-tier Infrastructure (client/middleware server/HPC resources).

For applications that require simple inputs and produce minimal outputs, a web interface to the batch application server can be constructed. A web-based portal, such as GridPort 3 [3], could be used to create this interface. However, with inputs and outputs that require interactive visualization and larger amounts of data transport, it becomes necessary to provide a client application that is very responsive to object manipulation (xyz rotation/translation) and middleware services that can track and archive job output files.

The Computational Chemistry Grid (CCG) uses the above client-server model for a community of computational researchers to run chemistry applications at five different centers (CCS, CCT, OSC, NCSA, and TACC) [4]. The following sections describe the components of the model: the CCG infrastructure, desktop client, middle-

ware services, and scientific applications. The advantages of using this model for any batch-application need are also examined.

Overview of CCG Architecture

The CCG is implemented with a three-tier architecture, consisting of a desktop client (GridChem client), a grid middleware server (GMS), and compute and data resources (HPC systems). The GridChem client runs on a desktop and provides a user interface for input editing, job submission, monitoring and management, as well as pre- and post-processing of molecular structures through a GUI-based visual molecular editor. The job submission, status monitoring, job deletion and file manipulations are forwarded to the GMS for further processing. At present, GridChem supports three quantum chemistry packages: Gaussian 2003, NWChem, and GAMESS [5]. Within the next two years, six more applications will be integrated into the CCG framework.

The GMS runs the necessary services to ensure transparent job execution on any one of the different HPC systems that comprise the CCG. These services include authentication, data management, resource brokering, scheduling, and job deployment on the computational resource that meets the job specification. Existing middleware technologies like Globus, Condor, MyProxy, and GridFTP [6] are run on the grid middleware server to implement these capabilities. The workflow among the components is illustrated in Figure 2 below.

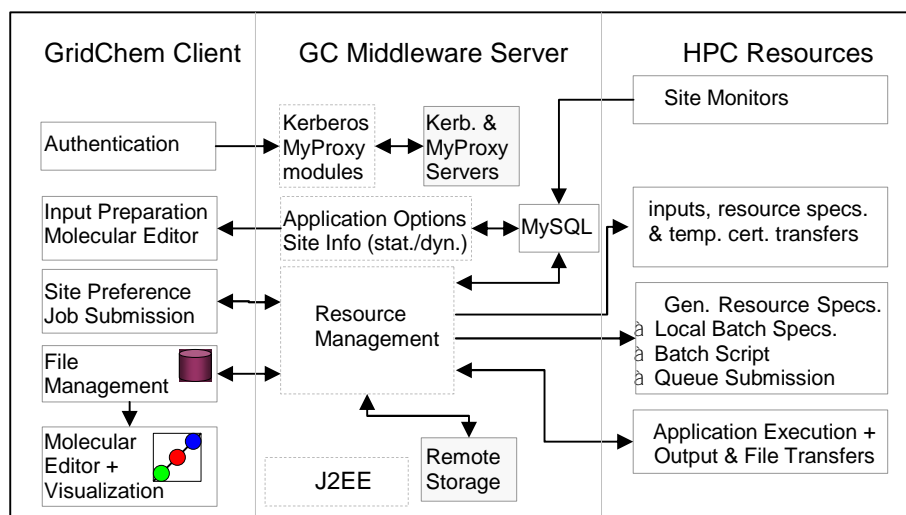


Fig. 2. GridChem Client-Server Components and Workflow.

The grid middleware server maintains information for each computational resource, including the type of platform, node memory, aggregate disk space, and available chemistry software. The GMS also manages file retrieval for intermediate analysis (monitoring) during job execution. Job details, status, and associated input/output files are written to mass storage when jobs complete and are readily available and downloadable to the user through the GridChem client. The server facilitates GridChem client submissions by routing jobs to a compute resource on the grid, storing job information and performing third-party data transfers to mass storage systems. In essence, the server is the gateway to the HPC computing resources. One of the important functions of the server is to provide a tier for grid-related software to operate, and thereby eliminate the necessity of installing, operating and maintaining a grid software stack within the user's desktop environment. For a "grid" consisting of a single HPC resource, the GMS services could be installed on the login/development node of the HPC system. A department might use the GridChem client/server model to provide batch access to applications through a wide variety of desktop platforms.

The computational and data resources of the CCG represent a diverse range of systems, including a large-scale HP Itanium 2 cluster, Intel Xeon systems, SGI Origin, and IBM Power4 systems. Although different resource managers are employed locally at each system (e.g., LoadLeveler, PBS, LSF), users of GridChem can submit jobs to any one of the compute platforms without having to learn the resource specification syntax unique to each batch facility. The generic resource requirements from GridChem are transformed into the batch-specific resource requirements by batch filters (scripts) on the local host. The scripts also make any job script adjustments for site-specific architecture characteristics, queue structures, and policies. Each compute resource provides static and dynamic scheduling information to the server for server-assisted routing of jobs. Once a robust grid scheduler becomes available [7], it will be employed to schedule jobs across the entire grid, and provide optional configuration selections for expediting throughput. Currently, users select a job submission site with the aid of monitoring tools. Over 3.5 million CPU hours per year are available to the chemistry community through the Computational Chemistry Grid project. The present aggregate disk storage in CCG is over 25 TB, and dedicated mass storage will be integrated later.

Desktop Client Interface

The GridChem client is implemented as a Java application and uses the JAVA Virtual Machine (JVM) of the J2SE 1.4 distribution [8]. For GUI clients that must execute on a wide range of platforms, JAVA compiled code has several advantages: a single source and interface removes the problems associated with maintaining platform-specific implementations (as both the client and platforms evolve); a consistent look and feel is presented on different platforms; and the compiled code operates on virtually every OS: Windows, Linux (desktop), Mac OS X, IBM AIX, etc. These advantages, however, come at a price. With a virtual machine, developers must focus more on making sure that the GUI components remain responsive to all user interactions within a virtual machine. Also, the client must be compatible with the J2SE en-

gine; but, it is relatively easy to install and update a JVM that is compatible with the GridChem client. (A window of versions will be maintained to accommodate irregularities in vendor adaptation timelines.)

The GridChem client consists of separate Java modules for user authentication, input data processing (including definition of job requirements), job submission/monitoring/management, and analysis, post-processing and/or visualization of job output. To achieve and manage each module's function, the client communicates with the middleware server, relaying job-related information, application data and configurations, and other state information back to the client. The workflow in GridChem client is described in Figure 3.

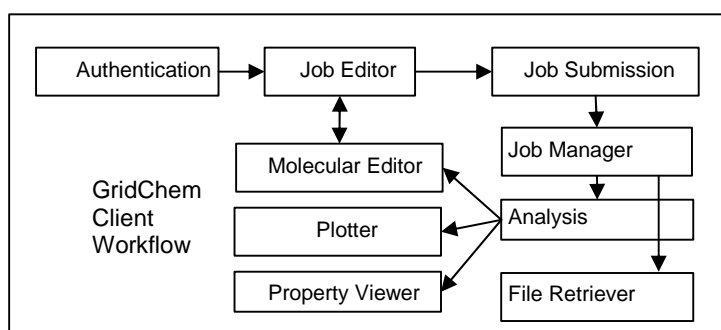


Fig. 3. GridChem Client-Server Components and Workflow.

The authentication module presents a panel in which users can choose the authentication method and provides user identification (login ID) and a corresponding passphrase/password. After authentication, a credential is generated for further secure access within the grid. It is maintained by the server for delegation to resource sites. Also, preference data for the session and job history data are retrieved from a database and server-side storage.

In the job creation module, a script is constructed and submitted once the following operations have been performed: the computational chemistry task (application) has been selected; the package-specific input data have been generated; the resource requirements have been specified, and HPC system for execution determined. The resource requirements include (wall-clock or CPU) time allocated for the execution, memory, and number of processors. Other information that defines resource requirements may include queue names, priorities, and disk space. For computational chemistry applications, the input consists of a description of the molecular system, the quantum chemistry methods to be used in the calculation and the basis set. There are multiple ways to include the molecular structure. The GridChem client has an integrated molecular editor for generating molecules from atomic or fragment components. To obtain a reasonable first guess for the set of initial atomic coordinates, the editor module features a molecular structure optimizer based on a molecular mechanics force field. The structure can also be defined by providing Cartesian coordinates for each molecular component or supplying relative coordinates using a Z-matrix. For

pre-existing input files, a file browser is included to retrieve files. In every instance, the job editor module can save the molecular data in a variety of formats for later reuse.

Once a job is ready, a submission is requested through a Submit Job button. Before sending the job to the middleware server for routing to an appropriate HPC resource, the GridChem client “preprocesses” the job input data to detect common errors that would cause the application execution to fail. Making the job submission interface foolproof is an ongoing and high-priority design goal of the development team. One planned enhancement of the editor module will dynamically present the available options (e.g., choice of methods and basis sets) as the structure is edited, for a chosen quantum chemistry application.

The job management module supervises jobs. It allows a user to select a job for monitoring from a job history data file (later from a database), check the status of the job in the remote production system, and retrieve and analyze its output. Intermediate or final output is parsed and relevant parameters are collected for display or listings. For instance, the structure of a molecule can be displayed in the molecular editor to visually monitor any changes in the structure during geometry optimization and transition-state search calculations. Additionally, tools for viewing molecular orbitals in 3D are being developed. Jobs that are not progressing satisfactorily can be aborted.

Middleware Server

The grid middleware server is responsible for user authentication, authorization, data management, as well as scheduling and job deployment on the computational resources. As an intermediary between the desktop and the computational backend, the GMS also presents job information to the client and makes intermediate data analysis possible while the job is running

User Authentication

Although the GridChem client contains an interface for user authentication, generation and handling of user credentials take place at the grid middleware server. Currently, three authentication mechanisms are supported.

Kerberos authentication can be selected when jobs are submitted to sites that support this authentication service. In this case, the GMS creates a Kerberos ticket on behalf of the user, using the name and password entered at the authentication panel of the GridChem client. The Kerberos ticket has a two-hour lifetime, and users are required to re-authenticate to access services after expiration of the credential. The re-authentication can be performed in the GridChem client.

A second authentication mechanism uses the Globus Security Infrastructure (GSI) which requires that the user has an existing x509 certificate validated by a recognized

CA, and a login name/Distinguished Name entry is assumed to exist in the grid-mapfile at each resource. The grid middleware server requests a proxy credential on behalf of the user and employs MyProxy for credential storage and management. GSI and MyProxy authentication offer three advantages: First, the use of credential delegation through a proxy prevents the user from entering a passphrase multiple times whenever authentication to a grid resource or authorization to use a service is required. Secondly, the use of MyProxy for credential management enables single sign-on capability on all of the grid resources. Third, MyProxy supports credential renewal, and this process may be automated at the GMS level so the user is freed from explicitly renewing the credential when it expires. This is the preferred method for long running jobs. MyProxy initialization can be performed directly in the GridChem client.

SSH authentication is integrated into GridChem, and implemented at the client-level only. Unlike GSI authentication, SSH does not allow single sign-in to all of the grid computational resources. The purpose of this method is to allow users to bypass grid middleware services and directly submit their jobs through a local account to a single HPC system. This mechanism is ideal for researchers that want to use the GridChem client to run applications on a local cluster where grid services are not installed, but the cluster administrator is willing to install a subset of the GridChem services (submission scripts).

Data Management

The grid middleware server handles three types of data: user data, infrastructure-related data, and system data. User data includes input, output, restart (if any), and job history files. Infrastructure files consist of accounting/usage data and application-specific information for customizing GUI client options. Data related to systems, including hardware information, resource characteristics, work load, queue information, paths to local launch scripts, and local policies, are stored at the grid middleware server.

Each user is allotted space on the middleware server for storage of job-related intermediate files and user-specific files required by different grid services such as proxy credentials, timestamps, and intermediate log/error files that are communicated to the client. (Much of the information in these “database” files is being converted to MySQL database tables in the next GridChem release.)

A database is maintained on the middleware server to manage static and dynamic information of the compute resources, as well as the application options. Before deployment of a metascheduler, the client will retrieve static, system-specific information (e.g., total CPU count, memory per node, aggregate disk space, available chemistry software) and dynamic information (e.g., system load, queue depths, available disk space), and present the data to help users select a system for job submission. Application methods and algorithm choices, functions, output verbosity control, and system run-time parameters are examples of application-specific data that will change with

software updates. Inclusion of this information in a database decouples the GridChem client from the evolving nature of the supported software and resource batch policies, and will prevent the client from having to be updated with every new revision of the application.

At the end of a job, output files are automatically sent to a remote storage device, such as a preferred mass storage, using UberFTP, a GridFTP-enabled ftp client. During the job, `gsiscp` commands extract intermediate output for the monitoring utility (for text display, energy plots, and/or visualization of the structure evolution). While options for retrieving the data sets to local disks are available in the client, no third-party transfer capabilities are available.

Resource Broker/Scheduling

In the initial phase of the CCG, GridChem client users will select a site for job submission, as outlined above. Within the next six months, the Condor workload management system will be integrated into the CCG infrastructure to provide metascheduling capabilities across the different grid resources. Condor will allocate resources to jobs by matching ClassAds, which are analogous to buyer/seller advertisements in the newspaper Classified Ads. Jobs are consumers of resources (buyers) and publish want ads, while HPC machines are providers of resources (sellers) and post sell ads to the Condor collector (newspaper). The Condor negotiator assigns jobs to machines based on conformance between the ClassAds of consumers (jobs) and providers (a pool of HPC machines) in a process called matchmaking.

Machine ClassAds contain static (e.g., CPU speed, memory per node, processor type) and dynamic attributes (e.g., current utilization), and can also include local conditions under which jobs are allowed to run on a host. Job ClassAds specify corresponding resource request attributes, such as wall-clock time, memory, number of CPUs, etc. For multiple matches, a rank attribute is computed to assign relative priorities to the matches. The ClassAds mechanism is extensible: new attributes can be constructed when predefined attributes are inadequate or insufficiently specific.

While Condor matchmaking ensures that jobs are assigned to machines that can satisfy their resource requirements, it performs no other scheduling functions. More intelligent job scheduling across a heterogeneous pool of HPC grid resources demands three essential components. These are: utilities for gathering local resource management information (queue priorities, queued job resource requirements, and the present job schedule and “fair share” policies), a repository for storing and accessing this information, and a scheduling algorithm that formulates optimal throughput decisions based on projected queue wait/job runtimes (may be calculated from current and historical workload information) and other intelligence-based criteria. In essence, a grid metascheduler will need to make judicious load-balancing decisions on a grid-wide basis that approach the efficiency of local resource managers (setting advanced reservations, understanding queue depths and backfilling, etc.). The development of a metascheduler across different grid resources is actively being pursued by the NMI

[9] community. Once available, the CCG will use this metascheduler to provide transparent job management and deployment on the grid.

Support and Accounting

In the second year of the GridChem project, a user portal implemented with GridPort will be employed for displaying the dynamic system information for each CCG computational site and will serve as a one-stop location where users can access support services (through the PCS [10] web-based ticketing system—presently implemented as a stand-alone web service), allocation information and management, discussion/ mailing lists, user guides, and site news.

The target user communities for CCG resources are: computational chemists who are local to participating sites, computational chemists who have large research activities and have NRAC (National Resource Allocations Committee) or other peer-reviewed allocations at one of the national NSF centers, and the “academic community at large” of experimental and theoretical chemists who occasionally need access to computational packages for their research.

Allocation policies adhere to national, site-specific, and community guidelines for duration, notification, refunding, and usage. Each site maintains allocations for the projects, a set of user logins under each project, and a gridmap file associating each login with a DN (Distinguished Name). Each day the batch logs are filtered for GridChem usage and the project balances updated. Also, there is a community allocation that requires job-specific accounting. Hence, job-specific accounting data must be reported back to the user through the client. The local and community balances are aggregated into a daily project balance within a database at the GridChem Middleware server.

Summary

The Computational Chemistry Grid (CCG) forms a virtual client-application server environment for computational chemistry users. Researchers interact with the CCG only through the GridChem client, and may submit jobs directly to a specific HPC system or leave the decision to a metascheduler, which will be integrated in CCG at a later time. Since the client is Java-based, it can be installed on Linux workstations, PCs and Macs with minimal effort. The application server layer consists of distributed resources for running applications, storage resources, and infrastructure software for data transfer and resource management. In reality, CCG is implemented as a three-tier architecture, composed of a desktop client (GridChem), a middleware server, and compute and data resources. All grid and data management tasks are handled transparently at the middleware level, without user participation. (For a small group of clients and a single cluster, the middleware services can be installed on the cluster front-end.)

By presenting the grid as an abstract application resource/environment, the CCG can increase user productivity and reduce the barrier of entry for using (chemistry) applications on high-end computational resources. The CCG desktop-client/application-server model uses a middleware server as a “gateway” to the grid, eliminating the need for heavyweight grid software on desktops. Since jobs can be deployed across multiple resources in a grid, turnaround can be increased. The most fundamental advantage, however, is the uniform access to multiple applications and HPC resources with a single desktop interface. A simplified interface is very helpful in increasing user productivity, by eliminating the need for the user to log on to different HPC systems and learn OS-specific commands and rules/syntax for job submission to the local resource manager (e.g., PBS, LoadLeveler, NQS, etc.).

References

- 1 GridChem: <http://www.gridchem.org/> The GridChem client was originally developed at NCSA.
- 2 Globus, <http://www.globus.org/>
- 3 GridPort: <http://gridport.net/index.cgi/>
- 4 CCG Centers: CCS, www.ccs.uky.edu; CCT, cct.lsu.edu; OSC, www.osc.edu; NCSA, www.ncsa.uiuc.edu; TACC, www.tacc.utexas.edu
- 5 Gaussian, http://www.gaussian.com; GAMESS, http://www.msg.ameslab.gov/GAMESS;
NWChem, <http://www.emsl.pnl.gov/docs/nwchem/nwchem.html>
- 6 MyProxy, <http://grid.ncsa.uiuc.edu/myproxy/>
- 7 Condor, <http://www.cs.wisc.edu/condor/>; NSF NMI Middleware Initiative, <http://www.nsf-middleware.org/>
- 8 JAVA, <http://www.sun.com/software/java/>
- 9 NMI, www.nsf-middleware.org
- 10 PCS, Portable Consulting System developed @ TACC: www.gridchem.org/consult

Cluster Computing through an Application-oriented Computational Chemistry Grid

Kent Milfeld and Chona Guiang, Sudhakar Pamidighantam, Jim Giuliani

Texas Advanced Computing Center (TACC)
National Center for Supercomputing Applications (NCSA)
Ohio Supercomputer Center (OSC)

Abstract

Over the last 20 years, personal computers and networking infrastructures have greatly enhanced the working environment and communication of researchers. Also, Linux clusters now flourish, providing significant resources for executing parallel applications. But there is still a gap between desktop environments of individuals and the wide assortment of Unix-flavored high performance computing (HPC) system environments. Grid technologies are delivering tools to bridge this gap, especially between HPC systems; but the difficulty of implementing the infrastructure software (installation, configuration, etc.) has discouraged adaptation of grid software at the desktop level. Hence, users who employ long-running parallel applications in their research still log into a grid-enabled machine to submit batch jobs and manipulate data within a grid. An infrastructure model adapted by the Computational Chemistry Grid (CCG) [1] eliminates dependence on grid software at the desktop, is based on the need to run chemistry applications on HPC systems, and uses a “client” interface for job submission. A middleware server with grid software components is employed to schedule and deploy jobs, and manage resources transparently. This same infrastructure can be used to implement other client/server paradigms requiring pre- and post-processing of application data on the desktop, and application execution on large (HPC) systems, as well as small departmental (Linux) clusters. This paper describes the structure and implementation of the CCG infrastructure and discusses its adaptation to other client/server application needs.

Introduction

The Cluster Revolution has significantly increased the number and availability of small and midrange Linux clusters for science and engineering researchers. The attraction of these systems can be attributed to advances in commodity 32/64-bit proc-

essors, interprocessor communication, compilers, and libraries. In the scientific and engineering communities, these clusters are often configured as batch systems for running scalable applications. Hence, a user logs into a frontend, devoid of a desktop environment, creates inputs and job scripts, and submits and runs jobs. Depending upon the application, further manipulation of the results may be performed on the cluster or the results may be copied back to the “comfort” of the user’s desktop environment for further analysis.

In certain fields such as computational chemistry, defined narrowly as electronic structure and molecular modeling for the scope of this paper, research community users invoke the execution of modules within large packages through inputs that specify the chemistry and physical model for determining structure, dynamics, and properties. These computationally intense parallel applications often require a large number of runs; and a high-performance computing (HPC) cluster with batch scheduling is an ideal platform for running these jobs. In reality, there is really very little need to log into a system if an application is to be executed under batch control. Ideally, it would be less cumbersome to submit batch requests directly from the user’s workstation or PC. There would be no need for a user to become familiar with different Unix variants (Linux, HP-UP, IRIX, UNICOS, Solaris, etc.), local environments, and batch resource specifications of various HPC systems.

While a single interface for remote batch submission mechanism can simplify submission for the user, there is a significant cost for providing this service. At the lowest level, a secure and reliable mechanism must be developed for the common desktop environments: Mac OS X, Windows, and Linux desktops. A common GUI must be created and each job request must be translated into the specific resource format for the target batch system.

Grid technologies have been developed to address security, data transport, and even job submission from desktop environments; but “heavyweight” approaches, such as the installation of Globus [2] on PCs and workstations, are often challenging to set up, install, maintain and use. An alternative approach is to push the heavyweight grid technologies to a server and provide a light-weight client on the desktop. The client interacts with the server, which in turn communicates with a remote HPC system running the applications. This “lightweight” desktop model is implemented in the Computational Chemistry Grid as a three-tier system (client/grid-server/HPC-resource), as illustrated in Figure 1; but the end user only sees a two-tier system (client/HPC-systems).

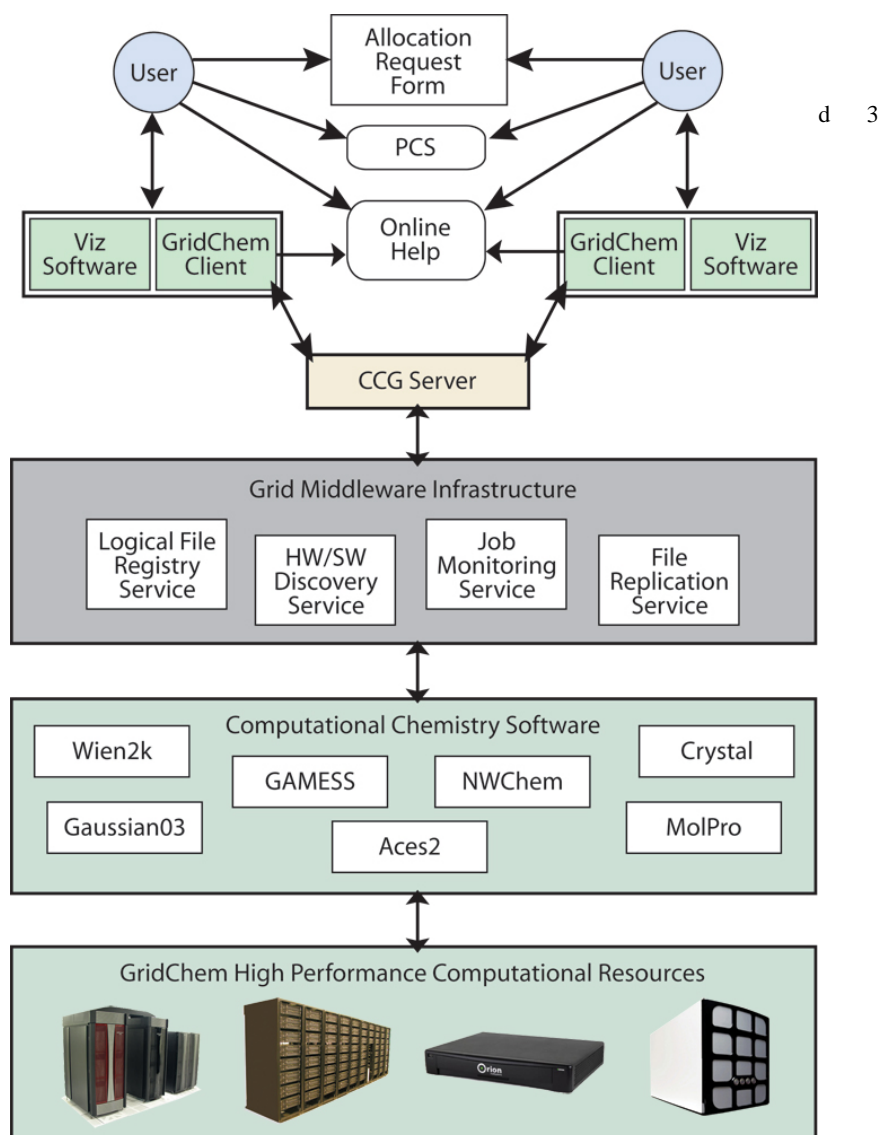


Fig. 1. CCG Three-tier Infrastructure (client/middleware server/HPC resources).

For applications that require simple inputs and produce minimal outputs, a web interface to the batch application server can be constructed. A web-based portal, such as GridPort 3 [3], could be used to create this interface. However, with inputs and outputs that require interactive visualization and larger amounts of data transport, it becomes necessary to provide a client application that is very responsive to object manipulation (xyz rotation/translation) and middleware services that can track and archive job output files.

The Computational Chemistry Grid (CCG) uses the above client-server model for a community of computational researchers to run chemistry applications at five different centers (CCS, CCT, OSC, NCSA, and TACC) [4]. The following sections describe the components of the model: the CCG infrastructure, desktop client, middle-

ware services, and scientific applications. The advantages of using this model for any batch-application need are also examined.

Overview of CCG Architecture

The CCG is implemented with a three-tier architecture, consisting of a desktop client (GridChem client), a grid middleware server (GMS), and compute and data resources (HPC systems). The GridChem client runs on a desktop and provides a user interface for input editing, job submission, monitoring and management, as well as pre- and post-processing of molecular structures through a GUI-based visual molecular editor. The job submission, status monitoring, job deletion and file manipulations are forwarded to the GMS for further processing. At present, GridChem supports three quantum chemistry packages: Gaussian 2003, NWChem, and GAMESS [5]. Within the next two years, six more applications will be integrated into the CCG framework.

The GMS runs the necessary services to ensure transparent job execution on any one of the different HPC systems that comprise the CCG. These services include authentication, data management, resource brokering, scheduling, and job deployment on the computational resource that meets the job specification. Existing middleware technologies like Globus, Condor, MyProxy, and GridFTP [6] are run on the grid middleware server to implement these capabilities. The workflow among the components is illustrated in Figure 2 below.

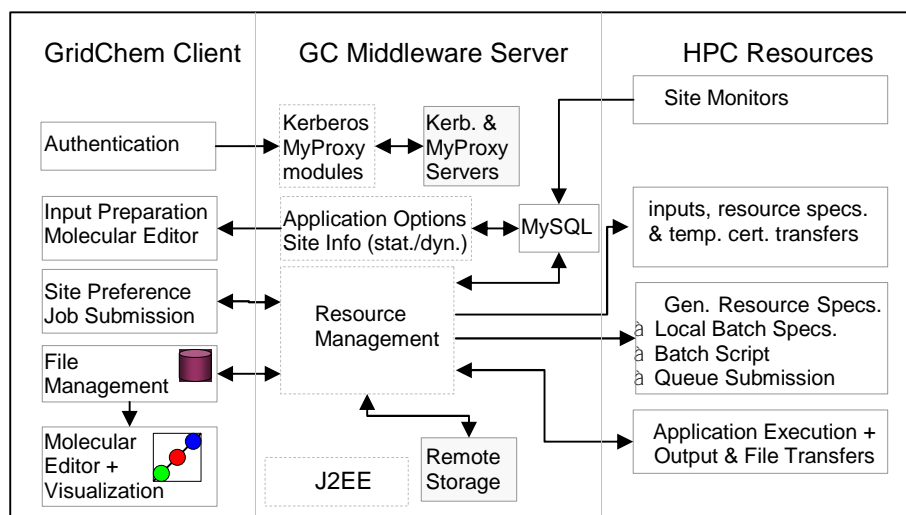


Fig. 2. GridChem Client-Server Components and Workflow.

The grid middleware server maintains information for each computational resource, including the type of platform, node memory, aggregate disk space, and available chemistry software. The GMS also manages file retrieval for intermediate analysis (monitoring) during job execution. Job details, status, and associated input/output files are written to mass storage when jobs complete and are readily available and downloadable to the user through the GridChem client. The server facilitates GridChem client submissions by routing jobs to a compute resource on the grid, storing job information and performing third-party data transfers to mass storage systems. In essence, the server is the gateway to the HPC computing resources. One of the important functions of the server is to provide a tier for grid-related software to operate, and thereby eliminate the necessity of installing, operating and maintaining a grid software stack within the user's desktop environment. For a "grid" consisting of a single HPC resource, the GMS services could be installed on the login/development node of the HPC system. A department might use the GridChem client/server model to provide batch access to applications through a wide variety of desktop platforms.

The computational and data resources of the CCG represent a diverse range of systems, including a large-scale HP Itanium 2 cluster, Intel Xeon systems, SGI Origin, and IBM Power4 systems. Although different resource managers are employed locally at each system (e.g., LoadLeveler, PBS, LSF), users of GridChem can submit jobs to any one of the compute platforms without having to learn the resource specification syntax unique to each batch facility. The generic resource requirements from GridChem are transformed into the batch-specific resource requirements by batch filters (scripts) on the local host. The scripts also make any job script adjustments for site-specific architecture characteristics, queue structures, and policies. Each compute resource provides static and dynamic scheduling information to the server for server-assisted routing of jobs. Once a robust grid scheduler becomes available [7], it will be employed to schedule jobs across the entire grid, and provide optional configuration selections for expediting throughput. Currently, users select a job submission site with the aid of monitoring tools. Over 3.5 million CPU hours per year are available to the chemistry community through the Computational Chemistry Grid project. The present aggregate disk storage in CCG is over 25 TB, and dedicated mass storage will be integrated later.

Desktop Client Interface

The GridChem client is implemented as a Java application and uses the JAVA Virtual Machine (JVM) of the J2SE 1.4 distribution [8]. For GUI clients that must execute on a wide range of platforms, JAVA compiled code has several advantages: a single source and interface removes the problems associated with maintaining platform-specific implementations (as both the client and platforms evolve); a consistent look and feel is presented on different platforms; and the compiled code operates on virtually every OS: Windows, Linux (desktop), Mac OS X, IBM AIX, etc. These advantages, however, come at a price. With a virtual machine, developers must focus more on making sure that the GUI components remain responsive to all user interactions within a virtual machine. Also, the client must be compatible with the J2SE en-

gine; but, it is relatively easy to install and update a JVM that is compatible with the GridChem client. (A window of versions will be maintained to accommodate irregularities in vendor adaptation timelines.)

The GridChem client consists of separate Java modules for user authentication, input data processing (including definition of job requirements), job submission/monitoring/management, and analysis, post-processing and/or visualization of job output. To achieve and manage each module's function, the client communicates with the middleware server, relaying job-related information, application data and configurations, and other state information back to the client. The workflow in GridChem client is described in Figure 3.

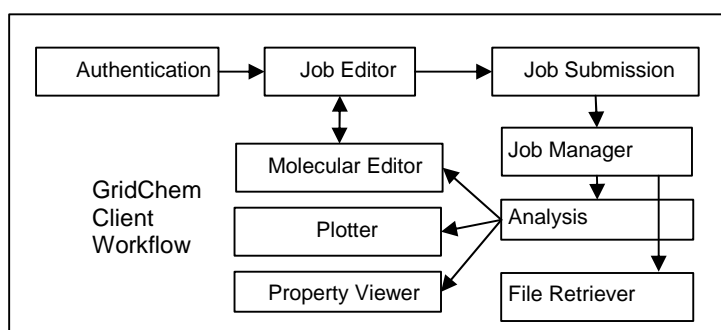


Fig. 3. GridChem Client-Server Components and Workflow.

The authentication module presents a panel in which users can choose the authentication method and provides user identification (login ID) and a corresponding passphrase/password. After authentication, a credential is generated for further secure access within the grid. It is maintained by the server for delegation to resource sites. Also, preference data for the session and job history data are retrieved from a database and server-side storage.

In the job creation module, a script is constructed and submitted once the following operations have been performed: the computational chemistry task (application) has been selected; the package-specific input data have been generated; the resource requirements have been specified, and HPC system for execution determined. The resource requirements include (wall-clock or CPU) time allocated for the execution, memory, and number of processors. Other information that defines resource requirements may include queue names, priorities, and disk space. For computational chemistry applications, the input consists of a description of the molecular system, the quantum chemistry methods to be used in the calculation and the basis set. There are multiple ways to include the molecular structure. The GridChem client has an integrated molecular editor for generating molecules from atomic or fragment components. To obtain a reasonable first guess for the set of initial atomic coordinates, the editor module features a molecular structure optimizer based on a molecular mechanics force field. The structure can also be defined by providing Cartesian coordinates for each molecular component or supplying relative coordinates using a Z-matrix. For

pre-existing input files, a file browser is included to retrieve files. In every instance, the job editor module can save the molecular data in a variety of formats for later reuse.

Once a job is ready, a submission is requested through a Submit Job button. Before sending the job to the middleware server for routing to an appropriate HPC resource, the GridChem client “preprocesses” the job input data to detect common errors that would cause the application execution to fail. Making the job submission interface foolproof is an ongoing and high-priority design goal of the development team. One planned enhancement of the editor module will dynamically present the available options (e.g., choice of methods and basis sets) as the structure is edited, for a chosen quantum chemistry application.

The job management module supervises jobs. It allows a user to select a job for monitoring from a job history data file (later from a database), check the status of the job in the remote production system, and retrieve and analyze its output. Intermediate or final output is parsed and relevant parameters are collected for display or listings. For instance, the structure of a molecule can be displayed in the molecular editor to visually monitor any changes in the structure during geometry optimization and transition-state search calculations. Additionally, tools for viewing molecular orbitals in 3D are being developed. Jobs that are not progressing satisfactorily can be aborted.

Middleware Server

The grid middleware server is responsible for user authentication, authorization, data management, as well as scheduling and job deployment on the computational resources. As an intermediary between the desktop and the computational backend, the GMS also presents job information to the client and makes intermediate data analysis possible while the job is running

User Authentication

Although the GridChem client contains an interface for user authentication, generation and handling of user credentials take place at the grid middleware server. Currently, three authentication mechanisms are supported.

Kerberos authentication can be selected when jobs are submitted to sites that support this authentication service. In this case, the GMS creates a Kerberos ticket on behalf of the user, using the name and password entered at the authentication panel of the GridChem client. The Kerberos ticket has a two-hour lifetime, and users are required to re-authenticate to access services after expiration of the credential. The re-authentication can be performed in the GridChem client.

A second authentication mechanism uses the Globus Security Infrastructure (GSI) which requires that the user has an existing x509 certificate validated by a recognized

CA, and a login name/Distinguished Name entry is assumed to exist in the grid-mapfile at each resource. The grid middleware server requests a proxy credential on behalf of the user and employs MyProxy for credential storage and management. GSI and MyProxy authentication offer three advantages: First, the use of credential delegation through a proxy prevents the user from entering a passphrase multiple times whenever authentication to a grid resource or authorization to use a service is required. Secondly, the use of MyProxy for credential management enables single sign-on capability on all of the grid resources. Third, MyProxy supports credential renewal, and this process may be automated at the GMS level so the user is freed from explicitly renewing the credential when it expires. This is the preferred method for long running jobs. MyProxy initialization can be performed directly in the GridChem client.

SSH authentication is integrated into GridChem, and implemented at the client-level only. Unlike GSI authentication, SSH does not allow single sign-in to all of the grid computational resources. The purpose of this method is to allow users to bypass grid middleware services and directly submit their jobs through a local account to a single HPC system. This mechanism is ideal for researchers that want to use the GridChem client to run applications on a local cluster where grid services are not installed, but the cluster administrator is willing to install a subset of the GridChem services (submission scripts).

Data Management

The grid middleware server handles three types of data: user data, infrastructure-related data, and system data. User data includes input, output, restart (if any), and job history files. Infrastructure files consist of accounting/usage data and application-specific information for customizing GUI client options. Data related to systems, including hardware information, resource characteristics, work load, queue information, paths to local launch scripts, and local policies, are stored at the grid middleware server.

Each user is allotted space on the middleware server for storage of job-related intermediate files and user-specific files required by different grid services such as proxy credentials, timestamps, and intermediate log/error files that are communicated to the client. (Much of the information in these “database” files is being converted to MySQL database tables in the next GridChem release.)

A database is maintained on the middleware server to manage static and dynamic information of the compute resources, as well as the application options. Before deployment of a metascheduler, the client will retrieve static, system-specific information (e.g., total CPU count, memory per node, aggregate disk space, available chemistry software) and dynamic information (e.g., system load, queue depths, available disk space), and present the data to help users select a system for job submission. Application methods and algorithm choices, functions, output verbosity control, and system run-time parameters are examples of application-specific data that will change with

software updates. Inclusion of this information in a database decouples the GridChem client from the evolving nature of the supported software and resource batch policies, and will prevent the client from having to be updated with every new revision of the application.

At the end of a job, output files are automatically sent to a remote storage device, such as a preferred mass storage, using UberFTP, a GridFTP-enabled ftp client. During the job, `gsiscp` commands extract intermediate output for the monitoring utility (for text display, energy plots, and/or visualization of the structure evolution). While options for retrieving the data sets to local disks are available in the client, no third-party transfer capabilities are available.

Resource Broker/Scheduling

In the initial phase of the CCG, GridChem client users will select a site for job submission, as outlined above. Within the next six months, the Condor workload management system will be integrated into the CCG infrastructure to provide metascheduling capabilities across the different grid resources. Condor will allocate resources to jobs by matching ClassAds, which are analogous to buyer/seller advertisements in the newspaper Classified Ads. Jobs are consumers of resources (buyers) and publish want ads, while HPC machines are providers of resources (sellers) and post sell ads to the Condor collector (newspaper). The Condor negotiator assigns jobs to machines based on conformance between the ClassAds of consumers (jobs) and providers (a pool of HPC machines) in a process called matchmaking.

Machine ClassAds contain static (e.g., CPU speed, memory per node, processor type) and dynamic attributes (e.g., current utilization), and can also include local conditions under which jobs are allowed to run on a host. Job ClassAds specify corresponding resource request attributes, such as wall-clock time, memory, number of CPUs, etc. For multiple matches, a rank attribute is computed to assign relative priorities to the matches. The ClassAds mechanism is extensible: new attributes can be constructed when predefined attributes are inadequate or insufficiently specific.

While Condor matchmaking ensures that jobs are assigned to machines that can satisfy their resource requirements, it performs no other scheduling functions. More intelligent job scheduling across a heterogeneous pool of HPC grid resources demands three essential components. These are: utilities for gathering local resource management information (queue priorities, queued job resource requirements, and the present job schedule and “fair share” policies), a repository for storing and accessing this information, and a scheduling algorithm that formulates optimal throughput decisions based on projected queue wait/job runtimes (may be calculated from current and historical workload information) and other intelligence-based criteria. In essence, a grid metascheduler will need to make judicious load-balancing decisions on a grid-wide basis that approach the efficiency of local resource managers (setting advanced reservations, understanding queue depths and backfilling, etc.). The development of a metascheduler across different grid resources is actively being pursued by the NMI

[9] community. Once available, the CCG will use this metascheduler to provide transparent job management and deployment on the grid.

Support and Accounting

In the second year of the GridChem project, a user portal implemented with GridPort will be employed for displaying the dynamic system information for each CCG computational site and will serve as a one-stop location where users can access support services (through the PCS [10] web-based ticketing system—presently implemented as a stand-alone web service), allocation information and management, discussion/ mailing lists, user guides, and site news.

The target user communities for CCG resources are: computational chemists who are local to participating sites, computational chemists who have large research activities and have NRAC (National Resource Allocations Committee) or other peer-reviewed allocations at one of the national NSF centers, and the “academic community at large” of experimental and theoretical chemists who occasionally need access to computational packages for their research.

Allocation policies adhere to national, site-specific, and community guidelines for duration, notification, refunding, and usage. Each site maintains allocations for the projects, a set of user logins under each project, and a gridmap file associating each login with a DN (Distinguished Name). Each day the batch logs are filtered for GridChem usage and the project balances updated. Also, there is a community allocation that requires job-specific accounting. Hence, job-specific accounting data must be reported back to the user through the client. The local and community balances are aggregated into a daily project balance within a database at the GridChem Middleware server.

Summary

The Computational Chemistry Grid (CCG) forms a virtual client-application server environment for computational chemistry users. Researchers interact with the CCG only through the GridChem client, and may submit jobs directly to a specific HPC system or leave the decision to a metascheduler, which will be integrated in CCG at a later time. Since the client is Java-based, it can be installed on Linux workstations, PCs and Macs with minimal effort. The application server layer consists of distributed resources for running applications, storage resources, and infrastructure software for data transfer and resource management. In reality, CCG is implemented as a three-tier architecture, composed of a desktop client (GridChem), a middleware server, and compute and data resources. All grid and data management tasks are handled transparently at the middleware level, without user participation. (For a small group of clients and a single cluster, the middleware services can be installed on the cluster front-end.)

By presenting the grid as an abstract application resource/environment, the CCG can increase user productivity and reduce the barrier of entry for using (chemistry) applications on high-end computational resources. The CCG desktop-client/application-server model uses a middleware server as a “gateway” to the grid, eliminating the need for heavyweight grid software on desktops. Since jobs can be deployed across multiple resources in a grid, turnaround can be increased. The most fundamental advantage, however, is the uniform access to multiple applications and HPC resources with a single desktop interface. A simplified interface is very helpful in increasing user productivity, by eliminating the need for the user to log on to different HPC systems and learn OS-specific commands and rules/syntax for job submission to the local resource manager (e.g., PBS, LoadLeveler, NQS, etc.).

References

- 1 GridChem: <http://www.gridchem.org/> The GridChem client was originally developed at NCSA.
- 2 Globus, <http://www.globus.org/>
- 3 GridPort: <http://gridport.net/index.cgi/>
- 4 CCG Centers: CCS, www.ccs.uky.edu; CCT, cct.lsu.edu; OSC, www.osc.edu; NCSA, www.ncsa.uiuc.edu; TACC, www.tacc.utexas.edu
- 5 Gaussian, http://www.gaussian.com; GAMESS, http://www.msg.ameslab.gov/GAMESS;
NWChem, <http://www.emsl.pnl.gov/docs/nwchem/nwchem.html>
- 6 MyProxy, <http://grid.ncsa.uiuc.edu/myproxy/>
- 7 Condor, <http://www.cs.wisc.edu/condor/>; NSF NMI Middleware Initiative, <http://www.nsf-middleware.org/>
- 8 JAVA, <http://www.sun.com/software/java/>
- 9 NMI, www.nsf-middleware.org/
- 10 PCS, Portable Consulting System developed @ TACC: www.gridchem.org/consult