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

Thomas D. Uram, Michael Wilde, Michael E. Papka, Mark Hereld
Argonne National Laboratory, University of Chicago Computation Institute

TeraGrid 2011
Motivation

Large projects can afford to have both a software and hardware budget, and they allocate teams of scientists to write custom software for the experiment. ... As an example, I have been watching the U.S.-Canadian ocean observatory Project Neptune allocate some 30 percent of its budget for cyberinfrastructure. In round numbers, that’s 30 percent of 350 million dollars or something like 100 million dollars! ... This is something that we computer scientists could help fix by building generic tools for the scientists. – Jim Gray, The Fourth Paradigm: Data-Intensive Scientific Discovery

- Based on our past gateway experience...
  - SIDGrid
  - Open Life Science Gateway
  - TeraGrid Visualization
    ...gateways tend to be domain specific, but much functionality is common among them
- Focus effort on generic tools for scientists and make it easy for them to customize
Computational Science Workflow

- Typical phases
  - Input data
  - Applications
  - Jobs
  - Results
  - Analysis
  - Visualization
  - Discovery
  - Repeat as needed

- Design Options
  - Train scientists in effective use of ssh and bash
    - Everything is possible
    - Not very user friendly
  - Provide a different abstraction (e.g. gateways)
    - A limited set of things is possible
    - Very user friendly (or can be)
Generic Portals for Science Infrastructure (GPSI)

- Identify and implement common components in computational science workflow
  - Data
  - Jobs
  - Results
  - Analysis
  - Visualization
  - Discovery

- Facilitate user customization to meet domain-specific requirements
  - Applications
  - Analysis
  - Visualization

- Facilitate developer customization (as a last resort)
Principles

**Principles derived from our experience building gateways and from surveys of gateway projects**

- **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.

- **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.

- **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.

- **Active developer 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 base**
  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.
Technology Choices

- Django / Pinax - Web Framework
- Swift - Workflow Language and Execution Engine
- JQuery - Javascript Library
- XML/XSL - Data Format and Transform Definitions
Technology Choices - Django/Pinax

- Model-View-Controller pattern
  This approach encourages sound 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.

- Web template language support consistent, intuitive interfaces
  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.

- Many standard and third party modules are available for Django and supported by the active development community. A few examples include support for OpenID, Facebook Connect, and user ratings. These make it easy to extend GPSI.

- 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.

- Extensive documentation
  The gateway is 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 GPSI easier for gateway developers to maintain.
Technology Choices - Swift

Swift is a system for the rapid and reliable specification, execution, and management of large-scale science and engineering workflows. It supports applications that execute many tasks coupled by disk-resident datasets - as is common, for example, when analyzing large quantities of data or performing parameter studies or ensemble simulations.

The open source Swift software combines:

- A simple scripting language to enable the concise, high-level specifications of complex parallel computations, and mappers for accessing diverse data formats in a convenient manner.
- An execution engine that can manage the dispatch of many (10,000) tasks to many (100) processors, whether on parallel computers, campus grids, or multi-site grids.

Swift users span the physical sciences, biological sciences, social sciences, humanities, computer science, and education. Swift users have achieved multiple-order-of-magnitude savings in program development and execution time.

Swift builds on and includes technology previously distributed as the GriPhyN Virtual Data System.

WHAT'S NEW?

PAPER PUBLISHED
Parallel Scripting for Applications at the Petascale and Beyond [pdf]

SWIFT 0.9 RELEASE - 27 APR 2009
The latest release of Swift, v0.9, is available from the downloads page.

You need to run an analysis pipeline on a thousand functional magnetic resonance imaging (fMRI) files. The files are in some funny directory structure, the TeraGrid is miles away, and neither writing an MPI program nor performing the many analysis tasks

turam@mcs.anl.gov
TeraGrid 2011
Data Management

- Files can be uploaded individually or in bulk; compressed archives will be expanded and the individual files catalogued.
- Files can be tagged, supporting categorization and search.
- Ajax-based file collection search on name, date, tags.
Files are displayed with relevant metadata, including a link to the originating job and a content-specific preview.
Compute Resource Management

- Users have a variety of compute resources available to them (TeraGrid, OSG, campus, project-specific), and should be able to integrate them without administrative intervention.
- In the current prototype instance, compute resources are controlled by the administrator using the Django admin interface.
- Compute resources registered with GPSI currently include:
  - Ranger (TACC)
  - QueenBee (LSU)
  - Lincoln (NCSA)
  - Blacklight (PSC)
  - UChicago PADS (Petascale Active Data Store)
Credential Management

- Credentials managed by user and associated with particular resources
- Credential types include
  - ssh
  - MyProxy
    - MyProxy server host, port, username and password
    - Credential renewed as needed
  - MyProxy OAuth
    - Avoids collection of MyProxy username and password by gateway
    - Have currently integrated pre-production TeraGrid MyProxy OAuth support
Application Management

- Multiple options for introducing applications to GPSI
  - Command line
  - Swift script
- Application arguments are parsed from command line or Swift script and stored in application definition
- Applications can optionally include an output file list in XML format, and an XSL stylesheet to control output representation. The output XML file can be static, or can be produced during execution. The XSL stylesheet can be modified online.
Swift script example

type file;

# <bus> is integer between 1 and 30
# <power> is in megawatts

app (file o) sgflow (int bus, int power)
{
    sgflow bus power stdout=@o;
}

string nbus = @arg("nbus","10");
string nplevel = @arg("nplevel", "3");
foreach bus in [1:@toint(nbus)] {
    foreach plevel in [1:@toint(nplevel)] {
        file o<single_file_mapper;
            file=@strcat("out.","bus.",bus,".pow.",plevel)>;
        o = sgflow(bus,plevel); <-- Swift submits a compute job for each call to sgflow
    }
}
Job Management

- Application command line arguments are presented as options in web form
- File inputs can be selected from the user’s file pool
- No support for argument validation yet. Scientists tend to know valid arguments (but not always)
- Command line applications will eventually support trivial parallelization using Swift concurrency (e.g. parameter sweeps)

Web interface generated from arguments to Swift script

turam@mcs.anl.gov
TeraGrid 2011
Job Management

- Job history is displayed with command line arguments, status, tags, etc.
- Jobs can be searched by any of these metadata
- Facilitates computational reproducibility: details of run are captured and can (soon) be used to launch similar runs

Note: “vl3” jobs have been selected in this view
Job Management

- For each job, a (partial) provenance record is stored
  - application
  - compute resource
  - parameters
  - input files
- Output files are listed, and represented according to the application-specific stylesheet
Output Processing

- Outputs are identified by user in initial application description
- Output filenames are generated by Swift
- At job completion, an XML output description is generated if the job has not produced one
- Standard XSL style sheet is applied to output description to produce the output HTML representation, if the application description does not include one

```xml
<?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>
    <file label="Lowest Energy">T1af7-25-100_predicted.pdb</file>
  </group>
</output>
```
Visualization

- Volume rendering job submitted to cluster nodes
- Generated HTML includes code to enable streaming of vis
- Resulting images captured and streamed to web browser (as JPEG images)
- Mouse/keyboard events captured in web browser and transmitted to volume rendering application
- Application-agnostic, similar to VNC but achieves higher frame rate and uses standard web technologies
Future Work

- Engage science partners
  - Deploy GPSI to identify gaps

- Infrastructural
  - Catalog user data across compute resources
    - Done previously in support of TeraGrid File Transfer gadget
  - Integrate code repositories to support scientific application development and dynamic provisioning
  - Enable Swift parallel scripting in web interface
  - Integrate basic tools for interactive analysis and visualization
  - Integrate alternative tools for target areas (e.g. workflows, analysis, visualization)
  - Implement API for applications to add simulation metadata to system, and to enable local scripting against GPSI service
  - Infrastructure as a Service
    - Click to deploy GPSI “instance”
  - Potential Globus Online integration
References

- Django
  - djangoproject.org
- Pinax
  - pinaxproject.com
- Swift
  - swift.ci.uchicago.edu
- OAuth
  - oauth.net
- TG MyProxy OAuth
  - security.ncsa.illinois.edu/teragrid-oauth
- vl3 volume rendering application
  - ask me or see TG11 talk “Interactive Large Data Exploration over the Wide Area” Thursday 11:30am
Acknowledgements

- Mike Papka, Mark Hereld, Joe Insley, Eric Olson, and the Futures Laboratory at Argonne National Laboratory
- Mike Wilde and the Swift team
- TeraGrid Gateways program
- NSF Award SCI-0503697
- XSEDE