Towards a Reference Architecture for Cloud-based Plant Genotyping and Phenotyping Analysis Frameworks

Banani Roy Amit Kumar Mondal Chanchal K. Roy Kevin A. Schneider Kawser Wazed
University of Saskatchewan, Canada
{*banani.roy, amit.mondal, chanchal.roy, kevin.schneider, kawser.nafi*}*@usask.ca*

***Abstract*—The domain of plant genotyping and phenotyping presents a number of challenges in the area of large data computation. Various tools and systems have been developed to automate the scientific workflows and support the computational needs of this domain. In this paper, we review a number of the popular systems (i.e., Galaxy, iPlant, GenAp and LemnaTec) in the domain of plant genotyping and phenotyping using the scenario-based architectural analysis method (SAAM). In particular, we focus on how different stakeholders are using these systems in a variety of scenarios and to what extent the systems support their needs. Our SAAM analysis shows that the existing systems have shortcomings. For example, they are limited in their support for high throughput processing of large amounts of heterogeneous types of data. Based on our findings we propose a reference architecture along with a preliminary evaluation in the subject domain. The reference architecture and its evaluation is aimed at helping developers/architects create suitable architectural designs and select appropriate technologies when developing plant phenotyping and genotyping systems.**

***Keywords*-Heterogeneous data; genotype; phenotype; software architecture; SAAM**

I. Introduction

Plant genotyping and phenotyping (PGP) are important for ensuring global food security. Plant genotyping and image-based plant phenotyping involves the generation and management of large amounts of data [11]. Genotyping involves the generation of DNA sequencing applying different methods (such as next generation sequencing) resulting in a huge volume of data from around the globe. Plant phenotyping is the appraisal of complex plant traits including growth, development, tolerance, resistance, architecture, physiology, ecology, yield and the basic measurement of individual quantitative parameters that form the basis for the more complex traits. Plant genotype and phenotype analyses involve numerous steps including physical plant sample collections, data curation, data conversion into different steps for generating users' expected end results, and making analysis results available to researchers and practitioners if needed. There are a number of challenges involved in automating the process of plant genotyping and phenotyping, e.g. reproducibility of experiments, high throughput processing of large amounts of data in various formats (e.g. structured, semi-structured and unstructured), identification of appropriate meta-data for the diverse uses of the data, collecting, abstracting, and loading data into easily accessible structures. Fortunately, several frameworks such as Galaxy [17], GenAp [21], iPlant Collaborative (or iPlant) [27], and LemnaTec [2] are already available to tackle many of the challenges. These technologies also attempt to tackle other problems, such as security, workflow management and accessibility of public datasets.

In this paper, we investigate and analyze the architectures of the candidate frameworks by combining a reverse engineering process (using various tools and manual analysis) and SAAM [19], a scenario-based architecture analysis method. Both software architectures and scenarios are important tools for understanding a system's behaviour. In our investigation, we attempt to understand the four candidate frameworks and determine their strengths and weaknesses by doing a comparison analysis with a set of scenarios using SAAM. We realized that there are three high-level requirements such as scalability, reusability and composability should be satisfied by a PGP system. In order to address these three requirements the PGP system should support (i) a data-centric model [7] capable of handling large and heterogenous data with support for high throughput data processing, (ii) a software component model [23] supporting loosely coupled interactions among different independent components of the system via message queuing, and (iii) an infrastructure model [5] defining services of the system across various machines in the cloud. However, our SAAM analysis reveals that none of the candidate architectures fully support these three models. For example, GenAp only partially supports the data-centric model and the infrastructure model, services of Galaxy are not separated into independent modules that are easily replaceable and pluggable without redeploying the entire system, and core subsystems of iPlant run on different machines that causes performance degradation. Consequently, the candidate systems are not flexible enough to support some advanced features necessary for plant phenotyping, such as high-throughput image analysis, geo-spatial data analysis, scientific and intelligent workflow management, and virtual plant modeling and simulation.

Realizing the shortcomings of the candidate architectures, we provide design recommendations in the form of a reference architecture that supports a wide range of requirements for the scientific research community of plant genotyping and phenotyping. Our conceptual architecture is a combination of the three models discussed above. These models are adopted from recent paradigms [5, 7, 14, 15, 23, 25] of designing cloud-based massive data computation systems. To the best of our knowledge no cloud-based reference architecture exists in the subject domain that can handle the data centric, software component and infrastructure models.

Along with discovering the capabilities of the candidate frameworks, we also discuss our experience with SAAM. For example, while applying SAAM to four different architectures supporting similar functionalities, we realized that when SAAM is applied to pre-built systems, product analysis is a particularly important step. It increases an evaluation team's insights and understanding of the systems.

In short, there are four major contributions of the paper:

- Scenario based architectural analysis of four popular systems for plant genotyping and phenotyping
- Product analysis identified as an important/explicit step in SAAM
- The design of a conceptual reference architecture
- A proof of concept subsystem implementation based on the reference architecture

II. Related Work

Software architectural evaluation provides assurance to developers that their chosen architecture will meet both functional and non-functional quality requirements [31]. There have been different architectural evaluation methods proposed [13, 30] including surveys and comparison frameworks [10, 22, 31]. Among the different methods, SAAM as proposed by Kazman et al. [19], is one of the most used methods. SAAM has been applied to numerous case studies: global information systems, air traffic control, and so on [4, 13]. SAAM identified different design problems in these systems and provided guidelines for fixing them.

In general, architectural evaluation is a central element during the entire software life cycle [28] and has been used for identifying architecturally significant requirements [9], for analyzing and evaluating architectures [13], for managing architectural knowledge [8], for architecture documentation [16], for architecture design [31], and for checking architecture/implementation conformance [32]. There is also recent work that use architectural evaluation methods for improving software product line management [28], for ensuring security of a software system [20], understanding sustainability of software architecture [22], and even early architecture evaluation for large-scale distributed systems [35]. Given that genotyping and phenotyping domains deal with terabytes of data, the architectures of the frameworks in these domains should be evaluated whether they meet not only the relevant quality attributes but also the challenges of Big Data and the specific requirements of the problem domain.

Unfortunately, to the best of our knowledge, there are no studies that evaluate the architectures or model a common reference architecture of Big Data frameworks in the subject domain. However, in recent years, studies of some advanced reference architectures for other cross-domain systems that handle mixed and large data are available [5, 6, 7, 14, 15, 23, 25, 29]. Highlights of these studies include standardizing on a data-centric model, an infrastructure model, and a software-component model that can adapt to the challenges of mixed data computation. We adopted them to introduce a common reference architecture for a cloud-based plant genotyping and phenotyping analysis system (CGPAS). Undoubtedly, these systems work on large and mixed data for diverse and varied sets of use cases. Demchenko et al. [14] discuss a number of aspects of why a data-centric model is necessary. Paakkonen et al. [29] define some reference architectures partially based on a data-centric model. Recently, Apache Beam (Google DataFlow) [7] has been developed based on a strict data model for processing massive-scale, unbounded, out-of-order data on the cloud. Some studies [14, 23, 29] propose new reference architectures for modular systems. We also include components to deal with the data interface, virtual plant modeling, and generic, semi-generic and customizable plugins integration on-the-fly. It is also necessary to describe a cloud infrastructure that fulfills the diverse requirements within the reference architecture. We adopt a heterogeneous cloud infrastructure [5, 15, 25, 26] for our conceptual architecture. Our analysis study is helpful for illustrating the challenges of developing a phenotyping system that handles Big Data, and developers will be able to implement such a system in a more advanced way by being more aware of the challenges.

III. Analysis Methodology

We adapted the architectural evaluation methodologies proposed by Kazman et al. [18, 19] and Roy et al [30] to evaluate the state-of-the-art platforms for genotyping and phenotyping. Our methodology combines different techniques in order to find out differences among the candidate platforms' architectures, including reverse engineering their software architectures, and scenario-based architectural and dynamic analysis. Reverse engineering software architectures [12, 34] alone does not provide a deep understanding of architectures. If the extracted architectures are refined with evaluation methods, they become more understandable to developers or architects [30]. Keeping this in mind, we extracted architectures of the candidate frameworks using tools and evaluated them using SAAM [19].

Steps of our developed methodology for evaluating candidate architectures is described as follows

Step 1. Product Analysis: In this step, we studied the candidate frameworks by using their executables, source code, online documentation, videos, asking questions of the product developers, and presenting and demonstrating them to various practitioners. As an outcome of the step, we devised a question answer table to guide the comparative analysis of the candidate products as shown in Table I. This table is helpful to learn about the products and their important differences

Table I
SOME IMPORTANT PROPERTIES OF THE CANDIDATE FRAMEWORKS

#	Property Text	Galaxy	iPlant	GenAp	LemnaTec
1	What cloud API is used for allocating virtual resources?	CloudMan	Atmosphere	Galaxy Cluster	Scanalyzer3D
2	What is the name of the cloud service provider?	Amazon Ec2	Atmosphere Server	Canada HPC	SUSE Linux
3	Does it provide Phenotype image analysis/ storage tools?	No	Yes, Bisque	No	Yes
4	Does it offer third party API support?	Yes, BioBlend	Agave API	MUGQIC	-
5	What kind of scheduling algorithm is used?	SLRUM	HTcondor	PBS scheduler	-
6	What is the underlying database?	SQLite	MongoDB	Spark SQL	SQLite
7	How large data files are stored?	No BigData yet	HPI-Cluster (IROD,FUSE)	CVMFS	LemnaBase
8	How meta data is stored?	Simple XML	Sematic Web(iRODS)	ADAM(R)	CSV/SQL
9	Does it support scientific workflow?	Yes	Yes (Taverna engine)	Yes	Yes
10	Access provided to publicly available datasets/analysis?	Yes	Yes	Yes	Yes
11	Does it support MapReduce for distributed computing?	No	No	Yes with pipeline	No
12	Does it support applications through web portals?	Yes	Yes	Yes	No
13	Does it fully dependent on third party developers?	Yes	No	Yes	No
14	What kind of visualization tools it offers?	IGV+UCSC+IGB+	CoGe	Same as Galaxy	LemnaMiner
15	Does it support independent 3rd party tool integration?	Cmd Line App	Cmd Line App	Cmd Line App	No
16	Does it support role based security management?	Admin Panel	Admin Panel	Admin Panel	No

quickly. The SAAM authors, Kazman et al. [18] used this step in their architectural analysis implicitly. In this paper, we make this step explicit in our proposed methodology. Our understanding is that when SAAM is applied to an already implemented system, it is an effective step to execute during the architectural analysis. Moreover, this step is also helpful for developing various usage scenarios.

Step 2. Candidate Architecture Extraction: In this step, we extracted the architectures of candidate frameworks using various tools such as PyDev, PyCharm, ObjectAid and Eclipse since most of the tools are Python based. These tools generate UML class diagrams based on the class relationship within each module. However, class diagrams alone are not enough to extract an architecture since it contains other scripts (such as Linux shell scripts). Therefore, we also manually analyzed the source code ourselves to mine crucial attributes and used the built-in Python script ModuleGraph to identify dependencies among different source code modules. Finally, we obtained UML activity diagrams based on our analysis.

Step 3. Scenario Development: In this step, we elicited scenarios. Each scenario is a representation of a particular quality attribute and expresses tasks illustrating the kinds of activities the system must support and the kinds of anticipated changes to be made to the system over time. These scenarios represent tasks relevant to different roles such as end user/customer, marketing, system administrator, maintainer and developer.

Step 4. Individual Scenario Evaluation: For each scenario, we determine if it is direct or indirect. A direct scenario describes behaviour that can be supported without any modification in the system whereas an indirect scenario describes behaviour that is not supported by the system. In order to support an indirect scenario, the system must undergo some changes. We discuss changes that are required in different modules/components of candidate systems to support the elicited indirect scenarios. This allows to understand the interactions between different modules of the systems.

Step 5. Candidate Architecture Comparison: In this step, we compare different architectures by identifying their outstanding properties along with their other strengths and weaknesses.

IV. PRODUCT ANALYSIS

In this step we first explore the features of the candidate frameworks. Our first candidate system, Galaxy, provides support for biomedical research. One of the important features of Galaxy is that it has History and Workflow options which are unique to Galaxy compared to the other tools. As well, Galaxy is recently able to be hosted in a cloud environment, which helps a user use it through the web. The second candidate system, GenAp [21] tried to design a more interactive database analysis framework over Galaxy. GenAp is a replication of Galaxy, but has a datahub management pipeline and individual server instance creating facilities. For dedicated plant phenotyping, potential candidate frameworks include: iPlant [27], and LemnaTec [2]. iPlant supports a number of applications related to plant phenotyping. In addition, iPlant facilitates flexible app creation, image annotation, Google map service for the image, and HPC options, which are unique to iPlant. LemnaTec is a desktop application which provides unique options for non-invasive plant data collection and life cycle observation of plant through imaging. This framework fails to provide support for collaborative research as it is a standalone desktop application.

As part of the product analysis step we did some dynamic analysis to learn the products' features and performance. For example, for evaluating scenario S10 (Table II), we consider the use case *Import data from 3rd party tool and convert the imported file into different format*. For Galaxy and GenAp, we imported a BED format genome file from the NCBI database and converted it into BAM format. Although all of them took < 2 minutes, Galaxy is faster than both iPlant and GenAp. However, GenAp is faster than iPlant.

Table II
SCENARIO-BASED COMPARISON OF THE CANDIDATE FRAMEWORKS

Properties	#	Scenarios	Galaxy	iPlant	GenAp	LemnaTec
Usability	S1	A biologist or clinician wants to explore their data using tools available in the web-based platform.	Direct, well structured	Direct, well structured	Direct, updated and well structured	Mainly desktop
	S2	A computational biologist wants to use state-of-the-art pipelines to analyze their data without the burden of installing and maintaining all of the associated bioinformatics tools.	Direct, well structured	Direct, well structured	Complex initial setup	Complex
	S3	A sequencing centre pushes raw data into a Web based project created by a user to facilitate data access and analysis.	Indirect; Client, Scripts and Lib need to be changed	Indirect; Services, Atmosphere, Models need to be updated	Partial support; use unified data interface	Limited, not web
	S4	A computational agriculturist wants to annotate an image (leaf, root) with paint and brush (if possible), and also wants to save and update the annotation.	Indirect; Client, Tools, Scripts and Lib need to be updated	Direct, Well structured	No tools found	Support
	S5	A developer wants to plug-in his image analysis tool or pipeline for doing image analysis, registration and segmentation.	Indirect; Config, Cron, Tools and Scripts need to be updated	Direct	Less supportive, difficult to integrate	Difficult
Flexibility	S6	A developer wants to upload an intermediate processed work flow data in the system for further analysis.	Direct, updated and well structured	Direct but needs to be improved, not user friendly	Direct, updated and well structured	Not supported
	S7	A computational biologist wants to add a new RNA analysis tool HISAT in the existing system.	Direct but not well structured	Direct and user friendly	Direct but not well structured	Not supported
	S8	A phenotypic researcher wants to work with a visual map interface to automatically extract environmental information (weather, soil info, landscape etc.) of a certain region and integrate it with plant traits analysis.	Indirect, Client, Config, Lib and Scripts need to be changed	Indirect, input parameter list needs to be updated for Migration and Model components	Not supported yet	Not supported yet
Security	S9	A user wants to share data from a project with another scientist to do joint data processing and exploration.	Secured	Secured	Secured	Not secured
Performance	S10	A computational agriculturist wants to convert a 8.6MB BAM file to a BED file for visualization.	Fastest, 23 sec.	Slower than Galaxy and GenAp, 73 sec.	Slower than Galaxy, 47 sec.	–
Modifiability	S11	Get an update after execution of each of the steps of a workflow or image processing pipeline.	Indirect; Need to modify Client, Config, Lib and Workflow design	Indirect; Models and Core functionality need to be updated	Source Code not analyzed	Not supported yet
	S12	A user wants to modify any part of code base of an available workflow and wants to save it in his private history as a new version of the workflow.	Not Available; Need to modify Config, Lib and Workflow design	Not available; Models, Core and Service design needs to change	Not available	Not available

We also developed a question answer framework (shown in Table I) to summarize important features. The table helps us to quickly determine important differences between the candidate frameworks and helps us in our scenario-based architectural analysis described in Section VI.

V. SCENARIO DEVELOPMENT

As discussed in Section III, we carefully selected 12 scenarios (shown in the third column of Table II). We selected the scenarios based on our product analyses and discussions with various bio-computation researchers, agriculturists and image processing researchers who work with the agriculture group at U of S. We also collected a few scenarios from the GenAp website. These scenarios show important usages of the system while reflecting various quality attributes (e.g., usability, flexibility, security and performance) as required by different stakeholders, such as computational agriculturists, computational biologists and developers. For example, scenario *[S2:] A computational biologist wants to use state-of-the-art pipelines to analyze their data without the burden of installing and maintaining all of the associated bioinformatics tools* expresses a usability requirement of the system for data analysis by a computer biologist. Additionally, the scenario *[S7:] A computational biologist wants to add a new RNA analysis tool HISAT in the existing system* is a flexibility requirement of software developers.

VI. EXTRACTED CANDIDATE ARCHITECTURES AND SCENARIO-BASED ANALYSIS

In this section, we briefly describe our extracted architectures of the frameworks that handle plant genotyping and phenotyping along with our scenario-based evaluation.

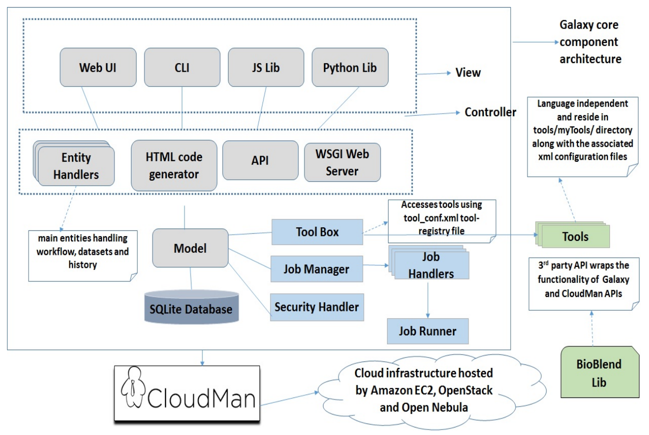
Galaxy core component architecture
Web UI
CLI
JS Lib
Python Lib
View
Controller
Language independent and reside in tools/myTools/ directory along with the associated xml configuration files
Entity Handlers
HTML code generator
API
WSGI Web Server
Accesses tools using tool_conf.xml tool-registry file
main entities handling workflow, datasets and history
Model
Tool Box
Tools
Job Manager
Job Handlers
3rd party API wraps the functionality of Galaxy and CloudMan APIs
SQLite Database
Security Handler
Job Runner
BioBlend Lib
CloudMan
Cloud infrastructure hosted by Amazon EC2, OpenStack and Open Nebula

Figure 1. Galaxy Cloud Architecture

A. *Galaxy*

The Galaxy [17] architecture consists of reusable software components as shown in Figure 1 (extracted following Section III, Step 2). Galaxy infrastructure is developed using the Model View Controller pattern[1] as shown in Figure 1. The Model is responsible for adding and retrieving items from the relational database SQLite. Data entities handling workflow, datasets, and history are mapped on to SQLite database tables. There is a controller for each of the main entities handling workflow, histories and datasets. One of the important controllers is the API enriched with libraries for various programming languages helping the developers in building different applications and tools in the Galaxy platform, e.g., BioBlend [33] is developed using the Galaxy API. The Job Manager is responsible for executing jobs with allocation of resources and keeping track of jobs by adding sequence number whereas the ToolBox collects and manages all types of tools in a language independent manner using the tool_conf.xml registry file. In addition, Galaxy allows third party visualization tools as plug-ins.

1) Individual Scenario Evaluation of Galaxy Architecture: In this step (discussed in Section III), we executed the scenarios onto the extracted Galaxy architectural artefacts including the conceptual architecture (Figure 1) and on derived UML diagrams (e.g. Figure 2). From different scenarios discussed in Table II we noticed that scenarios S1, S2, S6, S7 and S10 are directly supported by the Galaxy architecture. The rest of the scenarios (such as S3, S4, S8, S11 and S12) are indirect scenarios that are not supported by Galaxy but can be integrated or updated with the present architecture by modifying the system. The changes that are required to support those scenarios with the present architecture are stated in Table II.

For scenario S3, our study shows that Galaxy does not yet support processing of heterogeneous data. It mainly works on a processed dataset which is already collected and manually inserted in a SQLite database without having a unified data input mechanism. To add this feature, it would be necessary to change Galaxy's Client, Scripts and Lib modules. Or a wrapper can be added to support a Unified Data Interface in the Galaxy architecture. One of our candidate architectures, GenAp, which incorporated their architecture with the Galaxy public instance, already added this feature in their work.

Similarly, S4, which pertains to image processing behaviour is also not available in Galaxy. Galaxy is mainly focused on searching for data in a flat file database (SQLite) with a user friendly query. Galaxy developers focused on manually extracted (by a researcher or a biologist) genotype and phenotype information from different genes and chromosomes. Thus, they did not focus much on the image processing pipelines or analysis. But Galaxy supports different options to adapt this behaviour. First, Galaxy developers need to build a tool to support the functionality of S4. Then they can add the tool to Galaxy's Toolshed or, like other available tools, they can integrate the tool in their web version by changing the Galaxy Client, Scripts and Lib modules. Or they can provide direct web-based support for this scenario. S5 relates to the integration of image processing pipelines and as with S4, the main Galaxy instance would need to be modified. Galaxy's Config, Cron and Tools modules need to be changed. As Galaxy is an open source project, a group of developers have developed BioMina [1] based on Galaxy to support image analysis. But they do not do image registration or segmentation which is required for phenotyping analysis.

S6 focuses on workflow management and by analyzing the source code of Galaxy, we noticed that its workflow management system is well-structured and intelligent. S11 and S12 also deal with workflow or the image processing pipeline. For S11, in Galaxy, all the submitted tasks and jobs of a user developed workflow run as a background process and the user gets an update only once the whole process executes or fails. If it fails, Galaxy provides logs, which is not particularly user friendly. To make it more usable, it would be good to inform the user after execution of each step as a flash message or as a notification. To provide this feature Galaxy's Client and Workflow code design needs to be updated. For scenario S12, from our study, it is observed that sometimes a researcher or a biologist needs to modify steps or the number of time a tool is executed in an existing fully developed workflow. To add this flexibility, Galaxy's workflow architecture and code should be updated.

In order to determine the coupling among different components in Galaxy, we worked on the direct scenario S7 where we look into how a tool can be added to Galaxy. Basically, the same approaches discussed for S4 is needed to add the tool. From the activity diagram of Figure 2 we notice that tool integration involves many actions, such as tool shed configuration, data, datatype and metadata handling and loading utility library for tools from the toolbox. Each of the actions interact with the various modules shown in

[1]http://www.laputan.org/pub/papers/POSA-MVC.pdf

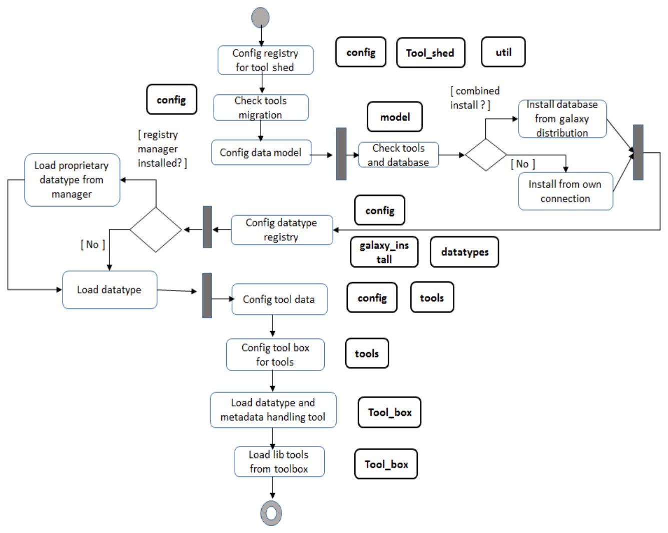
Config registry for tool shed
config
Tool_shed
util
config
Check tools migration
[combined install ?]
Install database from galaxy distribution
model
[registry manager installed?]
Config data model
Check tools and database
[No]
Load proprietary datatype from manager
Install from own connection
config
Config datatype registry
[No]
galaxy_install
datatypes
Load datatype
Config tool data
config
tools
Config tool box for tools
tools
Load datatype and metadata handling tool
Tool_box
Load lib tools from toolbox
Tool_box

Figure 2. Activity diagram for tool integration in Galaxy.

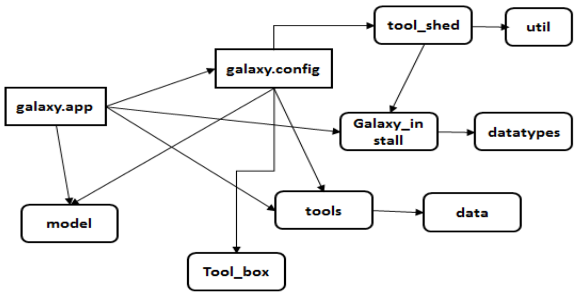

Figure 3. Component interaction diagram for tool integration in Galaxy

Figure 3. For this scenario, it is clear that the Galaxy system interchanges execution flow with numerous components, and hence, we conclude it is a tightly coupled system.

S8 describes important parameters for phenotyping research and analysis. Although Galaxy has a phenotype association tool available, it does not support weather, climate or soil information for phenotyping analysis. To support this, Tools, Lib and Client modules, especially in the phenotyping association tool, need to be updated.

In summary, our architectural analysis reveals that Galaxy's underlying architecture is modularized. The modules in the architecture are tightly coupled as different modules interact with each other in order to execute a scenario while maintaining moderate separation of concerns. The public instance of Galaxy does not support image-based phenotyping as it does not offer an image processing pipeline. This can be added through Toolbox with source code modification as discussed above. However, we have noticed that the image processing pipeline has been added in one of the galaxy servers, called Image Analysis and Processing Toolkit[1]. At present, one of the Galaxy instances, called BioMina [1] supports image analysis.

[1]http://cloudimaging.net.au

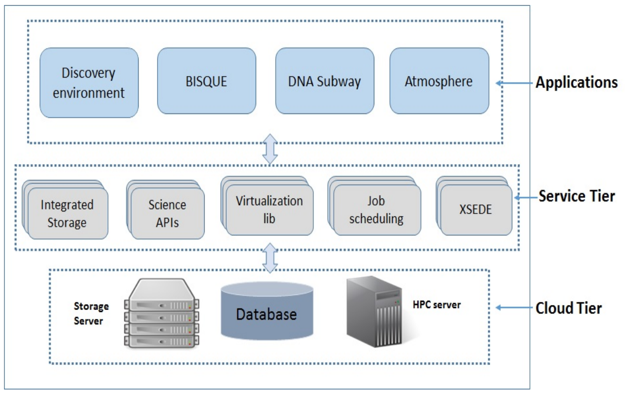

Figure 4. iPlant Collaborative architecture

B. *iPlant Collaborative*

The iPlant [27] architecture is a combination of independent applications as shown in Figure 4 and discussed below.

Atmosphere is a cloud service that allows users to launch their own virtual machines, whereas Discovery Environment (DE) is the primary web interface and platform to access the powerful computing, storage, and analysis application resources. DNA Subway makes high-level genome analysis broadly available to students and educators, and Bisque is the image analysis tool [24]. This platform provides links to different scientific APIs, and users can go to a link to access an API. For job scheduling, HTcondor is used and for creating cloud VM machines, OpenStack is used. iPlant has two approaches for storing data: a central database for smaller datasets for meta data and some private datasets; and, IRODS for storing images.

1) Individual Scenario Evaluation of iPlant Architecture: For iPlant, scenarios S1, S2, S4, S5, S6 and S7 are directly supported whereas the others are not without the architecture being added to or updated (see Table II for details).

For S3, as of Galaxy, iPlant does not support raw data input directly. IPlant developers work on a processed dataset. To support S3, the source code of the Atmosphere, Models and Services modules need to be changed. Moreover the DE source code needs to be redesigned.

Scenarios S6 and S7 focus on the workflow management and the tool integration process respectively. Workflow and tools integration for iPlant are implemented in DE, a Java project. We use a free tool ObjectAid to extract class dependency diagrams then manually draw the module interaction diagram (Figure 6) from the activity diagram (Figure 5). DE uses Google GIN API for handling some critical actions of app handling such as building and executing an app. The interaction diagram in Figure 6 only presents some abstract modules, but at the source code level each of the modules has numerous sub-packages. However, from the source code analysis and class dependency diagram we did not find any reference class or package for workflow. It appears that the workflow is implemented in an app (there

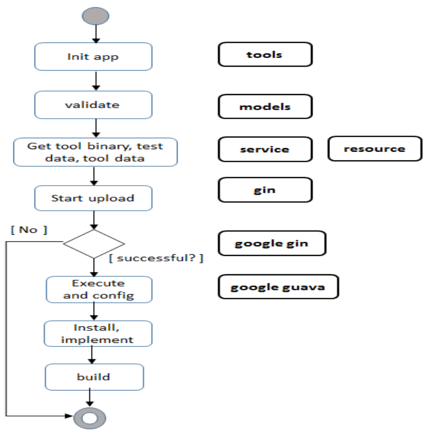

Figure 5. Activity diagram for tool integration in iPlant DE

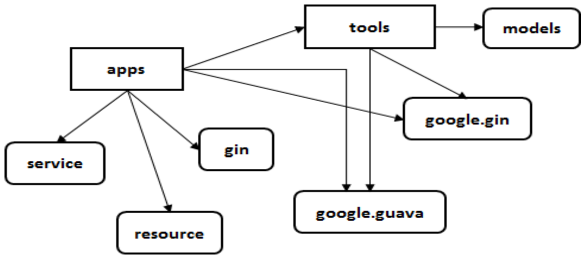

Figure 6. Interaction of components for tool integration in iPlant

is an option in DE for creating an app like workflow), and there is no special module for workflow handling. A user can upload a working workflow or create it inside the DE.

After careful observation of the architecture and source code, we noticed that the DE of iPlant has complex module interaction, thus imposing heavy coupling (it is even hard to find the reference code for the workflow implementation). Apart from this, iPlant uses third party APIs for managing tools and apps. From our source code analysis of DE, Atmosphere and Bisque, we observe that there is little interaction among these components. Although this is convenient as a loosely coupled system, they were developed with different technologies which makes it more difficult for a developer to modify source code and add new functionality. Besides, various application servers need to be deployed for various technologies. There is no separate module for workflow management which violates the separation of concerns principle of system design.

C. GenAp

GenAp [21] integrates its whole system with one of the Galaxy instances which is locally maintained with a focus on Big Data handling. GenAp offers its own MUGQIC pipeline tools which does not require any command-line knowledge. For each project creation, they communicate with Galaxy over HTTP which is a performance issue for GenAp. As discussed in Section IV, our dynamic analysis reveals that GenAp is much slower than other frameworks. They developed a central database for storing smaller files and indexing data. At the same time, GenAp communicates with an open source

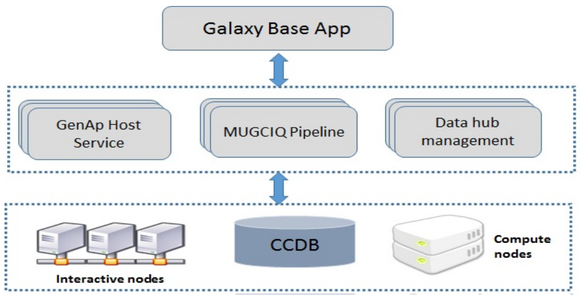

Figure 7. GenAp architecture

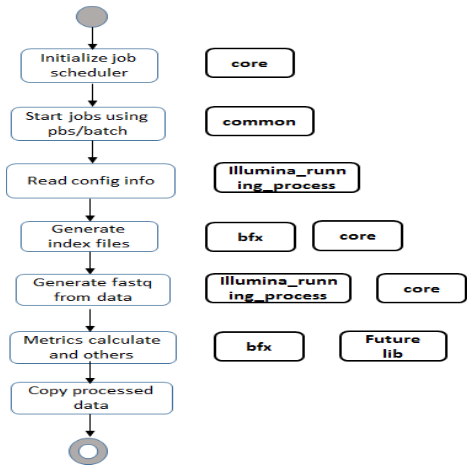

Figure 8. Activity diagram of Illumina pipeline service

database for reference data. Their own large size files are stored in a different large size database, and CVMFS is used for managing their file system. Comparatively, it is an easy architecture but is fully dependent on 3rd party developers[1].

1) Individual Scenario Evaluation of GenAp Architecture: The majority of the GenAp source code is based on the Galaxy project, and workflow and tool integration are similar to Galaxy. Therefore, most of the scenarios are similar to Galaxy. However, GenAp has an additional data hub pipeline MUGQIC. So for scenario S2 (details in Table II), we evaluated the Illumina pipeline processing for workflow execution. By following similar steps for Galaxy and iPlant, we obtained an activity diagram for scenario S2 (Figure 8). Running Illumina pipelines involves a number of actions such as generating index files, fastaq conversion, metrics calculation, and even blast tool execution. In summary, from our source code analysis and the activity diagram analysis of Illumina processing, we noticed that every step of all the pipelines are executed through a job scheduler, therefore the GenAp DataHub pipeline can be considered a loosely coupled system. As GenAp is based on Galaxy, it also does not support a message queuing architecture.

D. LemnaTec Software

LemnaTec OS [2] is a well-known plant phenotyping system in the world. An integrated set of modules provides

[1]https://bitbucket.org/mugqic/mugqic_pipelines/src

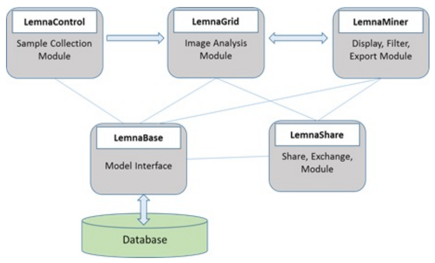

Figure 9. LemnaTec image analysis base plant phenotyping architecture

rich functionality to control any hardware configuration and then record and analyze the resulting sets of data. LemnaTec's modular architecture is shown in Figure 9.

1) Scenario-based analysis of LemnaTec Architecture: As LemnaTec is commercial desktop software, its source code is not available, so we studied their documentation and explored only LemnaShare features and extracted the abstract architectural view. Like other analysis, we tried to evaluate the LemnaTec architecture based on our 12 scenarios (mostly provided by the LemnaTec development team) which is discussed in Table II.

VII. Comparison of Candidate Architectures

We compared the four candidate architectures based on our individual scenario evaluation and dynamic analysis findings. Our comparison result is shown in Table II, along with a mapping of the scenarios to different quality attributes, such as usability, flexibility, security and performance. A summary comparison with a number of advanced features are shown in Table III. From the tables, we see that iPlant supports most of the scenarios, except scenarios S3, S8, S11 and S12, whereas Galaxy, GenAp and LemnaTec do not have support for many scenarios. If we consider S3 and all kinds of data interchange operations, none have fully supportive components. We found that GenAp supports a heterogeneous and open cloud infrastructure module for data-read write operations from a user's data-machine (distributed storage too); others do not. For the performance scenario S10, we noticed that Galaxy and iPlant are much faster than GenAp since GenAp communicates with the Galaxy server via an HTTP request. In addition, during our scenario analysis, we revealed that in all the four candidate architectures, components are tightly coupled as none of the architectures support message queuing for component interactions. Also none of the architectures support image bundling or grouping using MapReduce technology for high performance. Moreover, for scenario S4, we found that Galaxy and GenAp do not provide image analysis pipelines to measure plant leaf growth, whereas iPlant and LemnaTec do support this behaviour.

We were unable to identify a defined data flow model interface or message queuing module at the architecture level for any of the candidate architectures. However, all had components to support a mixed data model, and an infrastructure as service module (although Galaxy has a complex structure).

Table III
Summary Architecture Comparison w.r.t. Advanced Features

Tools	UDI	MDM	DFM	MRDM	MQM	PIM	HIOM	ISM
GenAp	$\sim$	$\sim$	$\odot$	$\oplus$	$\odot$	$\odot$	$\sim$	$\oplus$
Galaxy	$\sim$	$\sim$	$\odot$	$\odot$	$\odot$	$\sim$	$\odot$	$\sim$
iPlant	$\sim$	$\oplus$	$\odot$	$\odot$	$\odot$	$\odot$	$\odot$	$\oplus$

UDI - Unified data interface, MDM - Mixed data model
DFM- Data flow model component, MRDM- Map reduce module
MQM - Message Queuing module, PIM - Plugin integration module
HIOM - Heterogeneous and open cloud infrastructure module
ISM - Infrastructure as service module.
$\oplus$ - Full, $\sim$ - Variant/partial, $\odot$ - Doesn't exist or not found

Nonetheless, some other advanced components presented in Table III are not included in the architecture of all these tools. Therefore, this analysis led us to define a common architecture for advanced cloud-based plant phenotype and genotype systems.

VIII. A Conceptual Reference Architecture

A cloud-based system facilitates reproducible science by sharing datasets and high-throughput processing pipelines with collaborators in the domain of genotype and phenotype analysis. Galaxy, iPlant, GenAp and LemnaTec are the most widely used systems for the subject domain. An off-the-shelf reference architecture would be valuable for a variety of stakeholders, including eScience developers, and researchers involved with plant genotyping and phenotyping systems. Unfortunately, no common reference architecture (RA) is available that can serve as a design guideline considering the wide-spread support of cloud-based genotyping and phenotyping analysis (CGPA). As a result, we extracted the design architectures of these systems using SAAM evaluation techniques based on some use-cases and scenarios described in previous sections. Adapting those use-cases, scenarios and requirements of the P2IRC project of University of Saskatchewan [3] we propose a conceptual RA. An RA can be of two [29] types: (i) High Level, and (ii) Subset of System Functionality Level. Here we consider the high-level approach, and adapt reference architectures [23, 29] from cross domain systems (those which handle heterogeneous and large data). Additionally, we observed that providing only a software component model of such cloud-based systems is not sufficient to define a reference architecture; therefore, we include other important aspects in our proposal. The conceptual reference architecture diagram is presented in Figures 10 and 11. In fact, our conceptual architecture is a set of design guidelines described as follows:

Data-centric Model: Recently researchers have been using a data-centric model [7, 14] for efficiently analyzing, sharing, and handling a varied set of massive data in a cloud-based system. Defining a standard model for the management, processing and interchange of data for a cloud system is one of the most important aspects of an RA. Cloud-based genotyping and phenotyping analysis systems work with structured, unstructured and semistructured data, and with various datasets, such as genome, geospatial, and image datasets. These data are processed and analyzed with various workflows and pipelines. Researchers recently have been focusing on specific data models [7] in order to provide a more effective and collaborative workspace. As well, a common data-flow model is useful for implementing discoverable and sharable data processing logic. Thus, we recommend defining a specific data-flow interface for each category of data so that it can be easily inherited from common workflows and pipelines in the system. Moreover, data may be input to the cloud-based system from heterogeneous sources in the open cloud. In our architecture, we recommend using a unified data interface (Figure 10) for interchanging input and processed data between the system and the user source. Various data-storage mechanisms should be defined for the cloud system based on the categories of data rather than a common storage for all categories of data.

Software Component Model: The modular architectural paradigm for designing a cloud system that handles large and heterogeneous data has recently focused on a loosely coupled message-based structure [6, 23, 29] in order to support components and sub components of the system being updated, replaced or plugged-in by an open community during the lifetime of the product. To develop a flexible and adaptable system with this dynamic context, the components must be independent unlike a traditional MVC or three tier architecture. In this case, independent means loosely coupled, smoothly replaceable and pluggable. Along with the common modules defined in the architecture model of the candidate systems, our model includes some new components which are presented in Figure 10. The new modules in our architecture are: a cluster processing module for a big-data platform, a unified data interface module, a high-throughput plugin module, a web-based plant modeling module, a GIS module, and a remote sensor module. Easily replaceable and pluggable modules without redeploying the entire system are represented with a dashed line in the figure. Apart from these, we also recommend providing API libraries to easily develop workflows, pipelines, and plugins integrable to the base system. A module that automatically integrate dependable libraries on-the-fly while uploading the plugins by the users would make the system more extensible. The last but not least principle is that multiple workflows, pipelines or data-analysis job execution should be handled by a message-queuing [6] module (MQM) irrespective of whether they run in the base server or in the cloud-cluster.

Infrastructure Model: Cloud-based genotyping and phenotyping analysis systems (CGPAS) need to interact with various machines including open cloud infrastructures. Thus, the infrastructure model is an important part of designing such systems [15, 25] to firmly define which parts of the infrastructure serve which services, which parts are public, which parts have restricted access, and what are the

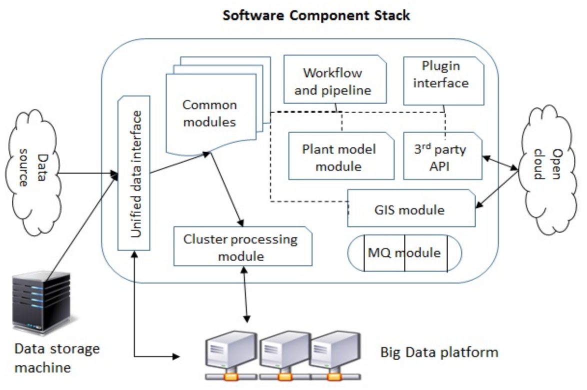

Figure 10. Software component model

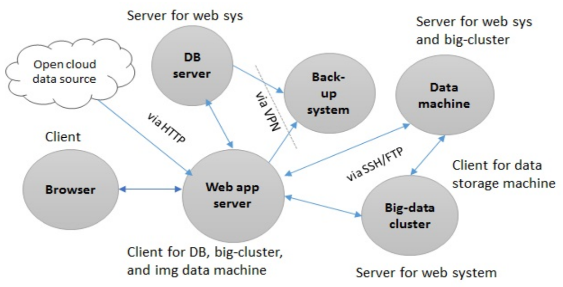

Figure 11. Heterogeneous cloud infrastructure model

communication protocols. We recommend a heterogeneous cloud infrastructure model for CGPAS as presented in Figure 11, where heterogeneous refers to supporting diverse communication protocols and network models.

We developed a subsystem for plant phenotyping to help prove our conceptual architecture. Since the studied candidate systems do not support high-throughput image analysis pipelines (HIAP), we designed a cloud-based system to support that. In this paper, we briefly discuss a few processing pipeline use cases in mapping with our conceptual architecture (full analysis is beyond the scope of this paper). First, we set up a heterogeneous cloud infrastructure as defined in our architecture including a four-node Spark cluster. Then, we defined a common data-flow model for image processing pipelines. This model leverages the development of HIAPs executed on a Big Data cluster. Additionally, we developed an API library (based on PySpark) for developing high-throughput plugins for our cloud system following the defined data-model. Pre-built HIAPs in our subsystem inherit the data-flow interface in the API. This data-model and API library also leverages the effort of researchers to smoothly work with image analysis pipelines/workflows for plant phenotyping while ensuring flexible collaboration among researchers. Furthermore, we designed a unified data access module that can load large image datasets from any remote data machine or other cloud sources (a test case of data loading from a remote machine is presented in Table IV). We test the high-throughput image analysis pipeline, plugin integration, and remote data loading use-case with more than 10K images with our small Spark cluster (results are shown

Table IV
EXECUTION TIME OF VARIOUS IMAGE PIPELINES IN OUR SYSTEM

Pipeline	**No. of Images**	**Execution Time**
Image Matching	10K	42 Mins
Clustering (5)	10K	1.6 Hrs
Image registration	$> 10K$	1.9 Hrs

in Table IV). Thus, our subsystem follows both the data-centric model, and heterogeneous infrastructure model. Apart from these we introduce an independent module for plant modeling and simulation based on WebGL. We compare the web-graphics module with a traditional graphics library and found that the web graphical library is more effective for plant simulation and modelling. In the cloud-based system, producing a plant model with the GL library is more flexible on the server side but less-interactive with a bandwidth bottleneck at the client side. The independent API, unified-data access module, plant simulation module, HIAPs module, and HIAP plugin uploading without re-deployment of the system indicate that the design of our system supports a software component model for plant phenotyping. Moreover, our system is capable of MapReduce distributed processing of large data with promising performance.

IX. CONCLUSION

Plant genotyping and phenotyping analysis (PGPA) is required to handle large datasets and involves numerous steps, from physical plant sample collection to sharing analysis results and scientific workflows. Thus, it is a challenge to develop an efficient and cost effective integrated environment. In this paper, we described our analysis of some existing popular tools for PGPA using SAAM, examined weaknesses and strengths of those frameworks, and conducted a comparative analysis (Table II and III). Our comparative analysis determined that iPlant is a strong tool that, unlike the others, provides support for image processing and, using Google Maps, for geo-spatial data. Unfortunately, it is not as flexible as Galaxy for workflow creation. GenAp is based on Galaxy with an extra DataHub server. LemnaTec is a desktop-based commercial tool dedicated for plant phenotyping. Since each of these tools has limitations, we introduced a conceptual reference architecture to better support the broad range of requirements, and developed a subsystem as a proof of concept. We believe that our study would be helpful to overcome the challenges of developing a cloud-based PGPA system that deals with large and mixed datasets. As future work, we will update our conceptual architecture with additional functionality detail, and test its efficacy with various evaluation techniques and an upgraded implementation.

ACKNOWLEDGEMENTS

We thank the developers of LemnaTec for helping us evaluate their system. This work is supported in part by the Canada First Research Excellence Fund (CFREF) under the Global Institute for Food Security (GIFS).

REFERENCES

[1] BioMina. http://biominavm-galaxy.biomina.be/galaxy/.

[2] LemnaTec. http://www.lemnatec.com/products/.

[3] P2IRC Project. http://p2irc.usask.ca/. University of Saskatchewan.

[4] SEI-SAAM: http://www.sei.cmu.edu/architecture/scenario_paper/ieee-sw2.htm.

[5] J. M. F. V. T. J. Afgan E, Chapman B. Using cloud computing infrastructure with cloudbiolinux, cloudman, and galaxy. In *Curr Protoc Bioinformatics*, 2012.

[6] D. Agarwal and S. K. Prasad. Lessons learnt from the development of gis application on azure cloud platform. In *2012 Proc. Cloud Computing*, pp. 352–359, 2012.

[7] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. J. Fernández-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt, and S. Whittle. The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *Proc. VLDB Endow.*, pp. 1792–1803, 2015.

[8] M. A. Babar, T. Dingsyr, P. Lago, and H. van Vliet. *Software Architecture Knowledge Management: Theory and Practice*. Springer, 2009.

[9] M. Barbacci, R. Ellison, A. Lattanze, J. Stafford, C. Weinstock, and W. Wood. *Quality Attribute Workshops QAWs*. 3rd edition, 2003.

[10] H. P. Breivold and I. Crnkovic. A systematic review on architecting for software evolvability. In *Proc. of ASEC*, pp. 13–22, 2010.

[11] A. Caciula. Optimization techniques for next-generation sequencing data analysis. In *ScholarWorks GSU*, pp. 446–58, 2014.

[12] E. J. Chikofsky and J. H. Cross II. Reverse engineering and design recovery: A taxonomy. *IEEE Softw.*, pp. 13–17, 1990.

[13] P. Clements and M. K. R. K. Kazman. *Evaluating Software Architectures: Methods and Case Studies*. Addison-Wesley Professional, 2002.

[14] Y. Demchenko, C. de Laat, and P. Membrey. Defining architecture components of the big data ecosystem. In *2014 Proc. CTS*, pp. 104–112, 2014.

[15] C. S. Feng Yu and K. A. Schueller. A design of heterogeneous cloud infrastructure for big data and cloud computing services. *Open Journal of Mobile Computing and Cloud Computing*, pp. 1–16, 2014.

[16] D. Garlan, F. Bachmann, J. Ivers, J. Stafford, L. Bass, P. Clements, and P. Merson. *Documenting Software Architectures: Views and Beyond*. Addison-Wesley Professional, 2nd edition, 2010.

[17] J. Goecks, A. Nekrutenko, J. Taylor, and Galaxy Team. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome biology*, pp. R86, 2010.

[18] R. Kazman, G. Abowd, L. Bass, and P. Clements. Scenario-based analysis of software architecture. *IEEE Software*, pp. 47–55, 1996.

[19] R. Kazman, L. Bass, M. Webb, and G. Abowd. SAAM: A method for analyzing the properties of software architectures. In *Proc. of ICSE*, pp. 81–90, 1994.

[20] J. Klein, R. Buglak, D. Blockow, T. Wuttke, and B. Cooper. A reference architecture for big data systems in the national security domain. In *Proc. of ICBDSE*, pp. 51–57, 2016.

[21] C. Kozanitis and D. A. P. Patterson. GenAP: a distributed sql interface for genomic data. In *BMC Bioinformatics*, 2016.

[22] H. Koziolek. Sustainability evaluation of software architectures: A systematic review. In *Proc. of the Joint QoSA and ISARCS*, pp. 3–12, 2011.

[23] M. Krämer and I. Senner. A modular software architecture for processing of big geospatial data in the cloud. *Comput. Graph.*, pp. 69–81, 2015.

[24] K. Kvilekval, D. Fedorov, B. Obara, A. Singh, and B. Manjunath. Bisque: A platform for bioimage analysis and management. *Bioinformatics*, pp. 54452, 2010.

[25] L. Liu. Computing infrastructure for big data processing. *Frontiers of Computer Science*, pp. 165170, 2013.

[26] M. Masoodian and S. Luz. Heterogeneous client-server architecture for a virtual meeting environment. In *Proceedings 8th Euromicro Workshop on Parallel and Distributed Processing*, pp. 67–74, 2000.

[27] N. Merchant, E. Lyons, S. Goff, M. Vaughn, D. Ware, D. Micklos, and P. Antin. The iplant collaborative: Cyberinfrastructure for enabling data to discovery for the life sciences. *PLoS Biol*, pp. 1–9, 2016.

[28] F. G. Olumofin and V. B. Mišić. A holistic architecture assessment method for software product lines. *Inf. Softw. Technol.*, pp. 309–323, 2007.

[29] P. Pääkkönen and D. Pakkala. Reference architecture and classification of technologies, products and services for big data systems. *Big Data Res.*, pp. 166–186, 2015.

[30] B. Roy and T. C. N. Graham. *An Iterative Framework for Software Architecture Recovery: An Experience Report*, pp. 210–224, 2008.

[31] B. Roy and T. C. N. Graham. *Methods for evaluating software architecture: A survey*, Queens Technical Report:545, 82pp, 2008.

[32] N. Sangal, E. Jordan, V. Sinha, and D. Jackson. Using dependency models to manage complex software architecture. *SIGPLAN Not.*, pp. 167–176, 2005.

[33] C. Sloggett, N. Goonasekera, and E. Afgan. BioBlend: automating pipeline analyses within galaxy and CloudMan. *Bioinformatics*, pp. 1685–6, 2013.

[34] F. Trias, V. de Castro, M. López-Sanz, and E. Marcos. RE-CMS: A reverse engineering toolkit for the migration to cms-based web applications. In *Proc. of ICSAC*, pp. 810–812, 2015.

[35] A. Zalewski and S. Kijas. Beyond ATAM: Early architecture evaluation method for large-scale distributed systems. *J. Syst. Softw.*, pp. 683 – 697, 2013.