Enabling Multi-task computation on Galaxy-based Gateways using Swift

Ketan Maheshwari*, Alex Rodriguez†, David Kelly*†, Ravi Madduri*†,
Justin Wozniak*†, Michael Wilde*†, Ian Foster*†
*MCS Division, Argonne National Laboratory, Argonne, IL 60439
†Computation Institute, University of Chicago & Argonne National Laboratory

***Abstract*—The Galaxy science portal is a popular gateway to data analysis and computational tools for a broad range of life sciences communities. While Galaxy enables users to overcome the complexities of integrating diverse tools into unified workflows, it has only limited capabilities to execute those tools on the parallel and often distributed high-performance resources that the life sciences fields increasingly requires. We outline here an approach to meet this pressing requirement with the Swift parallel scripting language and its distributed runtime system. Swift's model of computation – implicitly parallel functional dataflow – is an elemental abstraction to which the core computing model of Galaxy maps very closely. We describe an integration between Galaxy and Swift that is transforming Galaxy into a much more powerful science gateway, retaining its user-friendly nature while extending its power to execute highly scalable workflows on diverse parallel environments.**

Science gateways have high potential to accelerate productive computations for science. They support users by allowing them to perform computations via tools, workflows and data with little managerial overheads from a simplified and unified environment such as a web browser. The Galaxy [2] gateway is one such popular environment used, supported and developed by a large life sciences users community. Galaxy effectively handles many of the challenges faced by computational scientists by providing ready-to-use, interactive environment for data analysis, management and computational operations.

Swift [5] is a parallel scripting language specifically created to execute multiple instances of ordinary programs concurrently on diverse parallel and distributed resources. Swift is a powerful programming model for expressing and orchestrating the kinds of workflows that Galaxy users create. Swift specializes in rapidly creating complex application flows and readily executing them with a diverse range of remote computation systems. Arbitrary highly-concurrent workflow patterns can be composed via Swift script's set of programming constructs.

Swift offers a scripting language which enables expression of a parallel and distributed execution environment with capabilities to encode arbitrary computations into workflows and run on multiple remote environments via a standard Linux command-line. Compact, C-like Swift scripts provide powerful semantics to extract non-obvious concurrencies from complex application flows.

Within the familiar abstraction of the Linux (POSIX) environment, concise Swift scripts can result in rapid prototyping at a small scale and a large-scale execution without loss of computational logic expressed in the script. It transparently performs data movement and supports a wide variety of resource managers, distributed middleware and data transport services.

In this paper we describe our work in progress to integrate the Swift and Galaxy gateways. This effort is producing a powerful platform with access to a broader community and significantly more computational resources. We view the result of this integration as an environment which will enable and support parallel and distributed computations while providing the same portable, user-friendly and simplistic experience to end-users. We describe a set of integration levels and our experience implementing and using them. We report on the current developments and future paths for a science gateway leading to an environment which is expected to benefit the broad Galaxy user-base and apply well to many similar portals and gateways. Our experience to date shows that such a gateway will enable user communities to readily leverage distributed parallel resources without changing the model of workflow execution and provenance tracking that has made Galaxy so productive for them.

I. Motivation

In this section we describe the motivation behind our concept of adding implicitly parallel and transparently distributed workflow to Galaxy, and discuss the expected benefits of the integration of Swift into Galaxy gateways.

The general notion of science gateways includes web-based portals or web-oriented access as an integral part of user's interaction and experience. While simple and easy to adapt, a web interface can often be restricted in the ways operations can be performed over it. This can be considered a conscious design trade-off to lower the adoption barrier and avoid steep learning curves for non-programming user communities.

While being simple in user interactions and processing operations Galaxy currently has limited capabilities with respect to interfacing with large scale computational systems and running workflows and tools in a parallel and distributed manner. With respect to execution, Galaxy workflows are limited to either single machine multi-threaded systems or are interfaced to limited large-scale systems via an ad-hoc LRM/DRM (Local/Distributed Resource Manager) interface such as PBS, SGE and Condor pool. To the best of our knowledge the Galaxy DRM capabilities are limited to a single environment with local shared file system requirements.

Galaxy allows users to manage their data, tools and workflows in an easily accessible web-based environment. Galaxy provides server space for users to store the 'histories' of their

computations. The interface allows easy management of such histories.

However, scientific users often need more control over how and where their computations should be executed. A standard text-based Swift-enabled scientific gateway could be thought of as a powerful and flexible access point to a broader variety of computational resources for scientific tools and libraries. Such a gateway will have all the necessary tools implemented in the form of commands and command-line parameters executable from a standard Linux terminal.

These capabilities when done well will provide the power of running large scale analyses in parallel with the flexibility of the web-based portal environments so that the end user, researcher does not have to become an IT expert in order to perform science at a larger scale.

Some of the distinct benefits of such an environment will be as follows:

- An ability to steer computations to multiple, independent resources.
- An ability to add new resources to the deployment profile of existing workflow.
- An ability to run a particular analysis through a set of files or running a set of analyses in parallel.

Swift users who are already familiar with the Swift execution framework will be able to better manage their computations and results via Galaxy interface. Galaxy users will benefit from Swift's capabilities of interfacing with a broader computational systems. A generic Swift execution tool running arbitrary Swift scripts from within Galaxy interface will allow test and prototyping. Static application tools for special purpose execution with flexibility of user provided application parameters, execution sites will let users experiment with their allocations.

II. Swift-Galaxy Integration

Swift and Galaxy workflows can interoperate because of their similar model of basic interactions with external systems. Both operates on executables by invoking them via operating system utilities. The overarching goal of integration is to enable an environment by combining the strengths of each system. We aim to achieve this with minimal invasion and modification on either of the systems. The basic mechanism we employ is to use Swift as internal, low-level execution engine while Galaxy as the higher-level, external, user-visible framework.

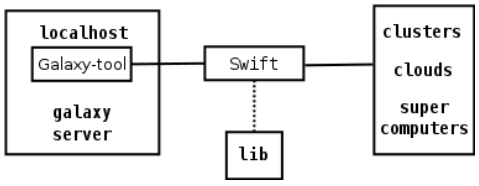

Fig. 1. Swift acting as an interface between locally installed Galaxy tools and remotely located resources

Figure 1 shows a general theme of Swift's role in connecting Galaxy to a wide range of scientific tools and resources. Under this scheme, we are developing modalities through which a variety of Swift enabled tools can be readily used as Galaxy tools. Galaxy tool is executed as Swift script using Swift-enabled scientific tools and libraries.

In the rest of this section we describe our experience with the development concerning the integration of Swift and Galaxy gateways. We develop different 'schemes' of integration and discuss benefits of each.

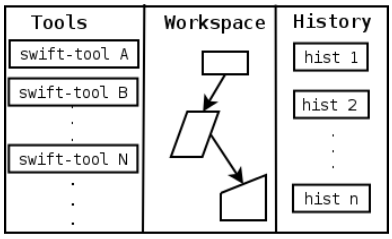

Fig. 2. Swift-Galaxy integration view: Custom Swift wrapped galaxy tools use swift runner

The basic working unit in Galaxy is a "tool". Tools run as custom execution units for existing program interpreters or "runners" such as shell, perl and python. As a first integration scheme, we developed a generic tool with a capability of execution of user provided arbitrary Swift scripts. The tool is set up such that user can select various configuration parameters from available choices. This allows users to provide arbitrary parameters specific to the application and the execution specific parameters such as target computational site.

Shown in figure 2 is a user's view of Galaxy interface with preset, static Swift wrapped tools that are available for use individually or as part of workflows. In addition to tool's own configurable properties, Swift specific properties allow users to chose on which remote resource to run the tool with custom configuration such as desired degree of parallelism, job distribution among sites and so on. With this scheme users can run individual tools as Swift scripts as well as can stitch tools together running a workflow made up of other Galaxy tools and Swift tools.

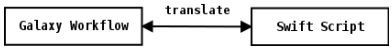

Fig. 3. Interoperability between Galaxy workflows to Swift scripts

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 gateways. Galaxy workflows are currently expressed as JSON [1] documents linking the Galaxy tools which are expressed as XML documents. Swift scripts are C-like expressions which are translated into an XML syntax by the Swift compiler. This poses an opportunity to translate and express Galaxy workflows as Swift representation and vice versa. A two-way translator will enable the workflows expressed in either formats thus being able to run under two environments.

Complex workflows with a large number of tasks often lose visual appeal and can be hard to understand. A readable and searchable textual representation aids well in understanding the flows and links of a complex multistage computation.

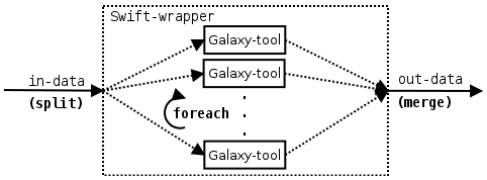

Fig. 4. Swift wrapper to Galaxy tool splitting data, running the tool under a swift `foreach` and merging the results

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 large dataset to run in parallel on a split dataset. The results of individual operations then get merged into a final result from the tool 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. One requirement however for this tool to be successfully applied would be that the operations on split data are associative of each other.

III. Data Management

Most application life cycles begin with raw and unprocessed data and end with processed data in the form of results. Data management is often the central part of automated systems. Both Galaxy and Swift offers various advantages to its users with respect to data management. Swift allows expressing disc resident or remote data directly into scripts and operate upon it like program variables. Swift implements the notion of data providers which are interfaced with various data movement protocols and can manage data motions at runtime without manual intervention.

Galaxy provides a user-friendly way of managing data and results. A user can import external data into Galaxy's persistent dataset system and operate upon it for computations. Galaxy's engine supports multiple custom data types and formats and can recognize them for operations, storage and display purposes. Additionally Galaxy supports data transfer protocols such as FTP and HTTP. Swift's data providers handle data movement via TCP sockets and managed APIs such as GridFTP and a support of GO is under development.

IV. Interface to Computational Infrastructures

Swift framework implements 'providers' suitable to interface with a wide variety of computational systems. HPC systems such as supercomputers have customized and optimized implementation of LRM/DRM. These systems employ schedulers and middleware in order to access the computational resources. Additionally, accessing the systems involves different modes often dictated by administrative policies of the system. Some are accessed from local head nodes directly while the others are accessed via remote machines using ssh based connections. Furthermore, there are general purpose clusters, clouds and bag of workstations which are accessed directly via machine addressing. These heterogeneous modalities results in complex resource access patterns. Swift's providers implements codes that handle these modalities for LRM/DRM, clusters and cloud-based systems simultaneously. Swift's pilot job implementation–coasters [3] can efficiently manage resource queues and schedule a group of tasks via jobs submitted as pilot agents.

Both Galaxy and Swift use cloud resources as one of their execution platforms. Clouds have played a crucial role in recent advancement of Galaxy usage. Galaxy offers Cloud managers in order to configure user-owned Galaxy server and run both the server and computations on cloud resources. This results in more user control and less load on the main Galaxy server. Swift's coaster execution providers have been shown to be useful cloud execution providers [4]. A cloud based Galaxy environment thus naturally suits Swift driven executions.

Swift provides structured and managed interfaces to such systems exposing configurable properties to users. The integration of Swift and Galaxy thus enables an aggregation of support for these clusters into Galaxy environment. As a result making use of these resources at large-scale a turnkey solution.

V. Conclusions and Future Work

In this paper we describe the benefits of an integration of 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 that of the latter. Such an integration will bring together an existing community of each of the platforms and will provide a turnkey solutions to usage and adaptation to distributed computing resources.

A deeper Swift-Galaxy interoperability effort is underway which will result in a robust translation implementation in the future. We are exploring possibilities for data interoperability between the two platforms via extension of Swift features (e.g. variable to file mappers) that will integrate with Galaxy dataset. The result will be a capability of operating upon Galaxy data by Swift's providers.

VI. Acknowledgments

This work was supported by the U.S. Department of Energy, Office of Science, under Contract DE-AC02-06CH11357.

References

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

[2] 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.

[3] M. Hategan, J. Wozniak, and K. Maheshwari. Coasters: uniform resource provisioning and access for scientific computing on clouds and grids. In *Proc. Utility and Cloud Computing*, 2011.

[4] K. Maheshwari, K. Birman, J. Wozniak, and D. V. Zandt. Evaluating cloud computing techniques for smart power grid design using parallel scripting. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, 2013.

[5] 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.