A solution looking for lots of problems: Generic Portals for Science Infrastructure

Thomas D. Uram, Michael E. Papka, Mark Hereld, Michael Wilde
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439, USA
{turam,hereld,papka,wilde}@mcs.anl.gov

ABSTRACT
Science gateways have dramatically simplified the work required by science communities to run their codes on TeraGrid resources. Gateway development typically spans the duration of a particular grant, with the first production runs occurring some months after the award and concluding near the end of the project. Scientists use gateways as a means to interface with large resources. Our gateway infrastructure facilitates this by hiding away the various details of the underlying resources and presents an intuitive way to interact with the resource. In this paper, we present our work on GPSI, a general-purpose science gateway infrastructure that can be easily customized to meet the needs of an application. This reduces the time to deployment and improves scientific productivity. Our contribution in this paper is two-fold: to elaborate our vision for a user-driven gateway infrastructure that includes components required by multiple science domains, thus aiding the speedy development of gateways, and presenting our experience in moving from our initial portal implementations to the current effort based on Python [15] and Django [16].

Categories and Subject Descriptors
H.3.5 **[Online Information Services]:** Web-based services

General Terms
Design, Experimentation,

Keywords
Science Gateway, Django, Python, Swift, Web2.0, Workflow

1. INTRODUCTION
Despite the many reusable technologies available for developing science gateways, portal development is still typically a one-off integrative development exercise, which is costly and time-consuming for new research collaborations. In this paper, we analyze technology choices made in our past portal development efforts to identify common patterns and pitfalls, and describe our effort to build a framework for generic portals for science infrastructure (GPSI).

Science gateways have been proven as a tremendously valuable tool to bridge the gap between computational scientists and their available compute resources [1]. Based on our work with several gateways, we have identified components and patterns that are common across gateways in several science domains. Common components are used to manage data, metadata, compute resources, credentials, job submission and monitoring, and browse simulation outputs. The primary distinguishing factor among gateways is the specific applications that are used to process the data and view the results. GPSI exploits this scenario by combining usable, familiar common components with support for users to define applications and output representations.

As compute infrastructures and data sizes grow larger, we need more sophisticated tools for managing data, running and monitoring simulations, tracking data and execution provenance, analyzing and visualizing results, and collaborating with colleagues. Furthermore, these tools should be intuitive and use common interaction patterns, so scientists can begin running simulations in their gateways as close to the start of their projects as possible. We have sought to make GPSI a tool that will help scientists without getting in their way, so they don't despair at having to learn yet another system to do their work.

2. BACKGROUND AND MOTIVATION
The GPSI effort is derived from our earlier work on several portals, described here briefly.

With the TeraGrid Visualization Portal (TGViz) portal [2], users could visualize their data using a ParaView server running on TeraGrid compute resources, viewing the rendered images with a ParaView client on their desktop. This required tracking job execution so that when it became active and the ParaView server had started, the portal could inform the user of the host and port against which to run their local ParaView client. The portal was developed on uPortal [19] and then GridSphere [20], using GRAM to submit jobs to the TeraGrid. This effort successfully enabled remote visualization using ParaView, but clarified the need for remote file browsing and facilities for introducing arbitrary remote applications.

In the Social Informatics Data Grid project [3], we developed a data warehouse and computational science portal for social and behavioral scientists to upload their large data stores and scale their multi-modal analyses to TeraGrid compute resources. As is typical, individual scientists had large caches of data and proprietary algorithms that they wanted to share with the other members of their team. Also, privacy concerns were significant as the data was largely derived from human studies. We worked closely with the scientists to integrate their applications into the portal and design their output representations. SIDGrid was built using PHP/Drupal and Python for the web frontend, and used Swift [7] for job execution. The SIDGrid portal demonstrated useful practices in data management (such as tagging and faceted search), web services (via SOAP), and desktop integration. SIDGrid could have benefited from the use of a web framework and object relational management (ORM) system, and a more tightly integrated collection of components.

The Open Life Science Gateway [4] used Java and OGCE to expose bioinformatics applications in a portal interface. In OLSGW, we developed support for defining application execution descriptions and integrating them into the portal with a generated web interface for execution.

The Open Social scientific application framework [5] was a Java-based effort to build a portal using OpenSocial gadgets as web components and Swift as the execution engine. In this work we established the importance of letting users define applications in the portal, as this allows them to improve their simulations without the aid of a gateway developer and analyze their results more quickly. This gateway also used OpenSocial gadgets, allowing users to arrange them to construct their own working environments in the web browser, and to use portions of the portal in third party sites like iGoogle. In applying this portal to protein folding simulations being run by the OOPS group at the University of Chicago, we developed support for users to build custom representations of their simulation outputs, another key user functionality.

From these efforts, we identified several functional components that appeared repeatedly in the various gateways. Users obviously required control over how their data is imported, how it is identified, and how they search and organize it. While portals are typically built around a static collection of resources, users often have multiple compute resources available to them and should be able to control where their jobs are run. The user also will clearly need to control how those jobs are run, including which application is used and the parameters and files used to run it, and various parameters germane to the job submission itself. Output processing is a hybrid of the general and the particular: the handling of many outputs can be anticipated, such as displaying an image on the output webpage, while others are more complex and domain-specific and must be controlled by the scientists (for example, how a molecular model might be displayed).

The infrastructure we envision is for computational scientists what WordPress is for bloggers. In GPSI, we have encoded much of what is required by a science gateway and support click-through interfaces for scientists to adapt it to their needs. Following simple interfaces to integrate their applications with the portal, scientists can describe their compute resources, run jobs, and view basic representations of their results. They can easily customize these representations according to the idioms of their particular science domain. Further, as with any gateway, GPSI can be customized by developers to meet the more complex requirements of science domains.

TECHNOLOGY CHOICES

2.1 Principles

The following principles guided our choices in designing GPSI. They are based on lessons learned in our earlier work on several portals in diverse science domains, and the findings of a recent study of TeraGrid Gateway developers [6].

Enforce sound software development practices

Choosing technologies that enforce good development practices will produce better quality code, which aids initial development as well as maintenance and customization efforts.

Adopt proven, well-supported dependencies

By choosing stable dependencies, the gateway will be stable and will receive bug fixes, enhancements, and security patches applied in future updates to the dependencies.

Emphasize intuitive interfaces

The gateway will be deployed to a diverse set of users with varying technical proficiency. Striving to make the user interface intuitive will help users be more self-sufficient and productive, reducing the support burden on the gateway developers.

Leverage development by the broader community

Projects with an active community are likely to produce reusable components that can be adopted off-the-shelf for future gateway enhancements.

Large developer community

Good developers are hard to find, and gateway projects suffer when they lose developers midstream. Choosing technologies that have a wide developer base increases the availability of professional developers.

2.2 Technologies

A comparison of the technologies used in earlier portal efforts and in GPSI is given in Table 1. In GPSI, we have chosen the following technologies.

Python/Django

GPSI benefits from the adoption of Python/Django in several respects, which we compare with earlier technology choices here.

- Python, being an interpreted language, enables developers to modify the live code to add minor functionality or fix errors, without requiring a compile/deploy step. Language preference is always subjective; we chose Python because its scripting nature allows us to modify code quickly, and due to its rich standard library and enforcement of good coding practices.
- Django employs and enforces a model-view-controller (MVC) methodology, a common development pattern for separating presentation from application logic. This approach encourages good development practices and produces more maintainable code. MVC web frameworks also exist in Java. Our choice here was largely constrained by our choice of Python; nonetheless, Django is a mature framework that has since been emulated by other web frameworks, notably the Java-based Play.
- Django templates provide a presentation layer consistent across webpages, making the user interface usable and attractive. We have used templates in other systems, and defined styles in CSS to present unified styles, but the Django template system is well documented and easy to use, and will be familiar to other developers that come to the system.
- Django is well documented, making the entire gateway easier to maintain than our original Java portal, which was coded from the ground up. As with the templating system, the

existence of comprehensive documentation makes the GPSI application easier for gateway developers to maintain.

- Many standard and third party modules are available for Django. A few examples include support for OpenID, Facebook Connect, and user ratings. Many web frameworks and rich internet application toolkits are now available and achieve a similar level of pluggability.
- Django is the most widely used Python web framework, with a very well established community of developers. Pylons and Pyjamas are alternatives, but Django is the most common and we judge the most likely to continue into the future.

Swift

For describing and executing workflows, we have chosen the Swift language and execution engine. Swift provides both a parallel scripting language for developing data-intensive workflow applications, and an execution engine for dispatching jobs to local and remote compute resources. Three advantages of supporting Swift in the framework include: (1) the Swift language blends a familiar C-like syntax with functional programming characteristics, which is designed to expose opportunities for parallel execution; (2) the Swift language allows researchers to easily link their scripts into a data-driven computational workflow; and (3) the Swift workflow engine is very lightweight for integration into a science gateway.

JQuery

JQuery [17] is a widely used JavaScript library that fulfills our principles in many respects, primary among them being the usability that it brings to the application. Further, JQuery is used in many stable web applications and is supported by a large community with many active developers.

XML/XSL

GPSI can identify the outputs of a simulation, but in cases where the researchers want to represent a constrained subset of the outputs, they required a mechanism for describing it. We have chosen to use a simple XML format that would be trivial for their simulation codes to produce, but would be sufficiently descriptive for us to process. In addition, many researchers are already familiar with basic XML syntax, so this choice is rather natural for them.

For transforming the XML output description to HTML, we have chosen to use XSL [12]. XSL is admittedly less familiar to scientists, but is the de facto standard for transforming XML documents. Also, GPSI includes a default XSL stylesheet which will be used in most cases, so only in special cases will scientists be exposed to XSL syntax.

3. PORTAL DESIGN

In this section, we describe the functional aspects of GPSI, and give examples of each. These examples are drawn on data and jobs from TeraGrid resources and from PADS [9], a compute cluster at the University of Chicago.

3.1 Data Management

Effective data management is essential to the ability of simulations to produce meaningful results, and will become more important with the growth of compute resources, and simulations, and resulting datasets. Users begin with large datasets already and must migrate them to compute resources before beginning their simulations. Simulations are routinely producing terabytes to petabytes of data, and this data must be managed in ways that maximize its utility to the researchers.

Table 1 Comparison of technologies in our earlier portals and in GPSI

	Earlier Portals	**GPSI portal**
Language	Java, JSP, Python, PHP	Python
Web Framework	None	Django
ORM	Hibernate	Django
Database	MySQL	multiple*
JS library	JQuery	JQuery
Application generation	PISE, Mobyle, StringTemplates	Django
Execution	VDS, Swift	Swift
Output generation	XSL	XSL

* Django includes support for many databases

Users introduce their datasets to GPSI in the same manner they would any other site, through simple uploads. In the case of large imports, the user can provide a compressed archive either by upload or by URL, and the portal will decompress the archive and add the files to the user's account. Users can share their portal files with other users, granting or revoking access at their whim. We have found that the ability to flexibly share files is essential amid the flux typical in research teams.

The file view in Figure 1 is a virtual table populated by calls to the Django file services by DataTables [18], an AJAX table control used throughout the GPSI site. The performance of this control is key to the ability of GPSI to quickly deliver views of large server-side data; paging or searching through thousands of files is done with very low latency. Files can be arbitrarily tagged, with support from the django-tagging module. Files can be searched by filename, date, and tags.

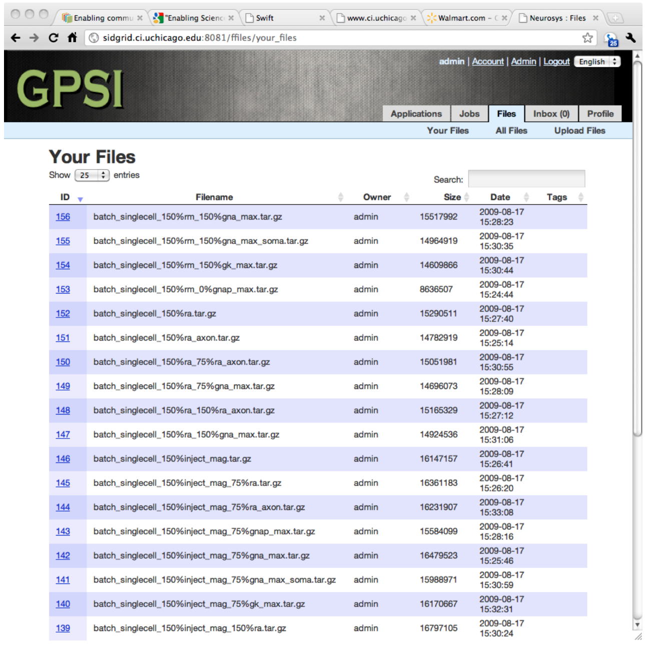
Individual files are presented in GPSI with associated history, metadata, tags, and a link to the process by which they were produced, as shown in **Figure 2**. The file can be previewed in the portal or downloaded to the user's computer.

3.2 Compute Resource Management

While TeraGrid gateways typically allow users to run their jobs on TeraGrid resources, we recognized in GPSI that users often have multiple compute resources available to them, including campus resources or other large, distributed grids. Users should be the arbiters of where their jobs run, selecting from their unique pool of resources.

GPSI users can manage their available compute resources, adding any resources for which they have a valid credential. Each resource definition includes the hostname and type of the job submission endpoint, and the hostname and type of the data transfer service. These are used to create Swift [7] configuration files used to drive data staging and job submission.

As part of the definition of a resource, a credential type is specified. At the time of job submission, the user is required to provide a suitable credential. Currently supported credential types include secure shell (ssh) for the local PADS cluster, and MyProxy [10] credentials for TeraGrid resources.



Your Files

Show 25 entries

Search:

ID	Filename	Owner	Size	Date	Tags
156	batch_singlecell_150%rm_150%gna_max.tar.gz	admin	15517992	2009-08-17 15:28:23	
155	batch_singlecell_150%rm_150%gna_max_soma.tar.gz	admin	14964919	2009-08-17 15:30:35	
154	batch_singlecell_150%rm_150%gk_max.tar.gz	admin	14609866	2009-08-17 15:30:44	
153	batch_singlecell_150%rm_0%gnap_max.tar.gz	admin	8636507	2009-08-17 15:24:44	
152	batch_singlecell_150%ra.tar.gz	admin	15290511	2009-08-17 15:27:40	
151	batch_singlecell_150%ra_axon.tar.gz	admin	14782919	2009-08-17 15:25:14	
150	batch_singlecell_150%ra_75%ra_axon.tar.gz	admin	15051981	2009-08-17 15:30:55	
149	batch_singlecell_150%ra_75%gna_max.tar.gz	admin	14696073	2009-08-17 15:28:09	
148	batch_singlecell_150%ra_150%ra_axon.tar.gz	admin	15165329	2009-08-17 15:27:12	
147	batch_singlecell_150%ra_150%gna_max.tar.gz	admin	14924536	2009-08-17 15:31:06	
146	batch_singlecell_150%inject_mag.tar.gz	admin	16147157	2009-08-17 15:26:41	
145	batch_singlecell_150%inject_mag_75%ra.tar.gz	admin	16361183	2009-08-17 15:26:20	
144	batch_singlecell_150%inject_mag_75%ra_axon.tar.gz	admin	16231907	2009-08-17 15:33:08	
143	batch_singlecell_150%inject_mag_75%gnap_max.tar.gz	admin	15584099	2009-08-17 15:28:16	
142	batch_singlecell_150%inject_mag_75%gna_max.tar.gz	admin	16479523	2009-08-17 15:25:46	
141	batch_singlecell_150%inject_mag_75%gna_max_soma.tar.gz	admin	15988971	2009-08-17 15:30:59	
140	batch_singlecell_150%inject_mag_75%gk_max.tar.gz	admin	16170667	2009-08-17 15:32:31	
139	batch_singlecell_150%inject_mag_150%ra.tar.gz	admin	16797105	2009-08-17 15:30:24	

Figure 1 A table view of a user's files. The AJAX table control fetches only the visible files, or the results of file searches, to provide a quick view into large data collections.

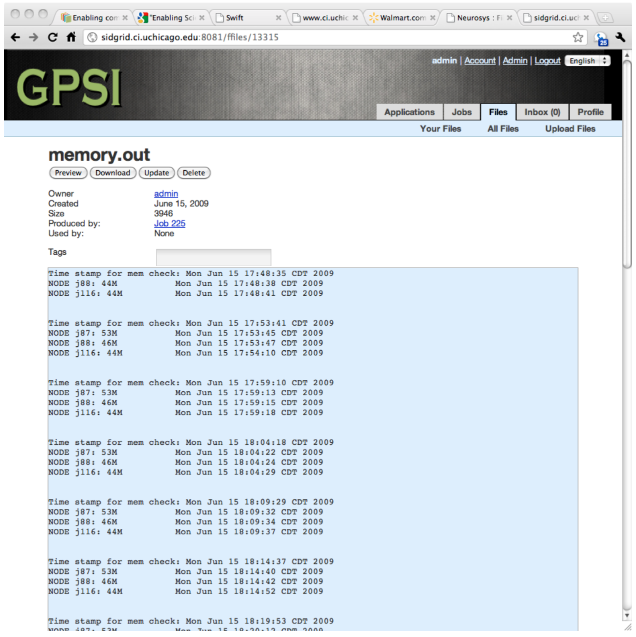
3.3 Application Management

While many aspects of gateways are common—they all must manage data and execute jobs--applications are the key differentiator between science gateways. Scientists understand their computational tools best, as they have typically run them on their local resources many times. Users should be able to describe their application, including the parameters and default values used to run it, and have the portal produce a web interface for specifying those parameter values and running the job. Taking this approach, a GPSI installation transitions from a generic portal to a domain- or scientist-specific portal.

Applications are introduced to GPSI either as simple command line or as Swift scripts. This flexibility allows scientists to begin running their command line applications in the portal immediately, on multiple compute resources. GPSI wraps command lines with Swift code internally for execution and, because of this, Swift's support for concurrency is also exposed in the portal. For example, given a command line and a set of execution parameters, the user can specify a range of values for one or more of the input parameters, and Swift will submit a job for each of the unique parameter combinations, blocking as necessary on data dependencies. This support takes users quickly from execution of a single instance of their code to large ensemble runs.

Alternatively, users can define an application by uploading a Swift script and any required supporting files. This scenario is applicable for users with existing Swift scripts, and those requiring more fine-grained control over their workflow definitions. In this case, GPSI parses the uploaded Swift script to determine the execution parameters and their default values, and stores these in the database with the application definition.

In some cases, the user will continue to maintain their code in their existing external repository, and would like to update the portal instance so the new code is executed for future runs. GPSI



memory.out

Owner	admin
Created	June 15, 2009
Size	3946
Produced by:	Job 225
Used by:	None
Tags	

```
Time stamp for mem check: Mon Jun 15 17:48:35 CDT 2009
NODE j88: 44M          Mon Jun 15 17:48:38 CDT 2009
NODE j116: 44M         Mon Jun 15 17:48:41 CDT 2009


Time stamp for mem check: Mon Jun 15 17:53:41 CDT 2009
NODE j87: 53M          Mon Jun 15 17:53:45 CDT 2009
NODE j88: 46M          Mon Jun 15 17:53:47 CDT 2009
NODE j116: 44M         Mon Jun 15 17:54:10 CDT 2009


Time stamp for mem check: Mon Jun 15 17:59:10 CDT 2009
NODE j87: 53M          Mon Jun 15 17:59:13 CDT 2009
NODE j88: 46M          Mon Jun 15 17:59:15 CDT 2009
NODE j116: 44M         Mon Jun 15 17:59:18 CDT 2009


Time stamp for mem check: Mon Jun 15 18:04:18 CDT 2009
NODE j87: 53M          Mon Jun 15 18:04:22 CDT 2009
NODE j88: 46M          Mon Jun 15 18:04:24 CDT 2009
NODE j116: 44M         Mon Jun 15 18:04:29 CDT 2009


Time stamp for mem check: Mon Jun 15 18:09:29 CDT 2009
NODE j87: 53M          Mon Jun 15 18:09:32 CDT 2009
NODE j88: 46M          Mon Jun 15 18:09:34 CDT 2009
NODE j116: 44M         Mon Jun 15 18:09:37 CDT 2009


Time stamp for mem check: Mon Jun 15 18:14:37 CDT 2009
NODE j87: 53M          Mon Jun 15 18:14:40 CDT 2009
NODE j88: 46M          Mon Jun 15 18:14:42 CDT 2009
NODE j116: 44M         Mon Jun 15 18:14:52 CDT 2009


Time stamp for mem check: Mon Jun 15 18:19:53 CDT 2009
```

Figure 2 View of an output file in the GPSI gateway. Files can be tagged by users, previewed in the web page, and downloaded.

includes support for users to define applications based on external repositories, in which case the application definition includes the relevant repository location details; taking Git [14] as an example, the additional information would consist of the Git repository URL and a username and password. GPSI also provides source code repositories for users who wish to maintain their code in a new repository hosted by the portal.

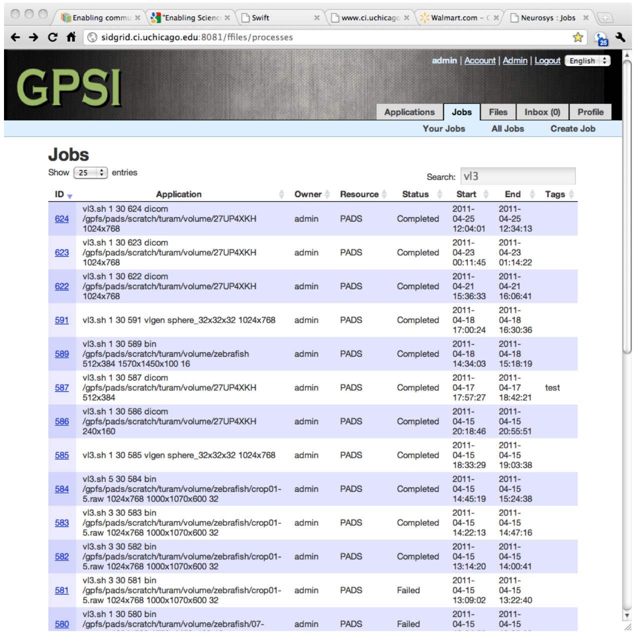
A user can share applications with other users or make them public. Available applications can be tagged and searched by tag to facilitate lookups.

3.4 Job Management

Job management is the core functional component of science gateways, handling the specification, execution, and monitoring of compute jobs. To run a job, the gateway collects information about how the job should be run (e.g. compute resource, number of nodes, queue, duration) and which application to run, including parameter values and input files. Users may have any number of jobs running on various sites at a given time, and a long history of past jobs. These data also become part of the data management scenario, as researchers need to track past jobs and their inputs and results, and do so quickly.

When submitting a job, GPSI renders the input form for the selected application for the user to specify applications parameters and execution parameters. The application parameters are derived from the application definition, and can be either a string or a file. If the input is a string argument, the user may specify a single value. If the input is a file argument, the user can select the file using a remote file browser or begin typing a filename to use the built-in autocomplete functionality, which queries the user's portal-resident files.

After selecting a compute resource and specifying a credential and accompanying job submission parameters, the user can submit the job. Upon submission, the job is recorded in the database with related input parameters and files.



Jobs

Show 25 entries

Search: vl3

ID	Application	Owner	Resource	Status	Start	End	Tags
624	vl3.sh 1 30 624 dicom /gpfs/pads/scratch/turam/volume/27UP4XKH 1024x768	admin	PADS	Completed	2011-04-25 12:04:01	2011-04-25 12:34:13	
623	vl3.sh 1 30 623 dicom /gpfs/pads/scratch/turam/volume/27UP4XKH 1024x768	admin	PADS	Completed	2011-04-23 00:11:45	2011-04-23 01:14:22	
622	vl3.sh 1 30 622 dicom /gpfs/pads/scratch/turam/volume/27UP4XKH 1024x768	admin	PADS	Completed	2011-04-21 15:36:33	2011-04-21 16:06:41	
591	vl3.sh 1 30 591 vlgen sphere_32x32x32 1024x768	admin	PADS	Completed	2011-04-18 17:00:24	2011-04-18 16:30:36	
589	vl3.sh 1 30 589 bin /gpfs/pads/scratch/turam/volume/zebrafish 512x384 1570x1450x100 16	admin	PADS	Completed	2011-04-18 14:34:03	2011-04-18 15:18:19	
587	vl3.sh 1 30 587 dicom /gpfs/pads/scratch/turam/volume/27UP4XKH 512x384	admin	PADS	Completed	2011-04-17 17:57:27	2011-04-17 18:42:21	test
586	vl3.sh 1 30 586 dicom /gpfs/pads/scratch/turam/volume/27UP4XKH 240x160	admin	PADS	Completed	2011-04-15 20:18:46	2011-04-15 20:55:51	
585	vl3.sh 1 30 585 vlgen sphere_32x32x32 1024x768	admin	PADS	Completed	2011-04-15 18:33:29	2011-04-15 19:03:38	
584	vl3.sh 5 30 584 bin /gpfs/pads/scratch/turam/volume/zebrafish/crop01-5.raw 1024x768 1000x1070x600 32	admin	PADS	Completed	2011-04-15 14:45:19	2011-04-15 15:24:38	
583	vl3.sh 3 30 583 bin /gpfs/pads/scratch/turam/volume/zebrafish/crop01-5.raw 1024x768 1000x1070x600 32	admin	PADS	Completed	2011-04-15 14:22:13	2011-04-15 14:47:16	
582	vl3.sh 3 30 582 bin /gpfs/pads/scratch/turam/volume/zebrafish/crop01-5.raw 1024x768 1000x1070x600 32	admin	PADS	Completed	2011-04-15 13:14:20	2011-04-15 14:00:41	
581	vl3.sh 3 30 581 bin /gpfs/pads/scratch/turam/volume/zebrafish/crop01-5.raw 1024x768 1000x1070x600 32	admin	PADS	Failed	2011-04-15 13:09:02	2011-04-15 13:22:40	
580	vl3.sh 1 30 580 bin /gpfs/pads/scratch/turam/volume/zebrafish/07-	admin	PADS	Failed	2011-04-15	2011-04-15	

Figure 3 View of jobs in the GPSI gateway. Jobs are searchable by application name, parameter values, and tags.

Jobs are executed asynchronously by the GPSI execution daemon. The GPSI daemon queries the database periodically for jobs to run. It gathers the execution parameters and input files and uses the Swift execution engine to submit jobs to TeraGrid and local compute resources. The daemon executes each job from a separate thread and waits on its completion, reporting progress back to the database, including whether and how many jobs have been submitted, are active, or have completed. Using the site definitions configured by the user, Swift stages files to the selected compute resource, executes the job, and stages the files back to the portal. If the user cancels the job while in process, Swift terminates jobs running on the compute resources and exits, and the job is labeled as canceled.

As with other data on the site, jobs are represented in a dynamic table built on AJAX queries (**Figure 3**). Jobs can be searched by date, application name, execution parameters, or tags. The job list can also be sorted by any of the columns.

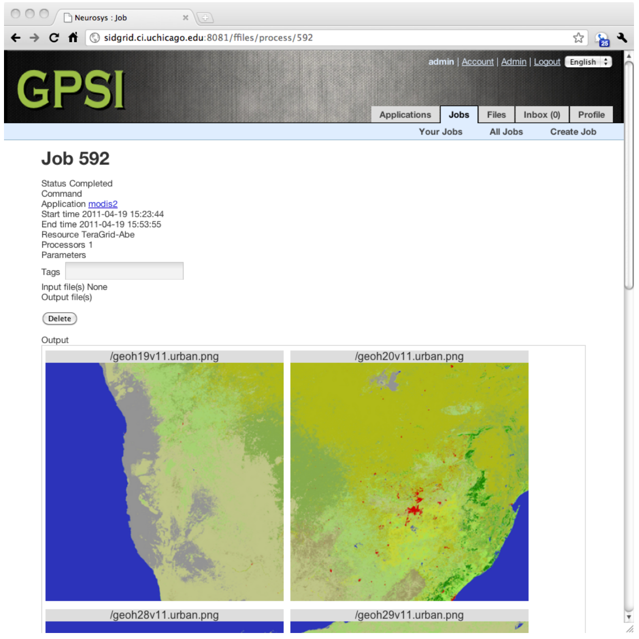
3.5 Output Processing

The results of computational science jobs take many forms. Many of these are common and admit to general handling: images, movies, or columnar text output; these can all be satisfactorily rendered using standard representations. Some of these basic forms, and certainly any of the more complex, domain-specific outputs, will be better represented with some guidance from the scientist. We seek to provide standard mechanisms for many of the standard data formats, and to allow users to customize output representations for all data formats.

Upon job completion, the portal processes the outputs to record provenance relationships and produce an HTML representation. By default, all outputs produced by the job are recorded as outputs and appear in the generated HTML representation. Alternatively, applications can describe the output files that should be tracked by listing them in a simple XML file; an example is given in Figure 4.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<output>
<file label="Data Summary">summary.csv</file>
<group name="T1af7-25-100">
<file label=""> T1af7-25-100_scatter.png</file>
<file label="Best Structure">T1af7-25-100_best.pdb</file>
</group>
</output>
```

Figure 4 An example of an XML file to describe the outputs of a job. This is an optional file used to select a subset of the outputs produced by the job.



Job 592

Status Completed
Command
Application modis2
Start time 2011-04-19 15:23:44
End time 2011-04-19 15:53:55
Resource TeraGrid-Abe
Processors 1
Parameters
Tags
Input file(s) None
Output file(s)

Output

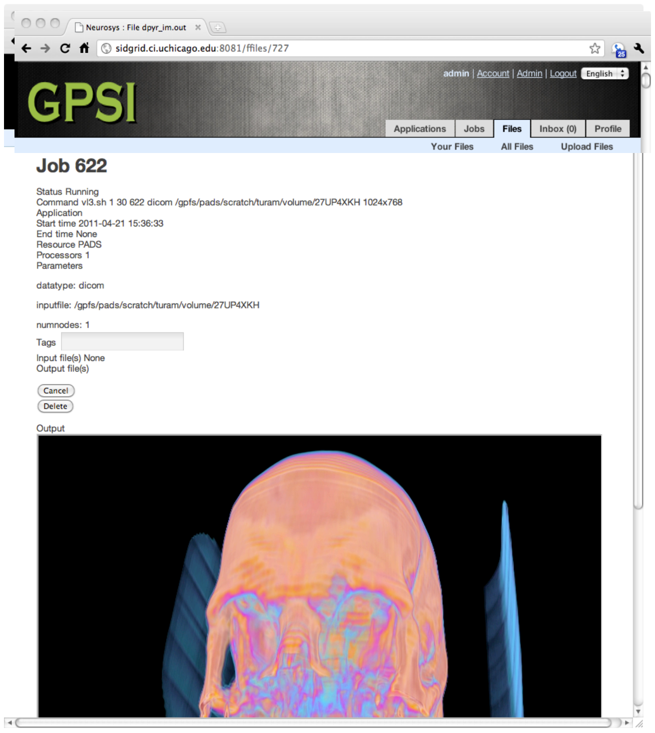
Figure 5 View of a MODIS job in the GPSI gateway. Pertinent job details are presented, as is the generated output view, including images produced.

The portal produces the final HTML representation by transforming the output XML file with an XSL [12] style sheet. The portal uses the default style sheet, unless an alternate style sheet has been provided by the application. An example of HTML output generated using the default style sheet is given in Figure 5. In this case, the output was produced by an analysis of MODIS data to colorize land mass images based on population density. These images appear because the default XSL stylesheet includes handling for images.

3.6 Visualization

Visualization is a key component of scientific analysis but has traditionally been delivered to web browsers in the form of individual images or low-resolution animations. Modern web browsers, with support for HTML5 video, are changing that landscape by providing support for the latest standards in streaming video, such as the h.264 and WebM codecs.

Figure 6 shows a results page from a VL3 volume rendering job run from within the portal. The job was run on the local PADS cluster, with VL3 producing the rendering, and accompanying software scraping and streaming the JPEG images to the browser. Mouse and keyboard interactions are captured from the web browser and transmitted to the running visualization on the compute resource to manipulate the dataset. We are also experimenting with streaming the visualization to web browsers using the WebM codec.


Neurosys : File dpyr_im.out
sidgrid.ci.uchicago.edu:8081/ffiles/727
GPSI
admin | Account | Admin | Logout
English
Applications
Jobs
Files
Inbox (0)
Profile
Your Files
All Files
Upload Files
Cancel
Delete

Job 622

Status Running
Command vl3.sh 1 30 622 dicom /gpfs/pads/scratch/turam/volume/27UP4XKH 1024x768
Application
Start time 2011-04-21 15:36:33
End time None
Resource PADS
Processors 1
Parameters

datatype: dicom

inputfile: /gpfs/pads/scratch/turam/volume/27UP4XKH

numnodes: 1

Tags

Input file(s) None
Output file(s)

Output

Figure 6 View of a VL3 job in the GPSI gateway. JPEG Images are streamed from the compute resource to the web browser, where the user can interact with the running volume rendering process.

4. RELATED WORK

HUBzero [8] is a mature and widely used platform for building science gateways. It is built atop the open-source Joomla CMS, written in PHP. The HUBzero project and GPSI share many common functional goals, including abstracting scientific applications, and integrating execution, analysis, and visualization. Applications are integrated using Rapptor definitions (analogous in some sense to the Swift application descriptions used in GPSI), from which desktop applications are built using Tcl and transmitted to the web browser using VNC [11]. In GPSI, we have tried to exploit modern web browser functionality, relying on Django to produce interfaces for specifying application inputs, and HTML5 to stream content to users. HUBzero's support for data analysis is, nonetheless, exemplary.

Open Grid Computing Environments (OGCE) is a comprehensive toolkit for building science gateways, and includes a Java-based portal foundation, grid compute tools, SOAP web services, a JavaScript Grid library, and OpenSocial gadgets support. These components have arisen out of very successful gateway projects (e.g., LEAD). Many of these components are good models for the underlying abstractions of GPSI. The key difference is that OGCE targets gateway developers, whereas GPSI is targeting science users.

SimpleGrid [13] is a toolkit for developing gateways for the geosciences community. Its goal is to simplify the process of developing gateways and provide common infrastructure that can mature as a gateway platform. Like OGCE, SimpleGrid targets gateway developers, so applies at a different level.

5. DISCUSSION

In this paper, we have described our work on GPSI, a generic portal infrastructure for science gateways. GPSI is the result of evaluations of technology choices, user engagement, and lessons learned in previous science gateway development efforts. From these we found that many science gateways could be built on a common foundation, avoiding the usual overhead of bootstrapping a new gateway. GPSI provides a foundation that includes data management, application management, job management and monitoring, and output processing facilities. These operate in default modes, with options for user customization.

Our work so far has concentrated on building the GPSI infrastructure and applying it in test scenarios from earlier science gateways. Three areas in which we plan to expand GPSI are:

- analysis using a combination of client-side graphing and server-side statistical analysis tools;
- rendering datasets on remote visualization resources and streaming to the web browser;
- web service access to GPSI services for data browsing, upload and download, and job submission.

In the future, we intend to apply GPSI more intensively in collaboration with science partners to identify shortcomings and additional requirements.

6. ACKNOWLEDGMENTS

This work was supported in part by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Dept. of Energy, under Contract DE-AC02-06CH11357, and the National Science Foundation by grant OCI-0504086.

7. REFERENCES

[1] Wilkins-Diehr, N., Gannon, D., Klimeck, G., Oster, S., Pamidighantam, S. 2008, "TeraGrid Science Gateways and Their Impact on Science," Computer , vol.41, no.11, pp.32-41, Nov. 2008.

[2] Dahan, M., Insley, J. A., Papka, M. E., Uram, T., and Gaither, K. P., 2007. "Enabling Science through the TeraGrid Visualization Gateway" TeraGrid '07 Conference, Madison, WI, June 14 - 18, 2007.

[3] Bertenthal, B., Grossman, R, Hanley, D., Hereld, M, Kenny, S., Levow, G., Papka, M., Porges, S., Rajavenkateshwaran, K., Stevens, R., Uram, T., and Wu, W. 2007. Social Informatics Data Grid, E-Social Science 2007 Conference, October 7-9, 2007, Ann Arbor, Michigan

[4] Wu, W., Papka, M.E., and Stevens, R. 2008 , "Toward an OpenSocial Life Science Gateway," *Grid Computing Environments Workshop, 2008. GCE '08* , vol., no., pp.1-6, 12-16 Nov. 2008

[5] Wu, W., Uram, T., Wilde, M., Hereld, M., and Papka. M.E., 2010. Accelerating science gateway development with Web 2.0 and Swift. In *Proceedings of the 2010 TeraGrid Conference* (TG '10). ACM, New York, NY, USA, Article 23, 7 pages.

[6] Wilkins-Diehr, N., and Lawrence, K.A. 2010, "Opening science gateways to future success: The challenges of

gateway sustainability," Gateway Computing Environments Workshop (GCE), 2010 , vol., no., pp.1-10, 14-14 Nov. 2010

[7] Wilde, M., Foster, I., Iskra, K., Beckman, P., Zhang, Z., Espinosa, A., Hategan, M., Clifford, B., Raicu, I. 2009. Parallel Scripting for Applications at the Petascale and Beyond, *IEEE Computer* 42(11):50-61, Nov. 2009.

[8] McLennan, M., and Kennell, R. 2010, "HUBzero: A Platform for Dissemination and Collaboration in Computational Science and Engineering," *Computing in Science & Engineering* , vol.12, no.2, pp.48-53, March-April 2010

[9] PADS, http://pads.ci.uchicago.edu

[10] Basney, J., Humphrey, M., and Welch, V., 2005. The MyProxy online credential repository: Research Articles. *Softw. Pract. Exper.* 35, 9 (July 2005), 801-816.

[11] Richardson, T.; Stafford-Fraser, Q.; Wood, K.R.; Hopper, A., "Virtual network computing," *Internet Computing, IEEE* , vol.2, no.1, pp.33-38, Jan/Feb 1998

[12] XSL transformations (XSLT) version 1.0

[13] Wang, S., Liu, Y., Wilkins-Diehr, N., and Martin, S. 2009, SimpleGrid toolkit: Enabling geosciences gateways to cyberinfrastructure, Computers & Geosciences, Volume 35, Issue 12, December 2009, Pages 2283-2294, ISSN 0098-3004, DOI: 10.1016/j.cageo.2009.05.002.

[14] Git, git-scm.com

[15] Python, www.python.org

[16] Django, www.djangoproject.com

[17] JQuery, www.jquery.org

[18] DataTables, www.datatables.net

[19] uPortal

[20] GridSphere