The CIPRES Workbench: A Flexible Framework for Creating Science Gateways

Mark A. Miller, Terri Schwartz, Paul Hoover, Kenneth Yoshimoto, Subhashini Sivagnanam, and Amit Majumdar
San Diego Supercomputer Center, University of California, San Diego, 9500 Gilman Drive La Jolla CA 92093-0505
mmiller,terri,phoover,kenneth,majumdar@sdsc.edu

ABSTRACT

Here we describe the CIPRES Workbench (CW), an open source software framework for creating new science gateways with minimal overhead. The CW is a web application that can be deployed on a modest server, and can be configured to submit command line instructions to any resource where the application has submission privileges. It is designed to be highly configurable / customizable, and supports GUI-based access to HPC resources through a web browser interface as well as programmatic access via a ReSTful API. Using browser access, the CW architecture creates an environment with secure user accounts where user input data, job results, and job provenance are stored. Using ReSTful access, it allows users with a registered a client application to deliver command lines to analytical codes and return of results from any compute resource. A development effort is underway to make allow the CW to submit jobs via the Science Gateways as a Platform (SciGaP) services hosted at Indiana University.

Categories and Subject Descriptors

D.2.2 [Software Engineering]: Design Tools and Techniques – Modules and interfaces

General Terms

Design.

Keywords

Software, Workbench, Science Gateway, ReSTful services. Open Source, SciGaP.

1. INTRODUCTION

Making it easy for domain researchers to access HPC and data resources is a difficult problem, and one that is of increasing importance. As digital tools become ever more sensitive, and sensors become more numerous, there is an ongoing explosion in the amount of data available for analysis. This in turn drives an increasing need for easy access to computational resources that are adequate for the data analysis jobs at hand.

Science Gateways are domain-focused web applications that address this problem by providing access to community data resources and/or analytical codes run on large HPC resources [1].

Conference'10, Month 1–2, 2010, City, State, Country.

Typically, Gateways provide access through a web browser and serve entire communities of practice that require a common set of analytical codes and/or data. A large number of Science Gateways have been created over the past decade, each addressing the needs of a specific community [2, 3]. In general, these Gateways have been created independently through separate, and often significant software development efforts.

An ongoing issue for Science Gateways is the question of how best to simplify the task of Gateway development. All Gateways require a common feature set (login, data storage, job submission, data retrieval, etc), as well as some features that are unique to a particular community of practice (legacy codes, interactive analytical interfaces, etc). There have been several significant efforts to create software infrastructure that provides the common infrastructure required by all gateways, while at the same time facilitating the addition of custom features required for a specific domain. Exhaustive review of these efforts is beyond the scope of the paper, but here we note some exemplars of development efforts to date.

One approach to creating a generic Gateway infrastructure relies on a self-contained gateway software package that creates a Gateway application (e.g. Galaxy [4], HubZero [5], and the Workbench Framework [6]). The Galaxy platform and the Workbench Framework are organized around submission of command line instructions and data queries to any available remote data or compute resource. The HubZero package, on the other hand, presented each user with a provisioned virtual machine, and most operations were carried out within that context. It has since been modified to submit jobs to remote resources in some instances. All three platforms were designed to make tool addition scalable, through different techniques. The HubZero and Galaxy platforms have been used to create Gateways outside of their original domain [7-9], and several implementations of the Workbench Framework have been created [10, 11]. The distributable web server design has succeeded in simplifying Gateway creation, but each installation is distinct, and must be installed and maintained by dedicated developers/system administrators on local hardware.

The second design philosophy involves creating a set of core Web services that provide the common core functionalities required by all Gateways. In this design paradigm, functionalities required by all Gateways are provided through a central server as a set of services that can be consumed by a client created by the Gateway creator. This philosophy minimizes the costs associated with middleware development and system administration, but individual Gateway developers must create a domain-specific client interface that consumes the services, and provides its users with access. The Agave [12], NEWT [13], and SciGaP (http://scigap.org/) services are provided from a central location under the control of project administrators. However, the

underlying NEWT framework is publicly available, which allows Gateway developers to create their own custom RESTful access outside of the NERSC environment.

Each of these two design philosophies has merit. Gateways can be created very quickly by using a generic, customizable interface to access centralized infrastructure services offered by the providers such as SciGaP, NEWT and Agave. This strategy eliminates the cost of creating and managing middleware for job submissions for individual gateways. On the other hand, self-contained gateway installations are not limited to resources and capabilities supported by the available RESTful service providers.

Here we describe an attempt to capture the best of both worlds, by adapting an existing framework for gateway development [6] to create a generic infrastructure that can support access both to generic compute or data resources, and to public ReSTful job submission services. This strategy is intended to provide maximum flexibility to gateway developers so they can take advantage of tools and capabilities available through ReSTful as these services expand and mature.

2. Design/Architecture

The CIPRES Science Gateway was created through two loosely connected Java software packages: the Workbench Framework (WF), which manages the executive functions of CIPRES, and a Portal Application (PA1), which is a Struts-based application that allows users to configure/create submissions and manage results via the WF through a browser interface [11]. The WF/PA1 design was intended to support delivery of command lines and database queries to any available remote resource without regard for its location or domain. All specific information about supported codes and databases is contained in a central registry, making the PA1/WF a generic, domain-independent software framework for Science Gateways. Accordingly, the PA1/WA have been used by other groups to create the Neuroscience [14] and PoPLAR [10] Science Gateways, as well as the Next Generation Biology Workbench for which it was created [15].

The WF has evolved significantly over the past six years in response to a growing user population, increased traffic and job submission loads, and growing size of input and output files. This evolution has driven the development of many improved features and capabilities [16, 17]. In addition, we recently released a public RESTful API to permit programmatic access to CIPRES for job submissions [18]. The RESTful services are provided via a second Portal Application (PA2) based on Jersey [19]. PA2 validates and submits instructions and input files from user client applications to the WF. As a result, developers outside the CIPRES project can access XSEDE or local HPC resources through client applications, whether it be a sophisticated desktop application like Mesquite [20], a web application like MorphoBank [21] or ViPR [22], or a simple ad hoc script created by user. Details of the RESTful API implementation can be found elsewhere [18].

As the software package used by CIPRES is now quite stable, and highly evolved for high traffic gateway use, we elected to refactor it into a distributable open source software package. The new package is designated as the CIPRES Workbench (CW), and it is designed to facilitate installation and customization of a gateway application in service of any domain. The goal in creating an open source project is to make it possible for us to share advances and improvements made by ourselves and other users of the software. The software will be useful for Gateways that require robust execution of command line instructions to codes run on a variety of remote execution hosts.

2.1 Design of the CIPRES Workbench

As noted above, the CIPRES Workbench package consists of a JAVA software development kit (WF) and two separate JAVA portal applications, one that provides access through a browser interface (PA1), and one that supports access through a RESTful application programmer interface (PA2). The overall design is represented in Figure 1. The CW requires a relational database to store user information, uploaded data, job statistics, and results, but it is agnostic on the specific RDBMS. Jobs are submitted to any remote execution hosts where submission privileges have been established. The design also allows for the submission of queries to remote databases.

2.2 Workflow for the CIPRES Workbench

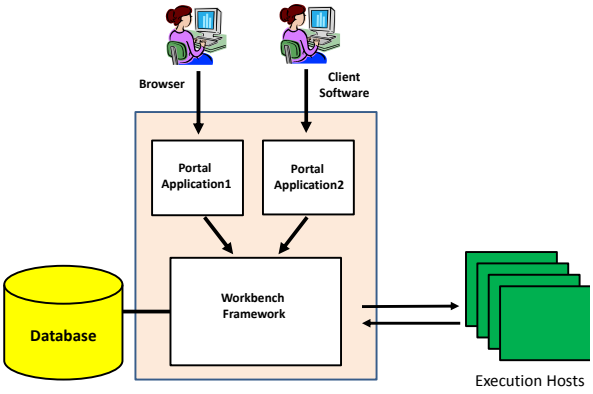

Figure 1. Overall design of the CIPRES Workbench

The basic workflow for the CW is shown in Figure 2. User access to the WF is provided by PA1 and PA2. Users of the browser interface (PA1) configure job submissions via a set of web forms. One or more web forms manage submissions to each code supported by the CW. Each web form is generated at compile time from a <tool>.xml document in the WF (see below). The xml tool description is based on the PISE specification [23]. The web forms provide JavaScript controls that guide and inform users in job configuration, and prevent submission of commands that are nonsensical, or will fail immediately.

For submissions to the ReSTful API (PA2), a client application or script must submit essentially the same parameters and input files that the web form in PA1 would provide. These parameters are validated on the server, and if valid, are used to populate the parameter map to configure a given run and generate a command line. Submissions from PA2 are subject to the same validation processes as submissions from PA1, using code generated from the <tool>.xml file; the same auxiliary files and command lines are created by the WF.

On submission from either PA1 or PA2, a "job handle" is assigned to uniquely identify the job, and job information is entered into the Task table of the CIPRES database. Submitted key value pairs are transmitted to the WF where they are validated. The validation code is generated from the same <tool>.xml documents that are used to create web forms used by PA1.

The selections made in the web form or submitted by PA2, together with information in the <tool>.xml document determine run configuration: number of nodes, cores, which parallel version of the code is called, which command line options are set, etc. The parameter map and <tool>.xml are also used to create any

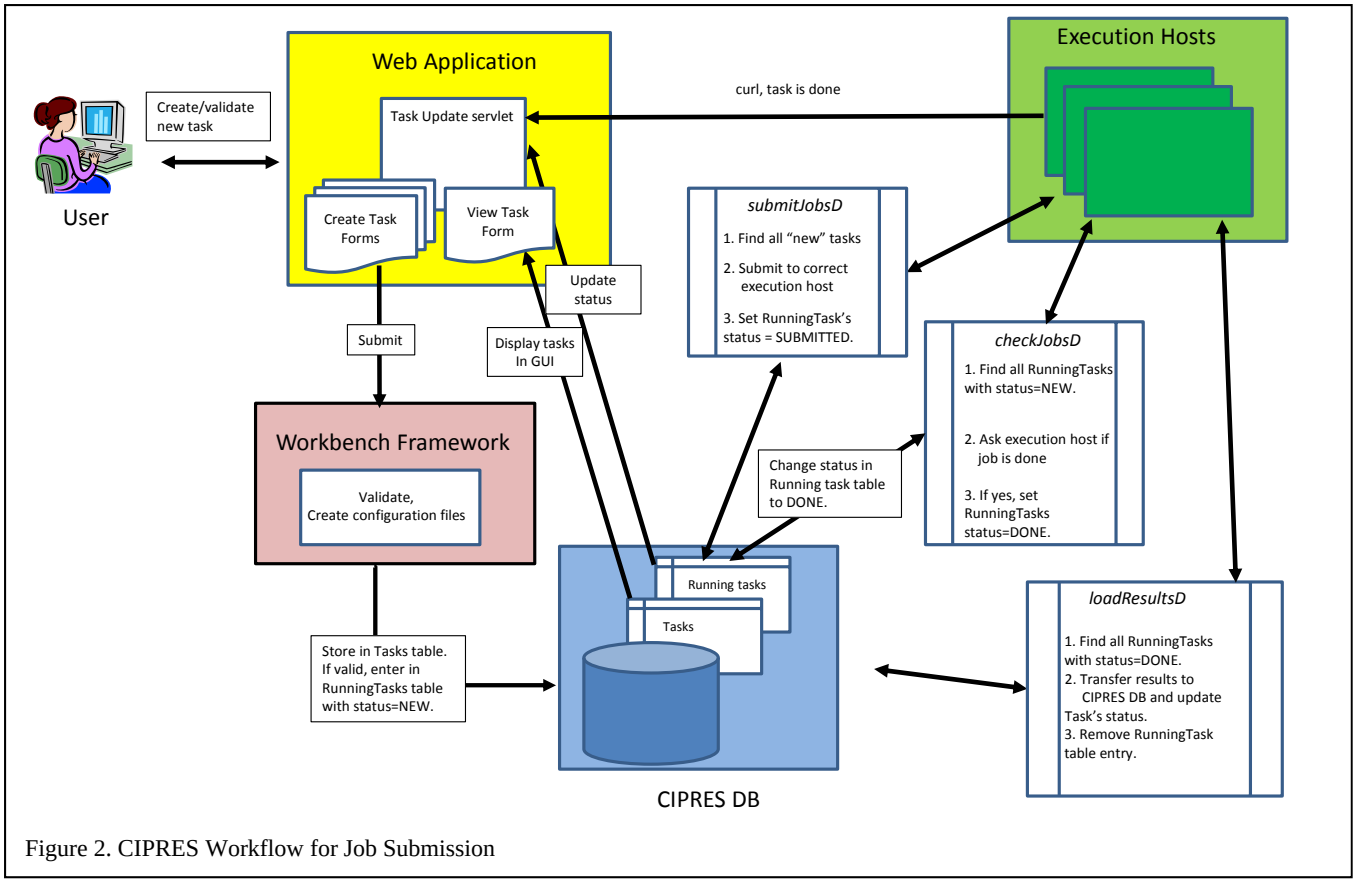

Figure 2. CIPRES Workflow for Job Submission

additional files required for the job run. The final step in job submission step is to create a RunningTask table entry for the job that contains the jobhandle and has its status set to "new".

Next, the job must be submitted to the execution host. Remote job execution is managed by a set of three daemons. The submitJobD daemon manages file staging and submission. SubmitJobD selects all Running Task table entries with status "new" and uses a thread pool to process them. For each job, the daemon creates a working directory with the name "jobhandle" on the execution host using the FileHandler class specified in the tool registry. (The WF includes a variety of FileHandlers such as LocalFileHandler, GridFtpFileHandler, SFTPFileHandler; it is easy to create FileHandlers for additional protocols.) The submitJobD daemon copies input files from the Task table of the database into the working directory, and creates an additional file named _JOBINFO.TXT that contains the name and email address of the user running the job, the jobhandle, and some additional information. If the execution host has a job scheduler, a custom "submit" script must be installed on the host to create a submission script for appropriate for that host's scheduler and to submit the script to the scheduler (e.g. for Torque, qsub is used). After creating the working directory and staging the input files, submitJobD uses the ProcessWorker class specified in the tool registry (see below) to run the "submit" script on the execution host (the CW includes ProcessWorker's for a variety of schedulers and communication protocols, and new ones are fairly easily added). On successful submission, the "submit" script prints the job ID from the scheduling system on stdout, SubmitJobD reads the jobID and updates the Running Task entry with the jobID (providing the RunningTask entry with both the CIPRES job handle and the remote system's job ID) and sets the status to "submitted".

Monitoring Job Status on Execution Hosts/Returning Results

Job status notification is managed by two methods. The primary mechanism occurs when a job completes: the job submission script issues a curl command back to a servlet in the web application when the job stops running. The servlet changes the Running Task's status to "done". A secondary mechanism uses daemon, CheckJobsD, which was implemented to make results notification robust against outages of the server or the execution host, which would disrupt the primary notification mechanism.

CheckJobsD monitors the progress of running jobs by selecting all Running Task entries with status = "submitted" and querying the execution hosts about those jobs. It does this by using ssh or gsissh to run a custom "check" script on the remote hosts, which uses qstat (or the appropriate command for the scheduler) to determine which of the jobs are finished and returns their IDs. When a job is finished, checkJobsD changes its status in the Running Task table to "done".

A third daemon, loadResultsD, selects all Running Task entries with status = "done". LoadResultsD, uses a FileHandler to transfer the results to the database and delete the RunningTask table entry. Before transferring results, the loadResultsD daemon examines the size of working directory to be transferred. If the working directory is below a configurable size limit (in the case of CIPRES this is 4GB) the directory is transferred to the database for indefinite storage. If the working directory contents are larger than the specified limit, the results are instead moved to an NFS file directory outside of the application for separate processing. For CIPRES, large results directories are managed by a script that

zips large results directories every 6 hours and moves them to a cloud storage resource for user access. When loadResultsD detects a large working directory, it posts a message to the user with a url where they can download the zip archive from the cloud resource via http. The zip archives of large results directories are retained for a period of one month.

After results are transferred to the database, the working directories on the execution host and their contents are retained for a specified period of time (2 weeks) before they are deleted by a cron job. The availability of the working directories after job completion simplifies the administrator's task in to debugging jobs that fail due to system and user errors. Frequencies of large file movement and times for archive retention are configured locally as cron jobs.

2.3 Installing and configuring the CW

The CW platform is designed to make creating a new Gateway as simple as possible. The basic procedure involves user configuration of a set of build properties that identify the location of key resources required by the application to operate in the designated environment. The package cleanly separates the generic executive capabilities of the WF from those specific to the Science Gateway being created.

Prerequisites:
Installation of the CW application requires Maven 3.0 or newer, Tomcat 7.0, Python 2.7, Java 1.7, and MySQL 5.5.5 or newer. Use of virtualenv is recommended to avoid permissions problems, as the build process involves the installation of python libraries. It also requires a relational database management system that uses SQL (the schema is found in the distribution at source/sdk/scripts/database/cipres.sql) and a web server where static content can be presented.

Set-up:
Creating a new gateway requires creation of a *config* directory (typically named *<newGateway>-config*). The config directory contains a build.properties file and all other information required for the build process. In addition, a set of five directories must be created for scripts, logs, a tool registry, database documents, and control of submission to specific tools/execution hosts. An example template is provided with the distribution that can be installed to conduct basic testing of the installation. The example can be copied and modified to create new Gateway applications. The build.properties file provides compile time properties that are used to configure the application for the local environment. A minimum of thirty values must be edited by the user to complete a new installation. The build.properties specifies paths to the five required directories noted above, the application's base url, the url and login information for the database instance, and properties of all local and remote submission hosts. Other build properties specified in this file include: availability and base url for REST access, usage limits, mail server information for notifications, polling intervals for daemons, Globus credential information, build script information, and information about submission hosts and tools that are known to this installation. The property values read from the build.properties file supplement and/or override information in the common code base. The build.properties file is customizable; a developer can add any needed properties to this file and reference those properties in resource files (e.g. the tool registry, see below).

Configuring New Execution Hosts.

The tools that the gateway can run and the execution hosts they run on are specified in a tool registry located at <newGateway>-config/sdk/src/main/resources/tool/tool-registry.cfg.xml.

To add an execution host, one must add a ToolResource element to the registry. For example, the CIPRES tool registry contains the following entry for Gordon:

```
<ToolResource id="gordon" type="GLOBUS"
        class="org.ngbw.sdk.tool.DefaultToolResource"
        filehandler="${gordon.filehandler}"
        processworker="org.ngbw.sdk.tool.SSHExecProcessWorker">
        <Parameters>
          <Parameter key="runner"
value="org.ngbw.sdk.common.util.GsiSSHProcessRunner" />
          <Parameter key="login" value="${xsede.gordon.login}" />
          <Parameter key="fileHost" value="${xsede.gordon.host}"/>
          <Parameter key="filePort" value="2811"/>
          <Parameter key="workspace" value="${xsede.gordon.workspace}"/>
          <Parameter key="rc" value="${xsede.gordon.rc}" />
          <Parameter key="submit" value="${xsede.gordon.submit}" />
          <Parameter key="check" value="${xsede.gordon.check}" />
          <Parameter key="cancel" value="${xsede.gordon.cancel}" />
          <Parameter key="accountGroup" value="xsede" />
          <Parameter key="chargeNumber" value="${cipres.charge.number}" />
          <Parameter key="coresPerNode" value="16" />
        </Parameters>
      </ToolResource>
```

Rather than hardcoding locations, class names, and account information in the tool registry, one may use properties and add them to the application build.properties file. For example, the registry entry shown above has these corresponding properties in CIPRES's build.properties:

```
xsede.gordon.workspace=/projects/ps-
ngbt/backend/gordon_test_workspace
xsede.gordon.host=trestles-dm1.sdsc.edu
xsede.gordon.user=
xsede.gordon.login=cipres@gordon.sdsc.edu
xsede.gordon.rc=/projects/ps-
ngbt/home/cipres/.bash_profile
xsede.gordon.submit=gordon_submit_v2.py
xsede.gordon.check=checkjobs_v2.py
xsede.gordon.cancel=delete_v2.py
gordon.filehandler=org.ngbw.sdk.tool.LocalFileHandler
```

These properties specify where working directories will be created on Gordon, the name of the file transfer host, login information, the location of an environment initialization script, the names of the "submit", "check" and "delete" scripts, and the FileHandler class to use with Gordon. The submit and check scripts are used by the daemons described above, while the delete script is invoked by a method in the WF when a user cancels a submitted job.

Job submission requires installation of several scripts to be on the execution host. Although the scripts are not part of the distribution, examples are included in the distribution, as they are meant to be customized. The example scripts include the "submit", "delete" and "check" scripts mentioned above, and an optional configuration file (typically a shell script to be "sourced"). Normally this script will set the PATH environment variable so that the "submit", "cancel" and "check" scripts are on the PATH.

Submissions through Airavata. The SciGaP project is creating a set of web services that provide infrastructure for new and existing Gateways. The goal of the SciGaP project is to decrease

the overhead for gateway creation and maintenance by centralizing the core infrastructure/middleware services such as authentication and job submission required by all Gateway projects. The software being developed to provide this infrastructure is Airavata [24], an open source Apache project.

As a first step in consuming the services offered by Airavata, we are developing code to manage job submission through an Airavata client called "Airavata-remote". This is a Python script that can access SciGap functions and data structures. The first step is to create a new CW ProcessRunner class that uses Airavata-remote (instead of gsissh or ssh with qsub or slurm) to communicate with execution hosts. The three daemons mentioned in the job submission workflow above will be able to use the new SciGapProcessRunner.

This option can be included in CW-based gateways by configuring selected ToolResources to use the SciGapProcessRunner. For example, one could change the parameter key "runner" in the Gordon ToolResource shown earlier from:

```
<Parameter key="runner" value= "org.ngbw.sdk.common.util.GsiSSHProcessRunner" />
```

to:

```
<Parameter key="runner" value="org.ngbw.sdk.common.util.SciGapProcessRunner" />
```

The new process runner will use the Airavata-Remote client, and if that fails for some reason, the runner will return to the GsiSSHProcessRunner as a fall back protocol. This will make submissions robust against any failures using Airavata alone.

In addition to creating the new process runner for SciGaP, the integration involves the following steps: making batch submission work through the SciGap API using the AiravataRemote client, incorporating SSH key management into the gateway setup process, incorporating SciGap API tokens (authentication and authorization from gateway to the SciGaP server) into SciGaP client. The motivating factor for taking advantage of SciGaP for submissions is improved reliability over gsi-ssh, easier credential handling, and improved capability for remote resource status monitoring.

Adding New Codes. Codes run on execution hosts are referred to as "tools" in the CW. New codes are added to the application in a two-step process. First, one or more <tool>.xml documents must be created to describe the new code's usage. These files contain all the necessary instructions for the application to create the command line, name any user-supplied input files, and create any other input files required by a given job run. In CIPRES, for example, each <tool>.xml specifies creation of a file named scheduler.conf. The scheduler.conf file specifies the job's maximum allowed runtime, the number of nodes and cores to use, etc. CIPRES "submit" scripts expect to find scheduler.conf in the job's working directory, and use the information to configure the job run.

The <tool>.xml documents are also central to job submissions by the CW in that they are used to generate PA1 web forms with JavaScript validation controls to configure job submission and to generate validation code on the backend that is shared by PA1 and PA2. A schematic of the use of the <tool>.xml files is shown in Figure 3.

The <tool>.xml files reside in a single subdirectory within the <newgateway>-config directory. Any <tool>.xml file in this directory can be activated in the application by adding it to the tool-registry. A sample tool registry entry looks like:

```
<Tool id="BEAST_TG"
      name="BEAST: Bayesian Evolutionary Anaysis by Sampling Trees"
      configfile="pisexml/beast_tg.xml" toolresource="gordon"
      commandrenderer="org.ngbw.pise.commandrenderer.PiseCommandRenderer">
</Tool>
```

The *Tool ID* value is determined by the name of the <tool>.xml file, *toolresource* specifies the execution host where the job is to be run, which is also identified in the tool-registry file as described above.

The tool registry allows tools to be grouped and entire groups can be enabled/disabled at build time. Individual tools and ToolResources (i.e. execution hosts) can also be selectively enabled/disabled at run-time, to temporarily prevent submissions to a host that is undergoing maintenance or is malfunctioning or to prevent use of a code that is causing problems.

Customizing the Web Applications.

The Struts-based PA1 application is customized primarily by modifying the template .jsp and .html files that are distributed with the application in the directory \scigap\trunk\example \portal\src\main\webapp. These can be easily edited to add relevant application-specific text and images; look and feel are adjusted by modifying the css.

The CW web application requires a static web site of some type for display of help pages. The application automatically generates links to help pages based on <staticurl><toolname>.html. The static site can be simply html pages placed at the appropriate urls relative to the base url of the static site. For convenience, and to manage versioning easily, the CIPRES static site is currently based on the Expression Engine CMS [25], while the NSG static site is based on Bootstrap [26].

Base urls for the static site, addresses for REST services, REST support, and REST documents, as well as location of bug tracking software can all be added as build properties

3. Case Study: Customization for the NeuroScience Gateway

Individual Gateways often require minor changes to accommodate the specific needs of the community codes they present. As an example of how these changes were made, we describe the adaptation of CW for use by the NeuroScience Gateway (NSG) [14], a neuroscience modeling gateway. Several changes were required to adapt the CW to the NSG use case. First, the NSG requires vetting of individual users prior to prior to allowing registration. Accommodating this requirement was a simply a matter of removing the registration links from the browser interface and posting a vetting form by editing the login.jsp page. The next change required was handling of input data. Codes offered by CIPRES require input data that consist of individual text files containing sequence data. For NSG codes, the input is models, which typically consist of source and executable code, along with input text files that vary depending upon the specific model. The user uploads a gzip-compressed (denoted by the .zip suffix), directory containing a binary, single file obtained from the ModelDB database [27]. The compressed upload contains models with one subdirectory under which the other files and directories are contained.

To accommodate this non-text file, the JavaScript upload code presented in the user's browser was modified to convert the binary .zip file to ASCII using the uuencode process. During job

submission, this ASCII encoded file is copied to the working directory for the task on the submission host. The NSG version of the "submit" script on the submission host then uses the uudecode utility to convert the ASCII encoded file back into a .zip file, then the gunzip utility is used to extract the complete subdirectory structure.

Since the subdirectory created within the working directory can have an arbitrary name, processing the data correctly required logic for determining the appropriate subdirectory name. We initally assumed a single subdirectory was present, as in the ModelDB database, then added additional logic to ignore MacOS specific directories. It was also necessary to adapt the system to tolerate white space in directory names, which is a common convention for non-Linux system users. This required modification of both the "submit" script and ibrun (an XSEDE MPI launcher) on the submission host.

A final modification required by NSG was introduction of a pre-processing step prior to job submission. The popular code NEURON [28] employs custom code (.mod files) that must be preprocessed into C, then compiled into code and placed into a special library for use during simulation. The design allows investigators to incorporate custom membrane mechanisms efficiently into simulations on diverse architectures. To accommodate this requirement in NSG, an additional compilation procedure had to be executed prior to job submission. To do this, the submit script was modified to call a simple shell script that gathers all .mod files into a single compilation directory, then runs a pre-processing and compiling tool (nrnivmodl) on that directory to creates the model library.

4. Future directions

Future development of the CW will aim to provide ever more flexibility in Gateway creation. One area we are exploring currently is integrating other SciGaP services into the CW application. Specific SciGaP features we have currently targeted are 1) *Scalable identity management.* Gateway such as NSG must validate users carefully, because users are allowed to upload and compile code on the server. The SciGaP project is exploring scalable identity management solutions, and we hope to incorporate this solution as a service with the CW. 2) *Workflow capabilities.* CIPRES offers codes that perform both sequence alignment and tree inference. While these two steps are part of normal phylogenetics workflow, the CW does not support the configuration and deployment of the two steps as a single process presently. We hope to be able to use submissions through SciGaP to add this capability. 3) *Interactive visualization* is a feature that is not supported easily by the CW platform at present. We will explore exposing this capability through services offered by the SciGaP project.

A second direction for future CW development is making the platform accessible as a cloud image. The Galaxy project has had success in developing CloudMan [29], which allows users to deploy Galaxy quickly on any cloud resource where they have established credentials, with little additional overhead. This capability will provide users with more flexibility in where they run their jobs, so they are not necessarily limited by queue or resource use policies on publicly available shared resources. We believe the same model may work for high end users who require more computational resources than they can receive through a Gateway community allocation.

5. Conclusions

The basic architecture of the CW has supported thousands of users of the CIPRES Science Gateway, and has been flexible enough to allow many adaptations and improvements as traffic and data size has increased over the years. The addition of ReSTful access through PA2 makes the package much more flexible in terms of access. Gateway developers can use the browser interface supported by PA1 to achieve rapid, out-of-the-box functionality, or create their own client application that manages data and job organization to whatever extent is required to meet their user community's needs. Our hope is that this package will prove a simple and convenient way for gateway developers to create a robust and stable gateway environment that supports large numbers of users. Our intention is also to make it possible to share innovations made by our group with existing adopters of the software, and to integrate improvements and enhancements made by other groups into the distribution. The architecture of the system provides significant flexibility in adopting new compute resources, and will also support query of any available remote databases as well, should the use case require it.

6. ACKNOWLEDGMENTS

The work described here was supported by NSF DBI-1262628, NSG DBI-1146949, and NSF ACI-1339856. Development support was also received from allocation award TG-DEB090011 from the XSEDE project, which is sponsored by the National Science Foundation. Initial software development for the workbench framework was also supported by NIH GM73931-01.

7. REFERENCES

[1] Wilkins-Diehr, N., Gannon, D., Klimeck, G., Oster, S., and Pamidighantam, S. (2008) TeraGrid Science Gateways and Their Impact on Science. *Computer* 41,(11) 32-41.

[2] (2015) SCI-BUS Gateway Listing. http://www.sci-bus.eu/wiki/-/wiki/Public/Publications

[3] (2015) XSEDE Gateway Listing. http://www.xsede.org/web/guest/gateways-listing

[4] Giardine, B., Riemer, C., Hardison, R.C., *et al.* (2005) Galaxy: A platform for interactive large-scale genome analysis. *Genome Res.* 15,(10) 1451-1455.

[5] McLennan, M. (2008) The Hub Concept for Scientific Collaboration. https://hubzero.org/resources/12

[6] Rifaieh, R., Unwin, R., Carver, J., and Miller, M.A. (2007) SWAMI: Integrating Biological Databases and Analysis Tools Within User Friendly Environment. In *Data Integration in Life Sciences (DILS 07)*, pp. 48-58.

[7] Chard, R., Sehrish, S., Rodriguez, A., *et al.* (2014) PDACS: a portal for data analysis services for cosmological simulations,. In *9th Gateway Computing Environments Workshop*, pp. 30-33, IEEE Press

[8] (2015) HubZero Powered Sites.

[9] Montella, R., Brizius, A., Elliott, J., *et al.* (2014) FACE-IT: a science gateway for food security research. In *9th Gateway Computing Environments Workshop*, pp. 42-46, IEEE Press

[10] Rekapalli, B., Giblock, P., and Reardon, C. (2013) PoPLAR: Portal for Petascale Lifescience Applications and Research. *BMC Bioinformatics* 14,(Suppl 9) S3.DOI: http://www.biomedcentral.com/1471-2105/14/S9/S3

[11] Miller, M.A., Pfeiffer, W., and Schwartz, T. (2010) Creating the CIPRES Science Gateway for Inference of Large Phylogenetic Trees In *SC10: Workshop on Gateway Computing Environments (GCE10)*

[12] Dooley, R., Vaughn, M., Stanzione, D., Terry, S., and Skidmore, S. (2012) Software-as-a-Service: The iPlant Foundation API. In *5th IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS)*.

[13] Cholia, S., Skinner, D., and Boverhof, D. (2010) NEWT: A RESTful service for building High Performance Computing web applications. . *Gateway Computing Environments Workshop (GCE)* 1-11.DOI:

[14] Sivagnanam, S., Majumdar, A., Yoshimoto, K., Astakhov, V., Bandrowski, A., Martone, M., and Carnevale, N.T. (2014) Early experiences in developing and managing the neuroscience gateway. *Journal of Concurrency and Computation: Practice and Experience*, 27,(2) 473–488.DOI: http://dx.doi.org/10.1002/cpe.3283

[15] Rifaieh, R., Unwin, R., Carver, J., and Miller, M.A. (2007) SWAMI: Integrating Biological Databases and Analysis Tools Within User Friendly Environment. In *Data Integration in Life Sciences (DILS 07)*, pp. 48-58.

[16] Miller, M.A., Pfeiffer, W., and Schwartz, T. (2011) The CIPRES Science Gateway: a community resource for phylogenetic analyses. In *2011 TeraGrid Conference: Extreme Digital Discovery*, pp. 1 - 8

[17] Miller, M.A., Pfeiffer, W., and Schwartz, T. (2012) The CIPRES Science Gateway: enabling high-impact science for phylogenetics researchers with limited resources. In *Proceedings of the 1st Conference of the Extreme Science and Engineering Discovery Environment: Bridging from the eXtreme to the campus and beyond*, pp. 1-8, ACM

[18] Miller, M.A., Schwartz, T., Pickett, B.E., *et al.* (2015) A RESTful API for Access to Phylogenetic Tools via the CIPRES Science Gateway. *Evolutionary Bioinformatics* 11,(4726-EBO-A-RESTful-API-for-Access-to-Phylogenetic-Tools-via-the-CIPRES-Science-.pdf) 43-48.DOI: 10.4137/EBO.S21501 www.la-press.com/a-restful-api-for-access-to-phylogenetic-tools-via-the-cipres-science--article-a4726.

[19] (2013) Jersey. http://jersey.java.net/

[20] Maddison, W.P. and D.R., M. (2008) Mesquite: a modular system for evolutionary analysis. Version 2.5. http://mesquiteproject.org

[21] O'Leary, M.A. and Kaufman, S. (2011) MorphoBank: phylophenomics in the "cloud". *Cladistics* 27,(5) 529-537.DOI: 10.1111/j.1096-0031.2011.00355.x http://dx.doi.org/10.1111/j.1096-0031.2011.00355.x

[22] Pickett, B.E., Sadat, E.L., Zhang, Y., *et al.* (2012) ViPR: an open bioinformatics database and analysis resource for virology research. *Nucleic Acids Res.* 40D593-598.

[23] Letondal, C. (2007) PISE (*), a tool to generate Web interfaces for Molecular Biology programs. http://www.pasteur.fr/recherche/unites/sis/Pise/

[24] Marru, S., Gunathilake, L., Herath, C., *et al.* (2011) Apache airavata: a framework for distributed applications and computational workflows. In *2011 ACM workshop on Gateway computing environments.*, pp. 21-28, ACM Press

[25] (2015) Expression Engine https://ellislab.com/expressionengine

[26] (2015) Bootstrap CMS. https://github.com/BootstrapCMS/CMS

[27] Davison, A., Morse, T., Migliore, M., Shepherd, G., and Hines, M. (2004) Semi-automated population of an online database of neuronal models (ModelDB) with citation information, using PubMed for validation. *Neuroinform* 2,(3) 327-332.DOI: 10.1385/NI:2:3:327 http://dx.doi.org/10.1385/NI%3A2%3A3%3A327

[28] Hines, M.L. and Carnevale, N.T. (2003) *The NEURON simulation environment. In: The Handbook of Brain Theory and Neural Networks.* MIT Press, pp.

[29] Afgan, E., Baker, D., Coraor, N., Chapman, B., Nekrutenko, A., and Taylor, J. (2010) Galaxy CloudMan: delivering cloud compute clusters. *BMC Bioinformatics* 11,(Suppl 12) S4.DOI: http://www.biomedcentral.com/1471-2105/11/S12/S4