The Perl Commodity Grid Toolkit

Stephen Mock, Mary Thomas, Maytal Dahan, Kurt Mueller
University of California at San Diego, San Diego Supercomputer Center
{mock, mthomas, mdahan, kurt}@sdsc.edu
Gregor von Lazewski
Argnonne National Lab
gregor@mcs.anl.gov

Corresponding Author: Stephen Mock
San Diego Supercomputer Center, MC 0505
9500 Gilman Drive
La Jolla, CA 92093-0505
Telephone +1 858 534 8398
Fax +1 858 534 5117

Summary

The Perl Commodity Grid Toolkit (Perl CoG) is a software project aimed at bringing the complexities and power of the computational Grid to developers of Perl applications. In this paper we describe the history and motivation of the project, the benefits of bringing the grid to Perl developers, the architecture and status of the Perl CoG Kit, and the future plans for the project.

1. Introduction

The Computational Grid is the term applied to the infrastructure being constructed to coordinate the use of widely distributed computational resources for high end problem solving. Complex services are deployed to add functionality like authentication, remote access to resources, resource management, and directory services to the Grid. The development of the Grid has been focused on high-end computing, rather than user interface. Using these complex services provides challenges to users and developers of Grid software.

Web portals like the NPACI HotPage [1] simplify the use of Grid services by hiding the complexities of the Grid from portal users while giving them powerful tools accessible through an easy-to-use interface. Without a toolkit that provides interfaces to the Grid services, the developers of each new portal must master the complexities of the Grid in order to incorporate the Grid services into their software. The Commodity Grid project is working to overcome these difficulties by creating what are called Commodity Grid Toolkits (CoG Kits) [2]. "A Commodity Grid Toolkit (CoG Kit) defines and implements a set of general components that map Grid functionality into a commodity environment/framework." [3] The Commodity Grid project is developing CoG Kits in Java, Python, CORBA, and Perl. The Perl CoG Kit, like the others, provides an API to Grid services for developers of higher-level Grid applications.

The Perl CoG Kit is intended to be used by any developer who wants to use the Perl programming language, and has a need to use Grid services in the application that they are developing. In particular, web portals are a good example of applications that use grid services, although there are any number of other example applications which fall into this category.

The NPACI HotPage is a portal that utilizes the Grid to provide users with web access to computational resources at NPACI [4]. The HotPage began as a portal to give users information about resources across the NPACI organization. The need to add interactive features like file manipulation and job submission provided the motivation for creating software to help interface the portal and the Grid. The code that was developed to give HotPage access to Grid services, like job submission, was packaged into a toolkit called the Grid Portal Toolkit, or GridPort [5]. GridPort supports the development of web based Grid portals. It provides access to authentication, job submission and management, and file manipulation via a Perl library interface, as shown in figure 1. A number of portals, including the Telescience Portal [6], LAPK [7], and GAMESS [8] use GridPort. GridPort is a set of Perl libraries, rather than Perl modules, and it does not offer object-oriented interfaces to any of its code. Our experiences in building GridPort were very useful in creating the Perl CoG because the functionality that the two toolkits provide intersect over a great portion of their features.

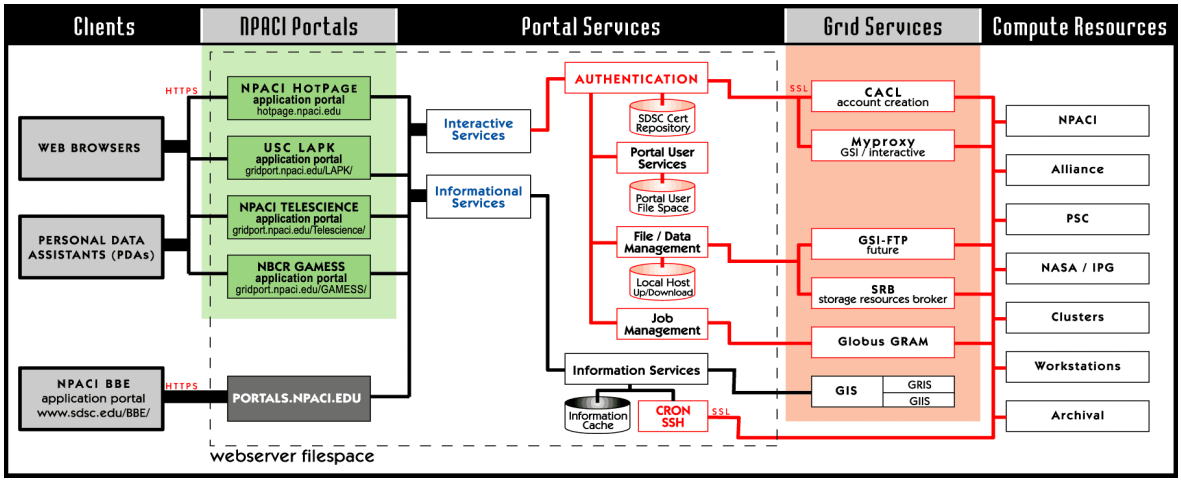

Figure 1. The diagram shows the layered architecture used by the portals based on the GridPort toolkit. The Perl CoG supplies the glue that binds the Portals to the Portal Services and the Grid Services.

2. Grid Computing with Perl

Perl is a stable, useful, and versatile programming language. It can be installed on almost any operating system, and most Unix and Unix-like operating systems come with Perl as part of the standard installation. Consequently, Perl is a highly portable language used for everything from system utilities to web applications. In particular, the power of Perl for parsing text has made it popular as a web scripting language and has given it widespread use by everyone from scientists to web programmers. The system administrator does not need to make extensions to a webserver to allow it to run Perl/CGI scripts. Perl can be object-oriented if a developer so chooses, depending on the complexity of the application being developed. If the programmer is developing a simple script, the overhead will often outweigh the benefits of object oriented programming. However, if the developer is writing a more complex application, the organizational benefits of object oriented programming often make an object oriented approach attractive. In this case, the developer can use the object-oriented features of Perl to keep the code clear and well organized. The popularity and strengths of Perl make it an excellent choice for marriage to grid technologies.

3. Requirements

We developed the Perl CoG Kit with Perl version 5.0 or higher to run on Unix and Unix-like systems. In order to install the Perl CoG, Perl version 5.0 or higher must be installed on the target system. The CoG Kit modules use a few other Perl modules, such as Net::LDAP.pm and Expect.pm to accomplish their tasks. The modules used by the Perl CoG Kit are either included with most Perl installations, or are available for free download from CPAN, the Comprehensive Perl Archive Network[9].

Much of the Grid services functionality of the Perl CoG Kit relies on services provided by Globus. The Perl CoG Kit uses its modules to contact the Globus gatekeeper on the remote machines in order to submit jobs, and it uses the Globus MDS to gather information. The Perl CoG Kit also uses the installed Globus [10] client utilities in some cases to make requests to the server side Globus daemons. Some of the modules are wrappers to the binary executables of the globus installation such as ‘globusrun’ and ‘globus-job-submit’. The CoG Kit also uses the installed GSI authentication utilities, like ‘grid-proxy-init’, that come with the Globus client tools to implement much of the authentication functionality. The Perl CoG uses these system utilities in cases where rewriting the same code in Perl would be difficult and time consuming. In order to quickly get a few of the core modules developed and working, the choice was made to develop some of the modules as wrappers to these utilities first, then evaluate making pure Perl implementations to alleviate the requirement of installing the Globus client utilities. The installation of the Globus client utilities may present some challenge, although the Globus team is making the installation easier with each update.

The installation scripts for some modules query the installer for information about the system, such as where the local Globus installation is located. The Perl CoG Kit installation follows the CPAN [9] conventions of providing a makefile and a test script for each of the modules. With the exception of the Globus client utilities requirement, the requirements for the Perl CoG are minimal, and the installation is simple.

4. Perl CoG Architecture

A layered approach that describes the architecture of the grid has been proposed [11]. This architechture consists of five layers:

- The fabric layer provides the resources to which shared access is mediated by Grid protocols. This layer provides interfaces to the local shared resources of entities on the grid.
- The connectivity layer defines the communication and authentication protocols required for network transactions and operations on the Grid.

- The resource layer builds on the fabric and connectivity layers to create communication and authentication protocols that facilitate the sharing of single resources.
- The collective layer contains the protocols and services that allow interaction with resources as collections rather than single resources.
- The application layer is the end user application, which uses the rest of the layers to become Grid enabled.

The Perl CoG Kit fits into this layered architecture by providing the glue between the application layer and the rest of the layers. The Perl CoG provides the developing application with the appropriate connectivity to the rest of the layers, as seen in figure 2.

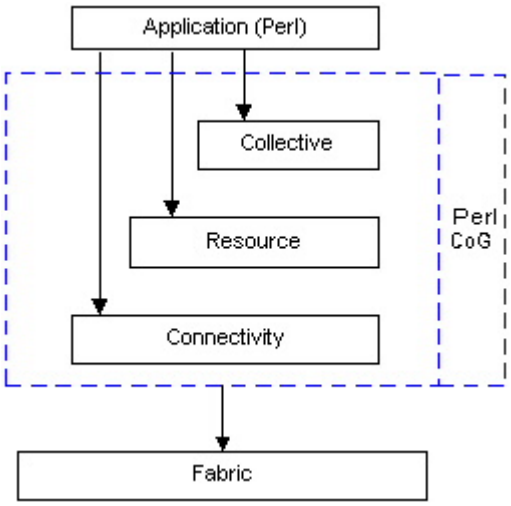

Figure 2: The Perl CoG provides the application with access to the rest of the layers of the Grid.

The initial list of modules to be created in the Perl CoG Kit arose from collaboration between the Perl CoG developers and members of the Globus Team. Modules were selected that would support the needs of the developers of the Perl CoG Kit, yet also serve the needs of the general developer community. In addition, modules that would provide a way to make a large impact quickly were developed first, so that the Perl CoG Kit could attract more users and generate feedback quickly. Additionally, we intend that this toolkit will contain input from other developers in the Perl community.

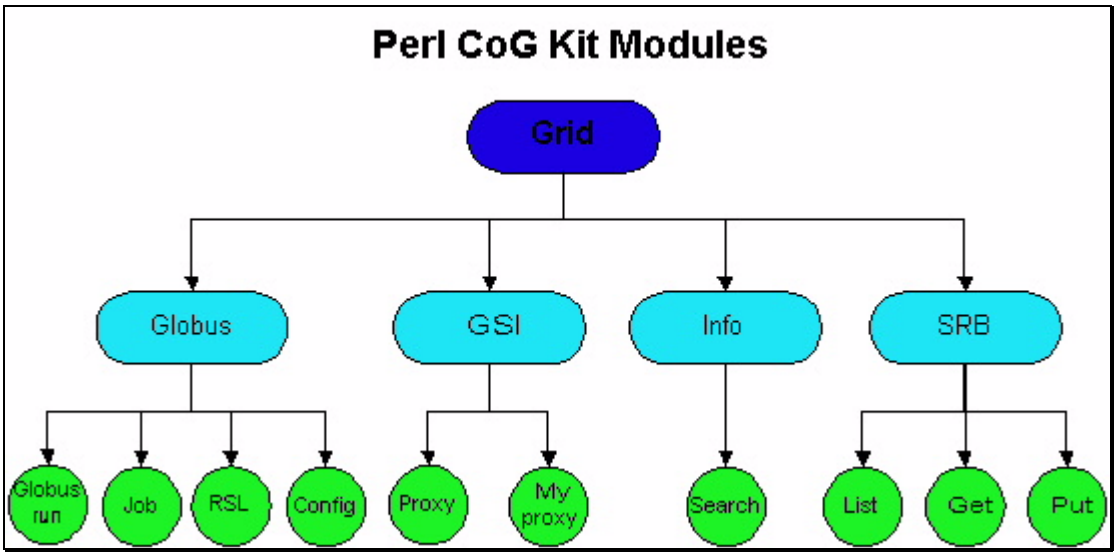

Figure 3. Perl CoG Module Architecture

The architecture of the Perl CoG Kit modules is shown in figure 2. The Perl CoG architecture was designed to be extensible. As the Grid community develops new technologies, new modules can be added at the middle layer alongside Globus, GSI, Info, and SRB. This architecture allows the Perl CoG kit to grow and mature with the grid community.

The Globus branch of the Perl CoG tree contains the modules related to the Globus toolkit which use Globus for job creation, submission, and manipulation. The GSI branch of the tree was put at the same level as the Globus branch, despite the fact that the Globus Team developed GSI. This decision was made because software packages other than Globus are beginning to adopt GSI as a mechanism for authentication. The Storage Resource Broker, for example, now supports GSI authentication. The Info branch is meant to contain the modules which facilitate information discovery on the Grid. Right now the only module for information discovery is the Search module, which performs MDS queries. A full description of each module and every subroutine is beyond the scope of this paper. For complete documentation, please refer to the Perl CoG Kit project page [12], or the Perl CoG distribution itself.

The ability for users to submit remote jobs using the Grid is very important. The Perl CoG Kit has modules for dealing with both interactive and batch jobs. The distinction between interactive and batch jobs deserves some explanation. Interactive jobs are simple jobs that the user expects to run immediately and either return a result or perform an action on the remote system. This allows applications to perform system-level tasks on remote resources in a secure and authenticated manner, such as manipulating files or running virtually any program that would be accessible if the user were directly logged into that remote resource. Batch jobs are those that are assembled, submitted, and then detached from. Batch jobs typically run for extended periods of time, requiring the submitting user or software agent to detach from it, instead of blocking while waiting for the job to complete like the interactive jobs. The batch jobs are submitted to a queuing system running on the remote resource, where it waits for the system to allocate to the job the required computational resources. Simulations that use large portions of high performance resources are typically run as batch jobs.

The interactive jobs are managed through the Globus::Globusrun module. It allows for the assembly and execution of interactive jobs via Globus GRAM [13], and returns the output of the interactive job. The Globusrun module is designed as a utility module, with no object-oriented features. The module was designed with the idea that the job would be a simple interactive job, which the software agent would run, and perhaps collect the output from, then forget. We felt that the simplicity of the task accomplished by the Globusrun module did not warrant the overhead that Perl incurs when using an object oriented approach. Handling batch jobs is done in the Perl CoG Kit with the Globus::GlobusJob module. The GlobusJob module is an object-oriented module, in which developers create job objects, and then can set or get the parameters of the job object. When the job has been assembled, it can be submitted. The module provides mechanisms for checking the status of the job, canceling the job, and getting the output of the job. Both the Globusrun and Job modules depend on installed Globus client utilities to implement their functionality. The Perl CoG implements a clean simple interface to job submission through Globus.

The Resource Specification Language (RSL) is the language the Globus software uses as a common language to specify job requests [14]. The Perl CoG contains an RSL module for that provides an interface to create, manipulate, and verify RSL strings. This module provides an object-oriented interface to the creation of RSL strings, making it possible for users to assemble jobs based on RSL without ever directly having to manipulate the RSL string. The RSL module contains subroutines to create and delete RSL objects, add and delete attributes from the RSL objects, and to output RSL strings from the RSL objects. The user of the RSL module needs to know only the attributes and values of the desired RSL string. The RSL module will assemble the string for them, as well as verify the validity of the attribute that they wish to assign a value. The RSL module is pure Perl, meaning that it does not depend on any outside, non-Perl utilities or libraries to install and operate correctly. The RSL module works in conjunction with the the Globusrun and GlobusJob modules to submit jobs to remote computational resources on the grid as seen in Figures 4 and 5.

```
use Grid::Globus::Globusrun;
use Grid::Globus::RSL;
```

```
###Create an RSL string
my $myRSL = Grid::Globus::RSL->new();
$myRSL->addAttribute('count','1');
$myRSL->addAttribute('executable','/bin/ls');
$myRSL->addAttribute('arguments','-l');

#assemble and run job
my @output = globusrun(args=>"-s -r",
                       host=>"tfglobus.sdsc.edu",
                       rsl=>$myRSL->toString(),
                       timeout=>"25");
foreach(@output){
  print;
}
```

Figure 4: A simple example of using the Perl CoG Kit's Globus::Globusrun and Globus::RSL modules which shows assembly of an RSL string describing a job, submission of the job, and collection the output from the job for printing.

```
use Grid::Globus::GlobusJob;
use Grid::Globus::RSL;

###Create an RSL string
my $myRSL = Grid::Globus::RSL->new();
$myRSL->addAttribute('queue','express');
$myRSL->addAttribute('maxwalltime,'20');
…
my $job = Grid::Globus::GlobusJob->new(contact => 'tfglobus.sdsc.edu',
                       rsl  => $ myRSL->toString()
                       );
$job->submit();
if($fatalerror) {
  $job->cancel();
}
$stat = $job->get_status();
my @stdout = $job->get_stdout();
my @stderr = $job->get_stderr();
```

Figure 5: A simple pseudocode example of using the Perl CoG Kit's Globus::GlobusJob and Globus::RSL modules which shows assembly of an RSL string describing a job, submission of the job, job status discovery, and collection the stderr and stdout from the job.

The Config module found in the Globus branch is a module for configuring the CoG interface to the local Globus installation on the system. The installation of the Config module attempts to locate the Globus installation, or prompts the installer for help. This module provides the other modules that use Globus system tools with a simple and uniform access to those installed Globus utilities. It also provides an easy mechanism for the CoG maintainer to switch the installation of Globus that the CoG uses. The Config module is primarily a convenience mechanism for the developers and maintainers of the Perl CoG.

Grid Security Infrastructure (GSI) is emerging as a dominant authentication and security mechanism [15]. The Perl CoG supplies modules for managing GSI credentials through the GSI::Proxy and GSI::Myproxy modules. The Proxy module depends on some of the installed GSI system utilities that are included in the Globus client software installation; thus it is not pure Perl. The Proxy module facilitates the

generation of proxy certificates from an X.509 certificate and key pair. The creation of this proxy allows delegation of authority to access remote Grid services. The Proxy module also provides the ability to destroy a proxy, and to get information about a previously existing proxy file such as the subject, strength, type, issuer, and time to live of the delegated credential. In addition to the Proxy module, the Perl CoG Kit supplies the GSI::Myproxy module to support the use of Myproxy[16] for managing GSI credentials. The Myproxy module has subroutines to insert a credential into a Myproxy server, to get a delegated credential from a Myproxy server, and to delete a credential from a Myproxy server. The Myproxy module depends on the installed Myproxy system utilities as well as the Expect Perl module. These two GSI modules were so constructed to allow the addition of more GSI authenticated grid services to be added to the Perl CoG Kit as the grid grows. The Proxy and Myproxy modules facilitate GSI authentication to grid services within a Perl application. Figures 6 and 7 show the simple interface that the Perl CoG Kit provides to GSI authentication.

```
use Grid::GSI::Proxy;
my $results = proxyInit($passphrase, $certificate ,$key, $proxyfile, $hours ,$limited,
                        $bits, $cert_dir);
```

Figure 6: Example of using the GSI::Proxy module to create a proxy certificate from an X.509 certificate and key.

```
use Grid::GSI::Myproxy;
$myp = Grid::GSI::Myproxy->new();
$myp->setMyproxyInit("$username", "$x509_passphrase", "$myproxy_passphrase",
                        "$server_name");
$myp->myproxyInit();

$myp->setMyproxyDelegation("$username", "$myproxy_passphrase", "$server_name");
$myp->myproxyGetDelegation();
```

Figure 7: Example of using the GSI::Myproxy module to store a proxy in a Myproxy server, then retrieving a delegated proxy from that Myproxy server.

Information discovery is handled in the Perl CoG through the Info::Search Perl module. The Info::Search module is an object-oriented module which facilitates the assembly and execution of queries against the Globus MDS [17] to access information about the resources on the Grid. The MDS is a server that is accessed using LDAP[18] and contains data in a standardized hierarchical structure of data objects. The developer creates a search object by specifying the MDS host and port and setting its parameters, and then executing a search. The search object contains the entries returned, which can be parsed and used to make decisions about further Grid activity. For example, a search could be performed to find a computer with the needed available compute resources for a particular job. The Search module depends on the two Perl modules, the Net::LDAP module, which it uses to connect and make queries to the MDS, and the LDAP::Entry module, which describes the data structure that the results from a search are returned in. The Search module is a pure Perl implementation; the only two requirements for it are Perl modules. The Perl modules it requires are freely available from the CPAN website. [9] A code example of using the Info::Search module to query the MDS is shown in Figure 8.

```
use Grid::Info::Search;

my $timeout = 120;
my $host    = ‘mds.sdsc.edu';
my $port    = 2000;
```

```
my $branch  = "o=Grid";
my $filter  = "(objectclass=GlobusLoadInformation)";

my $gis = Grid::Info::Search->new(
                                  host=>"$host",
                                  port=>"$port",
                                  filter=>"$filter",
                                  branch=>"$branch",
                                  timeout=>"$timeout"
                                 );
my @entries = $gis->execute();
my $entry;
foreach $entry (@entries) {
  my @atts = $entry->attributes;
  print "\n\n";
  foreach(@atts) { print "$_="; my $val = $entry->get_value($_); print "$val\n"; }
}
```

Figure 8: A simple example of using the Perl CoG Kit's Info::Search module to perform an MDS query, then print out the key/value data retrieved from the query.

The SRB [19] module is in development at this time, and will be available on the Perl CoG project page when complete [12].

5. Module Creation

The skeleton of each of the modules was created using a utility from the standard Perl installation called h2xs. The h2xs utility builds a Perl extension from any C header file, but also has the ability to simply put together the framework for building a new module. This includes providing a stub module file and testscript, creating a makefile generator, and installing all the necessary files and directories needed to create a module complete with manpages and documentation. The testscript has dual functionality: it supplies the installer with a mechanism for checking that the installation worked correctly, and it gives the developers a working example of how to use the module. So in addition to the Perldoc documentation that comes with each module, the developers can refer to the testscripts for additional examples.

6. Project Status and Future Plans

The Perl CoG Kit is still in the early stages of development. A small subset of the target functionality has been implemented already, though some of the modules already completed are being redesigned. The core functionality of the CoG Kit is in place. Modules for job assembly, batch job submission, interactive job submission, authentication, and information discovery have been completed and tested. The Perl CoG source code is available from the Perl CoG project web page found at http://gridport.npaci.edu/cog.

Since the Perl CoG is a new addition to the CoG Project it has many more modules to be developed. Modules for other Grid services are either being developed now, or will be developed in the future. We are working on modules to interface with the Storage Resource Broker (SRB) [19] for distributed file collection access, as well as modules for transferring files between remote systems using gsiftp. Some of the modules that require system utilities like the Globus utilities are under examination to determine if they can be implemented as pure Perl, the result of which could be a pure Perl implementation of GRAM. This benefit of taking this approach is that the CoG could be a replacement for the utilities that it provides wrappers for, alleviating the necessity to install software toolkits like the Globus client utilities. Since Perl is highly portable, a pure Perl implementation could lead to Grid access from new platforms.

The GridPort toolkit is in the process of being rewritten using the Perl CoG, making GridPort a cleaner and more portable package. The GridPort toolkit itself will include modules for doing

authentication and running jobs that are particularly specialized for use with web portals based on the modules of the Perl CoG.

The Perl CoG team is currently evaluating the use of SWIG [20], the Simple Wrapper and Interface Generator, to aid in the development of core Globus functionality like GRAM. The Python CoG Kit has demonstrated successful use of SWIG to interface directly with the Globus C code. This approach allows reuse of the Globus Team's code, while providing the Python language with access to the functionality contained in the Globus code. This provides a good starting point to enable another language, like Python or Perl, access to the software that Globus has already developed. SWIG has the ability to provide this sort of interface in several languages, including Python and Perl.

7. Conclusion

The Perl CoG brings the Grid to the Perl community, enabling more people and applications to become aware of and use Grid services. The potential usefulness of the Perl CoG extends beyond the web portal community: it enables the integration of Grid services into command line and system utilities that are written in Perl. The number of web-based portals under development is enormous, and many of them have a need for Grid services. Perl's easy integration into web-based applications makes it a great candidate for use in making these new portals Grid-aware, and the Perl CoG can make the process significantly easier.

8. References

[1] M. Thomas, S. Mock. The NPACI HotPage User Portal. https://hotpage.npaci.edu.

[2] Commodity Grid Kits Project Web Page. http://www.globus.org/cog.

[3] Gregor von Laszewski, Ian Foster, Jarek Gawor. CoG Kits: A Bridge between Commodity Distributed Computing and High-Performance Grids. http://www-unix.mcs.anl.gov/~laszewsk/papers/cog-final.pdf

[4] National Partnership for Advanced Computational Infrastructure (NPACI). Project Web Page. http://www.npaci.edu.

[5] M. Thomas, S. Mock, M. Dahan, K. Mueller, D. Sutton, J. Boisseau. The GridPort Toolkit: a System for Building Grid Portals. *10th IEEE Symp. on High Performance Distributed Computing.* Aug. 2001.

[6] The Telescience Web Portal. https://gridport.npaci.edu/Telescience/.

[7] The Laboratory of Applied Pharmacokinetics Web Portal. https://gridport.npaci.edu/LAPK.

[8] The General Atomic Molecular Electronic Structure Systems (GAMESS) Web Portal. https://gridport.npaci.edu/GAMESS.

[9] The Comprehensive Perl Archive Network. http://www.cpan.org.

[10] The Globus Project Web Page. http://www.globus.org.

[11] I. Foster, C. Kesselman, S. Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *Intl. J. Supercomputer Applications*, 15(3), 2001.

[12] The Perl CoG Kit Project Web Page.
https://gridport.npaci.edu/cog.

[13] The Grid Resource Allocation Manager.
http://www.globus.org/gram/.

[14] The Globus Resource Specification Language.
http://www.globus.org/gram/rsl.html.

[15] I. Foster, C. Kesselman, G. Tsudik, S. Tuecke. A Security Architecture for Computational Grids. *Proc. 5th ACM Conference on Computer and Communications Security Conference*, pg. 83-92, 1998.

[16] Myproxy (v 1.0).
http://dast.nlanr.net/Projects/MyProxy.

[17] The Metacomputing Directory Service.
http://www.globus.org/mds/.

[18] T.A. Howes and M. Smith. 1995. A scalable, deployable directory service framework for the internet. Techincal report, Center for Information Techology Integration, Univeristy of Michigan.

[19] The Storage Resource Broker.
http://www.npaci.edu/Thrusts/DI/SRB/.

[20] SWIG, Simplified Wrapper and Interface Generator.
http://www.swig.org.