Extending the Galaxy portal with parallel and distributed execution capability

Ketan Maheshwari,* Alex Rodriguez,† David Kelly,† Ravi Madduri,*
Justin M. Wozniak,* Michael Wilde,* Ian Foster*

*Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL USA
ketan,madduri,wozniak,wilde,foster@mcs.anl.gov

†Computation Institute
University of Chicago
Chicago, IL USA
arodri7,davidk@ci.uchicago.edu

ABSTRACT

The Galaxy platform is a web-based science portal for scientific computing supporting the life sciences user community. While user-friendly and intuitive for doing small to medium-scale computations, it currently has limited support for large-scale parallel and distributed computing. The Swift parallel scripting framework is capable of composing ordinary applications into parallel scripts that can be run on multiscale distributed and performance computing platforms. In complex distributed environments, often the user end of the application lifecycle slows because of the technical complexities brought in by the scale, access methods, and resource management nuances. Galaxy offers a simple way of designing, composing, executing, reusing, and reproducing application runs. An integration between the Swift and Galaxy systems can accelerate science as well as bring the respective user communities together in an interactive, user-friendly, parallel and distributed data analysis environment enabled on a broad range of computational infrastructures.

Keywords

Swift, Galaxy, Big Data, scientific applications

1. INTRODUCTION

The advent and impact of big data are evident from the ever increasing growth of data generation. Vast amounts of scientific data lie unexplored even as more algorithms and codes are developed to analyze big data in novel ways. Computational science today increasingly is converging to big data analytics. Furthermore, scientific applications are expanding their domains, resulting in an increase in the generation and processing of data and in the burgeoning demands on computational resources. Web based collaboration, online processing, and sharing of applications via portal-based environments have become commonplace. Furthermore, requirements such as analysis of results with visual aids, post-processing of execution traces, and customized visualization have added to the complexity.

The Galaxy [3] system addresses many of these demands, offering a web-based, interactive platform for data analysis via reusable and reproducible workflows. While excelling in the usability aspects, the Galaxy environment has however a limited parallelism capabilities and limited capabilities for interfacing with diverse, scalable computing infrastructures. The user can draw workflows in which multiple copies of a task are created to operate on multiple inputs at once, but this is not a scalable approach. Thus, integration with an existing highly distributed, portable system is desirable.

Swift [8] is a scripting framework for concurrent execution of ordinary programs. The Swift runtime contains a powerful platform for running user programs on a broad range of computational infrastructures, such as clouds, grids, clusters, and supercomputers, out of the box. Users rapidly prototype complex application flows on small, local resources and then deploy them on a diverse range of remote computation systems. Compact, C-like Swift scripts provide powerful semantics to exploit nonobvious concurrencies in complex application flows. The concise and powerful `foreach` loop can be used to generate thousands of concurrent tasks with just two lines of code. Thus, highly concurrent applications may be constructed in a small amount of script logic without explicitly managing process ids, batch queues, graph construction, or other tedious details.

Moreover, the Swift framework is specialized to run scientific computations on multiple, remote resources while being well interfaced with diverse schedulers, transport protocols, and security measures. Swift handles the data movement among distributed file systems and can leverage powerful data movement mechanisms. Swift works well with ssh, PBS, Condor, SLURM, LSF, SGE, Cobalt, and Globus to run applications and with scp, http, ftp, and GridFTP to move data.

In this paper, we report on our efforts to extend the Galaxy environment with parallel and distributed execution systems via the Swift framework. The result is an integrated environment with the same user-friendly Galaxy frontend and Swift as a backend. We motivate the integration by arguing that a combination of the best of both worlds will result in a powerful solution useful to the wider user community (§2). We support our argument with a description of various integration schemes and resulting benefits (§3). We describe

our experience with some real-world application implementations (§4). We provide perspectives on related systems by comparison with similar work (§5). Finally, we conclude with a summary and a description of current and future work (§6).

2. MOTIVATION

More and more computational resources are being acquired by organizations such as universities, national laboratories and research consortia to meet an increasing demand for computational and storage capabilities for big data analytics. Similarly, private owners such as Amazon, Google, and Microsoft have started offering resources in the form of clouds. Science users are likely to have allocations from many of these resources with heterogeneous architectures, accessibility modes, and resource managers. Thus, to effectively use each system in isolation or in combination, users need to expend additional efforts that can be a significant distraction and may require special expertise. Consequently, just the provision of resources is not sufficient. While the demand for computational resources is being met, a wide gap still exists between effective utilization of these systems by end-users. Portal-based systems such as Galaxy specialize in providing user-friendly computational analysis environments. Galaxy workflows largely run on local resources. Galaxy providers offer their own servers as execution platforms. As the number of users increases however, servers face spikes in demands. As a result, Galaxy offers a cloud-based solution such that users run their own Galaxy servers in cloud and manage cloud resources (e.g. via ad hoc Condor pools). Although community developments are aimed at running Galaxy tools on clusters, using them requires special tool configurations and constraints, for example, a shared file system between Galaxy server and the target cluster.
The following points summarize the motivation behind our conceptualization of the Swift-Galaxy integration.

- Benefits to users from the Galaxy community in interfacing their computations to large-scale parallel systems will include remote clusters, clouds, and supercomputers. An interactive distributed and parallel computing environment resulting in powerful capabilities of defining computations, orchestrating tasks to remote machines and monitoring the executions and results from a portal-based platform will give unprecedented power to Galaxy users.

- Swift users will be able to easily manage their applications and workflows. Swift users can share, reuse, and publish their distributed computations with Galaxy-style conventions and practices that are well adapted by a large user base. This can result in a wider community uptake of new computational codes and expansion of existing capabilities.

- The integration will result in a clear separation of concerns. Task coordination and orchestration will be handled by Galaxy while parallel execution on remote nodes and related data management will be handled by the Swift engine. Application developments benefit by this clean separation of responsibilities; distribution will be readily available without additional configuration on either side.

- Currently, the Galaxy server is operated on a single cluster at the University of Pennsylvania. With an expanding user community and increasing demand for computational resources, a broader capacity of resources and capability of execution will help meet this demand. This will result in a productive cycle where the growing demand in computation is met by deploying Galaxy servers on diverse systems, thus reaching wider science and engineering domains and resulting in more such deployments.

3. INTEGRATION SCHEMES

Both the Swift and Galaxy systems are powerful in different and complementary aspects, since they were designed with different goals in mind. Galaxy is a web-based portal for visual workflow composition, execution, and data analysis. Swift is a distributed computing framework for text-based workflow composition. The Galaxy environment is easy to adapt, simple, with limited control over resources. Swift is flexible, implicitly parallel, and robustly interfaced to a wide range of execution platforms. Clearly they form two different paradigms of computing for addressing the challenges faced in data science.
Swift and Galaxy workflows can interoperate because of their similar model of basic interactions with external systems.
A unit of execution in Galaxy, called a "tool", is defined by an XML document describing the command line, executable, interpreter to execute the command line, and any required inputs and outputs. The user controls tool inputs and outputs through the Galaxy workspace interface. Tools are composed together into workflows described internally as JSON [1] documents. The Galaxy engine processes the connections between the tools, identifies dependencies among tools, and schedules them for execution accordingly. Workflow execution results are saved as reproducible "histories", which may be shared among multiple users.
Both Swift and Galaxy operate on executables by invoking them via operating system utilities. The overarching goal of our integration is to enable a powerful environment by combining the strengths of each of the systems. We aim to achieve this with minimal modification of either systems. In other words, we take both systems "as is" and explore ways to integrate them. The basic mechanism we employ is to use Swift as the internal, low-level execution engine and to use Galaxy as the higher-level, external, user-visible framework.

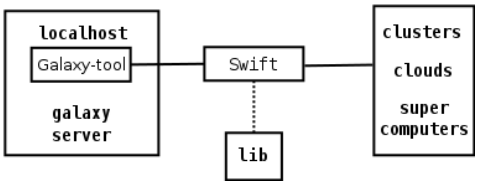

Figure 1: Swift acting as an interface between locally installed Galaxy tools and remotely located resources.

Figure 1 shows a general scheme of Swift's role in extending Galaxy to a wide range of scientific tools and computational

resources. Under this scheme, we are developing modalities through which a variety of Swift-enabled applications will be readily used as Galaxy tools. The execution starts from a local host as a local Galaxy run. The Galaxy tool invokes Swift application, using preconfigured application libraries written in Swift. The Swift engine in turn orchestrates and sends tasks to remote resources.
In the rest of this section we describe our experience in integrating the Swift and Galaxy gateways. We develop different schemes of integration and discuss the benefits of each.

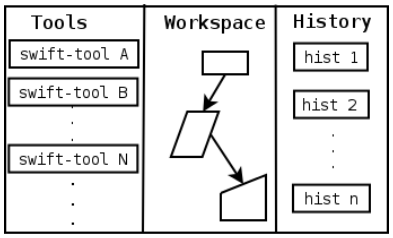

Figure 2: Swift-Galaxy integration view: custom Galaxy tools call Swift.

3.1 Scheme 1: Swift as Galaxy tool

As a first integration scheme, we developed a generic tool with a capability of executing user-provided arbitrary Swift scripts. The tool is set up such that users can select various configuration parameters from available choices. This capability allows users to provide arbitrary parameters specific to the application and the execution-specific parameters such as target compute site.
Figure 2 presents the user view of the Galaxy interface with preset, static Swift-wrapped tools that are available for use individually or as part of workflows. In addition to the tool's own configurable properties, Swift-specific properties allow users to choose the remote resource on which to run the tools with custom configuration such as desired degree of parallelism, and job distribution among sites. With this scheme users can run individual tools as Swift scripts, as well as stitch tools together running a workflow made up of other Galaxy tools and Swift tools.

3.2 Scheme 2: Translating Galaxy to Swift

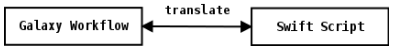

Figure 3: Interoperability between Galaxy workflows to Swift scripts.

Figure 3 shows a scheme where predefined Galaxy workflows are transliterated into Swift scripts via a post processing script. This development will enable interoperability between the two platforms. Galaxy workflows are currently expressed as JSON [1] documents linking the Galaxy tools, which are expressed as XML documents. Swift scripts are expressed as C-like programs. A two-way translator will enable the workflows to be expressed in either format thus being able to run under two environments. Additionally, the fact that Swift scripts are translated into intermediate XML representation makes this process easier in that no effort is required to write new modules in either of the workflow engines.

3.3 Scheme 3: Data splitting within tool

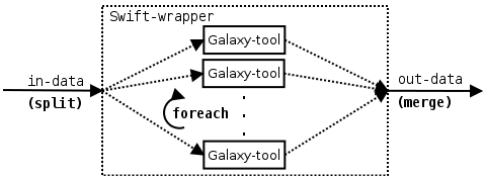

Figure 4: Swift wrapper to Galaxy tool. Input data from the Galaxy tool input is split, the base tool is run under a Swift `foreach` on each fragment, and the results are merged and presented as the Galaxy tool output.

Figure 4 shows a scheme of running Galaxy tools wrapped into a Swift `foreach` parallel loop. The scheme enables running ordinary sequential tools that perform homogeneous processing on a large dataset to run in parallel on a split dataset. The results of individual operations then get merged and emerge as output for consumption downstream. The scheme resembles the popular MapReduce computational paradigm. The implementation of this scheme is straightforward thanks to Swift's indexed arrays that map to files. The splits and merge on data can be organized into arrays that map to file fragments and can be processed in the Swift script as variables. For this tool to be successfully applied, the operations on split data must be associative.

4. EXPERIENCE

We describe our experience in running applications encoded as Swift scripts running on large-scale remote resources from within the Galaxy environment. In our work to date, we have run Swift-Galaxy applications on two institutional clusters and one XSEDE system: the University of Chicago Midway and UC3 clusters and the XSEDE Stampede cluster.
RCC Midway (*rcc.uchicago.edu*) is the University of Chicago Research Computing Center cluster supporting university-wide high-end computational needs. The cluster has multiple resource partitioning dedicated to specialized computing such as HPC, HTC, and GPU computing and runs a SLURM batch queue scheduler. The UC3 cluster (*uc3.uchicago.edu*) at the University of Chicago is an open computing framework for connecting users to shared distributed high-throughput computing resources, both on and off campus. The cluster runs Condor DRM capable of flexibly extending compute resources to the OSG environment. XSEDE (*www.xsede.org*) is an NSF-funded, national cyberinfrastructure comprising multiple large-scale computation systems on sites across the United States. Stampede is one of the supercomputing systems offered by XSEDE. Stampede runs the SLURM scheduler for submitting user jobs.
Figure 5 shows a screen capture of an application workspace interface. The drop-down dialog allows users to select resources for a run. The rest of the text dialog provides an interface to enter application-specific values. Parameter values provided through this interface are passed to Swift via the Galaxy tool runner. Galaxy runs the tool that in turn

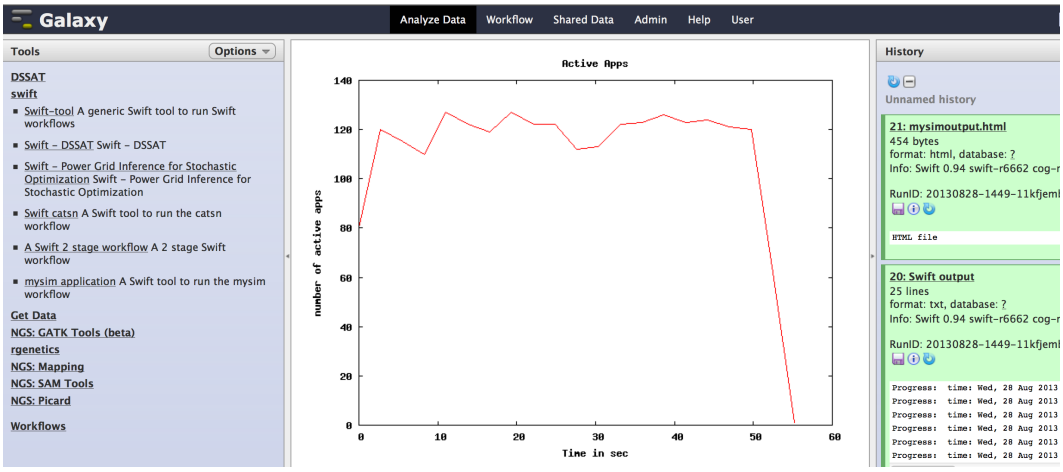

Figure 6: Screen capture of a post execution view of a Swift tool. The plot shows the number of parallel active jobs during the run of a 2000-task application.

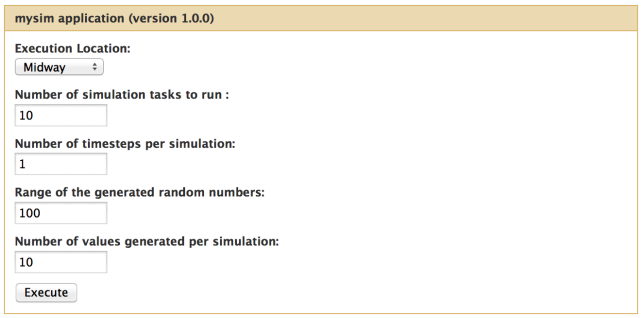

Figure 5: Screen capture of the Swift Galaxy tool showing the tool workspace options: target execution site and application specific options.

calls a Swift wrapper that in turn runs the application and relinquishes control back to Galaxy after the execution is completed. Figure 6 shows a screen capture of a post execution view for a Swift tool. The plot shows the number of parallel active jobs during the run of a 2000-task application on Midway. More than 120 active parallel jobs run at once over a period of 55 seconds to complete the 2,000 tasks. On the right pane is the history of this run captured by Galaxy that shows the standard output and an HTML page that has links to the results (not shown in figure) of the execution.
A brief description of applications enabled and ongoing work with the Swift-Galaxy interface follows.

Power Grid Analysis.

Large-scale analysis of power grid quantities is emerging as an important engineering application of national interest. Applications have many stakeholders, including the government, power companies, ISOs, and domestic and industrial consumers. In this work we enable an application concerning inference analysis of stochastic unit commitment models. The application involves hierarchical stochastic computations over a range of samples and batches. The code in Listing 1 shows a Swift script representation of the core application logic. A three-level-deep nested `foreach` loop generates about 100K concurrent tasks for a sample size of 5 and a batch comprising sizes (10, 20, and 30). Currently we implement the application under *Scheme 1* described above running on institutional and XSEDE resources.

```
1  foreach S, idxs in nS{
2   o = gensample (wind_data);
3   foreach B, idxb in [10:30:10]{
4    foreach k in [0:B]{
5     o = gensample (S, wind_data);
6     obj_out_l[idxs][idxb][k] = ampl_app_L(params);
7     o = gensample (S, wind_data);
8     obj_out_u[idxs][idxb][k] = ampl_app_U(params);
9  }}}
```

Listing 1: Swift code for power grid analysis application.

Climate Models.

The Decision Support System for Agrotechnology Transfer (DSSAT) application analyzes the effects of climate change on agricultural production [4]. Projections of crop yields at regional scales are carried out by running simulation ensemble studies on available datasets of land cover, soil, weather, management, and climate. The computational framework starts with the DSSAT crop systems model and evaluates this model in parallel. One study involves analysis of climate impact for a single crop (maize) across the conterminous USA (120K cells) with daily weather data and climate model output spanning 120 years (1981-2100) and 16 different configurations of fertilizer, irrigation, and cultivar choice. Currently, the climate model runs as a single, generic Galaxy tool wrapped via a shell script and callable by Galaxy as a tool.

Genomics.

A large section of the Galaxy community is geared toward genomics data analysis, mainly next-generation sequencing data (NGS). Many tools working on all aspects of genomics analysis are currently used, and many new tools are being developed. The analysis of NGS data can be parti-

tioned into a series of steps common among the multiple NGS applications (i.e., transcriptome, whole-exome, whole genome). One of the most popular applications for accurate variant calling of exome data is the Genome Analysis Toolkit (GATK) from the Broad Institute [2], which provides a best practices pipeline with optimal parameters in each step of the process. The pipeline includes running a number of GATK applications that can be computationally demanding and in some cases can be parallelized. As an ongoing development, a Swift-enabled GATK best practices pipeline is being integrated into Galaxy. We are currently porting the application using a combination of *Scheme 1* and *Scheme 3* described in §3.

The *Unified Genotyper* (UG) tool analyzes multiple chromosomes. By splitting the analysis by chromosome, we intend to run a `foreach` loop over chromosomes. The workflow consists of 123 input human genome files containing 23 chromosomes each. Thus UG can be run 23 times for individual chromosomes.

5. RELATED WORK

In this section, we briefly present our perspectives on related work. Many scientific gateways and portals are in use globally. Historically, problem-solving environments (PSEs) can be considered as ancestors of portal-based science gateways. With the advent of fast networks and sophisticated web programming technologies, PSEs have evolved into portals.

The driving force behind many separate developments can be attributed to user communities bonded by a scientific domain or adapted to a particular set of tools.

The GPSI [7] portal is a prototype portal using Swift as its workflow engine. A Python-driven web environment allows users to submit their Swift scripts to distributed resources and to manage their data from within the web-based environment.

Integration of the Vbrowser portal with the MOTEUR workflow engine [6] is another example of similar work geared toward combining network-based portals with a task orchestration engine. The target execution infrastructure in this case is the European Grid environment, and the target community is the medical imaging community. Nanohub [5] is another recent portal focused on the nanotechnology community.

6. CONCLUSIONS AND FUTURE WORK

In this paper we motivate the benefits of integrating portal-based execution frameworks with a large-scale parallel programming framework. We demonstrate the integration with Galaxy as a representative of the former and Swift of the latter.

The following points summarize the features of Swift integration with Galaxy:

- A generic Galaxy tool with the ability to run arbitrary Swift scripts from within a Galaxy interface.
- Static application tools for special-purpose execution with flexibility of user-provided application parameters, and execution site.
- Capability to arbitrarily select remote execution sites from a drop-down list of the tool's visual representation.
- Post execution analysis of runs and plots of job completion rate and parallel tasks.
- Capability to bring results into the Galaxy dataset system for reuse and publishing.

The following activities summarize our future work.

- Extending and combining of the integration schemes into a toolkit capable of encapsulating arbitrary Galaxy tools into Swift.
- Finer integration of external Swift data and managed Galaxy dataset system.
- Deeper integration of the Swift engine into the Galaxy framework. This will enable Galaxy workflows to be orchestrated as a Swift execution, retaining the Galaxy visual framework.

Acknowledgments

This work was supported in part by the NIH through the NHLBI grant: The Cardiovascular Research Grid (R24HL085343) and by the U.S. Department of Energy under contract DE-AC02-06CH11357. We are grateful to Amazon, Inc., for an award of Amazon Web Services time that facilitated early experiments. We thank Gail Pieper for proofreading help.

7. REFERENCES

[1] D. Crockford. RFC 4627: The application/json media type for JavaScript Object Notation (JSON). 2006.

[2] GATK Best Practices. www.broadinstitute.org/gatk/guide/best-practices.

[3] B. Giardine, C. Riemer, R. C. Hardison, R. Burhans, L. Elnitski, P. Shah, Y. Zhang, D. Blankenberg, I. Albert, J. Taylor, et al. Galaxy: a platform for interactive large-scale genome analysis. *Genome research*, 15(10):1451–1455, 2005.

[4] J. W. Jones, G. Hoogenboom, P. Wilkens, C. Porter, and G. Tsuji, editors. *Decision Support System for Agrotechnology Transfer Version 4.0: Crop Model Documentation*. University of Hawaii, 2003.

[5] G. Klimeck, M. McLennan, S. P. Brophy, G. B. Adams, and M. S. Lundstrom. nanohub.org: Advancing education and research in nanotechnology. *Computing in Science & Engineering*, 10(5):17–23, 2008.

[6] S. D. Olabarriaga, T. Glatard, and P. T. de Boer. A virtual laboratory for medical image analysis. *Information Technology in Biomedicine, IEEE Transactions on*, 14(4):979–985, 2010.

[7] T. D. Uram, M. E. Papka, M. Hereld, and M. Wilde. A solution looking for lots of problems: Generic portals for science infrastructure. Salt Lake City, UT, June 2011. ACM.

[8] M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz, and I. Foster. Swift: A language for distributed parallel scripting. *Par. Comp.*, 37:633–652, 2011.