Simplifying the Utilization of Grid Computation using Grid Wizard Enterprise

Release 1.00

Marco Ruiz[1], Neil C. Jones[2] and Jeffrey S. Grethe[1]

May 19, 2008

[1]Center for Research in Biological Systems, University of California at San Diego, La Jolla, CA
[2]Computer Science Department, University of California at San Diego, La Jolla, CA

Abstract

The field of high performance computing (HPC) has provided a wide array of strategies for supplying additional computing power to the goal of reducing the total "clock time" required to complete large scale analyses. These strategies range from the development of higher performance hardware to the assembly of large networks of commodity computers. However, for the non-computational scientist wishing to utilize these services, usable software remains elusive. Here we present a software design and implementation of a tool, Grid Wizard Enterprise (GWE; http://www.gridwizardenterprise.org/), aimed at providing a solution to the particular problem of the adoption of advanced grid technologies by biomedical researchers. GWE provides an intuitive environment and tools that bridge this gulf between the researcher and current grid technologies allowing them to run inter-independent computational processes faster by brokering their execution across a virtual grid of computational resources with a minimum of user intervention. The GWE architecture has been designed in close collaboration with biomedical researchers and supports the majority of every-day tasks performed by computational scientists in the fields of computational biology and medical image analysis.

Contents

1 Introduction

Research in the computational sciences seems to proceed, roughly speaking, in distinct phases. As an initial idea for a computational protocol is refined, a researcher will iteratively debug and improve applications on a small amount of handpicked data, tweaking parameters and inspecting output for correctness. Once the idea returns plausible results and can be considered publication-worthy, the program gets run on more and more data, often involving a new round of debugging as new classes of pathological inputs are discovered. During this phase, the computational protocol undergoes systematic characterization: for what parameter ranges does it return valid results? Can any immediate conclusions or further hypotheses be derived from running on publicly available data? Finally, once the computational protocol is ready to be released to the scientific community, a researcher must decide how to distribute the code. Increasingly, providing a publicly accessible website is an attractive alternative to making code available for download: releasing new improvements or bug-fixes to the code is substantially easier, and only the user interface requires documentation.

It is a lucky coincidence that the research process – including the hosting of an algorithm in a web environment – itself falls into the already useful class of embarrassingly parallel problems, that is to say, resource-intensive computational problems that can be broken into independent subunits that can run in parallel. A researcher who faces such a problem must deal with a number of tedious issues: how to determine what work needs to be done and how it should be broken into meaningful units (workload definition); how to assign individual work units to resources (application scheduling[1,3]); how to run and monitor executables (grid execution; e.g. Globus[2]); and how to deal with system-related and program-related failures (failure detection). Though each of these problems can be solved in a straightforward way, the combined solutions to all of them leads to a maintenance problem, interferes with distribution, and presents an additional barrier to a scientist trying to investigate a particular problem. It is our view that a researcher expert in a particular scientific discipline should not need to also become an expert in grid computing in order to produce an application that uses grid technology. It is also our observation that the bulk of computational scientists do not have at their disposal dedicated programmers and system administrators to plan, install, configure, and maintain a complex heterogeneous network of computers.

To reflect these needs we have designed a system, Grid Wizard Enterprise (GWE) that facilitates the above considerations, which we derived based partly on our own experiences performing computational science in bioinformatics and partly on observing others doing the same. It is important to note that GWE is not meant to be another grid middleware package, rather, it is meant to be a large-scale job launching and management tool that bridges the gulf between the biomedical researcher and current grid middleware by:

- Providing the researcher with the ability to easily configure the heterogeneous clustered/grid resources that they have access to.
- Allowing a researcher to easily specify large parametric computational jobs using the same general syntax as is used in the command line invocation of the analysis algorithms (e.g. see P2EL in Section 4) or through integration with community developed biomedical applications (e.g. see Slicer in Section 5).
- Managing the most common house keeping tasks required to ensure end-to-end success of a computation thereby relieving the researcher of this burden.

2 Processes Parallelization Problem

With current application scheduling frameworks (e.g. Condor[1], SGE[3]) it would appear that the end user has an easy to use platform to broker the execution of his processes, however, unless his processes are extremely straightforward and trivial, the end user faces a daunting challenge. Most real application scheduling requests require the resolution of issues that, even in the presence of a powerful resource manager, have to be resolved by the end user:

1. Uploading the data to be processed to the cluster (localization)
2. Submit all processes to compute nodes (queue jobs in resource managers).
3. Monitor processes execution progress (real time and querying on demand).
4. Send / receive custom alert notifications (certain interesting conditions reached such as a percentage of processes completed execution).
5. Failover and recovery from cluster and environment related problems.
6. Failover and recovery from processes related problems.
7. Gathering and compilation of processing results.
8. Uploading result data to the storage resource of your choice.
9. Cleaning up the original and result data from the cluster data storage resource.

In addition, clusters are, typically, shared multi-user organizational resources and cannot be monopolized; which means that every single one of the previous issues become more complicated to overcome; and new ones (such as coordination of user data workspaces, access to and utilization of multiple heterogeneous cluster environments, etc) expand the list. This solution is far from straightforward and requires a great deal of expertise from most end users to consider it practical.

Two different solutions have evolved, as a way to bridge this gap: users gathering a considerable level of technical knowledge, which takes time and effort away from their actual domain problems; and the creation, (by technically savvy users and/or IT departments), of highly customized scripts and applications tailored to specific parallelization problems. Both these avenues don't provide the broader community of biomedical researchers with the ability to easily harness the growing number of clustered computational resources available to them.

3 Grid Wizard Enterprise (GWE)

GWE is a distributed enterprise system (Figure 1) that was designed to be a practical solution for end users to easily and effectively parallelize and broker the execution of their inter-independent processes on clustered or grid environments they have access to. This system provides a solution for the issues previously mentioned in Section 2 and with a high degree of modularity which allows third parties to extend and customize the GWE system. In order to lower the barrier for use by the typical biomedical researcher, the only requirements the system has over the environment it would be running on are to have available java 1.5 or higher and operate over a SSH enabled network.

3.1 Distributed System

From a user's perspective, GWE is composed of the following subsystems:

a) GWE Client - System running on end users machines to communicate with a 'GWE Grid' in order to query the execution status of previously submitted requests and submit new ones. This client can be access through the command line or integrated within a biomedical application.

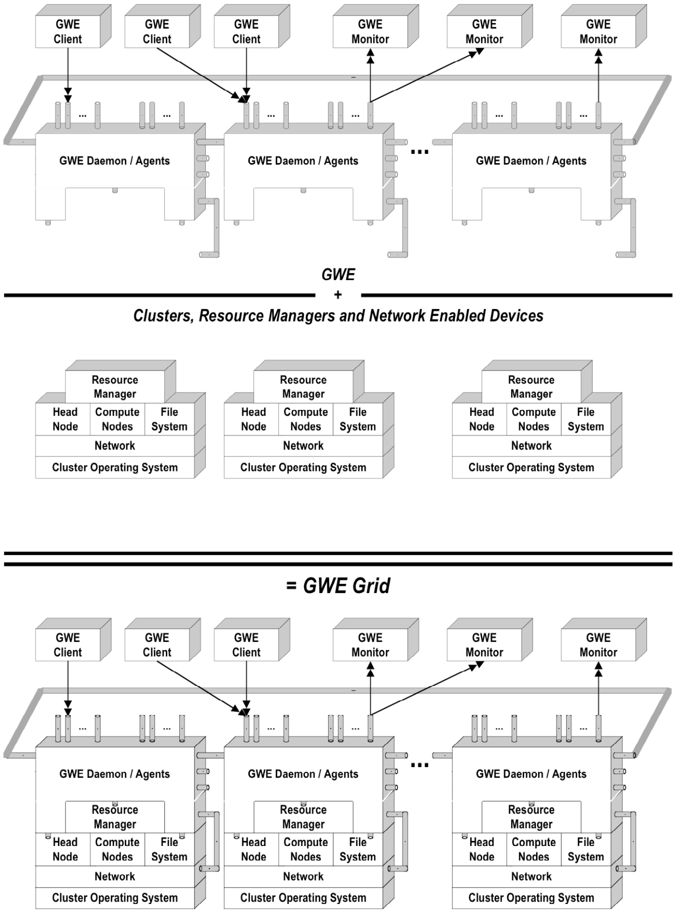


Figure 1: GWE Grid

b) GWE Daemon - System(s) running on clusters' head node to serve, on one end, as a listener for end user requests, on another end, as a job dispatcher and monitor for the respective cluster. In addition, GWE daemons can communicate with one another when a user has requested a computation to be executed on multiple clusters.

Typically, end users would connect to a particular ‘GWE daemon’ (running on a host on a reachable TCP/IP based network) using a ‘GWE client’. A GWE client’s configuration consists of: the list of clusters that compose the user’s accessible grid resources, SSH authentication information of all the networked resources to be accessed (clusters, file systems, etc) and locations of applications to auto-deploy.

GWE daemons are easily deployed through an automated process in the GWE client distribution and can be installed by any user with a valid SSH account on a cluster head node. Such an instance can be installed by the cluster’s administrator using a dedicated cluster account or by individual users setting up their own dedicated GWE system. At runtime, GWE daemons will silently spawn low level ‘agent type’ sub-systems, which are scheduled to run on the compute nodes (one ‘agent’ per allocated compute node) by the local cluster’s resource manager. These ‘agents’ will be in charge of the actual execution of the processes, monitoring status/progress/results and reporting back to the respective “daemon”.

GWE daemons’ configuration consists of multiple behavioral parameters. Among the most important ones are the ones used for its ‘compute node allocation policy’, such as queue size (maximum number of compute nodes allocated at any given time), “hijack” timeout (maximum time a compute node can remain allocated with active jobs before “releasing” its controlling agent) and “idle” timeout (maximum time a compute node can remain allocated with nothing to do before “releasing” its controlling agent).

3.2 Architecture & Design

All GWE subsystems have been architected as a set of independent modules and frameworks, glued declaratively using the ‘Spring’ application framework. Such modules have been designed with a robust and scalable infrastructure and with multiple levels of abstraction to provide a high degree of extensibility.

One of the most common extensible modules is the “GWE Client API” (Figure 2); used to build GWE client applications or empower applications with GWE client capabilities. Currently there are a few GWE client applications built using this API and they will be reviewed in more detail later in this paper. Internally this API contains many extensible sub-modules most of which have been further developed to reduce the effort required to add/change functionality. An example of this is the “abstract job descriptor” component; which can be extended to support languages other than P2EL, or even workflows with inter-dependent jobs.

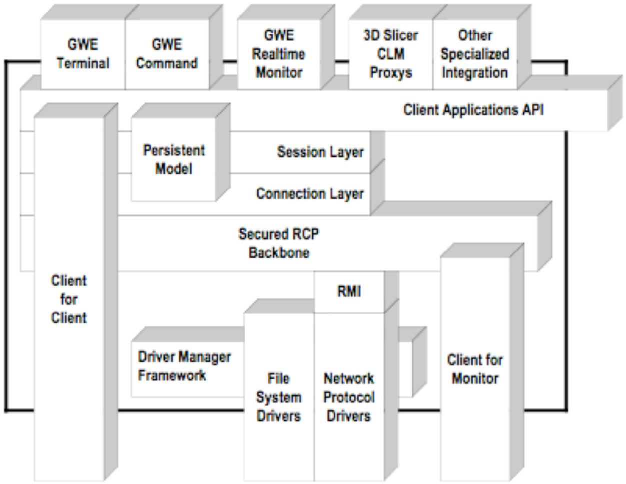


Figure 2: GWE Client Architecture

Finally, one set of extensible modules, which deserves a special mention are the ‘grid related resource drivers’. These drivers provide GWE with means to support new types of file systems, network protocols and cluster resource managers (which are auto detected per cluster when a GWE daemon is deployed). GWE comes out of the box with drivers to support the following ‘resources’:

- File Systems: Local, HTTP and SFTP.
- Network Protocols: Local and SSH.
- Resource Managers: Condor, SGE and PBS.

GWE daemons (Figure 3), include other robust enterprise level features. Some of the most important are: sandbox workspaces and virtual file systems per user to provide a scalable multi-user system; highly multithreaded services to avoid wait cycles and achieve maximum performance; embedded database to persist operational data (users, orders, jobs, clusters, etc) and automatic failover and recovery from cluster, environment and process specific related problems.

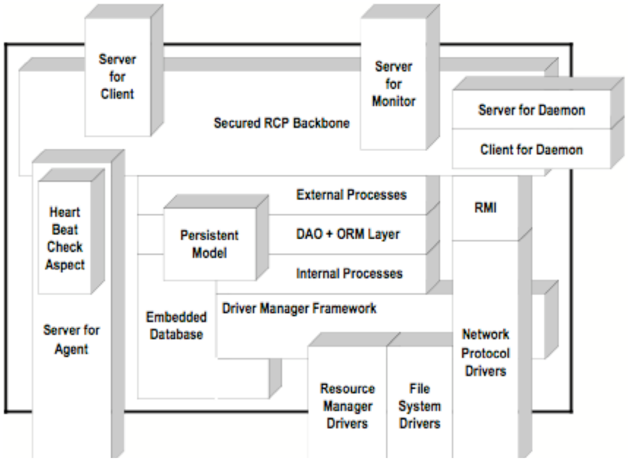


Figure 3: GWE Daemon Architecture

However, the GWE system was designed to allow any biomedical researcher to easily access their available compute resources with tools they normally utilize: the command line, SSH and domain specific applications. Therefore, the GWE inter subsystem communications infrastructure has been architected as a secured RPC backbone using a Java RMI network tunneled over SSH. This tunneling infrastructure consists of a framework, which transparently injects a series of hooks (socket proxy, heartbeat emitters and heartbeat checkers) into the communications using interceptors and AOP (aspect oriented programming). Besides providing a secured communication infrastructure this approach takes away the requirement for users or cluster administrators to open additional IP ports in clusters firewalls.

4 Usage and Integration with External Applications

4.1 GWE Client Applications

The GWE Client API, is a java API that allows developers to build highly customized GWE clients (see Slicer3 integration section). However, out of the box, GWE comes with two generic GWE client implementations:

a) GWE Terminal (Figure 4) is a console application; which keeps a live connection to a particular GWE daemon and allows the user to interactively query status information and submit requests. This application will remain alive and connected to the GWE daemon until an 'exit ' or 'quit ' command is issued. This application includes rich command line features such as inline editing, tab completion and command history. This application is ideal when the user is going to interact for a while with a GWE daemon.

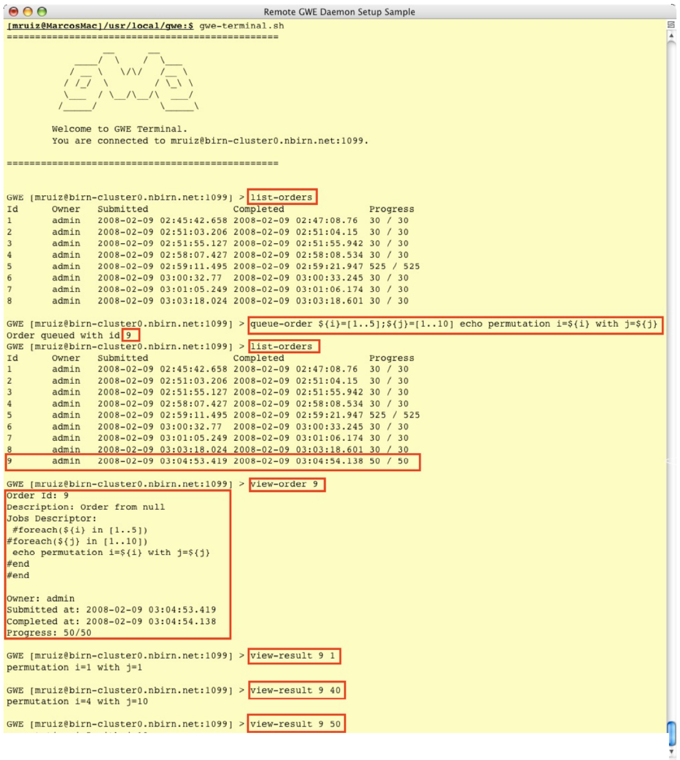
a) GWE Commands are Java command line applications; each meant to give the end user the capabilities to access a particular daemon. Usage of these applications is slightly more expensive than using the GWE Terminal since a brand new network connection has to be established every time they are invoked. However, they provide quick command line access to a GWE daemon and a basic API for integrating GWE requests programmatically.

This API provides a great number of functions; which can be categorized as execution request functions, query status functions and monitoring functions. While the first two categories of functions are initiated by end user communication, monitoring is requested by the user and initiated by the daemon as real time notification events.

4.1.1 P2EL (Process Parallelization Execution Language)

In order to provide the semantics for users to easily and effectively describe a group of process invocations including all related parallelization meta-instructions, a simple but powerful language (P2EL) has been designed for GWE (and an appropriate interpreter built into it). This language is a combination of pseudo bash and pseudo VLT (Velocity Template Language). This language has semantics to define:

- **Process Invocation Template** is a Bash like, meta-template containing iteration variables references (substitution expressions). This template will be used to generate all the process invocations of the respective P2EL statement.
- **Substitution Expressions** are "bash like" variable expressions embedded in the template to be replaced by the corresponding value-space.
- **Iteration Variables** are variables associated with a value-space set, which when applied to a process invocation template, generate a collection of processes invocations. This construct gets its values explicitly or implicitly through numerous ways including runtime value resolving functions.
- **System Variables** are variables associated with a system and/or contextual property resolved at runtime for a specific job (e.g. the iteration number generated by GWE and user home).
- **Script Loading.** Construct to instruct that the process invocation template shall be read from a specific file.
- **Staging Files Instructions and Locations**. Construct to instruct how to stage remote files into the context of process invocation (before executing it) and how to stage files produced by the process invocation out to selected destination locations.



```
[mruiz@MarcosMac]/usr/local/gwe:$ gwe-terminal.sh
================================================
              __      __
         ____/  \    /  \____
        /  __  \  \/\/  /__  \
       / /_/  \        / \_\ \
       \____  / \__/\__/\  ____/
      /_____/            \_____\

        Welcome to GWE Terminal.
        You are connected to mruiz@birn-cluster0.nbirn.net:1099.

================================================


GWE [mruiz@birn-cluster0.nbirn.net:1099] > list-orders
Id      Owner   Submitted               Completed               Progress
1       admin   2008-02-09 02:45:42.658 2008-02-09 02:47:08.76  30 / 30
2       admin   2008-02-09 02:51:03.206 2008-02-09 02:51:04.15  30 / 30
3       admin   2008-02-09 02:51:55.127 2008-02-09 02:51:55.942 30 / 30
4       admin   2008-02-09 02:58:07.427 2008-02-09 02:58:08.534 30 / 30
5       admin   2008-02-09 02:59:11.495 2008-02-09 02:59:21.947 525 / 525
6       admin   2008-02-09 03:00:32.77  2008-02-09 03:00:33.245 30 / 30
7       admin   2008-02-09 03:01:05.249 2008-02-09 03:01:06.174 30 / 30
8       admin   2008-02-09 03:03:18.024 2008-02-09 03:03:18.601 30 / 30

GWE [mruiz@birn-cluster0.nbirn.net:1099] > queue-order ${i}=[1..5];${j}=[1..10] echo permutation i=${i} with j=${j}
Order queued with id 9
GWE [mruiz@birn-cluster0.nbirn.net:1099] > list-orders
Id      Owner   Submitted               Completed               Progress
1       admin   2008-02-09 02:45:42.658 2008-02-09 02:47:08.76  30 / 30
2       admin   2008-02-09 02:51:03.206 2008-02-09 02:51:04.15  30 / 30
3       admin   2008-02-09 02:51:55.127 2008-02-09 02:51:55.942 30 / 30
4       admin   2008-02-09 02:58:07.427 2008-02-09 02:58:08.534 30 / 30
5       admin   2008-02-09 02:59:11.495 2008-02-09 02:59:21.947 525 / 525
6       admin   2008-02-09 03:00:32.77  2008-02-09 03:00:33.245 30 / 30
7       admin   2008-02-09 03:01:05.249 2008-02-09 03:01:06.174 30 / 30
8       admin   2008-02-09 03:03:18.024 2008-02-09 03:03:18.601 30 / 30
9       admin   2008-02-09 03:04:53.419 2008-02-09 03:04:54.138 50 / 50

GWE [mruiz@birn-cluster0.nbirn.net:1099] > view-order 9
Order Id: 9
Description: Order from null
Jobs Descriptor:
 #foreach(${i} in [1..5])
#foreach(${j} in [1..10])
 echo permutation i=${i} with j=${j}
#end
#end

Owner: admin
Submitted at: 2008-02-09 03:04:53.419
Completed at: 2008-02-09 03:04:54.138
Progress: 50/50

GWE [mruiz@birn-cluster0.nbirn.net:1099] > view-result 9 1
permutation i=1 with j=1

GWE [mruiz@birn-cluster0.nbirn.net:1099] > view-result 9 40
permutation i=4 with j=10

GWE [mruiz@birn-cluster0.nbirn.net:1099] > view-result 9 50
```

Figure 4: GWE Terminal

The following P2EL command illustrates a real use case example used in medical imaging to submit a large scale parameter exploration of an analysis algorithm which can result in 1000s of actual analyses. This command instructs GWE to run Slicer's "BSpline Deformable Registration" for each moving image found that matches the wildcard pattern 'sftp://srcHost/srcDir/moving-*.nrrd' against the fixed image 'http://otherSrcHost/otherSrcDir/fixed.nrrd?view=co', for each possible combination of values for: iterations (ITER = 10, 20, 30, 40 and 50), histograms (HIST = 020, 040, 060, 080 and 100) and samples (SAMP = 0500, 1500, 2500, 3500 and 4500). The output is saved under the remote directory 'sftp://destHost/destDir/' with the name 'out-[ITER_ID].nrrd', where '[ITER_ID]' is the unique auto-generated identifier of the job. It is important to note that the P2EL command line has the same general form as the original Slicer command (in italics) with the addition of the specification of the iteration variables and substitution expressions.

```
Slicer3 --launch /usr/Slicer3/lib/Slicer3/Plugins/BSplineDeformableRegistration
--iterations 10 --gridSize 5 --histogrambins 20 --spatialsamples 500
--maximumDeformation 1 --default 0
--resampledmovingfilename out.nrrd f.nrrd m.nrrd

queue-order  ${MOV}=f:expand(sftp://srcHost/srcDir/moving-*.nrrd)  ${ITER}=[10..50||10]
${HIST}=[20..100||020] ${SAMP}=[500..5000||1000]
Slicer3 --launch /usr/Slicer3/lib/Slicer3/Plugins/BSplineDeformableRegistration
--iterations ${ITER} --gridSize 5 --histogrambins ${HIST} --spatialsamples ${SAMP}
--maximumDeformation 1 --default 0
--resampledmovingfilename f:out(sftp://destHost/destDir/out-f:sys(ITER_ID).nrrd)
f:in(f.nrrd,http://otherSrcHost/otherSrcDir/fixed.nrrd?view=co) f:in(m.nrrd,${MOVING})
```

4.2 GWE-Slicer3 Integration

4.2.1 Slicer3

The National Alliance for Medical Image Computation's (NA-MIC) Slicer3[4] (http://slicer.org/) is a "free, open source software package for visualization and image analysis. Slicer's capabilities include: interactive visualization of images, manual editing, fusion and co-registering of data, automatic segmentation, analysis of diffuse tensor imaging data, and visualization of tracking information for image-guided procedures. Some of the core functionality that enables these applications include the capability to save and restore scenes using a format called MRML, a plug-in architecture to interface to external programs including ITK, a sophisticated statistical classification environment based on the EM algorithm, capabilities for rigid and non-rigid data fusion and registration, and processing of DTI MRI data."[5]

4.2.2 Objective

Researchers often encounter the need to run medical imaging algorithms over a large set of values to be permutated for different arguments (see P2EL example above), from algorithm calibration parameters to sets of images (i.e. sets of images utilized for testing an algorithm to large collections of images collected as part of a study that are ready to be processed). Executing all these processes on a single computer may take a really long time depending on the amount of processes and the number of file transfers. In addition, the gathering of results requires a lot of manual work and is highly prone to error.

Slicer3, provides an infrastructure to easily integrate medical imaging applications ('modules') as self-describing pluggable components. Taking advantage of this feature, GWE can be generically integrated with Slicer3 to allow researchers to execute their set of processes in parallel across distributed grid environments while handling all the 'side' tasks that otherwise would have to be done manually. This way, Slicer3 provides the means for users to dramatically reduce their processing time by facilitating a transparent "run-everywhere" philosophy for algorithm developers.

4.2.3 Design

Slicer3 modules execute as regular command line applications, which must conform to a straightforward, proprietary specification. This specification requires these modules to "self describe" when invoked with the predefined reactor argument of '--xml' which results in the generation of a proprietary XML descriptor stating the module's arguments metadata (flags, types of values, label, etc). Slicer3 uses this metadata to dynamically render an appropriate UI to collect the values for each argument and to generate the module command line invocation when needed to run the module per a user's request.

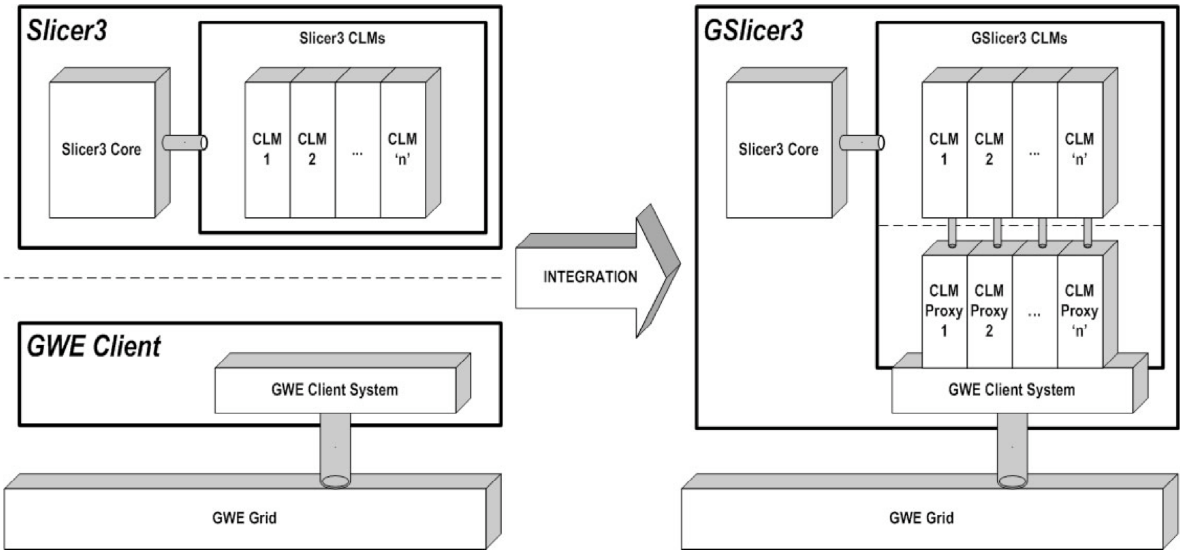


Figure 5: Slicer3 /GSlicer3 Architecture Comparison

The integration effort consists of installing the 'GWE Client API' inside a Slicer3 installation and generating new Slicer3 modules (Figure 5), one for each 'regular' Slicer3 module intended to be leveraged with grid computing capabilities. These new modules are called "GWE CLMPs" (command line module proxies) and the user can utilize them when trying to run a set of processes in a distributed grid environment, otherwise they can work as usual with the 'regular' versions. This effort includes a 'bundling' utility which installs GWE inside a Slicer3 distribution, introspects it to discover all its pluggable modules and dynamically generates a corresponding GWE CLMP for each of them. The end result is the grid-enabled version of Slicer3, which we call GSlicer3.

When GSlicer3 is launched, the end user will notice that (in the available modules menu) for every regular module, there is another named exactly the same with the additional suffix of " – GWE Powered" (Figure 6). These correspond to the CLMPs modules that have been auto generated by the 'bundling' utility.

GWE CLMPs are intelligent agents, which conform to the Slicer3 module specification responding as required to the predefined reactors and have an explicit association to their corresponding 'regular' version module. The proxied module creates their "self description" by retrieving the "self description" of their associated 'regular' version module and enhancing them with the appropriate grid computing related descriptions. Using these dynamically generated "self descriptions", Slicer3 is able to render a suitable UI (Figure 7) to capture the necessary arguments to execute invocations of the associated module on the grid. Also, such UIs support P2EL semantics so value added functions (file transfers, wildcard resolution, etc) are all available.

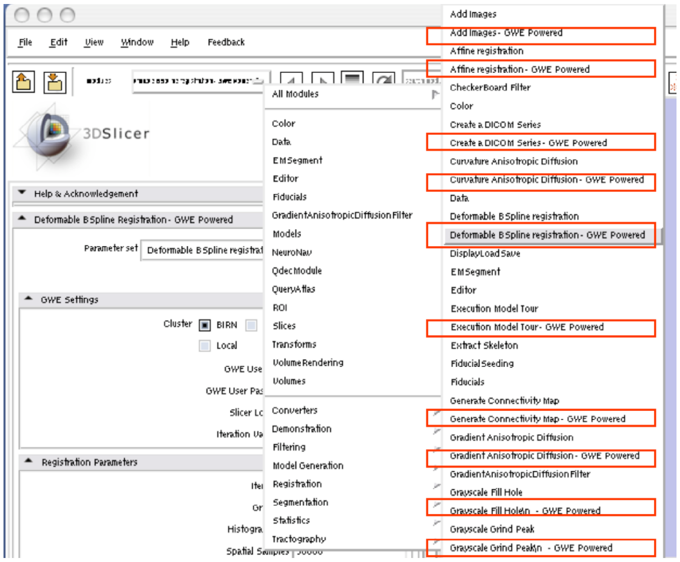
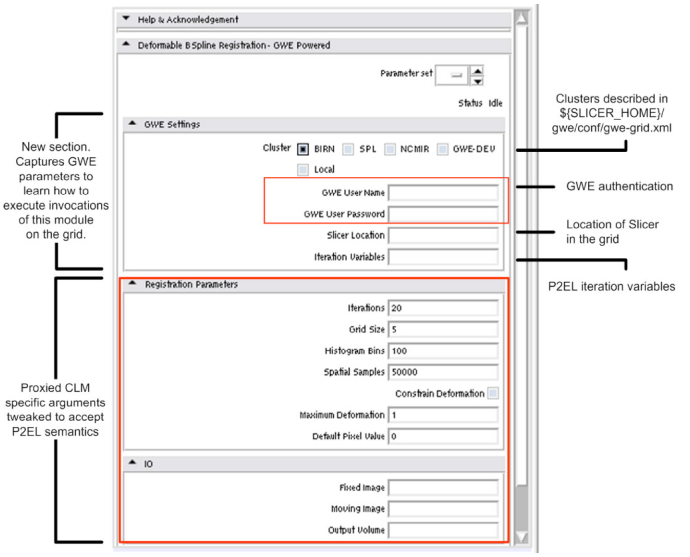


Figure 6: Grid enabled modules

A CLMP in essence is a “GWE client application” which at runtime will carry on the following tasks:

- Retrieve, add and modify the XML tags of their proxied CLM's XML descriptor in order to add arguments/fields to set/capture specific GWE parameters and P2EL value types.
- Generate a GWE order with the P2EL command corresponding to the user’s input appropriately translated to the selected cluster (Slicer3 location on the cluster, user home directory, etc).
- Install a customized Slicer result parser to the GWE order.
- Submit the GWE order created to the remote selected cluster, over the secured RPC GWE network.
- Monitor, in real time, the execution progress of the localized proxied CLM invocations (from the GWE order) on the selected cluster. This real time monitoring is also performed over the secured RPC GWE network.
- Keep track of the CLMP progress as a ratio of the number of proxied CLMs invocations already executed divided by the total amount of proxied CLMs invocations associated with the GWE order submitted.
- Notify Slicer3 of this progress using Slicer3 XML based progress API (<filter-XXX > tags sent to the standard output of the CLMP).



Status Idle

Cluster ☒ BIRN ☐ SPL ☐ NCMIR ☐ GWE-DEV

☐ Local

GWE User Name

GWE User Password

Slicer Location

Iteration Variables

Iterations 20

Grid Size 5

Histogram Bins 100

Spatial Samples 50000

Constrain Deformation ☐

Maximum Deformation 1

Default Pixel Value 0

Fixed Image

Moving Image

Output Volume

Figure 7: Grid enabled module UI

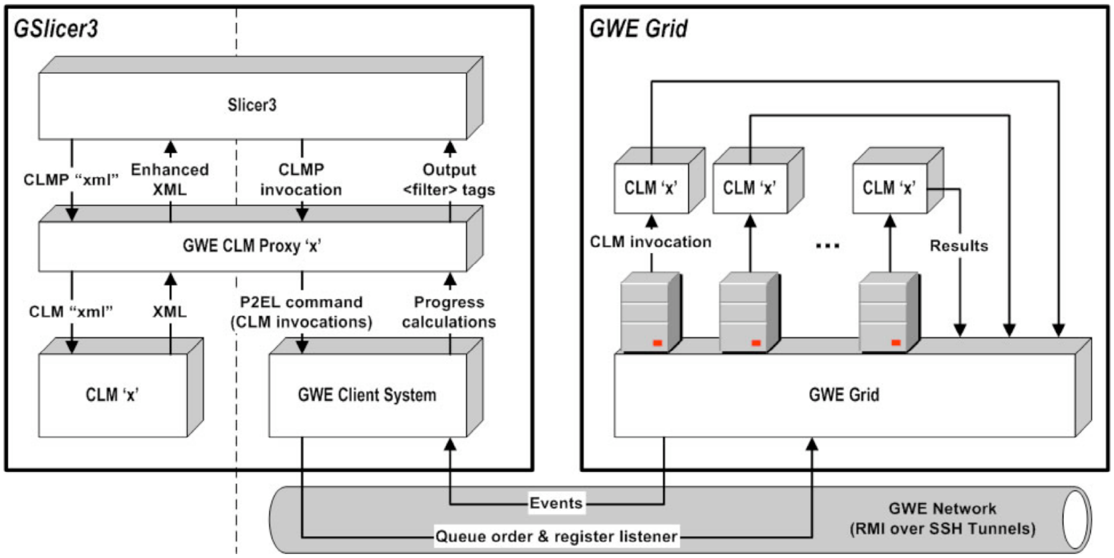


Figure 8: GSlicer3 Execution Flow

This integration effort provides a generalized, easy to use grid computing enabled interface to all Slicer3 CLMs that are "Standard Execution Model" compliant.

5 Future Work

This system is currently in its 4th alpha release. The current project plan is to release a new version every 6 weeks and release the first feature complete beta version in the beginning of 2009. In order to reach the beta release, the product is going through extended testing with biomedical researchers to elicit usability and functional requirements. As a result of this testing the following feature set is being finalized (a-d), implemented (e-h) and tested:

a. Application registry framework - provides capabilities to auto-deploy applications to running clusters on an as needed basis. Currently the Slicer3 integration described above assumes that Slicer3 is installed on the running clusters.
b. Multi-cluster module - provides true grid abstraction through the ability of GWE to distribute jobs to a specified set of clusters in a "daisy chain" configuration.
c. Array of iteration variables feature - provides the means to specify a set of values-sets to apply all at once as a single variable. For example: VAL_SET=[(0,10,20),(15,27,-3)] would create 1 permutation with the first set of 3 values and a second permutation with the second set.
d. Parametrical order behavioral logic - provides the means for end users to customize execution aspects of the jobs associated with an order, such as, job timeouts, launch mode, file system clean up policy, maximum concurrent jobs running, etc.
e. Alert notification module - provides the means for end users to specify job runtime conditions under which the system shall send notifications to customizable recipients (typically an email to a specified email address).
f. Job result parser framework - provides an infrastructure for the end user to create his/her own result parser to inject into a specific order so it can extract meaningful structured data out of the job results.
g. Additional grid related drivers for file system drivers (i.e. SRB, XCEDE based XML file catalogs, and GridFTP) and resource managers (e.g. Torque).
h. Portal client integration as JSR-168 portlets that can be deployed to any standards based portal container.

References

1. J. Epema, M. Livny, R. Dantzig, X. Evers, and J. Pruyne. A worldwide flock of condors. Journal on Future Generations of Compute Systems , 12, 1996.

2. I. Foster and C. Kesselman. Globus: A Toolkit-Based Grid Architecture, pages 259–78. Morgan Kaufmann, 1999.

3. W. Gentzsch. Sun grid engine: Towards creating a compute power grid. Cluster Computing and the Grid, 00:35, 2001.

4. S. Pieper, B. Lorensen, W. Schroeder, and R. Kikinis. The NA-MIC Kit: ITK, VTK, Pipelines, Grids and 3D Slicer as an Open Platform for the Medical Image Computing Community, Proc IEEE Intl Symp on Biomedical Imaging 2006; 1:698-701.

5. http://slicer.org/pages/Introduction, May 2008