Galaxy - a Gateway to Tools in e-Science

Enis Afgan[1], Jeremy Goecks[1], Dannon Baker[1], Nate Coraor[3], The Galaxy Team[2], Anton Nekrutenko[3], and James Taylor[1]

[1]Department of Biology and Department of Mathematics & Computer Science, Emory University
[2]http://galaxyproject.org
[3]Huck Institutes of the Life Sciences and Department of Biochemistry and Molecular Biology, The Pennsylvania State University

Abstract
eScience focuses on the use of computational tools and resources to analyze large scientific datasets. Performing these analyses often requires running a variety of computational tools specific to a given scientific domain. This places a significant burden on individual researchers for whom simply running these tools may be prohibitively difficult, let alone combining tools into a complete analysis, or acquiring data and appropriate computational resources. This limits the productivity of individual researchers and represents a significant barrier to potential scientific discovery. In order to alleviate researchers from such unnecessary complexities and promote more robust science, we have developed a tool integration framework called Galaxy; Galaxy abstracts individual tools behind a consistent and easy to use web interface to enable advanced data analysis that requires no informatics expertise. Furthermore, Galaxy facilitates easy addition of developed tools, thus supporting tool developers, as well as transparent and reproducible communication of computationally intensive analyses. Recently, we have enabled trivial deployment of complete a Galaxy solution on aggregated infrastructures, including cloud computing providers.

Keywords: Galaxy, bioinformatics, tool execution framework, cloud computing

1 Introduction

Rapid growth in both the production and the availability of scientific data has revealed the inadequacy of existing data analysis approaches, technologies, and tools. This is particularly true in experimental biology, where the volume of data produced by new technologies confounds the average experimentalist [1]. New tools and techniques are continually being developed to analyze this data. For the domain scientist, use of these newly available tools and technologies is often preceded by a steep or prolonged learning curve: discovering the most applicable

tools, deploying them on appropriate computational resources, learning new user interfaces, etc. Also, arriving at the desired results often requires the domain scientist to become familiar with several tools and compose those into an appropriate workflow. Because of the differences in tools and tool interfaces, this task can easily result in additional complexities that need to be addressed by the scientist directly (*e.g.*, developing wrappers to format tool input/output). Thus, in addition to the development of data analysis tools, there is a need for supporting tools and frameworks that make it possible to utilize otherwise available or upcoming cyber-infrastructure to process the data and deliver results directly to users [2], [3].

Users of such systems are often domain scientists focused on specific research problems that posses domain-specific knowledge but lack interest and knowledge to write computer programs. To support the scientific process, it must be possible for these scientists to share newly available scientific data and results with the remainder of the community. Critical to the scientific review process, these results should be easily verified and reproduced by others. When it comes to sharing results, thus far, scientists have primarily used the scholarly publishing process to announce their findings. However, the traditional publication process is not well suited for publishing the details of data intensive computational analysis in a completely reproducible manner. In the e-Science context, and with the rapid advances in data discovery, scientists seek to publish their results quickly and continuously in a medium that allows analysis to be expressed more naturally, leading to a shift toward web-based, lightweight publication methods such as wikis and blogs.

Because of these changes in the scientific process, there is a significant need to provide research scientists with streamlined access to a variety of domain specific tools that can be easily integrated and shared with others in the community. These are the aims of e-Science; they focus around delivery and sharing of highly focused solutions directly to domain scientists. Ideally, this enables a scientist to focus on her immediate work without requiring them to learn and understand how the underlying technology operates.

Here we describe an open-source software system, called Galaxy[1] [4], [5], that addresses many of the deficiencies associated with advancement of e-Science outlined above and facilitates scientists' focus on their domain. Galaxy provides an integrated analysis environment where domain scientists can interactively construct multi-step analyses, with outputs from one step feeding seamlessly to the next. The user interfaces are intuitive and consistent, requiring minimal user training and effort. Any software tool can easily be made available through Galaxy, and the underlying data management and computational details are completely hidden from the user, even when dealing with large-scale datasets and high-performance computing resources. The environment transparently tracks every analysis detail to ensure reproducibility, and provides a workflow system for constructing reusable analysis, as well as an intuitive interface for constructing workflows from existing analysis. The Galaxy data library system provides secure

[1] http://galaxyproject.org

data management and sharing, with fine-grained access controls and extensive metadata tracking. Thus, Galaxy provides an ideal solution for immediately delivering, integrating, and sharing e-Science tools directly to domain scientists, enabling scientists to focus on performing computational analyses rather than setting up and configuring computational tools.

The fundamental concept underlying Galaxy is a framework for tool integration. It focuses on accessibility, expandability, and result reproducibility. With that, it trades off some expressiveness in terms of user features for usability. Users may not possess the flexibility of complete programming language but they gain the ability to easily, rapidly, and consistently access a variety of domain specific tools. Galaxy achieves this by conceptualizing domain specific tools as an abstraction for doing data analysis. For domain scientists from historically non-computational fields (e.g., biology, psychology, sociology), this approach makes it possible to perform complex and reproducible analysis without programming or software engineering expertise.

With the continuous growth of the analysis data and thus increased computational requirements, alongside the usability features realized by Galaxy, there is a need to transparently provision necessary computational infrastructure. Galaxy supports this computational infrastructure demand at two levels: (1) by handling integration with an available workstation or a cluster and (2) by providing a ready solution to deploy Galaxy on aggregated computational infrastructures. This approach makes Galaxy extremely flexible and enables it to meet the needs of a spectrum of users, including tool developers, individual researchers, or entire labs.

This chapter describes the Galaxy framework in the context of a bioinformatics tool set and then generalizes the framework toward a tool execution and data analysis framework for any computational domain. Architecture, interface, and use cases are described, highlighting Galaxy's main features. Lastly, current and future directions for Galaxy are described, focusing on job execution across distributed computing infrastructures. Leveraging distributed computing enables Galaxy to process increasingly large volumes of data using ever-larger resource pools.

2 Galaxy - a Tool Integration Framework

Galaxy is a software framework for making computational tools available to users without informatics expertise. It is available as a self-contained, portable, open-source software package, which can easily be deployed on existing computational resources. It provides a web-based interface for interactively performing analysis using computational tools. Galaxy provides abstractions to make it easy to integrate almost any computational tool: any program that can be run from the command line can be integrated by providing an abstract description of how the program works (its parameters and the type of data it expects and generates). This serves both **researchers**, for whom a simple and intuitive user interface is auto-

matically generated, and **tool developers**, who can now rapidly deploy their tools in a way that their experimental colleagues can actually use. Galaxy can also integrate other sorts of tools (e.g., external database interfaces or other web services).

Using the concept of tools as discrete units of analysis with well-defined parameterizations – the Galaxy framework provides a powerful web-based environment for performing complex analyses (Figure 1). Analysis can be performed interactively, with Galaxy keeping a complete "history" that tracks every step of the analysis. Thus, unlike other approaches [6] that rely on careful engineering practices to ensure transparency and reproducibility, in Galaxy the analysis framework automatically ensures that every detail of every step of an analysis is recorded and can be inspected later. Galaxy also includes a workflow system for constructing more complex analyses that execute multiple tools, using the output of one tool as the input of another tool. Workflows can be automatically extracted from histories, providing a way to rerun and verify the reproducibility of any existing analysis. Histories and workflows can be shared in a variety of ways, including as part of a publication.

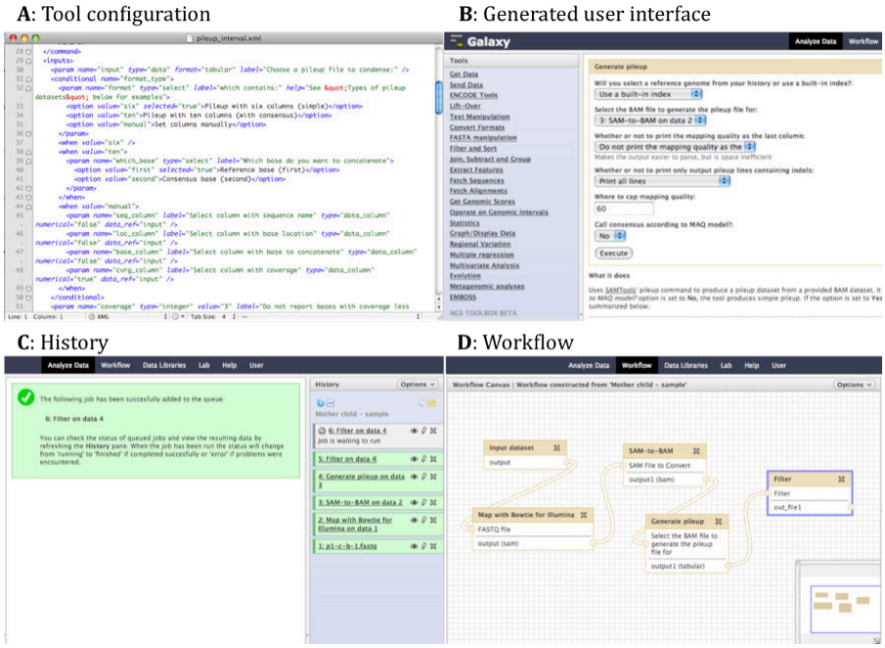

Figure 1. An overview of analysis in Galaxy: (**A**) Tool developers describe their software using Galaxy tool configurations. (**B**) Galaxy generates a user interface for the tool. (**C**) Users provide parameters for the tool, Galaxy manages tool execution and stores all parameters and results in the history. (**D**) The workflow system allows multi-step analysis to be reproduced, modified, or created from scratch.

2.1 Galaxy Goals

The aim of the Galaxy project is simultaneous impact on individual researchers that want to simply use computational tools and on tool developers that want to publicize their tools while minimizing tedious overhead work associated with the necessary tasks (e.g., data source integration, UI development). Thus, the design of Galaxy relies on the following principles:

Accessibility: The most important feature of Galaxy is the ability for users to access and use a variety of domain specific tools without a need to learn implementation or invocation details of any one tool nor worry about underlying infrastructural details. Galaxy enables users to perform integrative analyses by providing a unified, web-based interface for obtaining genomic data and applying computational tools to analyze the data. Accessibility is further supported by enabling users to import datasets into their workspaces from many established data warehouses or upload their own datasets. Lastly, full functionality of Galaxy is immediately accessible from a public web server[2], available for download and simple local installation (see Section 3.1), or readily deployable on aggregated infrastructures thus sustaining any level of desired hardware support (see Section 3.4).

Simplicity of extension: A key feature of the Galaxy framework is the ease with which external tools can be integrated. Galaxy is completely agnostic to how tools are implemented – as long as there is a way to invoke a tool from a command line, a tool could be written in any language whether interpreted (e.g. Perl, Python, R) or compiled (e.g. C, Fortran). Galaxy requires only a high level description of what parameters a user can provide for the tool, and how to construct a command line or configuration file based on those parameters. Note that when a user writes a tool configuration for Galaxy, they are not implementing a user interface; they are describing the abstract interface to that tool, which Galaxy can then use to adapt that tool to a specific user interface. As a result of such flexibility, functionality of Galaxy is not limited but can be expanded and adopted not only to a specific researcher or a lab but even to a specific domain.

Reproducibility of results: A key to good science is openness and thus the ability for others to repeat and validate derived results. Galaxy supports reproducibility by tracking derivation of users' analyses, by supporting the ability to repeat a given analysis (or part of thereof), as well as by effectively enabling tagging, annotation, and sharing of complete analyses. Such functionality enables a given analysis not only to be repeated but to also be given context and necessary description explaining underlying reasoning for a particular analysis step and to be easily shared with others.

---

[2] http://usegalaxy.org/

2.2 Galaxy components

As discussed in the previous section, Galaxy represents an integrated system that focuses on analysis accessibility, tool integration, and support for reproducibility of obtained results. To ensure flexibility and extensibility, Galaxy's architecture is highly modular, and is built for a set of distinct low-level components. These individual components operate at different levels and are not necessarily simultaneously utilized. Instead, when combined, they provide the integrated solution Galaxy has become known for. Figure 2 depicts these components.

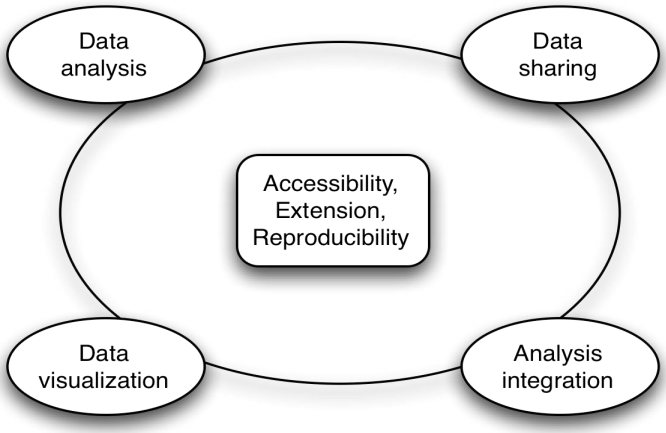

Figure 2. Symbolical representation of components that together enable Galaxy aims.

2.2.1 Data Analysis

The primary interface to a Galaxy instance is through the web (although the Galaxy framework allows other interfaces to be implemented; see Section 2.3). The Galaxy *analysis workspace* contains four areas: the upper bar, tool frame (left column), detail frame (middle column), and history frame (right column). The upper bar contains user account controls as well as help and contact links. The left frame lists the analysis tools and data sources available to the user. The middle frame displays interfaces for tools selected by the user. The right frame (the history frame) shows data and the results of analyses performed by the user. Pictured here are five history items representing one dataset and following analysis steps. Every action by the user generates a new history item, which can then be used in subsequent analyses, downloaded, or visualized. The Galaxy history page can display results from multiple genome builds, and a single user can have multiple histories. Figure 3 provides an overview of the analysis workspace.

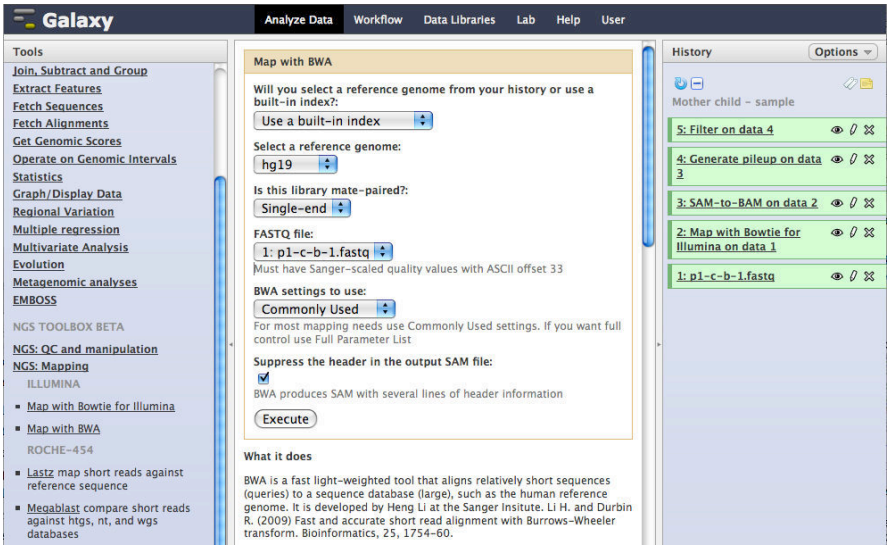

Figure 3. A screenshot of current Galaxy web interface.

The Galaxy *history* system stores all data uploaded by the user or generated by the analysis tools. Figure 4 illustrates how the history works. Each analysis step (Bowtie mapping [7] and format conversion in this case) generates a new history item, leaving the original datasets intact. Thus if the user makes a mistake or wants to experiment with different parameter settings, he/she can always go back to the original data.

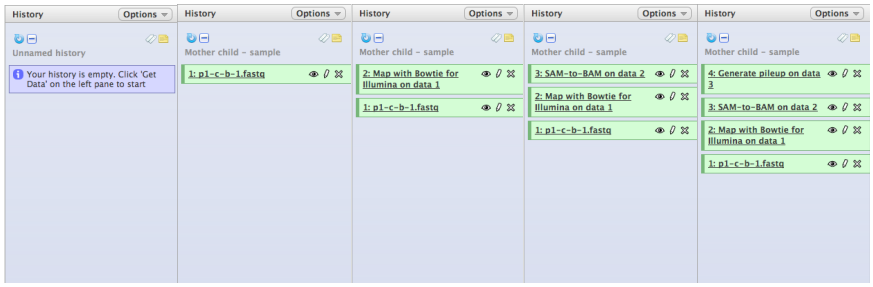

Figure 4. Galaxy history stores uploaded data and analysis results. Original dataset is always preserved and every subsequent analysis adds a new entry into the history pane.

The ability to perform an analysis in a step-by-step fashion and then be able to simply repeat any part of it (using the same data and parameters or changing those) represents a significant step forward in terms of enabling analysis and real-

izing reproducibility. However, due to the nature of the research problem, it is often the case that the same analysis needs to be repeated using different data but the same procedure (possibly using slightly different parameters). In order to enable effective realization of easy reuse, Galaxy supports the notion of *workflows*. The main feature of Galaxy that makes this possible is the structured abstract descriptions of tool interfaces. A Galaxy workflow consists of a specification of a series of tool invocations, where the input of a tool can be connected to the output of any previous tools. Input datasets and parameters are either specified in the workflow, or left undefined, in which case they become the inputs for invoking the workflow.

To make workflow construction as intuitive as possible for non-programmers, Galaxy allows workflows to be constructed from existing analyses. At any time after performing some analysis, a user will be able to extract a previous chain of analysis steps as a workflow (Figure 5A). Once extracted, the workflow can then be modified and run with different parameters or different starting datasets.

In addition to creation of workflows by example from existing histories they can also be created from scratch using the Galaxy workflow editor with an interactive graphical interface. A graphical editor is provided in which any tool can be added to the "canvas" (Figure 5B). A series of tools is then connected by links representing the flow of data. The workflow editor is aware of which tools can be chained together: if the output of tool A is compatible with the input of tool B, these two can be chained together.

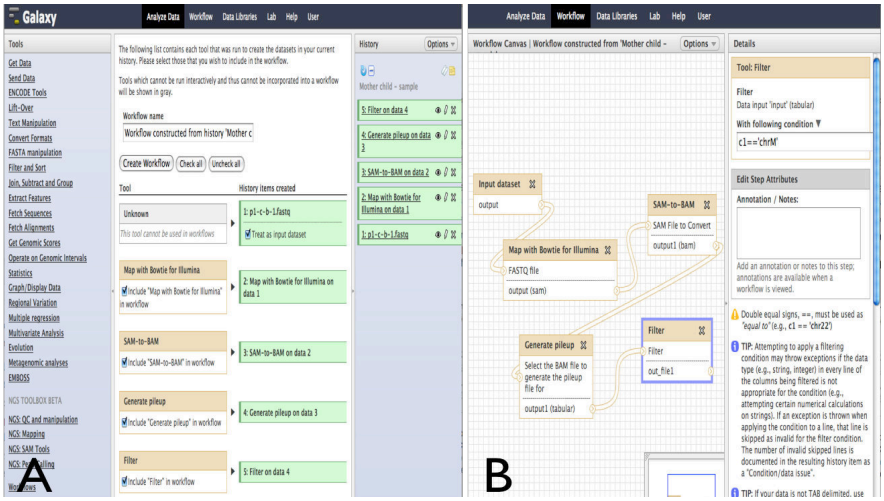

Figure 5. Two ways to create workflows in Galaxy: (A) Shows the interface for constructing a workflow from an existing history, (B) Shows the workflow editor for explicit workflow construction and editing.

2.2.2 Data Sharing

At the core of accessibility and reproducibility is the ability to share one's findings in a way that is both useful and usable to others. Within Galaxy, sharing is supported from the ground up and is tightly integrated into user's experiences, facilitating the transition from solitary analysis to collaboration. Users can share their findings at multiple levels and with varying scopes: a user can share individual datasets, parts of an analysis, an entire analysis, as well as analysis conclusions. This wide range of sharing options is supported through the following set of components: history sharing, tagging, annotations, data libraries, and Galaxy Pages.

An individual can share his history with specific Galaxy users, make it available via a persistent web link (thus not requiring a Galaxy account), or publish it, enabling all users to access it via a web link and find it via search. A user viewing a history shared with them can simply inspect the history or choose to import it into her local workspace and manipulate it as desired. Internal to Galaxy, history sharing is enabled by associating a URL with each history element followed by implementation of a broad set of access policies.

Galaxy also supports *tagging* (or labeling)—applying words or phrases to describe an item. Tagging has proven very useful for categorizing and searching in many web applications. Galaxy uses tags to help users find items easily via search and to show users all items that have a particular tag. Tags support reproducibility because they help users find and reuse datasets, histories, and analysis steps; reuse is an activity that is often necessary for reproducibility. Along the ideas of sharing, Galaxy implements notion of community tags; a user can choose to tag a dataset with a set of tags that will be visible and searchable by anyone using Galaxy or simply use personal tags that are accessible only to the given user.

Along with tagging, Galaxy supports *user annotations*—descriptions or notes about an analysis step. Annotations are a critical component of reproducibility because they enable users to explain *why* a particular step is needed or important. Automatically tracked metadata (i.e., history elements) records what was done, and annotations indicate why it was done, facilitating reproducibility by recording both the details and the context of an analysis. Galaxy users can annotate a complete history or workflow and can annotate individual history or workflow steps. Annotations are visible when a user shares a dataset, workflow, or history, as well as within Galaxy Pages (see last paragraph in this section).

With the exponential increase in size and number of bioinformatics data [1], it is increasingly challenging to manage, organize, and make available datasets. As a step in alleviating this, Galaxy supports notion of *data libraries*. Data libraries represent Galaxy instance wide repository of input data sets that can be easily shared and incorporated into users' histories. Not only does the data available in data libraries not need to be uploaded by each user, but any data library element can be used any number of times by any number of users without duplicating the single disk file. Data libraries provide a hierarchical container for datasets, mean-

ing that they can contain datasets, folders, and sub-folders. In addition, data versioning and sophisticated data access permission role system are implemented allowing role-based sharing of individual data library elements as well as making a data library public. Alongside data library sharing permissions, individual data library elements can enforce different actions based on selected user groups.

Galaxy *Pages* (Figure 6) unify Galaxy's functionality to provide users with a medium to communicate and share a complete computational analysis. Pages are custom web-based documents that enable users to communicate about an entire computational experiment, and Pages represent a step towards the next generation of online publication or publication supplement. A Page, like a publication or supplement, includes a mix of text, figures, and graphs describing the experiment's analyses. In addition to standard content, a Page also includes embedded Galaxy items from the experiment: datasets, histories, and workflows. These embedded items provide an added layer of interactivity, providing additional details and links to use the items as well. Like histories, Pages can be shared at a range of levels. Importantly, pages can be published and serve as online supplementary materials for a journal paper; a recent paper describing a metagenomic study that surveyed eukaryotic diversity in organic matter collected off the windshield of a motor vehicle used a Page for its supplementary material[3] [8].

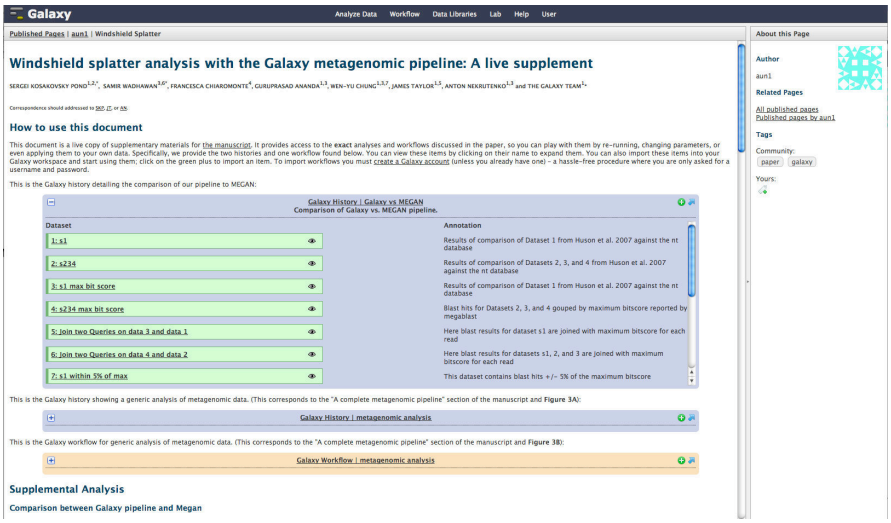

Figure 6. A screenshot of a published Galaxy Page showing supplemental information for performed analysis and associated scholarly publication. Any part of information included on this Page can be easily copied into user's workspace thus supporting notions of accessibility and reproducibility. Shown page is available at http://main.g2.bx.psu.edu/u/aun1/p/windshield-splatter

[3] http://usegalaxy.org/u/aun1/p/windshield-splatter

2.2.3 Data Acquisition and Visualization

Modern biological analyses frequently involve both locally produced experimental data and community data available through a number of well-organized data warehouses including NCBI, UCSC, Ensembl, TIGR, GMOD and others. These excellent resources provide users with the ability to query databases (e.g. with the UCSC Table Browser or the Mart system) and visualize features of the genomic landscape (e.g. with the UCSC Genome Browser, GBrowse, or Ensembl browser). In other words, these resources represent two termini of a typical analysis: the beginning (obtain the data) and the end (visualize the results). Galaxy complements these resources, enabling data manipulation and aggregation of local data together with data acquired from these external sources. Results of analysis can then be sent back to these resources for visualization.

The Galaxy distribution includes tool configurations for integrating several important data sources, including UCSC Table Browser, BioMart, InterMine, GBrowse, directly into the data analysis interface native to Galaxy. Galaxy makes implementing connections to external data-sources simple and straightforward. From Galaxy, any existing web based resource can be integrated into Galaxy's web interface through its "proxy" functionality, requiring no changes to the existing resource. Because of that, any Galaxy instance can immediately use these data connections with no custom configuration.

Upon completion of an analysis, Galaxy users can easily visualize analysis results. Galaxy implements connections to external data visualization tools, including the UCSC Genome Browser (see Figure 7), which can be accessed via web links within a dataset's history entry.

Figure 7. Visualization example with Galaxy and UCSC Genome Browser.

2.2.4 Access to Computational Resources

The Galaxy components discussed thus far focus on enabling and streamlining the process of analyzing data. However, increasing capabilities to produce large bioinformatics datasets also means an increasing need for computational resources. Thus, there is an obvious benefit—and, soon, a need—to enable the transparent acquisition of computing resources (e.g. computing power, storage space) to meet these demands. Galaxy makes it possible for users to acquire and utilize computational resources in two distinct ways. First, by providing an abstract interface to various compute clusters, allowing the use of existing resources users may have available. Second, by providing a self-contained solution for enabling Galaxy to be executed on distributed but aggregated infrastructures, for example, cloud computing [9] resources.

In the context of generating ever-growing data and the necessary transformation of this data into a biologically meaningful information, it is necessary to possess significant computational infrastructure and informatics support. In particular, meeting computational demands is especially difficult for individual researchers and small labs. For an experimental group with no computational expertise, simply running a data analysis program is a barrier, let alone building a compute and data storage infrastructure capable of dealing with DNA sequencing data. Galaxy as a whole represents the first step in hiding low level details, such as running and assembling various tools, from users. Galaxy abstracts and automatically handles all aspects of interaction between users, tools, and the system. Through a job abstraction approach, Galaxy describes and encapsulates a job in an internal representation. This representation enables Galaxy to easily move across various systems as well as to support reproducibility. Once a job is described and is ready to be executed, depending on the configuration of Galaxy, an appropriate job runner is invoked and the job is submitted to the underlying resource(s). The job runner polls the system for job status and handles the produced output to integrate it back into Galaxy for the user. Availability and actions performed by job runners are further described in Sections 3.2 and 3.3.

In addition to enabling streamlined execution of analysis jobs on local resources, provisions have been made to enable Galaxy to simply execute on virtualized compute infrastructures, including cloud computing resources. Cloud computing [10] has recently emerged and is ideally suited to the analysis of large-scale biological data. In this model, computation and storage exist as virtual resources, which can be dynamically allocated and released as needed. This model is well suited for many problems in bioinformatics data analysis, which intermittently require large amounts of compute power with fast access to enormous amounts of data. By coupling Galaxy and such environments, it is possible for anyone to acquire needed resources and perform desired analysis while not requiring informatics expertise. Section 4 discusses this functionality of Galaxy in great detail.

2.3 Galaxy Architecture

The Galaxy *Framework* is a set of reusable software components that can be integrated into applications, encapsulating functionality for describing generic interfaces to computational tools, building concrete interfaces for users to interact with tools, invoking those tools in various execution environments, dealing with general and tool specific dataset formats and conversions, and working with "metadata" describing datasets, tools, and their relationships. The Galaxy *Application* is an application built using this framework that provides access to tools through an interface (e.g., a web-based interface). A Galaxy *Instance* is a deployment of this application with a specific set of tools. The core components of the Galaxy Framework are the *toolbox*, the *job manager*, the *model*, and the *web interface*, *depicted in Figure 8*.

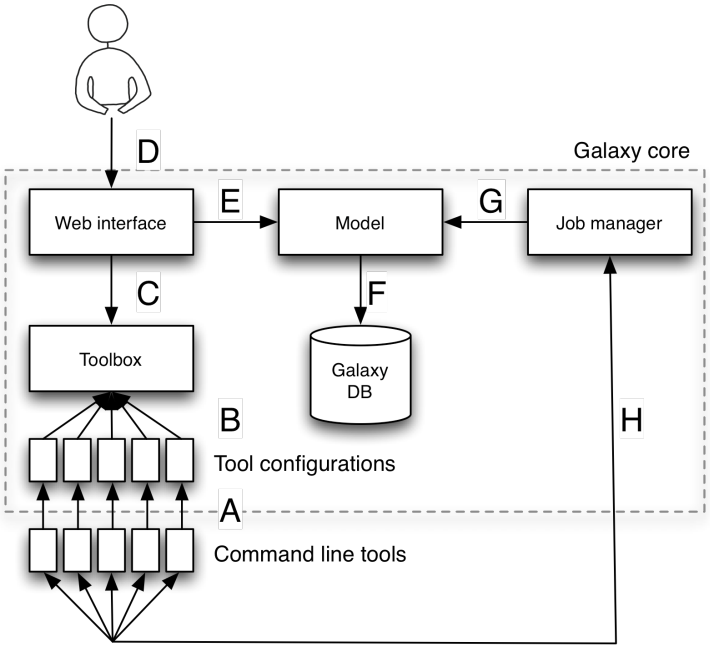

Figure 8. High-level architecture of core Galaxy framework components. (A) Command line tools are described within the tool configuration files and (B) validated within the Toolbox. Available tools are made available through the web interface (C). As users submit jobs (D), web controllers interact with the Model (E) to store the relevant information in Galaxy's database (F). As jobs become ready, the Job manager polls the database (G), prepares the jobs, and submits them to the underlying resources (H). As jobs complete, the Job manager handles and imports them back into the Galaxy framework.

The toolbox manages all of the details of working with command-line and web-based computational tools. It parses Galaxy tool configuration files (for an example of such a file see Section 3.2) that describe the interface to a tool – the parameters and input data it can take, their types and restrictions, and the outputs it produces – **in an abstract way that is not specific to any particular user interface.** This abstraction is critically important since it allows for changing how tools are displayed without needing to change their configuration (e.g., to leverage new accessibility features as web browsers improve, or to provide interfaces that are not web-based). The toolbox provides support for validating inputs to a tool, and for transforming a valid set of inputs into the commands necessary to invoke that tool. Additionally, the toolbox allows tool authors to provide tests for their tools (inputs and corresponding outputs) and provides support for running those tests in the context of a particular Galaxy instance.

The job manager deals with the details of executing tools. It manages dependencies between jobs (invocations of tools) to ensure that required datasets have been produced without errors before a job is run. It provides support for job queuing, to allow multiple users to each submit multiple jobs to a Galaxy instance and receive a fair execution order. The underlying method for execution is "pluggable". Currently jobs can be executed on the same machine where the Galaxy instance is running, or dispatched to a computational cluster using a standard queue manager (support for The Portable Batch System and Sun Grid Engine systems is included, and other dispatch strategies can be implemented and plugged in easily).

The model provides an abstract interface for working with datasets. It provides an object-oriented interface for working with dataset content (stored as files on disk) and "metadata" (data about datasets, tools, and their relationships; stored in a relational database). Beyond providing access to the data, this component deals with support for different datatypes, datatype specific metadata, and type conversions.

The web interface provides support for interacting with a Galaxy instance through a web browser. It generates web-based interfaces to the toolbox (for browsing and choosing tools), individual tools (building forms to accept and validate user input to a tool), and the model (allowing the user to work with all of the datasets they have produced). The web interface is currently the primary way to interact with a Galaxy instance, but the other underlying components do not need to know anything about the web, all web specific aspects of Galaxy are encapsulated by the web interface. The web interface is the primary interface type for Galaxy and this was dictated by the fact that web-browsers are present on every modern computer.

2.3.1 Implementation Details

The Galaxy framework is implemented in the Python programming language. Python has several advantages for our purposes. First, it is a lightweight dynamic

language that allows us to rapidly implement new Galaxy features. While Python is concise and easy to write, it is also a highly structured language that is generally easy to read and understand. This is important since it makes customizing and extending the Galaxy framework much easier for users. Additionally, Python has a very powerful standard library, as well as an amazing variety of third party open source components; some of the best of which we have been able to take advantage of in building Galaxy. However, an important aspect of the Galaxy architecture is the abstraction between the framework and the underlying tools. Because the Galaxy toolbox interacts with tools through command-line and web-based interfaces, there is *no* requirement that a tool author use Python (e.g., the example in Section 3.2 uses a tool written in Perl). While Python is a powerful language for scientific computing, and many of the tools we provide for comparative genomic analysis are implemented in Python, frequently another language may suit a particular problem better, or simply be preferred by a tool author. We want Galaxy to be able to provide easy access to all useful computational tools – as long as a tool can be run through a command line or the web, Galaxy can incorporate it.

Galaxy includes its own web server and embedded relational database (SQLite), and a Galaxy download includes all dependencies: a user needs to just edit the configuration file and run one command to start interacting with and customizing their own Galaxy instance (see Section 3.1). However, if a particular Galaxy instance needs to support higher throughput, they can customize the web server, the underlying relational database, and the job execution mechanism. For example, the public Galaxy instance maintained by the Galaxy team at Penn State is integrated with Apache as the web-server, uses the enterprise class relational database PostgreSQL, and executes jobs on a computational cluster with a queue managed by Torque PBS.

2.3.2 Software Engineering Details

Galaxy development follows a two-week cycle, in which tasks are identified, integrated, and tested every two weeks. Between cycles the team meets to discuss errors that occurred during the past cycle, review changes that were made, and establish tasks for the next development cycle. By identifying projects that can be completed quickly, these short cycles allow students who are involved in the project for only a short time to still make a satisfying and useful contribution.

Regular code reviews are also a critical part of our process, whenever changes are checked into the Galaxy version control system, they are emailed to all members of the team for review. Thus we ensure that all code checked into Galaxy is seen by more than one person. Galaxy includes unit tests both for the framework, and for individual tools. These tests are run automatically whenever changes are made, and test failures are emailed to the team so that any regressions are identified immediately. In addition to email, developers communicate through a software development oriented wiki, that allows for writing documentation and fea-

ture specifications in a collaborative way, as well as tracking bugs, feature enhancements, and other issues.

3 Deploying and Customizing Galaxy

This section provides an overview of the steps required to deploy and customize an instance of Galaxy and establishes a need for tools such as Galaxy to transition toward a ubiquitous computing platform that is not dependent on any given infrastructure and/or tool set.

3.1 The Installation Process

The Galaxy framework can be freely downloaded by anyone from anywhere and used locally to perform data analyses or to support development of local tools. To make Galaxy attractive for developers, its installation process very straightforward. Because Galaxy includes and manages all of its own dependencies, the only requirements for a Galaxy download is a local Python 2.4 or later interpreter and Mercurial[4] (an open-source version control system):

1. **Get the latest copy of Galaxy from the source repository.** We make the code available in several forms to ensure anyone can obtain it, but the easiest way to ensure you have a current copy of the Galaxy code is by using mercurial from the public repository available on bitbucket[5]:
   `hg clone http://bitbucket.org/galaxy/galaxy-dist/`
2. **Run the automatic initial setup.** When first installed, Galaxy needs to download dependencies for the platform it is installed on, create the initial configuration files, and initialize its database. This is all completely automated and requires no work from the user beyond running the initial setup script:
   `sh setup.sh`
3. **Run Galaxy**. At this point Galaxy is configured and ready to run, which is as simple as:
   `sh run.sh`

These three simple steps are all that is needed to have a running Galaxy instance, which can be immediately used to perform analyses using the default set of tools. This default configuration uses its own embedded web server and database, and executes analysis on the machine where it is installed. However, Galaxy can be easily configured to interface with an enterprise class database, high availabil-

[4] http://mercurial.selenic.com/

[5] http://bitbucket.org/

ity web server, or various computational clusters depending on the needs of a particular site (see Section 3.3). Additionally, users can customize many other aspects of a Galaxy instance, such as the appearance of the web interface, the datatypes the framework will recognize and understand, and the available computational tools. Ensuring a simple installation process allows anyone to utilize Galaxy without possible contention of public resources. In addition, developers can easily establish their personal environment for developing or customizing new tools.

3.2 Adding Tools to Galaxy

In addition to simple installation procedure, another key requirement for winning developers is to make tool integration effortless. For example, consider integrating a simple PERL script which is run from the command line as "`toolExample.pl FILE`" which reads FASTA format sequence data from the file name FILE and prints the GC content - a statistical measure of the content for a DNA sequence - of all of the sequences. To integrate this tool into Galaxy, we create a "tool config" file like that show in Figure 9. This configuration file provides an abstract interface to the tool that can then be integrated into any Galaxy instance. To make a particular Galaxy instance aware of the tool, it simply needs to be added to "`tool_conf.xml`" configuration file in Galaxy's installation directory. Note that a tool can be written in any language or be a compiled executable – the Galaxy framework only needs to know how to run it and to pass it the necessary parameters.

For tools that require more complex interfaces Galaxy provides additional constructs for describing tool input parameters. Galaxy allows groups of parameters to be repeated, and will automatically deal with the interface complexity of allowing the user to add and remove repetitions of the group (Figure 9D). For example, in a plotting tool this could be used to allow the user to define an arbitrary number of plot series, each built from a different dataset with different parameters. It is also possible to define conditional groups of parameters, allowing the specific inputs that are available in a given section to depend on previous inputs (Figure 9C). For example – again considering a plotting tool – it is possible to have different input parameters within a series, depending on whether the user has selected a line series or a point series. These grouping constructs can be nested to arbitrary depth, allowing input scenarios of substantial complexity to be specified simply and concisely.

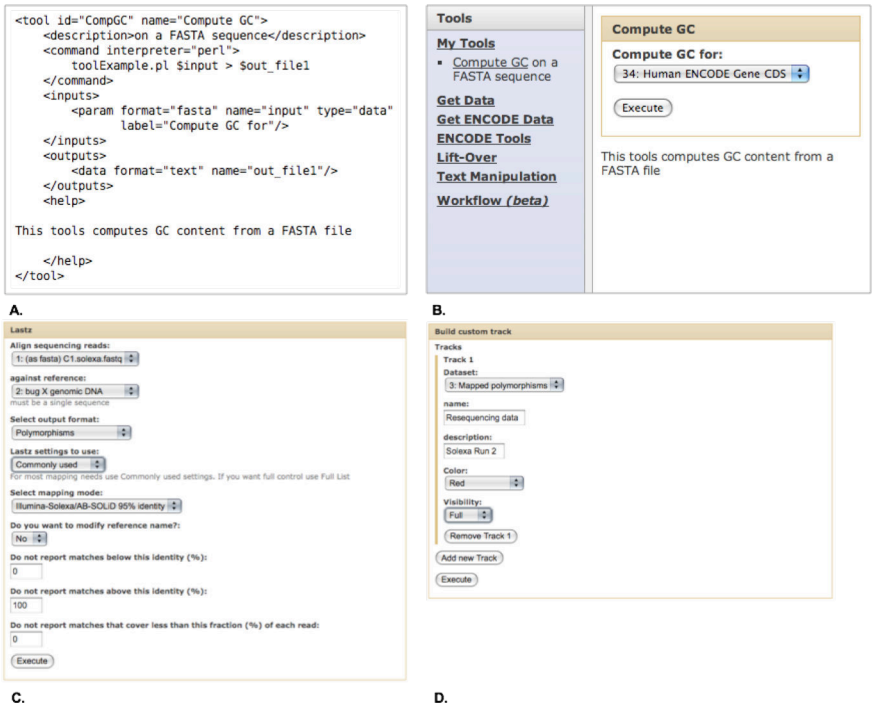

Figure 9. (A) Tool configuration file for the example tool described in Section 3.2 and (B) the user interface to the tool as generated by Galaxy. Note the correspondence between elements of `toolExample.xml` file and the user interface elements in the generated form. (C) An example of interface for lastZ short read wrapper. This interface uses a conditional construct allowing the user to switch between "Commonly used" parameters (shown) and "Full list" representing a multitude of options for this tool. (D). An example of interface utilizing group repetitions. Here a user can build multiple custom tracks.

3.3 Customizing Galaxy

In production environments where many users are intended to use Galaxy, the Galaxy instance should be customized to rely on high availability tools capable of handling user-generated load as well as to relegate job execution to a compute cluster, thus speeding up execution of jobs. The Galaxy framework supports such customizations via multiple abstraction levels implemented in the application. Specifically, all of database (i.e., model) interaction is handled through the

SQLAlchemy[6] toolkit that mediates interactions between Galaxy and the underlying database server. In turn, Galaxy is implemented on top of an abstraction layer that does not change with the underlying database implementation. As a result, SQLite, PostgreSQL, and MySQL are immediately supported as underlying database servers. Customizing a given instance of Galaxy instance to use an alternative database server is done simply by changing a global configuration parameter. In addition, the implementation of Galaxy provides database migration scripts that ensure smooth transition from one Galaxy update to the next without compromising data stored in the database.

By default, Galaxy executes user submitted jobs on the local system. Due to the computational demand such jobs impose on a given system, as the number of jobs grows, it is beneficial to farm those jobs out to a compute cluster and execute them in parallel. Comparable to the support for multiple database servers, Galaxy provides immediate support to execute jobs on the TORQUE PBS cluster manager and the Sun Grid Engine (SGE) cluster manager. Within Galaxy, support for multiple job managers is implemented at a conceptual level making it easy to add support for additional job management systems.

Figure 10 shows the architecture of the job manager component within Galaxy. The Galaxy web controller receives a job and all data about the job is stored in the database (e.g., input data, user-selected parameters) (steps 1 and 2). The job monitor detects the change and proceeds to create a job wrapper - a Galaxy-specific representation of the job that contains all necessary components forming a job (e.g., input dataset, reference to external index files, full path for invoking a tool, complete set of job parameters) (step 3) and adds it to the end of a local queue (step 5). Relevant job data is also stored in the local database to enable job recovery in case a job or a machine was to fail. Next, depending on the configuration of Galaxy, the job is picked up from the job queue by a job runner (step 6). A job runner is a modular and pluggable component that implements necessary details for running a job on the underlying system. In case of the basic local job runner, this simply entails composing the complete tool invocation command and spawning a new thread (steps 7 and 8). In case of a cluster manager such as SGE, it entails creating a job wrapper script and submitting the job to an appropriate queue as well as monitoring the job.

[6] http://www.sqlalchemy.org/

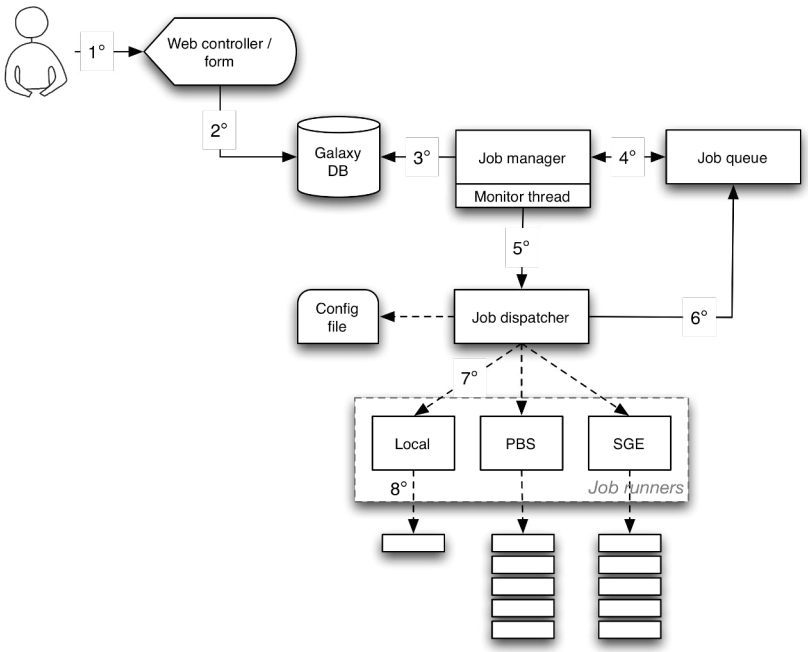

Figure 10. Architecture of the job management component within Galaxy.

3.4 Galaxy Accessibility

Anecdotal evidence suggests that Galaxy is usable for many biologists. Galaxy's public web server processes ~5000 jobs per day. In addition to the public instance, there are a number of high-profile Galaxy servers in use, including ones at the Cold Spring Harbor Laboratory and the United States Department of Energy Joint Genome Institute. All of Galaxy's operations can be performed using nothing more than a web browser, and Galaxy's user interface follows standard web usability guidelines [11], such as consistency, visual feedback, and access to help and documentation. Hence, biologists familiar with genomic analysis tools and comfortable using a web browser should be able to learn to use Galaxy without difficulty.

Finally, Galaxy has been used in numerous life sciences publications by groups not affiliated with the Galaxy team, and its sharing features were recently used to make data available from a genome-environment interaction study published in Science [12].[7]

[7] http://main.g2.bx.psu.edu/u/fischerlab/h/sm1186088

3.5 Galaxy Usage Example

To demonstrate the utility of Galaxy, here we show how it was used to perform and communicate a previously pbulished metagenomic study that surveyed eukaryotic diversity in organic matter collected off the windshield of a motor vehicle [13]. The choice of a metagenomic experiment for highlighting the utility of Galaxy and Pages was not accidental. Among all applications of Next Generation Sequencing (NGS) technologies, metegenomic applications are arguably the least reproducible. This is primarily due to the lack of an integrated solution for performing metagenomic studies, forcing researchers to use various software packages patched together with a variety of "in-house" scripts. Because phylogenetic profiling is extremely parameter dependent—small changes in parameter settings lead to large discrepancies in phylogenetic profiles of metagenomic samples—knowing exact analysis settings are critical. With this in mind, we designed a complete metagenomic pipeline that accepts NGS reads as the input and generates phylogenetic profiles as the output.

The Galaxy Page for this study describes the analyses performed and includes the study's datasets, histories, and workflow so that the study can be rerun in its entirety: http://usegalaxy.org/u/aun1/p/windshield-splatter To reproduce the analyses performed in the study, readers can copy the study's histories into their own workspace and rerun them. Readers can also copy the study's workflow into their workspace and apply it to other datasets without modification. Moreover, histories and workflows can be exported to other Galaxy instances, thus supporting additional means for result reproducibility.

Other recent examples of the use of Galaxy for analysis of genomic data include [14], [15] and [16].

4 Enabling the Next Step in e-Science

Overall, Galaxy provides an easy-to-deploy platform for data analysis, tool deployment, and analysis sharing and publication. An individual instance of Galaxy can easily be customized by adjusting its runtime environment (i.e., cluster support) to enable it to scale and meet the demand imposed by its users. However, this model of expansion does not fare well in several scenarios. For example, data analysis is often an intermittent activity: a research lab will often have periods when large amounts of data need to be analyzed followed by little or no data analysis needs. Alternatively, an individual researcher or a small lab may not have access to a large compute cluster needed to perform desired analysis; for such a scenario, it is often not worth purchasing and maintaining a compute system due to the associated cost, time and knowledge required to maintain it, and then dealing with system aging. Also, in academic environments, a research lab's comput-

ing demands can grow or shrink based on current projects, interests, and funding levels. These needs translate directly into a dynamic demand for computational infrastructure support.

Because of the global trend toward resource aggregation [10], it is likely that, with time, organizations, labs, and universities will aggregate and virtualize many of the dispersed computational resources into a few dense datacenters that will be shared among all the users on as-needed basis [10]. Grid computing [21] represented a first step in this direction; cloud computing [9] represents the second step. Having all resources aggregated in a single location where sharing and access policies are defined opens opportunities for individual researchers, small labs, as well as large institutions to gain access to desired resources when needed. However, tools and applications need to be able to adjust to such environments and utilize them transparently, and, ideally, effectively. Figure 11 depicts this scenario.

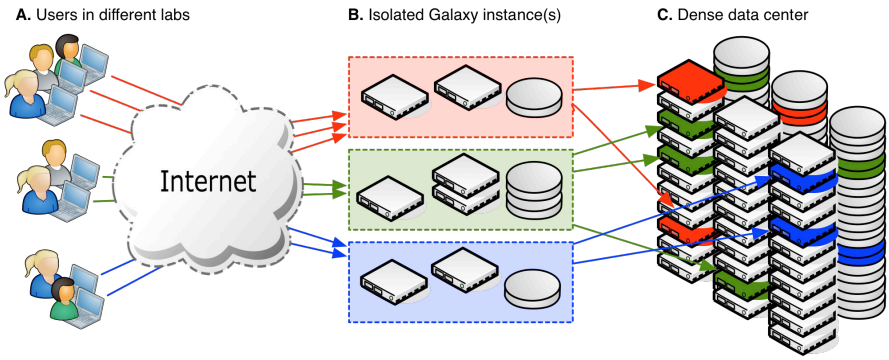

Figure 11: Simplified overview of an aggregated distributed infrastructure and it's perception by users: (**A**) Users in different labs access a dedicated application instance over the internet with nothing more than a web browser, (**B**) these application instances appear to the users to be dedicated infrastructure with apparently infinite compute and storage resources, but are in fact virtual resources (**C**) which are allocated on demand from a large shared pool.

In order to enable scientists to take advantage of these new models for computational resource allocation, we have focused on provisioning a general solution that can operate in the upcoming and developing infrastructures. One requirement that such a solution has is that it must not require or impose specific demands but, instead, must focus on utilizing the most general concepts that will persist beyond any single infrastructure configuration. In addition, like the Galaxy application itself, this solution should be easy to utilize and not require informatics expertise.

4.1 Galaxy and IaaS

To meet the goals stated above, we have decided to implement a solution that targets the bottom-most layer of an aggregated infrastructure – Infrastructure-as-a-Service (IaaS) level. Such solution relies only on the basic infrastructural components that can be expected to exist in a range of aggregated infrastructures (i.e., customizable operating system with access to persistent storage). Furthermore, our solution is implemented as a standalone application that does not require any external services (e.g., a broker service that coordinates user requests). This is an important design feature because users do not have to supply their credentials to anyone but the infrastructure providers. Overall, the described model minimizes dependencies and contributes to the robustness of the application (which is essential in self-managed, distributed systems).

To minimize user's exposure to the low-level infrastructure at which the system operates, there is a need to abstract user interaction so that only high-level operations are perceived by the user. Such solution effectively bridges the low-level IaaS design with the high-level, targeted Platform-as-a-Service (PaaS) solution. In return, the user is completely abstracted from the infrastructural details but the system design enjoys needed flexibility to be applicable in a range of custom environments.

Figure 12 depicts design of the derived solution. As shown in the figure, a user initiates interaction with the infrastructure through the local infrastructure console manager (step 1). Depending on the implementation of the infrastructure, the console manager may be a set of simple command line tools or a web portal. Through the given interface, the user instantiates the Galaxy Controller (GC) machine image (step 2). GC is represented by a customized machine image that contains necessary tools to configure, start, and support the Galaxy application. For the case of virtualized infrastructures, GC image is an operating system image; for the case of dedicated infrastructures, GC is a physical machine where GC has been preconfigured. After a machine boots, GC starts automatically as a standalone process (step 3). The benefit of GC running as a standalone process is that it is independent of other processes running on the instance, including the Galaxy application, and can thus be used to control, manage, and react to the state of those processes. With that, GC coordinates all of the required services and components that are required to deliver a ready, on-demand instance of Galaxy.

Because the GC image is common to all users of a given infrastructure and is thus a stateless instance, in order to personalize it to a given user and enable data persistence, GC needs to obtain user-specific data. This is realized by relying on the existence of an external data repository. After the GC image boots, it automatically retrieves GC-needed data from the repository (step 4). This data contains references to persistent storage volumes that are then attached by GC to the running instance as file systems and used by Galaxy application (step 5). If this data does not exist (as will be the case during first invocation of a given GC image by a

user), data is obtained from a public data repository and then updated and stored in the user-specific repository for future invocations.

Once the GC has completed the initial system configuration, the user interacts with it directly (through a web interface) to request desired resource allocation (step 6). Requested resources are used by the Galaxy application associated with given GC to run user submitted jobs. As requested resources are acquired, they are automatically configured as GC workers (GC-w) (step 7). Lastly, the Galaxy application is started and made accessible for end users (step 8). At this point, from the end user's standpoint, given instance of Galaxy application is used like any other (steps 9 and 10). However, from the application administrator standpoint, the application instance has the potential of scaling with user demand as well as exhibiting portability across infrastructures.

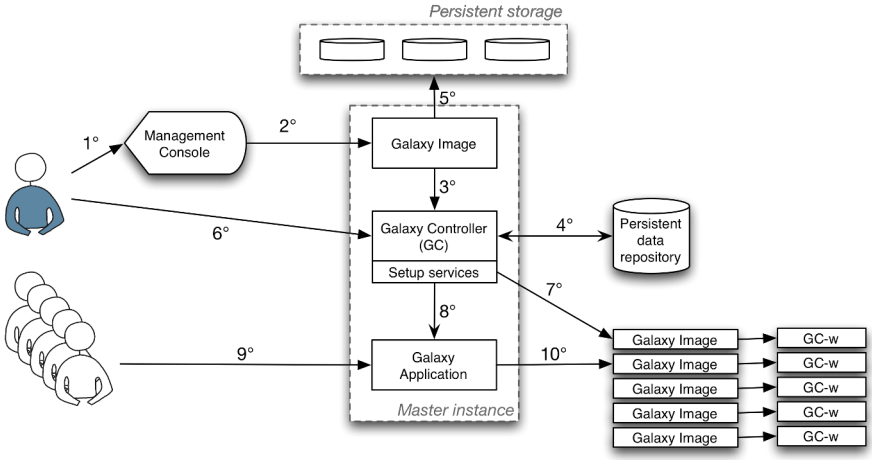

Figure 12. Architectural view of the process and components required to enable scalable and user-specific deployment of (Galaxy) application in aggregated infrastructures.

The described architecture where a generic machine image is contextualized at runtime [22] by the GC (step 4 in Figure 12) enables the same machine image to be used as a base for the master and worker instances. This allows more streamlined machine image administration (i.e., less maintenance overhead) and enables dynamic scaling of the size of a user's cluster. Once an instance boots, through the contextualization process, an instance is assigned a role of the master or worker. Then, through cross-instance messaging, appropriate configuration can be completed allowing an instance to be added to an existing cluster at runtime.

The described architecture relies on only the basic services expected from any IaaS provider, namely existence of a general-purpose user data repository an attachable/mountable persistent storage. Any kind of a content delivery system can serve as the persistent data repository, even if it is external to the IaaS provider. Internal to GC, interaction with needed components can be implemented similar to

the multiple job runners from the job manager component of the Galaxy application. As a result, it is easy to envision immediate availability of support for multiple IaaS providers.

In addition, the described approach allows a user to interacts with the system through a web-based interface with all the necessary steps accomplished through a guided wizard. Low-level technical details are abstracted from a typical user and are automatically handled thus making the system easy to use. Lastly, because of the distributed computing environment, the system was designed with an assumption that components will fail. By having GC run as a meta-application that manages lower level components and functionality, it is possible for it to automatically address potential failures.

4.2 Galaxy and AWS

Recently, cloud computing [10] has emerged as a computational model that is rooted in the concepts of aggregated infrastructure ideas discussed throughout this section. In this model, computation and storage exist as virtual resources, which can be dynamically allocated and released as needed. With many providers of cloud computing services (e.g., Amazon Web Services (AWS), RackSpace, GoGrid, Mosso), this model presents an ideal scenario to verify the architecture described in Section 4.1. We have selected AWS as the service provider to verify our design because it represents a *de facto* standard in this area, showcased by proven availability and reliability of its services. Furthermore, by initially focusing on a well-accepted and readily available technology, benefits delivered by a given tool are immediately available for consumption by potential users. Also, the infrastructure architecture and the API set forth by Amazon and used by an implementation of GC have been accepted by other cloud infrastructure management projects (e.g., Eucalyputs [23], OpenNebula [24], Nimbus [25]) allowing for smoother transition as those services become more prominent or infrastructures based on those managers emerge.

4.2.1 GC Implementation

In order to implement the described design, within AWS, there was a need to create a customized Amazon Machine Image (AMI) packaged with necessary applications and services. Because it has to be possible for the same machine image to be instantiated by multiple, independent users, the image could not be fully preconfigured to contain all of the necessary user information (e.g., [26]). As a result the image had to be configured to support boot time parameterization and contextualization [22]. In addition, because of the amount of time and effort required to update such image (e.g., each time an update to a tool is needed), we wanted to

minimize the number of tools embedded into the image and rely on more flexible components that an instance could be associated with at runtime. As a result, generated AMI was preconfigured only with the basic components, including Python interpreter, message queuing system, and necessary user accounts (e.g., galaxy, postgres). All of the domain specific tools, Galaxy application, and the Galaxy Controller are obtained at instance boot time from attachable disk volumes that are easy to associate with individual users, perform tool updates, and persist beyond lifetime of any one instance.

Internally, GC was implemented as a web app within the Galaxy framework and enabled for standalone execution. At the implementation level, the GC is represented by two services, namely GC master and GC worker. The distinction among services is determined at runtime, based on the given instance's role, which is determined at instance boot time. All of the instance contextualization, master-worker communication, and status reporting is performed through a messaging system implemented using the AMQP standard [27] and using a RabbitMQ[8] server deployed on the master instance.

Following the general steps described in Section 4.1, we next provide more detailed actions performed after created machine image is instantiated:

1. User instantiates a master instance
2. As part of startup process, start RabbitMQ server, download GC source code from either user's data repository or the public data repository and start the GC web application
3. Within GC, create attachable storage resources (EBS volumes in case of AWS) from public snapshots for the Galaxy application and necessary index files (i.e., datasets) used by several Galaxy tools; attach them to the running instance, and import existing file systems
4. Start NFS and enable sharing of relevant directories
5. Unpack and configure SGE
6. Allow a user to, through the GC web interface, configure their cluster: specify amount of persistent storage to be used for user data
7. Create the user data external storage resource and appropriate file system
8. Configure PostgreSQL database on the user storage resource to be used by the Galaxy application
9. Start the Galaxy application.
    a. Once Galaxy is ready, enable access to it from the web interface
10. The user can now start using the Galaxy application. As the need for cluster nodes increases (or decreases), through the web interface, the user may add (or remove) worker instances
    a. As a worker instance boots, they mount NFS directories and notify master as being 'alive'
    b. As instances report alive, authentication information to enable SGE to submit jobs is exchange with the master.

[8] http://www.rabbitmq.com/

In order to support dynamic scaling of a user's cluster, the master repeats steps 9 and 10. Within the GC implementation, there is no distinction between the initial cluster startup and later cluster size scaling. Namely, after a user requests to add a number of worker instances, the master starts the instances specifying as part of the user data that these instances are workers. As instances boot, they start the worker configuration process and, through the messaging system, exchange needed information with the master (e.g., public ssh keys, NFS mount points). This allows the master to alter the configuration of the cluster and include those worker instances into the resource pool. The same process is followed when down scaling the size of the cluster.

Upon user-initiated termination of worker instances, GC stops all relevant services, terminates worker instances, exports file systems and detaches external data volumes. Because data volumes used containing tools and index files are not modified during the life of a Galaxy instance, those disk volumes are deleted. Next time given instance is instantiated they will be created just the first time. User data volume is obviously left untouched. Information about given volume is stored in a user-specific persistent data repository (S3 bucket, in case of AWS).

4.2.2 Interacting with GC

GC, and thus the Galaxy application, is readily available for use by anyone[9]. A completely configured and functional cluster can be instantiated in as little as 5 minutes for a cost of less than $1. To instantiate an instance of Galaxy on the AWS cloud, and thus get access to the full spectrum of tools supported by Galaxy, the following steps are required:

1. Create an AWS account and sign up for Elastic Compute Cloud (EC2) and Simple Storage Service (S3) services
2. Use AWS Management Console to start an EC2 instance
3. Use GC2 web interface on started EC2 instance to start a desired number of compute instances
4. Enjoy your personal instance of Galaxy on the cloud

Because AWS services implement a pay-as-you-go access model for compute resources, it is necessary for every user of the service to register with the provider. Once registered, the user is assigned an AWS access key and accompanying secret key. Note that Step 1 is a one-time activity.

Step 2 is required every time a cloud instance of Galaxy is desired. As part of the instance startup process, the user interacts with the infrastructure provider management interface. In case of AWS, this can be either AWS Management Console or command line tools. As part of this step, the user needs to choose appropriate machine image and provide account information. The provided account information is needed and used to acquire persistent storage for user specific data.

[9] http://usegalaxy.org/cloud

Provided account information is used locally only by the particular instance user is working with and is never shared or transmitted to another service or tool.

Once and instance boots and GC becomes available, through the web interface, the user finalizes the cluster creation by specifying amount of persistent data storage they would like to associate with the given cluster. This is required because of the virtual and temporary structure of cloud instances: once an instance is shut down, all modifications that were performed on the given system are lost. Therefore, in order to preserve the user data beyond the life of an instance, the external storage medium is required.

After the master instance completes the initial cluster setup and starts the Galaxy application, the user starts a desired number of worker instances. Moreover, the user can dynamically scale the number of worker instances over the course of life of the cluster. This is performed simply through the web interface while GC automatically handled all aspects of instance startup and cluster configuration (as described in Sections 4.1 and 4.2.1). Figure 13 captures GC's current web interface. As can be seen in this figure, under cluster status, small icons represent individual worker instances. Furthermore, each icon graphically depicts the given instance's system load over the past 15 minutes (see Figure 14). Such representation allows a user to immediately and quantitatively visualize the status and load of their cluster.

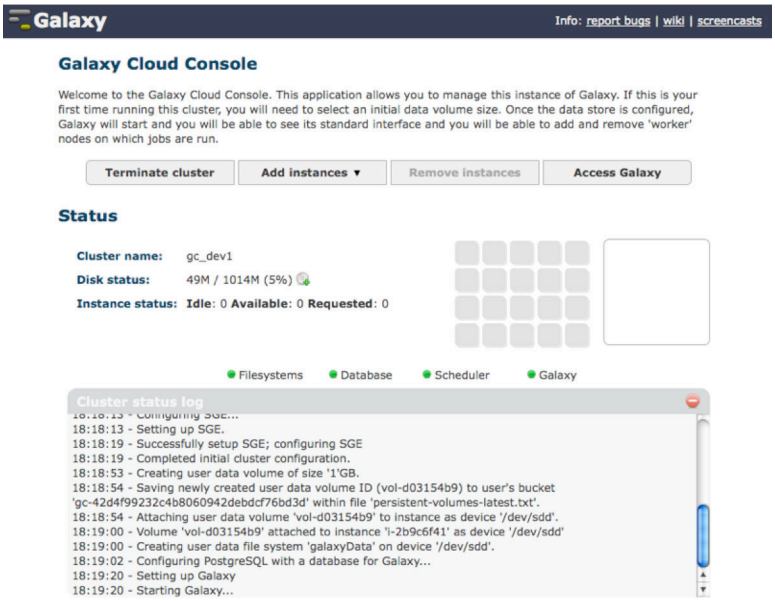

Figure 13. GC web interface showing user controls and the cluster status.

If the cluster becomes loaded and user decides more worker instances are needed, through the GC interface, the user can simply add additional worker in-

stances. GC starts the requested number of instances and automatically configures those to be used by the cluster job manager, thus distributing the cluster's workload. Likewise, if a cluster is underutilized, the user may specify a number of worker instances to remove. Without disrupting currently running jobs or users accessing Galaxy, GC reconfigures the cluster to remove and terminate those instances. As par of future work, we will enable such cluster scaling to be done automatically by GC (within user-specified bounds). Figure 14 depicts the process of cluster scaling from the user's standpoint.

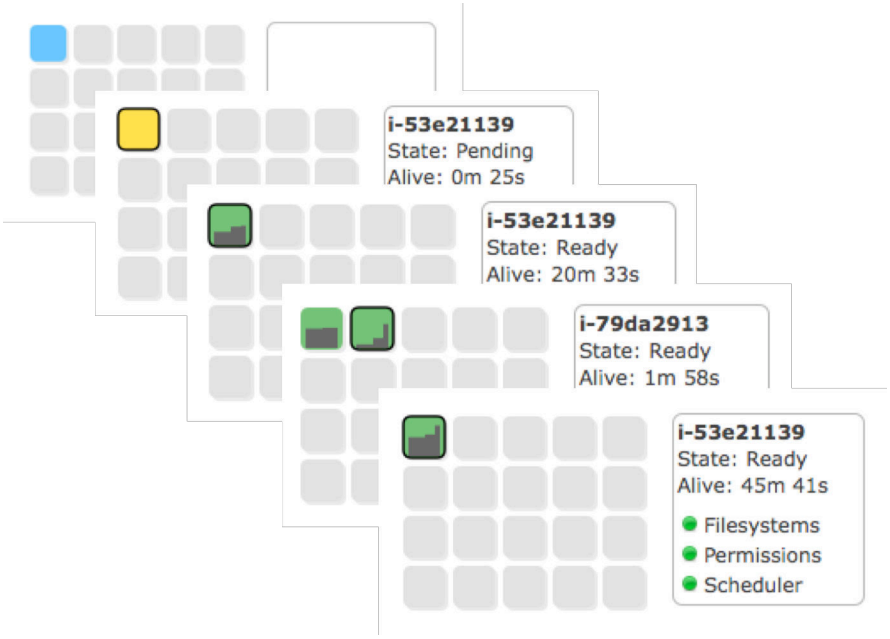

Figure 14. The process of cluster size scaling from the user's standpoint as indicated in the Galaxy Cloud web interface.

5 Related Work

Historically, several attempts have been made for the integration of biological analysis tools with the goal of making them available to bench biologists. These include ISYS [28], Biology Workbench [29], PLATCOM [30] and the Sequence Retrieval System [31]. ISYS is a downloadable platform, allowing software developers to add their own tools and databases. It includes a number of tools, such as a sequence and annotation viewer, a BLAST search launcher, and an Entrez sequence retrieval module. An important feature of ISYS is its DynamicDiscovery functionality, which suggests appropriate tools for a particular data type. How-

ever, ISYS requires programming experience and serves as a development platform rather than a ready-to-use tool. Biology Workbench is a comprehensive web-based col- lection of sequence analysis software. However, it is unsuitable for the analysis of large datasets, and cannot be used with genomic sequences and their associated annotations (these limitations are noted on the Biology Workbench website). PLATCOM provides a variety of utilities for comparative sequence analysis. However, this system lacks a history mechanism, and forces the user to choose a tool first and then the data, while Galaxy focuses on data and then provides the user with analysis choices. SRS has been successfully used for providing links to numerous databases and can, in principle, be used for tool integration using the complex "Icarus" language. Yet the existing public installations of SRS feature very few tools due to configuration difficulty (e.g., SRS at EBI only features EMBOSS, FASTA, BLAST, and HMMER) and cannot be scaled to analyze genomic datasets. Importantly, SRS is commercial software, which, even if obtained under an educational license, cannot be modified. On the other hand Galaxy is absolutely free: the Galaxy source code is open to everyone and designed to make extension and customization easy. This openness is a core principle of our philosophy – drawing as many developers as possible into the software design process will ensure the high usability and applicability of our product.

Recently a series of new approaches for designing pipelines and tool integration have been proposed. These include Taverna [32], [33] and Gene Pattern [6]. Taverna provides language and software components to facilitate building of analysis workflows; it can be used to construct very complex distributed analyses, but requires the user to install local software and manage many of the details of invoking an analysis. As a result, it is most useful for computational users building workflows over distributed systems and not immediately beneficial for experimental biologists. On the other hand, using Galaxy requires only a web browser, and the user does not need to worry about the details of how complex analyses are allocated to computing resources. In fact, Taverna and Galaxy are complementary – Taverna workflows could be integrated into a Galaxy instance as tools. However, our goal is to make workflows so simple that experimentalists can use them without reading manuals. Galaxy workflows are based on the existing history system, which has been attractive for many users due to its simplicity.

Gene Pattern is an analysis environment supporting gene expression, proteomics, and association studies. It is distributed as a large, complex client/server application requiring substantial expertise for installation and configuration. All data in Gene Pattern come from the user – there is no notion of integrated data sources available in Galaxy, namely direct connection to commonly used databases such as UCSC Genome Browser, HapMap, or BioMart. Although integration of external tools is similar within Gene Pattern, the configuration files available within Galaxy offer more flexibility in terms of tool descriptions. In terms of end user usability features, Gene Pattern does not provide support for user tags while annotations are limited to workflows in form of an external document. Tagging and annotation are tightly integrated within Galaxy, ensuring reproducibility

at all stages of an analysis. Gene Pattern supports sharing analyses and workflows with individuals or groups through external software tools. Sharing of items (datasets, histories, workflows) within Galaxy is supported at progressive levels and published to Galaxy's public repositories. Integration of sharing with Galaxy Pages thus supports embedding, publishing, and reuse of relevant analysis information. Lastly, Galaxy can readily be instantiated on a cloud computing infrastructure thus eliminating immediate resource availability concerns.

6 Conclusions

eScience came about in response to the growing need to streamline and merge computation and data analysis with scientific investigation. With the explosion of scientific data over the past decade, many sciences are becoming computationally driven. This is resulting in a shift in how science is done in those fields - a shift toward computationally intensive tasks. However, obtaining results from data analysis and computation does not come easy. Foremost, it requires development and availability of domain-specific tools. Next, it requires familiarity and ability to use those tools. Again, because computation has had a limited historical presence in many sciences, this step represents a barrier. The learning curve is difficult to be overcome because it requires scientists to step outside of their domain and become proficient in another science - computer science. Specifically, a scientist more often than not needs to learn how to write code and patch existing code. Not only is this often perceived as a significant burden by scientists but it leads to poor code design and likely poor tool development. Most importantly, because of the many *ad-hoc* scripts or interactive methods used to perform an analysis, obtained results are rarely reproducible. Taken together, these issues leads to poor science.

To facilitate computational science, streamlined access to data analysis is needed. There is a need for access to domain specific tools whose use does not mandate informatics expertise. Such an approach enables domain scientists to focus on their own research rather than being bogged down with low-level computational details. In response to this need, the Galaxy framework was developed.

Galaxy provides abstractions for tools that perform data analysis. Within Galaxy, seamless integration of one tool's output into another tool's input is supported from the ground up. Galaxy provides consistent and usable interfaces that shift the focus from running a tool to analyzing results. With a broad range of bioinformatics tools available by default within a Galaxy instance, Galaxy targets bioinformatics data analysis, with a specific focus on comparative and functional genomics, including the use of data generated with high-throughput DNA sequencing technologies [34], [35]. Furthermore, the Galaxy framework allows for easy addition of tools, thus enabling the framework to be extended to other do-

mains (one known example includes the machine learning domain[10]). With continuous research and development, Galaxy offers a ready solution to many researchers in eScience.

Nonetheless, Galaxy lacks the flexibility often encountered in research environments. Due to the continuous fluctuation of demand and supply for compute infrastructure, there is a need for Galaxy to be able to scale its computing usage accordingly. Furthermore, with the global increase in power and cooling demands of compute resources [36], coupled with the associated environmental impact, organizations are looking to reduce power usage and cut costs by aggregating all of the resources into dense data centers that can offer better economies of scale. As a result, there is a need for established tools to be able to utilize such infrastructures without being a major disruption to users. Galaxy Controller represents a step in that direction. Infrastructure independent design enables it to utilize upcoming infrastructures while providing a ready solution for current needs. Coupled with the Galaxy application, presented method provides a complete solution for end users: rooted in the basic requirements of IaaS while delivering SaaS functionality and avoiding exposure to the informatics details is a gateway to eScience.

Acknowledgements

Galaxy is developed by the Galaxy Team: Enis Afgan, Guruprasad Ananda, Dannon Baker, Dan Blankenberg, Ramkrishna Chakrabarty, Nate Coraor, Jeremy Goecks, Greg Von Kuster, Ross Lazarus, Kanwei Li, Anton Nekrutenko, James Taylor, and Kelly Vincent. We thank our many collaborators who support and maintain data warehouses and browsers accessible through Galaxy. Development of the Galaxy framework is supported by NIH grants HG004909 (A.N. and J.T), HG005133 (J.T. and A.N), and HG005542 (J.T. and A.N.), by NSF grant DBI-0850103 (A.N. and J.T) and by funds from the Huck Institutes for the Life Sciences and the Institute for CyberScience at Penn State. Additional funding is provided, in part, under a grant with the Pennsylvania Department of Health using Tobacco Settlement Funds. The Department specifically disclaims responsibility for any analyses, interpretations or conclusions.

References

[1] NCBI. (2009, February 3). *GenBank Statistics*. Available: http://www.ncbi.nlm.nih.gov/Genbank/genbankstats.html

[10] http://galaxy.fml.tuebingen.mpg.de/

[2] E. Huedo, R. S. Montero, and I. M. Llorente, "A Framework for Adaptive Execution on Grids," *Journal of Software - Practice and Experience,* vol. 34, issue 7, pp. 631-651, June 2004.

[3] E. Afgan and P. Bangalore, "Dynamic BLAST – a Grid Enabled BLAST," *International Journal of Computer Science and Network Security (IJCSNS),* vol. 9, issue 4, pp. 149-157, April 2009.

[4] D. Blankenberg, J. Taylor, I. Schenck, J. He, Y. Zhang, M. Ghent, N. Veeraraghavan, I. Albert, W. Miller, K. Makova, R. Hardison, and A. Nekrutenko, "A framework for collaborative analysis of ENCODE data: making large-scale analyses biologist-friendly," *Genome Research,* vol. 17, issue 6, pp. 960-964, Jun 2007.

[5] J. Taylor, I. Schenck, D. Blankenberg, and A. Nekrutenko, "Using Galaxy to perform large-scale interactive data analyses," *Current Protocols in Bioinformatics,* vol. 19, pp. 10.5.1-10.5.25, Sep 2007.

[6] M. Reich, T. Liefeld, J. Gould, J. Lerner, P. Tamayo, and J. Mesirov, "GenePattern 2.0," *Nature genetics,* vol. 38, issue 5, pp. 500-501, 2006.

[7] B. Langmead, C. Trapnell, M. Pop, and S. Salzberg, "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome," *Genome biology,* vol. 10, issue 3, p. 25, Mar 4 2009.

[8] P. Kosakovsky, S. Wadhawan, F. Chiaromonte, G. Ananda, W. Chung, J. Taylor, and A. Nekrutenko, "Windshield splatter analysis with the Galaxy metagenomic pipeline," *Genome Research,* vol. 19, issue 11, Oct 9 2009.

[9] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems,* vol. 25, issue 6, pp. 599-616, June 2009.

[10] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing," University of California at Berkeley UCB/EECS-2009-28, February 10 2009.

[11] J. Nielsen, *Designing web usability*, 1st ed.: Peachpit Press, 1999.

[12] S. Peleg, F. Sananbenesi, A. Zovoilis, S. Burkhardt, S. Bahari-Javan, R. Agis-Balboa, P. Cota, J. Wittnam, A. Gogol-Doering, and L. Opitz, "Altered Histone Acetylation Is Associated with Age-Dependent Memory Impairment in Mice," *Science,* vol. 328, issue 5979, pp. 753-756, 2010.

[13] S. Kosakovsky Pond, S. Wadhawan, F. Chiaromonte, G. Ananda, W. Chung, J. Taylor, and A. Nekrutenko, "Windshield splatter analysis with the Galaxy metagenomic pipeline," *Genome Research,* vol. 19, issue 11, pp. 2144-2153, 2009.

[14] K. Gaulton, T. Nammo, L. Pasquali, J. Simon, P. Giresi, M. Fogarty, T. Panhuis, P. Mieczkowski, A. Secchi, and D. Bosco, "A map of open

chromatin in human pancreatic islets," *Nature genetics,* vol. 42, issue 3, pp. 255-259, 2010.

[15] R. Kikuchi, S. Yagi, H. Kusuhara, S. Imai, Y. Sugiyama, and K. Shiota, "Genome-wide analysis of epigenetic signatures for kidney-specific transporters," *Kidney International,* 2010.

[16] J. Parkhill, E. Birney, and P. Kersey, "Genomic information infrastructure after the deluge," *Genome biology,* vol. 11, issue 7, p. 402, 2010.

[17] N. Singer. (2009, January 12). *More chip cores can mean slower supercomputing*. Available: http://www.sandia.gov/news/resources/news_releases/more-chip-cores-can-mean-slower-supercomputing-sandia-simulation-shows/

[18] D. Eadline. (2010, March 3) HPC Madness: March Is More Cores Month. *LINUX Magazine*. Available: http://www.linux-mag.com/id/7722

[19] A. Menon, J. Santos, Y. Turner, G. Janakiraman, and W. Zwaenepoel, "Diagnosing performance overheads in the Xen virtual machine environment," in *1st ACM/USENIX international conference on Virtual execution environments*, Chicago, IL, 2005, pp. 13-23.

[20] J. Hamilton. (2008, November 28). *Cost of Power in Large-Scale Data Centers*. Available: http://perspectives.mvdirona.com/2008/11/28/CostOfPowerInLargeScaleDataCenters.aspx

[21] *The Grid: Blueprint for a New Computing Infrastructure*, 1st ed.: Morgan Kaufmann Publishers, 1998.

[22] K. Keahey and T. Freeman, "Contextualization: Providing one-click virtual clusters," in *IEEE International Conference on eScience*, Indianapolis, IN, 2008, pp. 301-308.

[23] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in *Cloud Computing and Its Applications*, Shangai, China, 2008, pp. 1-5.

[24] I. M. Llorente, R. Moreno-Vozmediano, and R. S. Montero, "Cloud Computing for On-Demand Grid Resource Provisioning," *Advances in Parallel Computing,* vol. 18, pp. 177-191, 2009.

[25] K. Keahey, I. Foster, T. Freeman, and X. Zhang, "Virtual Workspaces: Achieving Quality of Service and Quality of Life in the Grid," *Scientific Programming Journal, Special Issue: Dynamic Grids and Worldwide Computing,* vol. 13, issue 4, pp. 265-276, 2005.

[26] H. Nishimura, N. Maruyama, and S. Matsuoka, "Virtual clusters on the fly-fast, scalable, and flexible installation," in *CCGrid* Rio de Janeiro, Brazil, 2007, pp. 549-556.

[27] A. W. Group, "AMQP - A General-Purpose Middleware Standard," ed, p. 291.

[28] A. Siepel, A. Farmer, A. Tolopko, M. Zhuang, P. Mendes, W. Beavis, and B. Sobral, "ISYS: a decentralized, component-based approach to the integration of heterogeneous bioinformatics resources," *Bioinformatics,* vol. 17, issue 1, pp. 83-94, Aug 14 2001.

[29] S. Subramaniam, "The Biology Workbench--a seamless database and analysis environment for the biologist," *Proteins,* vol. 32, issue 1, pp. 1-2, Jul 1 1998.

[30] K. Choi, Y. Ma, J.-H. Choi, and S. Kim, "PLATCOM: a Platform for Computational Comparative Genomics," *Bioinformatics,* vol. 21, issue 10, pp. 2514-2516, Feb 24 2005.

[31] T. Etzold and P. Argos, "SRS--an indexing and retrieval tool for flat file data libraries," *Bioinformatics,* vol. 9, issue 1, pp. 49-57, 1993.

[32] E. Kawas, M. Senger, and M. D. Wilkinson, "BioMoby extensions to the Taverna workflow management and enactment software," *BMC Bioinformatics,* vol. 7, p. 253, 2006.

[33] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. R. Pocock, P. Li, and T. Oinn, "Taverna: a tool for building and running workflows of services," *Nucleic Acids Research,* vol. 34, issue Web Server issue, pp. W729-32, 2006.

[34] A. Mortazavi, B. Williams, K. McCue, L. Schaeffer, and B. Wold, "Mapping and quantifying mammalian transcriptomes by RNA-Seq," *Nature methods,* vol. 5, issue 7, pp. 621-628, 2008.

[35] S. Pepke, B. Wold, and A. Mortazavi, "Computation for ChIP-seq and RNA-seq studies," *Nature methods,* vol. 6, pp. S22-S32, 2009.

[36] B. Moore, "Taking the data center: Power and cooling challenge," *Energy User News,* vol. 27, issue 9, p. 20, 2002.