

Motivation and Requirements for a User Interface to a Computational Grid in the UK

Ali Anjomshoaa*, Lorna Smith
Edinburgh Parallel Computing Centre
University of Edinburgh
Edinburgh, EH9 3JZ

January 2003

Abstract

In an area such as High Performance Computing (HPC), where the lifetime of application codes often exceeds that of most HPC machines, the ability to access these machines through a simple generic end-user interface is of real interest. By developing an end-user interface to a wide area computational Grid that includes HPC machines, we may be able to hide some of the complexity associated with the diverse and changing nature of the UK's HPC systems and their administrative domains, from the users.

This report examines the need for an end-user interface to a wide area computational Grid, which will be comprised of a heterogeneous environment of machines in diverse administrative domains, within various institutions across the UK. This need is first motivated, before a discussion of the requirements for an end-user interface is presented in the following sections. This discussion also addresses possible implementations of the end-user interface.

*Please e-mail comments and feedback to ali@epcc.ed.ac.uk

Preamble

The terms *machine* and *resource* are used interchangeably throughout this document to refer to physical computing resources used for computational needs.

The terms *executable* and *binary* are used interchangeably throughout this document to refer to the compiled and linked programs for execution on a machine.

The terms *batch system* and *resource management system* are used interchangeably throughout this document to refer to the software applications which manage resource usage through job management on a set of queues, according to a set of usage policies.

1 Introduction

Computing and information technology (IT) Grids are a new paradigm. They take many forms, from computational Grids, which involve the combination of many computing resources into a network for the execution of computational jobs, to data Grids, which should allow coherent and uniform access to distributed data. Other Grids may exist as a collaborating set of computing service applications in order to perform a particular task, such as various meteorological data services whose information may be combined to provide a weather summary of a particular part of the world at a specific time. It should be noted that, the goal of Grids is to virtually combine geographically distributed IT resources from many different administrative domains, into single customised resources that can perform a single or a series of specific tasks. Currently this is being done using the ubiquitous Internet or smaller Internets.

The distributed resources on a Grid, be they number-crunching computing nodes, or a plethora of heterogeneous databases, or a mixture of these and other resources such as network enabled electronic instrumentation, are likely to be from many administrative domains. This means that, these resources belong to different owners and are managed by different administrators, all of whom will conform to different sets of rules and configuration directives, and impose different usage policies on their users. It is highly unlikely that the administrators of individual domains will give up the autonomy that they enjoy over their resources, particularly where the Grids span unrelated administrative domains across organisational boundaries.

Why then should these resource owners wish to share their resources through a Grid by allowing inclusive outside access to users whom they do not know and can not directly contact and administer? It might be the case that these resource owners are service providers, in which case the Grid will just be the mechanism by which they provide access to their resources for users who in some way pay for their use. It might also be the case that the resource owners are academic institutions and other organisations, whom acting as service providers may well receive public or private incentives. Providing ubiquitous access to a scientific database, for example, can be compared to providing a library service.

Further, how will it be possible for the heterogeneous set of environments described above to be combined into a collaborative Grid of IT resources? Environments where not only are the resources wide and varied in nature, but so are their

owners and administrators and the policies under which their use is governed. Environments under which some homogeneity must govern the interaction of users with services, and services with each other?

As a first step toward providing a solution to this problem, one might look for a single interface, which will bind and interact with the various heterogeneous resources, so as to provide an abstract interface through which they may interact with each other. This interface may well take the form of middleware, software that mediates between application programs and the network of heterogeneous hardware that might be hosting them. This middleware may be compared to an operating system (OS) for a Grid, which would allow file replication and data transfer across networks, allow access to compute nodes, and handle security and permission issues. A more complete solution would be the development of suitable sets of native standards and protocols, which resources and their providers would adhere to in order to make them Grid enabled. The latter is currently the focus of the Global Grid Forum (<http://www.gridforum.org/>).

Grid middleware is currently available in many forms, not all of which follow the same architecture or set of principles. The Globus Toolkit (<http://www.globus.org/>) is a popular example of Grid middleware. This toolkit is comprised of modules, each of which provide a set of tools to undertake various key functionalities of a Grid OS. These functionalities include those that deal with security issues, data issues, resource management issues, and system information issues on the Grid. Globus provides a single interface to a heterogeneous Unix computing environment, by interacting with the common system tools found on those resources.

It is out-with the scope of this document to provide solutions to the various political, sociological, and technological problems that exist with the development and deployment of Grids. Instead, this document will concentrate on the relatively more simple and short term issue of the requirements for a user interface (UI) to a computational Grid. In particular one could focus on a wide area computational Grid based on the Globus Toolkit in the UK.

2 Motivating the use of a Computational Grid

In order to use a remote machine for computational jobs, users currently need to logon to that remote machine, develop a job and its environment on that ma-

chine, and then to submit the job, either interactively or to a batch system, on that machine. This has the administrative overhead that each user must have been authorised on, and have prior arranged access to that machine. In addition, each user has to familiarise themselves with the specific setup of the remote environment which they wish to use, such as the location of various tools and libraries on that system.

The goal of a UK computational Grid is to enable users to access multiple remote computing resources via Grid middleware. This should alleviate the need for those users to access each machine directly. Further, with a more advanced computational Grid infrastructure, users will not even need to know of the existence nor the availability of individual resources. In this scheme, users would simply submit a computational job, specifying its associated data, to the computational Grid, and would then receive the results where they choose. In addition, once appropriate Virtual Organisations (VOs) have been set up, users would simply become members of a VO that has been authorised on, and has prior arranged access to a plethora of machines. This system of VO membership distributes and condenses user management overheads and removes the need for replication of administrative efforts across the Grid. The use of VOs to administer users is an attractive feature of Grids.

The concept of VOs is a key issue in the architecture and use of Grids. Consider the simple scenario of a computational Grid to which users wish to submit computational jobs. These users may be classified into groups that have certain attributes in common. They may be grouped according to their funding bodies (e.g. in the UK, academic research is funded through government funding councils) or their funding scheme, or according to their interests, for example a collaboration of scientists. These groups will be geographically and organisationally distributed, for example in various institutions, organisations, and industries, distributed within a country or continent. In addition, they will be highly dynamic in nature, that is, the period over which they exist and the membership of the group and its size will change within the lifetime of the group. Another consideration maybe that membership of these groups may be governed by any number of rules and regulations, for example some may need tighter security checks within government or industry. These groups are the Virtual Organisations mentioned in the previous paragraph.

It should be noted that, introducing a new computing system and platform for scientific research, such as that offered by a ubiquitous UK wide computational

Grid, will always face some resistance from researchers who are used to the status quo of their working practices and, sadly, the limitations that they face regarding the capabilities of the computational resources that are available to them. Part of the effort in setting up a UK wide computational Grid, therefore, should include the marketing of this resource to the end-users. In order to do this, a user interface (UI) to such a new system needs to be carefully thought out and implemented, to the point where the user will find it an attractive feature, and the service offered by the computational Grid a benefit to their research. In addition, the UI should make the overhead of porting research from existing local and remote environments to the computational Grid as low as possible.

Consider a collaboration of scientists and engineers working on a project, for example, a consortium of cosmologists. Such a collaboration will need storage space for their data and access to compute power in order to process and analyse these data, and in order to perform simulations with which to understand the results. Currently, they would probably access resources located remotely at some high performance computing centre to which they have access. In the worst case, their initial point of entry would probably be a terminal in a Unix environment, which they would have accessed using a secure shell (ssh) connection. From there, an individual user would either develop their computational job on that machine and set up the correct environment for it using command line Unix, or transfer their job from their local environment to the executing environment (perhaps by means ftp or scp) and do some setting up of the Unix environment prior to the execution of that job. Invoking a job might be either done interactively on the command line, or by submitting that job to a batch system where it would sit on a queue until that resource management system submits it for execution. Further, the user must ensure that the data and other necessary input and output files are available to and accessible by the job.

This last example of the collaboration of cosmologists is one that fits well with the description of a VO given above. Individual members will be from a variety of different organisations and institutions, which will be geographically distributed. They use high performance computing remotely, which might be accessed through a computational Grid. In addition, this Grid will have more resources than they would normally have access to, and if these resources are all reached through the same uniform interface, then this would alleviate the need to introduce these end-users to new sets of command line interfaces and yet another set of development environments, especially on an infrastructure where hetero-

geneous resources exist on a geographically distributed network. Encouraging these users to utilise this new infrastructure will be no trivial task, and will face resistance at the outset. The move to a new environment will always have costly overheads that will affect the project management schedule, and face a mental barrier as a result. For this reason the introduction of a UK computational Grid needs to be well marketed and facilitated, so as to make the initial overhead of porting research to the computational Grid small if not negligible.

It should also be noted that most scientific end-users of computational services are not computing experts, and should not have to become one in order to carry out their research. Thus, they should not have to learn about various systems and their setup. When a service provider who operates a high performance computer upgrades their service and switches from one system vendor to another, it also needs to facilitate the migration of its users to the new environment. If this service provider is faced with changes to the interaction of a single UI with its service, it would make the task of migrating users and their tasks to the new environment far easier than having to deal with the needs of each user individually. This also has the benefit that the end-users see no change in their interaction with the new system as the interface itself and its functionality will not be changing. They would thus submit their jobs to this system in the same way as before. This example illustrates another use of a UI to a computational Grid, that of abstracting the direct interaction of end-users with the systems which they use. In the example here, it may be noted that if the computational service provided were a part of a wider computational Grid, users would not experience any disruption in service during the migration to a new system. This carries the condition that the VO, of which they are a member, must have authority to access other resources on that Grid.

An initial cost free computational Grid service implemented by academic institutions for academic research, and offering limited amounts of development and usage time on academic resources, is an attractive start to developing a computational Grid and building an end-user base. The tools supplied along with such a service, such as an end-user UI, are an important feature of the service and should be provided alongside a guarantee of system support and some assurance of Quality of Service (QoS). As a tool, the importance of a well thought out and implemented UI cannot be overstated in facilitating the move onto the computational Grid. It is the first point of contact and first impression of the look and feel of the service that end-users will base some of their judgement on. The

uptake and usage of a UK computational Grid, therefore, will depend on a good marketing strategy, which should highlight the benefits and the ease of use of this Grid. In order to be effective as a tool, however, the UI to a computational Grid cannot be a thin client. It must offer much more functionality than an interface for specifying jobs. These functionalities are discussed in the *Requirements for a User Interface to a Computational Grid* section of this report.

In Summary

The Globus Toolkit has a set of APIs that currently support the development of higher level client software, such as a job-submission UI. The advantages of providing a standard UI, with a number of core functions, for a UK computational Grid are;

- providing users with an easy to learn and use GUI based UI for job submission, monitoring, and control, and perhaps even job development,
- providing users with a standard de facto interface associated with the UK's computational Grid and high performance computing services.

The latter point being a key issue for marketing the said Grid. The provision of a standard UI must surely be part of the service provided by a computational Grid. Without it, the computational Grid would be alike to the World Wide Web without a web-browser, where users would have to retrieve and parse html documents on the command line or by using shell scripts!

3 Requirements for a User Interface to a Computational Grid

The requirements for a UI to a computational Grid are discussed below.

Requirements for Specifying a Single Computational Job

At the basic level, users will need to be able to specify single computational jobs that will not have any dependencies on other jobs, for submission to the Grid.

Such jobs will have well defined and independent input and output data associated with them. The user should be able to define these data and provide their location on local or remote resources (see later for *Access to Remote Data*).

Users should be able to submit a computational job as source code that will need to be compiled and linked for execution, or as a statically or dynamically linked binary that will be targeted to a specific platform for which it was compiled and linked.

Remote Compilation

In cases where an end-user wishes to submit a computational job in the form of source code to the Grid, a UI must provide the facility for the user to specify that the job must first be compiled and linked before it is submitted for running on a remote resource. Thus, the user may be given the ability to choose the platform of the remote resource and a compiler. In addition, the user may wish to provide a script or make file, which states how the job is to be compiled and linked and the system libraries required for this purpose. The greatest advantage of being able to submit a computational job in the form of source code is that end-users do not need to worry about the choice of the resource platform on which the job runs when they submit that job to the Grid. Further, users may not have direct access to a particular platform in order to develop their job for running on that platform. Access to specific platforms on the computational Grid, to members of a VO whom has been given authority to submit jobs to those platforms, is an invaluable feature of a computational Grid.

An important hurdle to overcome for the submission of source code to a Grid, is the availability of the necessary compilers and the reporting of compiler errors to the user. The UI must be able to invoke compilers and so there must be some method by which their location on remote resources is made known to the UI. One possible method by which to do this is to include information on compilers as part of the system specifications of the remote resource. It would then be a matter of using the discovery methods in place, for the UI to learn about the available compilers. In the event of build failure, the UI must be able to return the necessary information back to the user for debugging purposes. The user should then go through the cycle of debugging and resubmitting their job for recompilation. It would perhaps be advisable that the user builds and tests their application on a specific platform, before submitting it to the Grid. Any errors

which occur during build on the Grid, may then be attributed to the particular system on the Grid on which the building took place and failed. For debugging purposes, the user should target the platform, or indeed the same resource, on which the first build on the Grid had failed.

Dynamically Linked Binaries

The issue of dynamically linked binaries is a potentially serious hurdle to overcome. The user should be able to submit a dynamically linked executable. There are two issues which need to be overcome in this scenario. The target machine for the job must be one that,

- is of the correct platform for the compiled and linked job, and
- contains the libraries required for the dynamically linked executable in the correct location.

The latter constraint is non-trivial. The set up of Unix systems and other non-Unix platforms is sufficiently variable to make the issue of the execution of dynamically linked executables on remote machines difficult, especially without the overhead of moving or rearranging those libraries needed on the target resource. Possible options are,

1. to copy those libraries needed to a default location on the remote machine, for which the executable has been dynamically linked (for example, this might be the `/tmp/` directory or some other `/griduser/` directory on a Unix machine) and to which the user has write permission, or
2. to attempt to find those libraries on the remote machine by searching the standard paths on which they may exist.

In case 1., it should be within the remit of the UI to make copies of the necessary libraries on the remote resources. In addition, the user will have to re-link their dynamically linked executables with those libraries in the location to which they will be copied to on the remote resource by the UI. One must take into account the amount of disk space available on the remote machine for the libraries to be copied across.

In case 2., the executable must then be linked with the libraries in the path on which they exist on the remote machine. This is far from trivial. It would involve the user moving libraries around on their local machine, to the location on which they exist on the remote machine, and re-linking their executable, which in turn would require that the user has write permission to those locations on their local machine.

A temporal limit on either of these solutions may be a desirable attribute, that is, if either of these operations take longer than `time_limit`, then cancel request and inform user.

Alternatively, one could devise a system by which absolute paths (such as URLs) can be specified for the required libraries, so that upon execution on a remote resource, those libraries are read from the local or another remote location. The disadvantage of this solution is inherent in the remote access and transfer of libraries. In addition, when building the binary, these absolute paths must be used for linking.

Overall, one must take into account the reasons and advantages for submitting dynamically linked executables over statically linked executables. These may include,

- the size of a statically linked version of the executable being large enough to make its transfer to and storage on remote resources on the Grid not viable, especially for multiple submissions of a job. This could make the extra overhead involved with moving and copying libraries around and re-linking executables worthwhile.
- Dynamically linked executables, such as third-party and commercial software, for which the user has no access to the source code.

Following the above discussion on the submission of computational jobs as source code, or statically or dynamically linked binaries, it should be noted that the use of more portable languages such as Java leads to the simplification of many of the issues outlined. Traditionally, high performance applications have been developed using the Fortran and C and C++ programming languages. These languages are platform dependent and are not readily portable in that they must be compiled and linked with system libraries specific to the target platform, in order to produce machine code executables. Java on the other hand is compiled into byte-code which can be executed using a Java byte-code interpreter (commonly

known as the Java Virtual Machine). It is the interpreter which is platform dependent, but, which resides on each resource that is Java enabled. The end-user who submits a Java byte-code computational job to the Grid, need not worry about platform dependence. It should also be noted that the use of Java as a high performance computing language and one that can be used in parallel computing, though advanced, is currently in the research phase (<http://www.javagrande.org> and http://www.epcc.ed.ac.uk/computing/research_activities/JOMP/).

The issues of compiling and executing jobs on heterogeneous architectures on the grid, without loosing the basic principle of hiding the complexity of the grid from users, is a difficult one. There are currently no ideal solutions to the problems faced. What is clear, however, is that any UI that is developed, would have to offer users the ability to compile jobs remotely before running them, and to make individual decisions about dynamic versus static libraries. The issue is less important for VOs based around specific areas of science, where a limited range of codes are used by multiple users. Executables could be compiled on a range of platforms by one individual and made available to the users.

Requirements for Specifying a Job of Multiple Tasks

A user should be able to specify a complex job that involves the execution of multiple jobs, or tasks, the dependencies between which should be easily defined. These dependencies may be temporal and should be defined hierarchically, i.e. task A has to be done before task B. The input and output data for the tasks must be defined for each task, providing the ability to specify relational data, that is, the input for task B is the output of task A.

The UI must provide the user with the option of specifying the set of tasks and the relationships between them. If the set of tasks are multiples of the same task, that is, a very loosely coupled or uncoupled set of parallel tasks, then the user will wish to state the number of times the task runs and the input and output associated with each instance of the task. The Nimrod system is designed exactly for this type of parametrised job, where for each task a set of input parameters is varied (<http://www.csse.monash.edu.au/davida/nimrod/>).

On the other hand, the set of tasks associated with a job may not be loosely coupled or multiple instances of the same task. That is, the tasks may be a chain of different tasks whose input and output have dependencies on each other's. It is

then necessary for the UI to enable the user to specify the tasks independently. A graphical interface may be provided for the specification of task relationships and dependencies, for example a graphical representation of a Directed Acyclic Graph (DAG).

See the section on *Requirements for Specifying a Single Computational Job* for the discussion on individual job submission and the issues involved in dealing with dynamically linked executables.

4 Parallel Environments

The UI should enabled users to submit jobs to parallel environments and to specify resource requirements, such as the number of processors and the hardware attributes required for that job. To this end, users should be able to target particular types of resources (e.g. SMP machines), but, should not have to concern themselves with the type of resource management systems administering jobs on those machines nor their actual locations.

5 Resource Management

Users should not be concerned with resource management on the Grid, nor with resource management systems on individual resources on the Grid. The UI should itself deal with various resource management systems on remote resources, taking into account the user's job constraints such as time limits for job completion. The UI will itself need to be able to undertake higher level scheduling decisions based on information available from remote resources. A job that needs to be finished today cannot sit on a batch queue for 24 hours before execution begins. In addition, the UI will need to be able to evaluate and analyse constraints and the demand on Grid resources.

This requirement dictates the need for the UI to have some scheduling capability. A matching paradigm where user requirements for jobs are matched to resource capabilities of various machines on the Grid, is desirable. The Condor High Throughput software implements such a matchmaking system, where the requirements of users' jobs are matched to the machine attributes advertised by resources (<http://www.cs.wisc.edu/condor/>).

Of course the issue of resource management does not come into light if resources are being exclusively targeted. Further, on a more advanced Grid, where dedicated Grid schedulers exist, this functionality can be delegated from the UI to Grid schedulers, and jobs can subsequently be submitted via such schedulers. This advance will of course depend on the emergence of appropriate description languages for jobs and resources. Again, Condor's matchmaking paradigm uses the *ClassAds* (Classified Advertising) mechanism, where jobs and resources are described in ClassAds using a description language for the required and preferred attributes that are to be matched. A standard and portable implementation of a Job Description Language (JDL), perhaps using XML, for the submission of computational jobs, and a Resource Description Language (RDL) for matching resources to the requirements of those jobs, would be an invaluable addition to a computational Grid.

6 Job Monitoring and Control

As well as submitting jobs to a computational Grid, users should be able to monitor and control their jobs through the UI. Users should be able to query remote resources, or the Grid as a whole since they should not need to know where their jobs are running or are waiting in a batch system, for progress on their jobs. This would involve the UI registering handles for submitted jobs in order to keep track of which resources they were submitted to.

Users should also have control over the jobs they have submitted to the Grid. For example, if a job has been waiting in a batch system for too long, the user should have the ability to migrate that job through cancellation and resubmission, either in two steps or atomically. A user should be able to state policies for advanced job control by the UI, such as automated checkpointing and migration, at the time of job submission.

It may be desirable to provide the UI with the ability to collate information on all of the jobs submitted by a user. For example, in the case where a user has submitted a job of multiple tasks as outlined above, this will be useful in order to provide an overall picture of the status of the users' jobs. It may also be desirable to give the user, through the UI, the ability to customise what information about the jobs is collated and in what way. For example, a user may wish to know the elapsed time since the submission of a job of multiple tasks, and the status of each

task in the running order.

7 Access to Remote Data

The input and output data that the user specifies for a job may be stored remotely. The user will need to provide the location of the remote data which is to be accessed or written out, to the UI. It should be within the remit of the UI to access that data. Of course, ideally the user should not have to care where the data is on the Grid. If a ubiquitous wide-area file system were in operation on the Grid, the user would only have to care about the location of files and data with respect to some root location under which they exist. This would mean that geographically distributed files and data on file systems and databases would all have to map to a single root handle for their retrieval. This is a complex issue to address and will involve the development of new standards, mechanisms, and systems. The Andrew File System (afs) is an example of the basic infrastructure necessary to access files over a distributed file system. The issues of security, that is, authentication and authorisation, and data transfer both need addressing for a distributed file system on the Grid.

The attraction of the ability to store data on a distributed file system is the potential availability, at short notice, of copious amounts of disk space. This is very much in line with the idea of the availability of processing power on tap, which a computing Grid promises. Users will no longer have to worry about the disk quota available for their application at a particular site. They can instead take advantage of very large amounts of disk space available on and accessed through the Grid.

The picture of a distributed storage space on the Grid does, however, raise the question of reliability. What level of data redundancy is required in order to guarantee up-to-date back up of data and access upon request. Further, what form must this back up take? Is it possible to have multiple copies of all data that are asynchronously updated or mirrored? If so, how will the multiple copies of the data be identified as copies of each other, so that if one is unavailable, the requester may access another. Another important issue is that of provenance, that is, the history of the data. It is important to enable users to version data with ease, and to have that data backed-up.

On a more mature Grid, the user may only need to specify the virtual location of

their data store within a VO-space (the VO's virtual host environment or universe). The mapping between the virtual data store and the physical storage space may be specified and implemented by a dedicated service within the VO-space. The user or UI need not know about the physical location of the data store. The data can then be physically moved around on the Grid, for example when a database or storage are moved, upgraded or superseded, without affecting the user or application. It may be possible to have the UI hold references of the locations of data duplicates for the data specified by the user, in order for a job to have uninterrupted access to data. It will then be necessary to implement into the UI the functionality to monitor the availability of the data necessary for a job and to make the migration to a mirror site as necessary.

The mechanism for data transfer (https, ftp, etc.) should not be an issue for the user. The UI should undertake to find the most appropriate transfer mechanism depending on the client and server resources, and data transfer needs. For example, if a remote resource doesn't support the GridFTP mechanism provided by Globus 2.0, the UI should choose another method. It is extremely desirable for the resources placed on a Grid to provide an appropriate and secure method for serving and access to remote data.

Accessing data on a remote resource will require authentication and authorisation, both of which should be undertaken by the UI using the user's credentials. Authentication using Grid proxy certificates is outlined below, and authorisation through VO membership was mentioned above.

8 Authentication Mechanisms

The area of security on the Grid is both crucial and complex. Security breaks down into the two issues of authentication and authorisation. Authentication is the process by which a user's identity is authenticated as the user known to the system. Authorisation is the process by which the permission for the authenticated user to perform a particular task, such as query a system or run a computational job, is verified and granted. The basics of user authentication on a computational Grid are addressed in this section and those of authorisation in the next.

Users on the computational Grid will most likely make use of an X.509 certificate authentication scheme. The certificates can be digitally signed by a certificate

authority (CA) which is recognised by the resources on this Grid. Short-lived certificate proxies to delegate identity and undertake third-party user authentication and authorisation should be used. A proxy is a short lived copy of the user's certificate which has been created by the user or an agent, such as a UI. Users should not, however, need to generate certificate proxies manually. It is desirable for a certificate proxy to be automatically created by the UI when a user submits a job to the Grid. In addition, it would be a desirable feature if the certificate proxy was checked for adequate lifetime upon submission of further jobs, and/or for the proxy's lifetime to be dynamically extended as required. This lifetime management of a certificate proxy must only be undertaken for proxies that were created when a user submitted a job to the Grid from the same local machine.

An automated process for the lifetime management of user certificate proxies by the UI is desirable. It should be possible for the UI to extend the lifetime of the proxy as required by a user's jobs which are running on the Grid. The UI should also undertake to destroy certificate proxies when no jobs depend on that proxy. The UI, therefore, will need to map job handles that it retains to get access to jobs on remote resources, to the proxy used for their submission.

9 Authorisation Mechanisms

A central repository index of users' Distinguished Names (DN), which appears on their digitally signed X.509 certificates, should be kept by the issuing CA. If a CA is recognised and accepted by a resource, then the CA's repository index of users can be queried by that resource when it is propositioned for a job. This is a pull model, where the resource looks to the CA to verify authorisation.

The converse, a push model, would be one where the CA would publish its central repository to all the resources that recognise and accept it, upon the repository being updated. The resources receiving notification of the user repository update from a CA, may then choose whether or not to update their local repositories of authorised users.

Clearly a two way authentication and authorisation mechanism is required between the CAs and the resources, for both the pull and push models outlined above. Accepted CAs, therefore, must always exist in the local authorisation lists of all resources.

10 Accounting and Charging Mechanisms

An accounting mechanism needs to be in place in order to provide some Quality of Service (QoS) and fare-share between users. An accounting model is crucial for resource and user management on the Grid, a point which has been heavily neglected in Grid development thus far.

A charging mechanism can take a dynamic form with the use of tokens exchanged between users and resources. Use of appropriate economic and cost models are crucial for such a charging mechanism. This dynamic charging mechanism, that is, the payment of tokens for resource usage, must involve the UI. To this end, the UI needs to know what capital the user has left and what the cost of resources are. It is desirable for the UI, as part of its high level scheduling capability on the Grid, to take into account the cost of a job on various machines and to make an appropriate choice taking into account temporal and other constraints dictated by the user. In such a scheme, resources would need to advertise their cost model and tariffs on the Grid alongside their core attributes, such as their platform, size of memory, and so on.

The cost for resource usage will depend on the economic model exercised. This model could take into account premium charges for usage at peak and off-peak periods, for example time of day and week, and the quota allocated to the user. The complexity of the model is arbitrary, depending on the number of parameters used to control the allocated time for a user on the Grid.

11 Targeting Resources

It should be possible for a user to target resources of a particular type or in a particular location or domain through the UI. In addition, in the absence of exclusive resource selection by the user, the UI will need to be able to match jobs to suitable resources in its role as a high level hierarchical scheduler. It is therefore necessary for the UI to be able to query information about available resources. This can be done by a lookup service, or resource registry, with which resources register themselves and to which resources send system attributes and resource particulars such as the operating platform, as specified by their administrators. System particulars may include tools and other software for building and executing applications, as well as cost models under which the resource is charged

for.

Based on the information recovered from the resource registry, the UI will then be able to perform higher level scheduling of the jobs in its remit on the Grid. This operation depends on the matching between the requirements for the job and the availability of resources which can satisfy those requirements. The matchmaking system used by the Condor High Throughput software, where the user writes a classified advert (ClassAd) stating the requirements for their job as a set of attributes, and where resources advertise their abilities in resource ClassAds, is a good starting point as a model for doing this.

It has already been mentioned above, that the hierarchical scheduling ability of the UI may be delegated to dedicated Grid scheduling services, if and when any become available.

12 Logistics

The UI discussed above for use with a computational Grid, is the client application that should run on a local machine from which a user wishes to interact with and use that Grid.

With the current plans of the Engineering Task Force of the UK's Grid Support Centre, this will mean that the UI client application will need to sit on top of the Globus Toolkit. The Globus Toolkit APIs may be enough to enable the implementation of most of the functionality and requirements of the UI that have been discussed above.

The Grid Support Centre (<http://www.grid-support.ac.uk/>) will need to partake in the development and deployment of the UI, and to provide subsequent support for its use. The UI should ideally be a portable application, such that it will not be restricted to specific platforms. To this end, a UI developed using Java technologies seems ideal. It should be possible for such an application to make use of the Java based Commodity Grid (CoG) kit for Globus, which has been developed to integrate client applications with Globus using the Globus APIs.

Further, with the advent of the Open Grid Services Architecture (OGSA) (<http://www.ggf.org/ogsa-wg/>), it may be desirable to build a UI that fits within a Grid Services framework by including Grid Service interfaces.

13 Summary

The UI to a computational Grid should be implemented as a client side application which is a manifestation of the front-end to that Grid. All of a user's interactions with the back-end Grid should be facilitated through the UI. To this end, the user should be able to submit, monitor, and control single or multiple jobs, each of a single or multiple set of tasks. The UI should enable the user to specify dependencies between a set of tasks in one job and in turn to specify the dependencies between a set of jobs.

The user submitting a job to the Grid should be able to specify the location of the input and output files in a uniform way to the UI. In addition, the user should be able to specify a preference for the method used for the transfer of input and output data and other files associated with their jobs.

The creation of short lived security certificate proxies and their management for the duration of jobs that are dependent upon them, should be within the remit of the UI. It should be possible, therefore, for the UI to produce and store multiple proxies and to be able to extend their lifetimes. To this end, the UI must be able to associate a job with the certificate proxy instance which was used for its submission.

The UI should be able to select resources in its role as a high level hierarchical scheduler, and should be able to base this choice on cost models and tariffs of resources as well as other more traditional attributes such as a resources' OS. It is desirable that the UI undertake a form of accounting for the users' jobs submitted to or running on the Grid, in order that it may make more accurate scheduling decisions based on a user's available credit and priority ratings for fare-share.

14 Conclusion

In order to attract users to a wide area computational Grid, one must provide the tools necessary to facilitate the migration of jobs from conventional local and remote computational resources to that Grid. The interface to such a computational Grid is an area which requires addressing. A thick UI client tool with the set of functionalities addressed above, is prominent on the list of tools with which to facilitate use of a computational Grid. This UI should be the user's complete entry point to a computational Grid.