# An Evaluation of Galaxy and Ruffus-Scripting Workflows System for DNA-seq Analysis

**Submitted by: AJAYI OLABODE OLUWASEUN**

A thesis submitted in partial fulfilment of the requirements for the degree of Magister Scientiae at the South African National Bioinformatics Institute in the Faculty of Natural Sciences, University of the Western Cape.

**Supervisor:** Prof Alan Christoffels

Date: 15th August 2018

# Declaration

I declare that *An Evaluation of Galaxy and Ruffus-Scripting Workflows System for DNA-seq Analysis* is my own work, and that it has not been submitted before for any degree or examination in any other university, and that all the sources I have used or quoted have been indicated and acknowledged as complete references.

Ajayi Olabode Oluwaseun August 2018

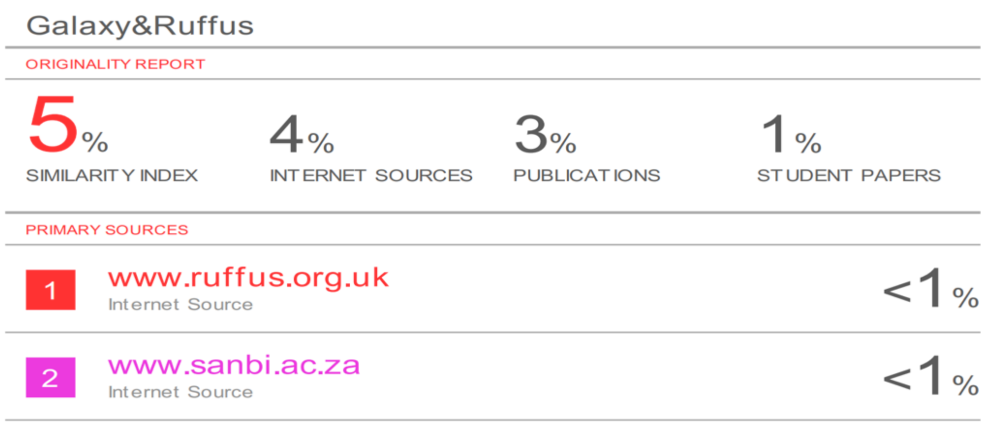
[illegible]

Signed: . . . . . . . . . . . . .. …

turnitin

Digital Receipt

This receipt acknowledges that Turnitin received your paper. Below you will find the receipt information regarding your submission.



Galaxy&Ruffus

ORIGINALITY REPORT

| 5% | 4% | 3% | 1% |
|---|---|---|---|
| SIMILARITY INDEX | INTERNET SOURCES | PUBLICATIONS | STUDENT PAPERS |

PRIMARY SOURCES

| | | |
|---|---|---|
| 1 | www.ruffus.org.uk<br>Internet Source | <1% |
| 2 | www.sanbi.ac.za<br>Internet Source | <1% |

# Abstract

**An Evaluation of Galaxy and Ruffus-Scripting Workflows Systems for DNA-seq Analysis**

*O.O AJAYI*

*MSc in Bioinformatics (Full Thesis), South Africa National Bioinformatics Institute, University of the Western Cape, Bellville, South Africa*

Functional genomics determines the biological functions of genes on a global scale by using large volumes of data obtained through techniques including next-generation sequencing (NGS). The application of NGS in biomedical research is gaining in momentum, and with its adoption becoming more widespread, there is an increasing need for access to customizable computational workflows that can simplify, and offer access to, computer intensive analyses of genomic data. In this study, the Galaxy and Ruffus frameworks were designed and implemented with a view to address the challenges faced in biomedical research. Galaxy, a graphical web-based framework, allows researchers to build a graphical NGS data analysis pipeline for accessible, reproducible, and collaborative data-sharing. Ruffus, a UNIX command-line framework used by bioinformaticians as Python library to write scripts in object-oriented style, allows for building a workflow in terms of task dependencies and execution logic. In this study, a dual data analysis technique was explored which focuses on a comparative evaluation of Galaxy and Ruffus frameworks that are used in composing analysis pipelines. To this end, we developed an analysis pipeline in Galaxy, and Ruffus, for the analysis of *Mycobacterium tuberculosis* sequence data. Furthermore, this study aimed to compare the Galaxy framework to Ruffus with preliminary analysis revealing that the analysis pipeline in Galaxy displayed a higher percentage of load and store instructions. In comparison, pipelines in Ruffus tended to be CPU bound and memory intensive. The CPU usage, memory utilization, and runtime execution are graphically represented in this study. Our evaluation suggests that workflow frameworks have distinctly different features from ease of use, flexibility, and portability, to architectural designs.

# Keywords

Data-intensive

Galaxy Framework

Functional Genomic

High Performance Computing

*Mycobacterium tuberculosis*

Performance Evaluation

Ruffus Framework

Runtime Execution

SNP (Single Nucleotide Polymorphism)

Workflow Systems

# Acknowledgement

I give thanks to God for making this project possible. This I believe is a reason I live and my purpose in life. With God on my side, this project turned to be an invaluable asset for rest of my career. Many a time, when I belittle my genius mind he restores hope back. He is my God of all sufficient who is near and cover-up my inadequacy. I thank my thesis supervisor Professor Alan Christoffels of South African National Bioinformatics Institute (SANBI), University of the Western Cape for giving me a platform to restore my past academic performance. The door to Prof. Alan office was always open whenever I had a question about my research or writing. He consistently allowed the thesis to be my own work but steered me in the right the direction whenever he thought I needed it. This thesis is written within the context of the Computational Bacterial Analytical Toolkit for tuberculosis project (COMBAT-TB), which is funded by the National Research Foundation of South Africa. Furthermore, I take this opportunity to express gratitude to all the department members for their help and support especially, Dr. Dominique for reviewing the entire project. I am gratefully indebted to her for the very valuable comments on the thesis. Thanks to Peter for sharing his thoughts and views during the project. His assistance was innumerable. His vast ideas sometime made me work hard whenever I thought about his computing knowledge. Also, thanks to Hocine, Sandiswa, Dr., Bunmi, Dada, and Natasha for believing in me. Their encouragement to believe in myself regardless of my inadequate reasoning sometime pushed me beyond the limit. Furthermore, I thank Oni's family, in fact my second family for their unceasing encouragement, support and attention. I am grateful for this unending love and support they rendered through this venture. Finally, I must express my very profound gratitude to my parents, brothers, and sisters for providing me with unfailing support and continuous encouragement throughout my years of study. This accomplishment would not have been possible without them believing in me.

# Dedication

*"Infuse your life with action. Don't wait for it to happen. Make it happen. Make your own future. Make your own hope. Make your own love. And whatever your beliefs honor your creator, not by passively waiting for grace to come down from upon high, but by doing what you can to make grace happen... yourself, right now, right down here on Earth".*

Bradley Whitford

This thesis work is dedicated to my lovely and beloved mother, who has wished to go to school in her life. I love you Mommy.

# Table of Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| API ………………. | Application Programming Interface |
| CLI ………………. | Command Line Interface |
| DNA-seq …..……. | Deoxyribonucleic Acid Sequencing |
| DRM ……………. | Distributed Resource Management |
| ENA ……………… | European Nucleotide Archive |
| GUI ………………. | Graphical User Interface |
| HPC ……............... | High Performance Computing |
| MTB ……………... | *Mycobacterium tuberculosis* |
| NGS ……………… | Next Generation Sequencing |
| RNA ……………… | Ribonucleic Acid |
| SGE ……………… | Sun Grid Engine |
| SNP ……………... | Single Nucleotide Polymorphism |
| VM ………………. | Virtual Machine |
| WMS ……............ | Workflow Management Systems |
| XML ……………... | Extensible Mark-Up Language |

# Chapter 1: Thesis Rationale

Functional genomics is a field of molecular biology that determines the biological functions of genes on a global scale, using large volumes of data obtained through techniques such as next-generation sequencing (NGS). NGS technology has been a key driver in accelerating knowledge gains in functional genomics, and molecular biology research. With data from the genomics field constantly increasing, the scientific community is finding the processing, analysis, and the discovery of research based on the datasets more challenging (Goble and Stevens, 2008).

The advent of new sequencing technology systems by companies such as Illumina, Oxford Nanopore, and PacBio (as illustrated in Figure 1), has resulted in an exponential growth of NGS data output, with the advantages of a reduction in the time and cost expenditure associated with sequencing projects (Metzker, 2010; Loman *et al.*, 2012). A recent advance in NGS technology has allowed scientists to re-sequence Deoxyribonucleic Acid (DNA) and Ribonucleic Acid (RNA) in a quick and affordable manner (Korpelainen *et al.*, 2014). Furthermore, the NGS process involves the input of biological samples from different organisms as well as the same organisms into NGS sequencer machine, which then produces a computer representation of genomic datasets as the output (Glenn, 2011). Due to the large volume of genomic datasets produced in NGS projects, an extraordinary demand has been placed on bioinformatics workflow systems (Stein, 2010). Consequently, there is an increased requirement for efficient data analysis pipelines for a multitude of applications in functional genomic research, before they can be routinely used by researchers.

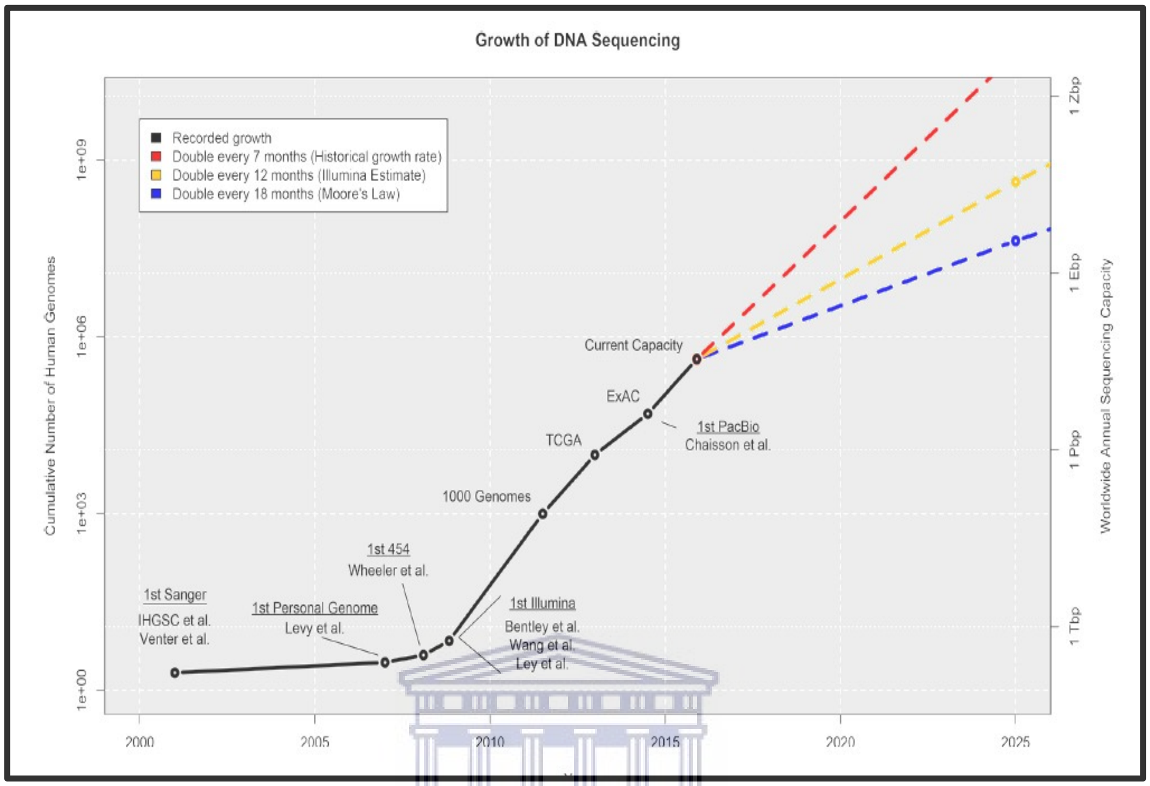


**Figure 1: Plot of the actual and predicted growth of DNA sequencing from 2001 to 2015.**

The plot illustrates the actual and predicted evolution of DNA sequencing together in the total figure of human genomes sequenced (left axis) as well as the worldwide annual sequencing capacity (right axis: terabase pairs (tbp), peta-base pairs (pbp), exa-base pair (ebp) (Stephens et al., 2015).

Considering that the logic to manually extract and transform this data requires considerable human effort, it has become a necessity to develop and utilize an automated, yet simple, workflow system that can serve biomedical researchers. Workflow steps that are particularly prone to errors and repetitions, and that need manual intervention from biomedical researchers, should be the specific targets for effective software solutions. One such solution which addresses the above-mentioned workflow problems is an analysis pipeline in the bioinformatics workflow framework. More research is needed to inform the development of the pipeline analysis techniques that include data quality checking, analysis, processing and interpretation of genomic data (Zhang *et al.*, 2010).

Curcin and Ghanem (2008), indicated that workflow systems have become a requirement for functional genomics. However, the complex nature of functional genomics datasets, and the bioinformatics workflows systems used to analyze and process these large volumes of genomics datasets from many resources, makes efficient processing difficult with standard environments (Schulz *et al.*, 2016). In general, -omics' research, including the field of proteomics, relies heavily on workflows containing relevant pipelines for data analysis (Fisch *et al.*, 2015). The notion of a genomic data analysis workflow systems becomes increasingly relevant when biomedical researchers start to use more than one bioinformatics tool (Torri *et al.*, 2012). Moreover, biomedical researchers often need to connect two or more bioinformatics tools to 1) assess the quality or feature of the genomic data sets, 2) convert the data to other formats, 3) visualize the data, 4) compare results and 5) perform other operations in a logical manner. For these reasons, a data analysis pipeline consists of different programs which are integrated together to perform tasks of varying complexity (Sanner, 1999). Bartlett and Toms (2005), developed a protocol that demonstrated the unique process which may be employed by an expert researcher using bioinformatics resources to address a specific research problem. For example, logical thinking and attention to detail may be utilized by a researcher to define acceptable input file-types, parameter values and resource management, as well as exception behavior, in an effort to answer specific bioinformatics research questions (Neron *et al.*, 2009).

Automation of frequently executed tasks can be incorporated into complex workflows, thereby decreasing time and effort spent by biomedical researchers in command-line sessions, non-reusable script writing, and general time-consuming software (Guimera, 2012). Workflow management systems manage workflow processes through software execution, the order of which, is driven by the software application which is installed on a local computer system, or clusters (Brown *et al.*, 2015). Efficient and comprehensive data analysis pipelines require these workflow management capabilities (Liu *et al.*, 2014). A workflow procedure consists of multiple steps (any repeatable series of steps that include creating, managing and providing output information experimental investigation) that are used to execute and automate a workflow process, thereby instituting a set of procedural rules to allow for the flow of tasks and information from one action to the next (Romano, 2008). In one step, the workflow outputs serve as input to the next step, according to a predefined network or graph topology that synchronizes the movement of data (i.e., extracting, transforming and loading as shown in Figure 2).

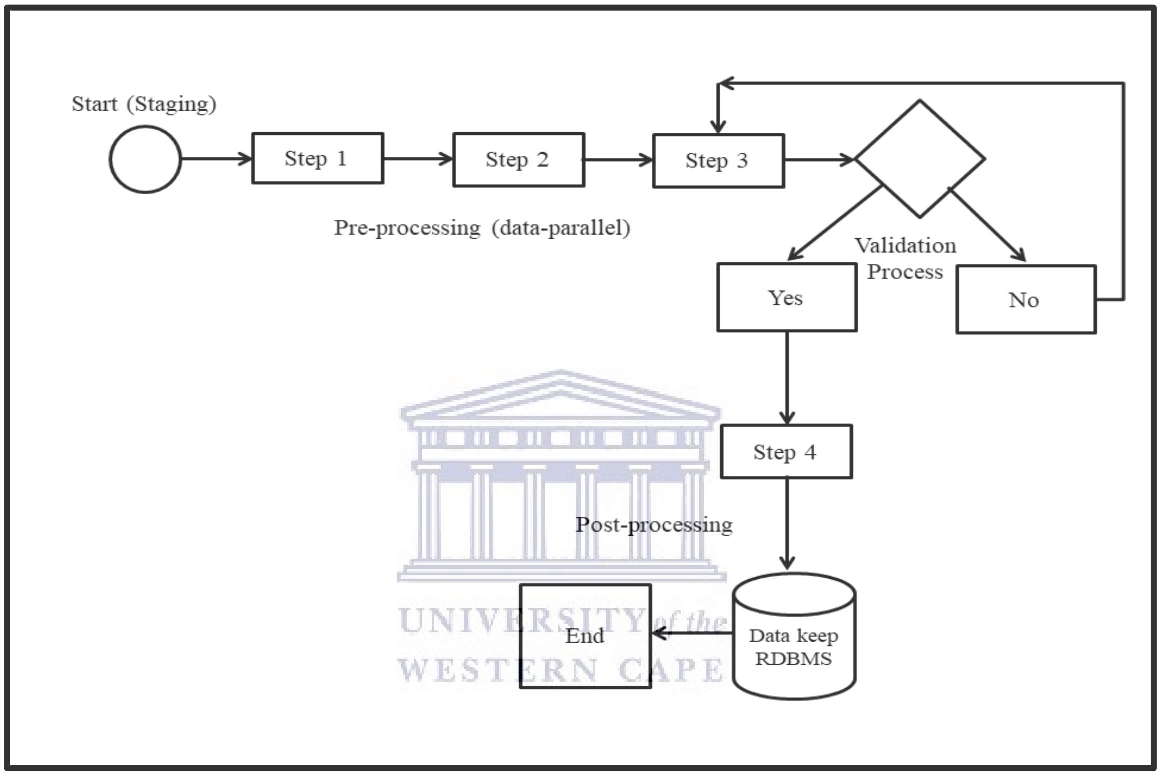


**Figure 2: Example of a simple workflow diagram.**

The diagram represents a flowchart that shows workflow starting point (i.e., staging), connecting step 1 to step 2. The output from one step is for further dissemination to the other steps. The database (such as MySQL data keep), is a relational database management system (RDBMS) for retrieving and storing biological data. The database can form part of the major requirement that support a workflow system development (Bhagwanani, 2005).

Workflow processes coordinate multiple workflow tasks. Workflow processes are further defined as sequences of activities that are necessary to complete tasks. A task can be defined as a process that cannot be split up any further (Van Der Aalst and Van Hee, 2004).

Alternatively, a task, also known as an activity, is an automated activity performed at the user-level of any workflow system.

A study by Spjuth et al. (2015) explains the idea of using workflow systems to assist researchers in their studies. They describe a workflow system as a multi-step procedure or task that runs on a distributed computing platform. In this way, a task represents the execution of a workflow process, such as integrating bioinformatics tools, submitting a query to a database server, submitting a job to on-premises high performance computing (HPC) systems, as well as cloud-based computing or invoking a service over the web-browser to use a remote resource (Goble and De Roure, 2009). HPC (or cluster) systems, is a cluster of parallel computers that are connected together to support data-intensive scientific applications on a large global scale (Spjuth et al., 2015). Workflow frameworks are important enablers for such capabilities (Kang et al., 1999).

Bioinformatics workflow systems, and its logic, are driven by software applications developed and written in different computer programming languages. Workflow activities are automated by compositions using the available open source packages or proprietary software (Deelman *et al.*, 2009). Bioinformatics workflow protocols are therefore ideal vehicles for biological data extraction and are becoming a standard for use in supporting functional genomics research worldwide, by managing genomics data pre-processing, and post-processing. Other benefits of bioinformatics workflow systems include;

a) Data automation, data format conversion, and pipeline analysis integration.
b) Provisioning of a graphical user interface (GUI) that manages experimental steps that enable biomedical researchers to build custom pipeline analysis for genomics data analysis or the use of predefined use cases.
c) Provisioning of a command-line interface to support scripting programs (Hinchcliffe *et al.*, 2014).
d) Provisioning of data management that include analysis tracking and pipeline staging (Deelman, 2010).
e) Offering access to tools that manage and execute pipelines.

f) Offering an efficient means of conducting sequencing analysis across diverse ‘omics datasets and applications (Goble and De Roure, 2009).
g) Allowing for both experienced and novice researchers to build analysis pipelines, without knowledge of complex programming language.
h) Serving as a platform for managing the growing pool of genomics data and
i) Allowing for independent computational analysis.

The overall purpose of this study was to evaluate the Galaxy and Ruffus workflow frameworks to assist biomedical researchers in processing and analyzing the *Mycobacterium tuberculosis* (MTB) genomic datasets to obtain high quality variant calls. The approach used in this study was to adapt an already existing open source genomics workflow framework with the view that utilizing different middleware software components on the cluster, and extending the analysis pipeline for re-use in a clear manner that simplify the automation of bioinformatics analysis, would: 1) solve challenges of large-scale data analysis 2) develop best practice workflows, and 3) fill the current gap amid computing infrastructure and bioinformatics applications (as discussed in Chapter 3). In addition, the assembled analysis pipeline would be a completely open source project and the workflow framework was benchmarked against each other based on system complexity and support for data storage management, provenance, and data retention policy. Here, the benchmarking activities for the bioinformatics workflows included the exploration of different design choices and metric gathering for systems performance and scalability, data storage, analysis pipeline process time and transfer speed. The abovementioned metrics are an essential consideration for both current and future workflow frameworks requirements. Moreover, predicting future bioinformatics workflows updates and how the metrics could affect the underlying infrastructure technology is of great importance to anticipate proper workflow system design and its limitations (Van Der Aalst and Van Hee, 2004; Furtaw, 2016). A further aspect of this study was to establish the genomic data source collaboration plan and source code control versioning system where data analysis pipeline development can continue-on a quick-changing running system. While system requirements can change at any time they require simplification, and a team of biomedical researchers require quick analysis results, thereby imposing added pressure on analysis pipeline design decisions. Therefore, additional care must be taken when developing/or building, testing and deploying the data analysis pipeline within the bioinformatics workflow frameworks. As such, this approach, accompanied with best practices was explored in-depth in this thesis.

Finally, in this study, proof of concept was demonstrated using experimental data sets. Different in-house research tools (i.e., open source bioinformatics tools in the computing environment) that solve particular needs were integrated into the bioinformatics workflow frameworks presented herein, using the methods developed in this study.

# Chapter 2: Literature Review

This chapter presents the historical background of bioinformatics workflow systems which are currently available for biomedical researchers, as well as the challenges faced by researchers when deciding on which workflow frameworks to utilize. This chapter describes, with examples, several workflow systems, particularly graphical user interface or command-line interfaces as well as the underlying infrastructure technology, that currently exist for analyzing genomics data. The chapter also tabulates workflow features and compares each feature to one another and concludes by motivating the rationale for choosing and evaluating the Galaxy and Ruffus workflow systems for processing of genomics data.

## 2.1: Historical Background of Bioinformatics Workflow Systems

The historical development and knowledge behind the study of scientific workflow management systems has expanded to the fields of bioinformatics and biomedical science. Clinicians and research scientists have seen the development of bioinformatics workflows as an essential part of research to compliment the traditional patterns of theory and wet-bench experiments. Next generation sequencing (NGS), also known as high throughput sequencing technologies, have led to sequencing at unprecedented speed, and in combination with low sequencing costs per base pair, has produced a huge amount of genomic data, that overwhelms the current workflow systems and resources (Altintas *et al.*, 2012; Kodama *et al.*, 2012).

This growing volume of genomics data being generated from NGS technology needs to be analyzed using bioinformatics workflows (Goesmann *et al.*, 2003; Wilke *et al.*, 2003). Genomics data analysis involves the processing of data files through a series of computational steps and transformations, referred to as an analysis pipeline. These steps can usually be achieved by installing third party GUI- or CLI- based software that can execute the data analysis pipelines (Hinchcliffe *et al.*, 2014). As suggested by Li and

Chen, the bioinformatics research field requires robust, accurate and precise workflow systems (Li and Chen, 2014). In addition, the authors took into consideration factors relating to big data and suggested that data volume, data processing velocity, data source variability, as well as data quality veracity or authenticity requires special technology and workflow management systems (Li and Chen, 2014).

The ability to actively incorporate genomic datasets into modern studies strongly depends on effective bioinformatics workflows with capabilities to handle downstream analyses and give interpretable results. Therefore, the concept of workflow systems or frameworks, requires significant informatics knowledge and expertise to design a pipeline for detailed analysis and interpretation of sequencing data that can be applied in clinical settings (Kanwal *et al.*, 2017). Moreover, a workflow framework is regarded as a platform for managing workflow activities, as well as coordinating computing resources and behaviors (Zhao *et al.*, 2005). A workflow framework provides an enabling meta-environment that has gained increased interest in the fields of genomic science and technology. It has further been indicated that workflow frameworks have assisted in rapid development of distributed and parallel data analysis pipelines (Zhao *et al.*, 2005; Deelman *et al.*, 2009; Spjuth *et al.*, 2015).

A typical biological sequencing data analysis pipeline in bioinformatics workflow systems has several phases that include experimental design and sample collection, sequencing and data processing for subsequent downstream analysis (Kanwal *et al.*, 2017). For instance, an analysis pipeline in bioinformatics workflows consists of a series of connected steps that transform raw input (e.g. a FASTQ file from an NGS sequencer) into meaningful or interpretable outputs (e.g. variant call-sets).

To understand a complex system (such as Kepler), it is necessary to have a birds-eye view in order to determine how the different pieces fit together (Altintas *et al.*, 2004). Bioinformatics workflow systems require software applications to prosper, and to build bioinformatics pipelines, encapsulation is needed, which can be used as basic building blocks for another bioinformatics project. This generates knowledge in the effective use of workflows.

## 2.2: Systems Infrastructure for Bioinformatics Workflow Systems

Due to the high demand of bioinformatics workflows for analyzing the vast amounts of data generated by NGS, the systems infrastructure for workflow integration has become a vital requirement in biomedical research. This demand is further confounded by the significant cost reduction in sequencing which has allowed instrument manufacturers to decrease the cost per genome produced by NGS machines, thereby increasing the feasibility of including this technology in biomedical research (Anderson and Schrijver, 2010). While the complexity of workflows varies significantly in different applications, a data analysis pipeline in bioinformatics workflow frameworks may typically require days of computing time and a great amount of computing power (Brown *et al.*, 2015). Brown *et al.* (2015), explains that setting up a bioinformatics workflow system is not a straightforward process, and often require extensive technical skills and coding experience to setup. Furthermore, finding an accessible method to keep and process genomics data in the most efficient, metadata-rich, secure and transparent way is not a simple task (Kanwal *et al.*, 2017). Likewise, the challenges of integrating bioinformatics workflow management systems with personal computer functionality, such as system resource and data storage, are increasing (Figure 3). Therefore, high performance computing (HPC) and cloud computing facilities have been introduced as solutions to challenges faced by biomedical researchers and are shaping new developments in the bioinformatics field (Jamalian and Rajaei, 2015; Nishanth and Kihoon, 2015).

### 2.2.1: High-Performance Computing Environments

HPC is increasingly becoming an important tool in biomedical research, and currently enables researchers and computer scientists to solve complex problems requiring several computing capabilities, to increase the pace of research discovery (Alyssa, 2016; Leading, 2016; Liu, 2016). HPC subsequently reduces the time and cost that scientists spend on analyzing genomics data. HPC coupled with bioinformatics workflow systems are essential for analyzing genomics data to obtain meaningful results, while additionally, maintaining the processing time and  speed at which genomic data outputs are being generated (Nishanth and Kihoon, 2015). In a scientific research environment for example, HPC resources generally consist of compute nodes with a greater level of computing performance when compared to general purpose computers. HPC with hundreds of thousands of 'off-the-shelf' processors run a Linux-based operating system with a batch queueing system (i.e., batch-queuing system is a scheduling system that helps to plan the execution of batch jobs) for scheduling jobs (Jamalian and Rajaei, 2015). Studies by Di Tommaso *et al.* (2017) have shown that the most efficient and effective bioinformatics frameworks are workflow systems, supported by these batch-queuing systems (e.g., PBS/Torque, SLURM, Sun Grid Engine). The components used to support a HPC environment, such as computer memory, cores, compute node and storage, as well as fabric and software (Figure 3), have been changing at unprecedented rates over the past two decades (Huang *et al.*, 2006). This has resulted in systems bottlenecks that are becoming increasingly imbalanced (Alyssa, 2016; Leading, 2016). Some bioinformatics analysis jobs, with highly specific resource needs, have forced the biomedical research community to implement discrete clusters which are dedicated to these jobs (Nyrönen *et al.*, 2012; Bianchi *et al.*, 2016). This has however contributed significantly to the overall development of workflow systems.

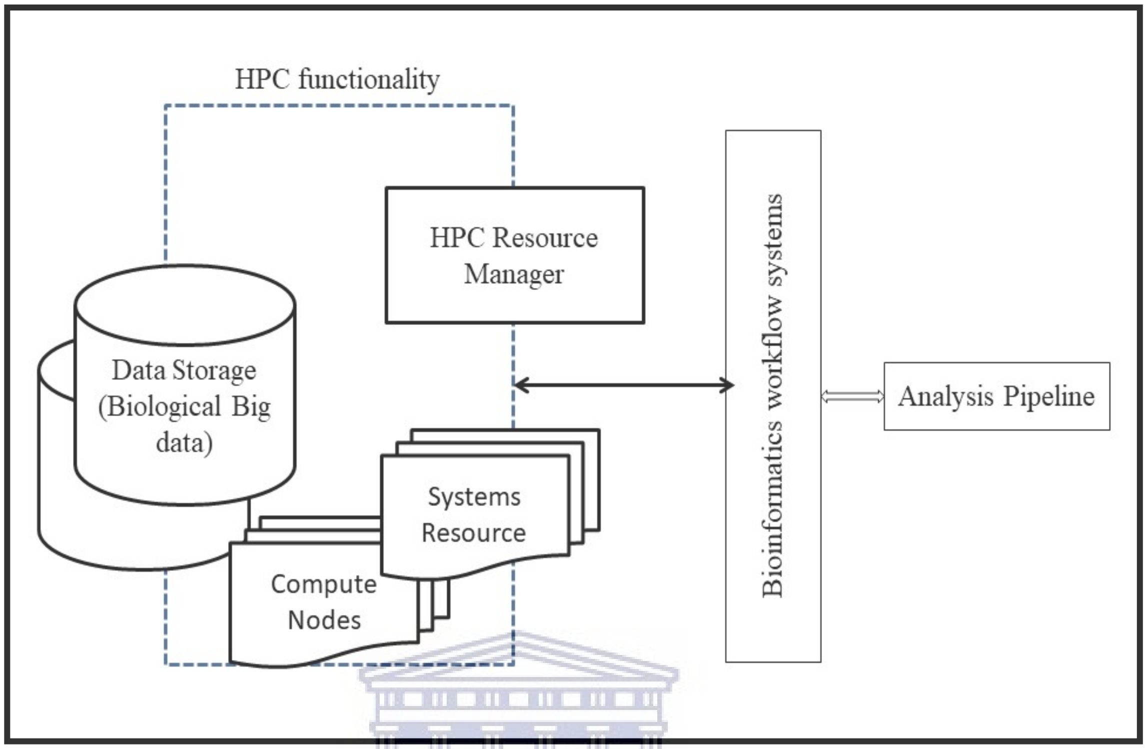


**Figure 3: The fundamentals of HPC system infrastructures.**

The diagram describes the several components of HPC functionality on a typical computing lab environment. The HPC system is made up of many processors and cores, high-speed networking, and large compute nodes for data stores (Alyssa, 2016). At the central of HPC is the manageable resource manager (e.g., hardware and systems software), which allow system administrator to dedicate energy to managing the HPC environment. The HPC allows software stack that supports the bioinformatics workflow systems.

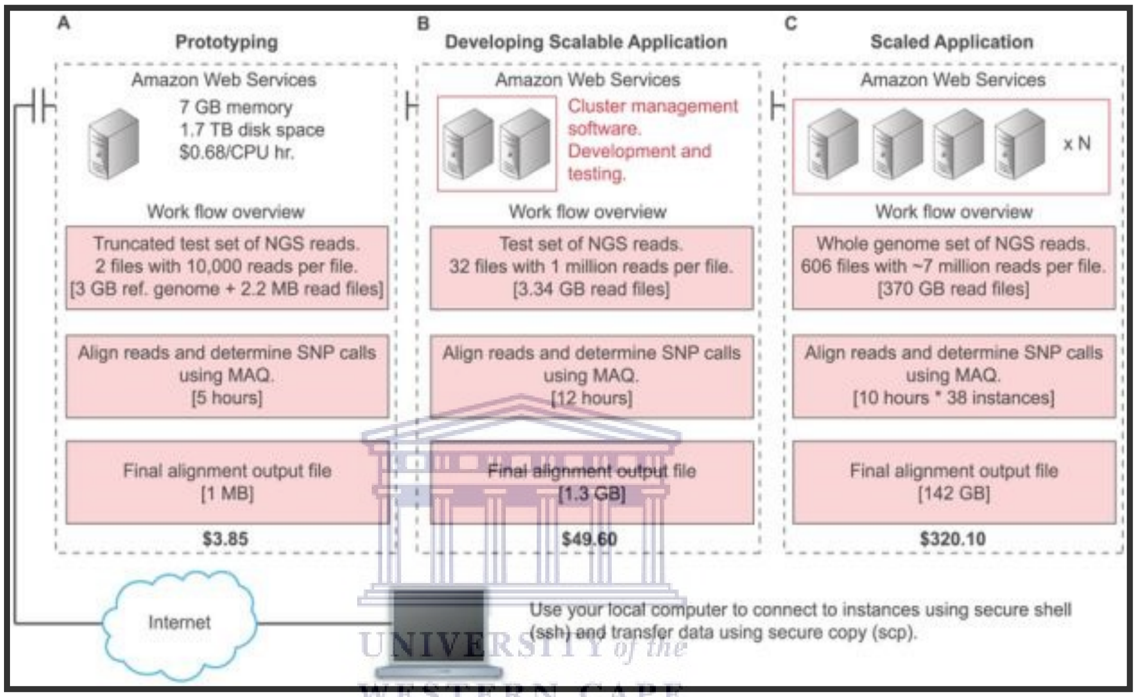
## 2.2.2: HPC in a Cloud Environment

Cloud computing has recently emerged as a supplemental technology, which offers virtualize environments (such as virtual machine and Dockers), and the capability to run custom virtual machine images (VMI) or containers (Afgan *et al.*, 2012; Spjuth *et al.*, 2015). Despite the advent of cloud computing, setting up virtual cloud server clusters for biomedical research requires knowledge about the pros and cons associated with different bioinformatics tools (O'driscoll *et al.*, 2013). Cloud compute storage solutions for biological data have been developed to tackle the challenges when implementing platforms for data-intensive NGS analyses (Doctorow, 2008; Li and Chen, 2014; Luna *et al.*, 2014; Stephens *et al.*, 2015), (Figure 3).

Since the volume of genomic data being generated is increasing exponentially, many biomedical research labs or institutes are considering cloud computing as a cost-effective alternative for the storage and processing of large volumes of biological datasets (Liu *et al.*, 2014). Fusaro *et al.* (2011) summarized that a cloud computing platform could be implemented and utilized as a platform for storing biological data, thereby facilitating analysis of petabyte sized datasets, in a more effective way (Figure 5). Cloud computing also enables the application of new data processing models, such as the MapReduce framework (Dean and Ghemawat, 2008; Afgan *et al.*, 2010) and its variants, which have been successfully implemented on processing large-scale clinical genomic data using virtual cloud clusters (Zaharia *et al.*, 2010; Zou *et al.*, 2013).

A study by Armbrust *et al.* (2010) found that cloud computing systems provide users with full control over virtual compute resources, by using virtualization technologies. Cloud computing includes hardware, software and systems infrastructure, and these are provided as services over the internet. In addition, Chine (2010) identified three major classes of cloud computing providers, which include Infrastructure-as-a-service (IaaS), Platform-as-a-service (PaaS) and Software-as-a service (SaaS) (Bhardwaj *et al.*, 2010). IaaS offers only virtual machines and compute storage systems for any purpose, whereas PaaS offers platforms for developing software applications. SaaS on the other hand, offers available software that can be used as is, or customized for an application (Stein, 2010).

As shown in Figure 4, analyzing and processing biological data using the bioinformatics resource in the cloud HPC cluster environment can be very challenging. Without correct automation, the setup and fine-tuning of virtual cloud clusters may become a difficult task, as there is a requirement for systems administrators to have considerable knowledge with regards to installation and configuration of different software tools. Organizing the different bioinformatics workflow frameworks that are developed and integrated in a cloud based system, is at the discretion of the system administrator or user if there are no best practice or operating procedures to guide it (Schindelin *et al.*, 2012). Deployment and provisioning scripts are therefore essential for the cloud computing model to be successful.



A
Prototyping
Amazon Web Services
7 GB memory
1.7 TB disk space
$0.68/CPU hr.
Work flow overview
Truncated test set of NGS reads. 2 files with 10,000 reads per file. [3 GB ref. genome + 2.2 MB read files]
Align reads and determine SNP calls using MAQ. [5 hours]
Final alignment output file [1 MB]
$3.85
B
Developing Scalable Application
Amazon Web Services
Cluster management software. Development and testing.
Work flow overview
Test set of NGS reads. 32 files with 1 million reads per file. [3.34 GB read files]
Align reads and determine SNP calls using MAQ. [12 hours]
Final alignment output file [1.3 GB]
$49.60
C
Scaled Application
Amazon Web Services
x N
Work flow overview
Whole genome set of NGS reads. 606 files with ~7 million reads per file. [370 GB read files]
Align reads and determine SNP calls using MAQ. [10 hours * 38 instances]
Final alignment output file [142 GB]
$320.10
Internet
Use your local computer to connect to instances using secure shell (ssh) and transfer data using secure copy (scp).

**<u>Figure 4: A cloud-based framework for creating a scalable NGS workflow system.</u>**

The diagram illustrated the step-wise cloud workflow framework for establishing a scalable NGS workflow system. A user, using a local computer can ssh into an instance of a virtual machine running in AWS cloud. Installing software programs, developing a scalable bioinformatics application tools together with utilities cloud cluster management software and testing the instance pipeline all depends on cost and consumption usage. The costs are representative of actual development time, data transfer into and out of the cloud, and the compute time (Fusaro et al., 2011).

Currently, there are existing 'off-the-shelf' bioinformatics workflow management system installations in the form of cloud virtual machine images that can be used to mitigate the otherwise steep learning curve experienced by biomedical researchers (Afgan *et al.*, 2012). This cloud virtual machine image is, in essence, a virtual representation of a physical hard disk drive, containing preinstalled data and bioinformatics software tools (Schindelin *et al.*, 2012). An added advantage of the virtual image is that depending on the experimental load requirement, multiple instances of this cloud machine can be made.

Unfortunately, several known concerns impacting the standard utilization of cloud HPC clusters exist and are still driving biomedical research labs or institutions away from utilizing the cloud model. For instance, HPC compute nodes are virtualized, and this has raised concerns with regards to virtualization overhead as well as virtual machine co-location. Moreover, a pay service to implement a cloud model for creating a scalable workflow application to fit small and large project is essential for the sustainability of HPC clouds (Curcin and Ghanem, 2008; Netto *et al.*, 2018). There are some institutions that cannot afford the cost of the cloud services, and therefore the burden of cost associated with investing in the required expertise can be inhibitory (Fusaro *et al.*, 2011). One of the most persistent problems facing biomedical researchers is not having access to working system infrastructure that facilitates progressive, sustainable and qualitative research outputs (Deelman, 2010; Truong and Dustdar, 2011; Emeakaroha *et al.*, 2013).

Another recurrent issue which has been raised by many experts relates to the latency and bandwidth of the network used by cloud infrastructure. For example, Amazon Web Service's Elastic Compute Cloud (EC2) functions differently compare to a typical, dedicated HPC cluster in national laboratories (Garfinkel, 2007; Jackson *et al.*, 2010; Marathe *et al.*, 2013). These differences can lead to new performance issues that necessitate different bioinformatics tools to gain prominence into the workflow systems and its underlying cloud-based infrastructure. In this way, bandwidth impacts the time it takes for transmission of big data to and from the cloud and has been a major setback for many research labs (Liu *et al.*, 2014; Luna *et al.*, 2014; Netto *et al.*, 2018). At the time of writing this thesis, existing resources were traditionally HPC clusters, and as such, the focus and limitations of this study fall within this area.

## 2.3: Bioinformatics Workflow Frameworks Reality

There are various bioinformatics workflow frameworks that aim to address issues that exist in the building of data analysis pipelines for the extraction of valuable genetic insights from large amounts of genomics data. A bioinformatics workflow framework enables integration of several bioinformatics tools, and the development of data analysis pipelines for annotating and exploring NGS datasets. This process can range from creation and composing data analysis pipelines, to evaluating usability in biomedical research.

Numerous bioinformatics workflow frameworks for composing pipelines have already been developed (e.g., Taverna is used for building bioinformatics data analysis pipelines) (Oinn *et al.*, 2004). However, due to the lack of continuous software development and community support, not all bioinformatics frameworks have all the features required to develop high-throughput data analysis pipelines (Taura *et al.*, 2013). Therefore, lessening barriers to entry on development and deployment for developers and user communities will significantly aid in building overall reusable and interoperable pipeline analyses (Stein, 1996).

The bioinformatics workflow frameworks employed by biomedical research labs for composing an analysis pipeline are essential when selecting which frameworks to use. According to Plale et al. (2011), a bioinformatics workflow framework is an integral platform that encourages pipeline configurations. There are frameworks that encourage biomedical research labs or institutes to share analysis pipelines and collaborate with other researchers around the world. For instance, the Taverna (Oinn et al., 2004) and Kepler (Altintas et al., 2004) interfaces have common characteristics that allow easy sharing of analysis pipelines, protocols and standard operating procedures (SOPs). Unfortunately, analysis pipelines reproducibility in the biomedical field is a goal hard to accomplish due to the complexity of workflow systems, usually involving series of analysis steps and protocols (Di Tommaso *et al.*, 2017)**.** For instance, Taverna and SnakeMake frameworks follows different language patterns and as such biomedical researchers, depending on experience, may wish to replicate or reproduce the CLI workflow framework over the GUI workflow framework (Oinn *et al.*, 2004; Koster and Rahmann, 2012; Spjuth *et al.*, 2015).

Another important factor to take into consideration when choosing a framework is the underlying technologies and process specification model languages, such as Yet Another Workflow Language (YAWL), which handles complex data transformations, and complete integration with HPC resources and external web services (Van Der Aalst and Ter Hofstede, 2005). Data model handling, such as the extensible mark-up language (XML) schema, allows for  native data handling that adheres to specific standards and conventions (Aldred, 2011). A typical example is the Arvados framework (Arvados, 2016) that starts with raw genomics data processing files such as FASTQ files, and after a number of steps, actions and commands, ultimately results in variants being called (Depristo *et al.*, 2011; Van Der Auwera *et al.*, 2013; Pabinger *et al.*, 2014).

A study by Mckenna *et al.* (2010), demonstrated an efficient, features-rich, and robust analysis pipelines for processing massive data sets generated by NGS machines. Pipelines for genome datasets follow a specific order of biological procedures from beginning to end. Most of the activities in the pipeline are performed by humans interacting with computer systems (Gorelick and Ozsvald, 2014). Many research labs, or institutes, are restricted by the difficulty of accessing and manipulating the data produced by NGS machines, and may not be aware of the possibilities and simplicity with which they can answer technical questions (Ison *et al.*, 2015). Therefore, workflow frameworks that make routine tasks and procedures, support pipeline reproducibility and offer measures for fault-tolerance are possible solutions which can be utilized in research settings (Spjuth *et al.*, 2015). Pipelines in the bioinformatics workflow frameworks combine knowledge from different areas of genomic fields and it is important for researchers in the biomedical field to understand the concepts related to composing pipelines. There are bioinformatics workflow frameworks that require in-depth knowledge of detailed documentation related to workflow design and modelling (Tolvanen and Kelly, 2008; Liu *et al.*, 2014). Moreover, a bioinformatics workflow framework is consider to be good framework if certain design criteria such as extensibility, restorability, ease of use and deployment and pipeline reproducibility are met (Lamprecht, 2013). Novice biomedical researchers lack the capabilities to identify efficient, yet simple workflows, and may not have the expertise to recognize the workflow systems design criteria (Williams *et al.*, 2014).

## 2.4: Command Line Interface Blueprint

Command line interface bioinformatics frameworks consist of collections of scripts written specifically to run on a modern GNU/Linux distribution terminal and that allows researchers to run commands in a shell terminal, or console window, that ultimately work together within an operating system. Researchers reply to a shell command prompt by typing a command on an identified line and accept a reply from the system, or series of shell commands for individual tasks that they want to implement. In this way, command line enables automation and execution of scripts through the terminal (Stevens and Rago, 2013; Oracle, 2017). Computer scripting languages such as Python (Foundation, 2016) and Perl (Perl, 2016) enable developers to write custom scripts and develop software applications for a special run-time. Scripts are sequences of commands written to accomplish a task and assist in executing already developed software tools. In the functional genomics field, computer scripting languages enables packaging of bioinformatics tools that automate tasks, or execute tasks one-by-one during workflow processes and allows integration of bioinformatics tools within the pipeline frameworks (Stein, 1996; Sanner, 1999). For instance, workflow frameworks, such as Bpipe (Sadedin et al., 2012), SnakeMake (Koster and Rahmann, 2012), GXP Make (Taura *et al.*, 2013) (Taura et al., 2013b), Omics Pipe (Fisch et al., 2015)) and Nextflow (Di Tommaso et al., 2017) are CLI programs written specifically for the UNIX run-time environment.

## 2.5: Graphical User Interface Blueprint

Graphical user interface is a type of user interface that enables users to interact with a computer by utilizing graphics, in combination with hardware (a keyboard and a mouse), to provide an easy-to-use interface to a program (Michael and William, 2014). A GUI provides wizards, windows, buttons, iconic images, pull-down menus, scrollbars, other icons, and in general, allows users to interact with the computer operating system or application (Lefkowitz, 2000; Oracle, 2017). GUI-based bioinformatics workflow, using a drag-and-drop graphical interface allows biomedical researchers to design data analysis pipeline by selecting and connecting integrated bioinformatics tools. A number of GUI-bioinformatics workflow frameworks exist, such as Arvados (Arvados, 2016) and Mobyle (Neron *et al.*, 2009), which have been developed mainly for application in the life sciences field. Arvados is a GUI-bioinformatics workflow that makes it easier for biomedical researchers to build analysis pipelines, allows bioinformatics software developers to create genomic web applications and system administrators to manage large-scale compute and storage resources (Arvados, 2016; Calabrese, 2018). Taverna (Abouelhoda *et al.*, 2012), a workflow management system, offers services that allow access to bioinformatics tools and/or permits the building complex analysis pipelines which are distributed across web-services, or local computing infrastructure. Other examples of GUI workflows include Kepler (Altintas *et al.*, 2004) and Chipster, which are used for composing and analyzing NGS generated datasets. Chipster, for example, is utilized in studies where RNA-seq  data is analyzed in order to determine differential expression of genes (Wang *et al.*, 2011). A GUI workflow platform enables researchers to share, publish, find and download workflows, with the goal of making the re-use of existing workflows as easy as possible (Lamprecht, 2013).

## 2.6: Comparison of Bioinformatics Frameworks Features

In the bioinformatics domain, workflow frameworks already exist and can be used to explore and analyze genomic datasets. Workflow frameworks such as Nextflow workbench enable biomedical researchers to build pipeline analysis to control the data analysis activities for large genomic projects (Kurs *et al.*, 2016). Even with the rapid change in workflow frameworks, which has been maddened by new technological developments, existing frameworks have been used successfully in a number of studies (Leipzig, 2016). On the other hand, some workflow frameworks have failed due to missing features and consequently, the biomedical research community has had trouble in deciding which framework to employ. In Table 1 below, a summary of the ten-different bioinformatics workflow framework features is demonstrated.

**Table 1: Bioinformatics workflow frameworks feature comparisons**

| **Tool names** | **Workflow syntax** | **Online analysis integration** | **Interface interaction** | **Web services support** | **Built in cloud support** | **Built in distributed cluster support** |
|---|---|---|---|---|---|---|
| Arvados | Explicit | Yes | GUI | Yes | Yes | Yes |
| Chipster | Explicit | Yes | GUI | Yes | Yes | No |
| Galaxy | Explicit | Yes | GUI | Yes | Yes | Yes |
| PegaSys | Explicit | Yes | GUI | Yes | Yes | Yes |
| Taverna | Explicit | Yes | GUI | Yes | Yes | Yes |
| Bpipe | Explicit | No | CLI | No | No | No |
| GXP Make | Implicit | No | CLI | No | Yes | No |
| Omics Pipe | Implicit | No | CLI | No | No | No |
| Ruffus | Explicit | No | CLI | No | No | No |
| SnakeMake | Implicit | No | CLI | No | Yes | No |

Arvados, Chipster, Galaxy, PegaSys and Taverna, workflow frameworks enable researchers with a limited background in computing, as well as limited technical resources and support, to still perform tasks effectively. These workflow frameworks together with resources that are public, and are used to construct highly complex biological sequence analysis pipelines for investigating the genomics data, all from a normal UNIX PC, or  with support from built-in distribute cluster (or HPC) (Oinn *et al.*, 2004; Nishanth and Kihoon, 2015).

Furthermore, Bpipe, GXP Make, Omics Pipe, Ruffus and SnakeMake provide platform for running bioinformatics jobs. What differentiates each framework is the fact they are written in different programming language and have different design philosophy and limitations (Kircher and Kelso, 2010). All of the abovementioned workflow frameworks support job parallelism, but lack the built-in support for cloud and distributed compute clusters (Di Tommaso *et al.*, 2017). GXP Make and SnakeMake extend their platforms from single node systems to cluster and cloud. An observed disadvantage of SnakeMake, however, is that processing of a job associated metadata becomes slow when more than a 1000 job have been submitted to the cluster. Bpipe, SnakeMake, GXP Make, and OmicPipe are not ideal for performance evaluation. Ruffus, however, is able to execute task on multiple nodes, with a common task scheduler keeping track of dependencies and support for automatic reporting of parameters used, execution runtimes and tool and data versions (Biostars, 2010; Koster and Rahmann, 2012; Taura *et al.*, 2013; Ruffus, 2016). None of the CLI tools mentioned in the table supports online analysis integration (Table 1), whereas GUI workflow frameworks, Arvados, Taverna, PegaSys and Chipster have established integration of web services in bioinformatics (Spjuth et al., 2015). However, the Galaxy project maintains a larger research community and offers the most popular web browser-based platform.

Many research labs or institutions have scripting language experience and use custom scripts to assist in job parallelization (i.e., linking compute nodes) as well as integration with HPC resource managers such as PBS, SLURM etc. possibly via DRMAAv1 or 2 (Neron *et al.*, 2009; Biostars, 2015; Jamalian and Rajaei, 2015; Netto *et al.*, 2018). The study by Spjuth *et al.* (2015) suggested that working with custom scripts should be fast and easy to learn as shell scripts are considered to be very simple and flexible (Vince, 2015). Experienced biomedical researchers working with workflows in bioinformatics may prefer writing their own custom scripts when constructing data analysis pipelines and in this way may find working with custom script much easier, and quicker to deploy on a local or HPC cluster environment, despite a possible non-optimal process of workflow automation (Nishanth and Kihoon, 2015). Desirable advanced features, such as workflow replicability and reproducibility of analyses couple with HPC cluster resource environment and integration may require development from scratch using established framework (Biostars, 2015; Santana-Perez and Pérez-Hernández, 2015) (Table 2).

**Table 2: Bioinformatics workflow framework design philosophy**

| **Tool Names** | **Ease of Use** | **Workflows Track and Commands** | **Reliability** | **Ease of Development** | **Workflow Complexity and Robustness** | **Workflow Reproducible** |
|---|---|---|---|---|---|---|
| Arvados | No | Yes | Yes | No | Yes | Yes |
| Chipster | Yes | Yes | Yes | No | No | Yes |
| Galaxy | Yes | Yes | Yes | No | Yes | Yes |
| PegaSys | Yes | Yes | Yes | No | Yes | Yes |
| Taverna | Yes | Yes | Yes | No | Yes | Yes |
| Bpipe | Yes | Yes | Yes | Yes | Yes | No |
| GXP Make | Yes | No | No | Yes | No | No |
| Omics Pipe | No | Yes | Yes | Yes | Yes | No |
| Ruffus | Yes | Yes | Yes | Yes | Yes | No |
| SnakeMake | Yes | Yes | Yes | Yes | Yes | No |

## 2.7: Analysis Pipeline Options

The wide variety of workflow frameworks which are currently available may inundate non-experienced biomedical researchers, ultimately leading to difficulties in selecting suitable workflow frameworks to analyze genomic datasets (Bianchi *et al.*, 2016). A variety of analysis pipelines, such as RNA-seq  (for evaluation of gene expression studies), Chip-seq, (for evaluation of the binding of regulatory elements to genomic locations), and DNA-seq or Exome or Whole-Exome (to evaluate encoding of structural or genetic variants such as short Indels, large-scale genomic rearrangements, single nucleotide polymorphisms (SNPs)), requires an efficient workflow framework combined with the HPC systems capability (Pepke *et al.*, 2009; Mckenna *et al.*, 2010; Nagalakshmi *et al.*, 2010). Pabinger *et al.* (2014), suggested that the best way to better manage the large volume of genomics data is to choose the most appropriate frameworks among the existing available computational and analysis tools. Bioinformatics workflow frameworks are non-static, and biomedical researchers around the world are faced with an evolving need to produce analysis pipelines for investigating genomic datasets.

Comparative evaluation of the different workflow frameworks has therefore become a crucial requirement to choose, and implement, the most appropriate framework for a particular-problem. Bioinformatics frameworks often include support for extending functionality and features by using dedicated scripting programming languages, such as Python (Foundation, 2016), and this allows for easy integration of systems and other additional workflow features, to promote workflow flexibility, efficiency and scalability. Workflow framework strength is in simplifying the management of workflow control and dataflow structure, while the weakness lies in its lower level features which are not easily programmable since it requires experienced programmers. In a study by Curcin and Ghanem (2008) a high-level framework for comparing workflow systems, based on control and data flow properties is provided. A disadvantage of workflow system was illustrated by Hillman-Jackson and co-workers (2012). Here, the authors suggest that novice users may experience difficulty with creating and modifying workflows. Furthermore, libraries which need to be implemented in workflows to develop tool wrappers does require bioinformatics experience and as such, it is recommended that users make use of informative tools (such as instructional videos) to gain an understanding of workflow features (Hillman-Jackson *et al.*, 2012).

## 2.8: Conclusion to the literature review

This research review's purpose is to help an inexperienced biomedical researcher with less computer programing knowledge understand different kinds of bioinformatics workflow frameworks that exist out there. This is important because working with a bioinformatics workflow system can be overwhelming and choosing among GUI or CLI workflow frameworks is totally a question of personal choice. Moreover, the choice of workflow framework should be well-informed both by the demands of bioinformatics pipeline analysis and the needs of those using it. The use of a bioinformatics workflow system is ultimately tied to reproducible research (Kurs et al., 2016). Reusable analysis that can be easily implemented and run in the HPC are often desirable in terms of full resource control and management, reproducible research and the type of collaborative work in modern NGS studies.

Within the local setting that this project will be carried out, the demand for bioinformatics workflow systems that support the exploration of MTB genomic datasets should be made available. This literature review confirmed that bioinformatics workflow frameworks have different features and compositions. Pipeline analysis construction is often developed within the frameworks. Having determined an exact focus for the project on an evaluation of workflow frameworks, further investigation of the workflow frameworks revealed that there is a need for efficient and customizable bioinformatics workflow systems, or compute facilities that support biological sequence analysis and data-provenance for data-intensive computational analysis, to build NGS data analysis pipeline. Examination of the existing state of workflow frameworks has confirmed that there exists a gap in workflow constructions that could feasibly be addressed by implementing a SNP-based analysis pipeline that can process and analyze MTB genomic datasets. Therefore, this study aims to:

1) Address these challenges by presenting a concise evaluation of NGS data analysis pipelines embedded in the Galaxy and Ruffus frameworks. The feature set and performance of the investigated workflow frameworks are demonstrated in this study with the aim to assist biomedical researchers in making informed decisions related to the frameworks.
2) Evaluate Galaxy and Ruffus performance using state-of-the-art variant calling pipeline tools for MTB datasets.
3) Report the performance of an NGS analysis pipeline in the bioinformatics workflow systems for processing, analyzing and annotating of regularly generated MTB genomic datasets and that efficiently manage the analysis of large genomic datasets.

In this thesis, Chapter 3: describes research design and methods, Chapter 4: describe the pipeline integration and benchmark of Galaxy and Ruffus workflow frameworks, and Chapter 5: Concluding with remarks. Source code is described in appendices A-F.

# Chapter 3: Designs and Methodologies

This chapter details the different software tools and methods which were used for implementing the workflow frameworks (Galaxy and Ruffus). Each workflow framework was setup and organized on the South National Bioinformatics Institute (SANBI) HPC system environment. Furthermore, the chapter describes the case study; how a SNP-based analysis pipeline was integrated in the Galaxy and Ruffus environment, and the criteria necessary for the benchmarking. The pipeline-based frameworks and the bioinformatics tools employed in this study are reported and an assessment of cluster resource management was also conducted to determine how the Galaxy and Ruffus frameworks function. The assessment included the investigation of the way the HPC resource manager controls the basic computational units and system resources on (page 12) (Figure 3). It also enabled the setup of the Sun Grid Engine (SGE) job manager for the virtual working environment (Nocq *et al.*, 2013) which uses the computing nodes on the cluster facilities (Van Deventer, 2014).

Additionally, the Distributed Resource Management Application API (DRMAA), a global cluster resource manager that enables higher levels of system integration, was also deployed (Brown *et al.*, 2015). Git, which tracks changes made in open source projects, was initialized and set up into our cluster virtual working environment. We used GitHub as the source code-based repository to store, track changes, and apply logs of version control to the software and libraries we implemented in this project. To this end, the work on integration of Galaxy and Ruffus frameworks with the SANBI HPC cluster facilities has enabled novel, stimulating, productive and simplified ways to launch bioinformatics computing workflows. An HPC infrastructure enables scientists and researchers to perform workflow tasks that require large amount of computing capabilities to process and solve complex genomic problems. HPC typically utilizes a message passing interface (MPI) to communicate between different nodes (Alyssa, 2016). That is, MPI allow data to be transfer from location (one process) to that of another process through two-way operations on each process. Therefore, we assembled a coherent and reusable SNP analysis pipeline for processing, analyzing and annotating genomic data sets. The following subsections illustrate the steps taken to organize the workflow working environment.

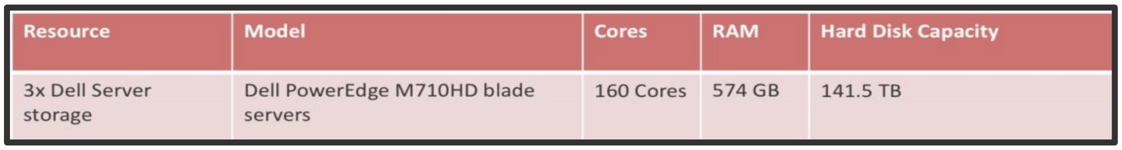
## 3.1: Distributed Software Control Version Systems

Distributed software control version systems allow easy access to source code repositories (Blischak *et al.*, 2016). GitHub as a control version platform assists in the management of the project software source code. Tracking code level changes is both a shared and required activity of today's open source community. For instance, in a software development environment, tracking of software source code is as essential as meticulous record keeping of lab procedures and protocols in the biomedical environment (Heller *et al.*, 2011). However, not all biomedical researchers are aware of the existence, or of the advantages using the distributed software control version systems as opposed to the traditional methods of source code repository (O'sullivan, 2009; Rother *et al.*, 2011; Altintas *et al.*, 2012). In this study, the Galaxy and Ruffus framework source code were derived from an existing open source repository (Appendix A, B and C).

## 3.2: Hardware Resource

In table 3, the technical hardware resources used in this study are summarized. The basics setup including the number of cores (processors), disk, and memory is illustrated.

**Table 3: Overall hardware resource used for the workflow frameworks**

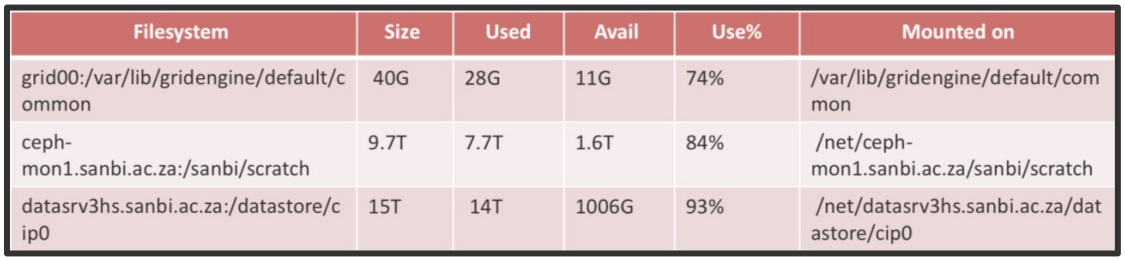


| Resource | Model | Cores | RAM | Hard Disk Capacity |
|---|---|---|---|---|
| 3x Dell Server storage | Dell PowerEdge M710HD blade servers | 160 Cores | 574 GB | 141.5 TB |

## 3.3: Virtual Working Environment

When working in a virtual distributed and shared HPC environment, there are several limitations when it comes to software integration, configuration, versioning, and management. A virtual environment (i.e. also known as virtualenv) is a toolbox, or container, that keeps the dependencies required by a project in separate places. Due to technical limitations in the study, Python virtualenv (Hale and Stanney, 2014) was the only viable system at our disposal, however, other options could be used outside of traditional HPC environments. With virtualenv we created isolated Python environments and avoided installing Python packages globally. That is, we installed the virtualenv using Python installer tools. The virtualenv allows us to create a folder called venv that contains all the necessary executables, Python libraries and packages needed in this study (Appendix A, B, C). The venv libraries provided support for creating a lightweight program, and also enabled us to integrate file systems with globally installed modules on the base system (Afgan *et al.*, 2012). The Python libraries and executable files used for building the workflows were kept within the virtual environment (Gorelick and Ozsvald, 2014). The workflow frameworks virtualenv was set up on HPC cluster (i.e., Linux base system integration) for software capabilities and compatibilities (Mcgough *et al.*, 2005; Kurs *et al.*, 2016). The workflow framework virtualenv provides support for batch queuing system such as Sun Grid Engine resource manager (Jamalian and Rajaei, 2015; Nishanth and Kihoon, 2015). The working environment resource is summarized in Table 4 (Appendix D).

**Table 4: Overall virtual working environment summaries**



| Filesystem | Size | Used | Avail | Use% | Mounted on |
|---|---|---|---|---|---|
| grid00:/var/lib/gridengine/default/common | 40G | 28G | 11G | 74% | /var/lib/gridengine/default/common |
| ceph-mon1.sanbi.ac.za:/sanbi/scratch | 9.7T | 7.7T | 1.6T | 84% | /net/ceph-mon1.sanbi.ac.za/sanbi/scratch |
| datasrv3hs.sanbi.ac.za:/datastore/cip0 | 15T | 14T | 1006G | 93% | /net/datasrv3hs.sanbi.ac.za/datastore/cip0 |

### 3.3.1: Virtualenv Setup for Sun Grid Engine

This study required mandatory decoupling of the reliance on a pre-defined computing environment to allow for switching between different HPC resources, without infrastructure constraints. To this end, a virtual Sun grid engine (SGE) was set up for the development and running of the SNP analysis pipeline on an HPC environment (Gorelick and Ozsvald, 2014). The SGE was configured to support execution on an HPC system. Furthermore, the SGE allowed for jobs to be scheduled and automatically distributed across the cluster resources. While ongoing jobs are running in the background on the HPC environments, jobs submitted to HPC resource may require continuous system integration, updates, and maintenance. SGE scheduler monitors all submitted jobs on cluster nodes and its deployment ensures that the cluster node does not get overloaded. SGE provides support for the Galaxy and Ruffus workflow frameworks. The distributed resource management application programming interface (DRMAA/PI) with SGE enabled jobs submission to the cluster. The DRMAA is a high level open grid that controls job submissions by using a distributed resource management (DRM) system, such as a Cluster or Grid computing infrastructure (Sun, 2007; Deelman, 2010). The DRMAA covers all the high-level functionality required for the Galaxy and Ruffus framework applications to control, query, submit and monitor jobs on execution resources in the DRM system (Guimera, 2012; Alyssa, 2016).The virtual machine setup is graphically represented in figure 5 (Appendix D).

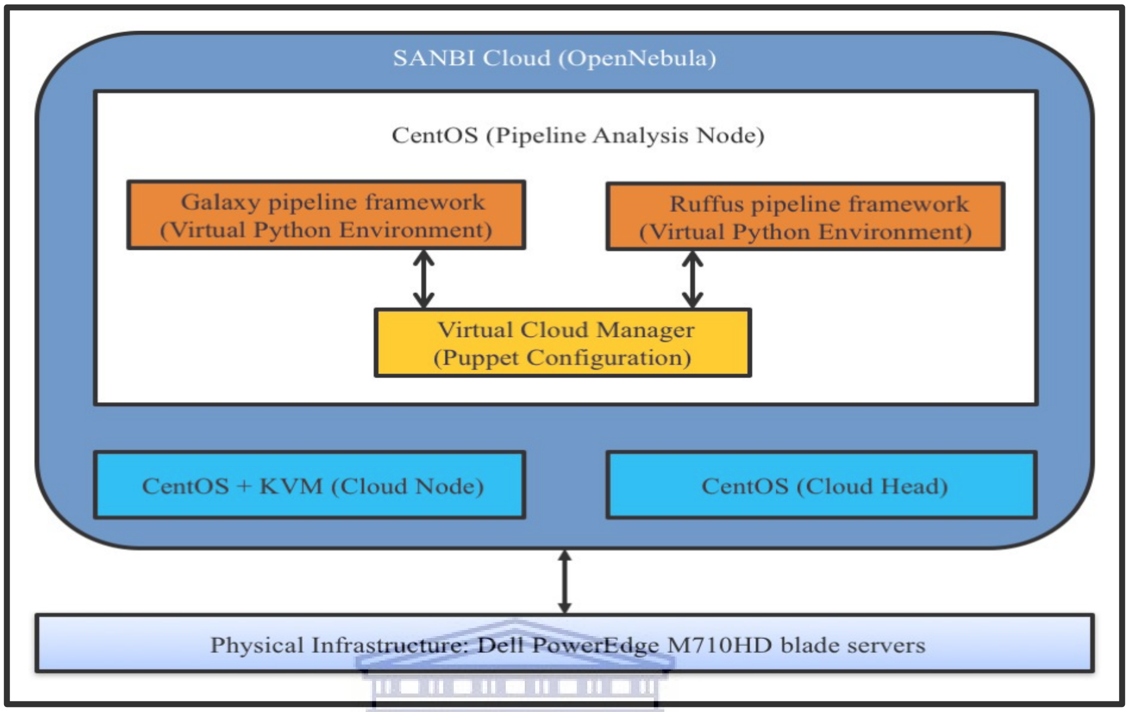

SANBI Cloud (OpenNebula)
CentOS (Pipeline Analysis Node)
Galaxy pipeline framework (Virtual Python Environment)
Ruffus pipeline framework (Virtual Python Environment)
Virtual Cloud Manager (Puppet Configuration)
CentOS + KVM (Cloud Node)
CentOS (Cloud Head)
Physical Infrastructure: Dell PowerEdge M710HD blade servers

**Figure 5: The virtual cloud infrastructure to physical layers.**

The virtual manager consists of a central management virtual node that runs the cloud controller and a number of cloud nodes that each run a supported hypervisor. The virtual manager interfaces with Virtual Machine (VM) that housed the virtual Python environment either libvirt, a Linux library that provides an abstract VM management interface, or the Amazon EC2 interface. At SANBI cloud nodes run CentOS 6.2 with the KVM hypervisor. The Dell PowerEdge M710HD blade server store the VM images on a Storage Area Network (SAN), accessed via iSCSI over a 10 gigabit Ethernet network (Van Heusden et al., 2012).

## 3.4: Implementation of MTB SNP Based Pipeline Analysis in Galaxy and Ruffus

The process overview of the data analysis workflow steps for the *Mycobacteria* datasets is presented below (Raman *et al.*, 2008). The SNP-based pipeline analysis allows for the raw reads coming off the sequencing machine to undergo numerous steps, ultimately generating variant call-sets. Each pipeline component phase was composed to execute a set of bioinformatics tools, using the distributed data-parallel execution patterns

(Altintas et al., 2012). Each step in the analysis pipeline which is capable of processing and analyzing the genomic data (e.g., MTB datasets) is presented in Figure 6.

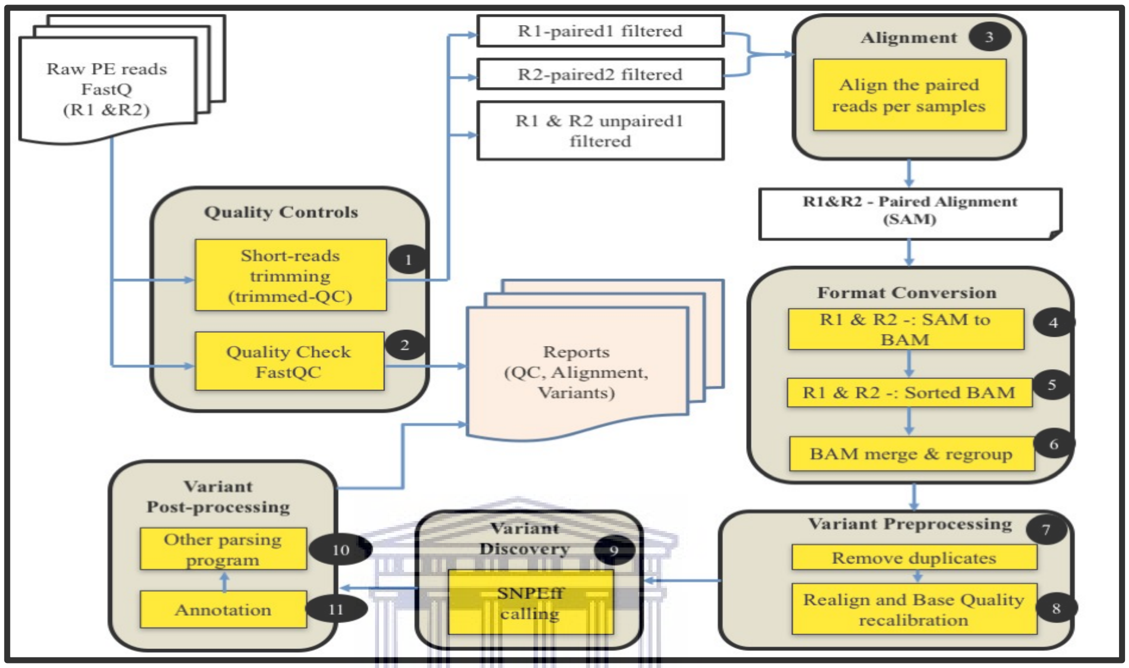


**Figure 6: Overview of the MTB NGS data analysis pipeline.**

We use the following standard to build the bioinformatics analysis workflow in the selected workflow frameworks (i.e., Galaxy and Ruffus). The analysis workflow consists of 10 major steps for exploring and annotating the MTB genomic data sets. The variant discovery step includes variant calling and annotation which leads to variant post-processing.

The analysis pipeline includes 11 tools, and performs data quality control and quality checking, filtering and trimming of sequence reads, alignment to a reference genome, post alignment analysis, and statistical evaluation and annotation of the detected variants (D'antonio *et al.*, 2013). To aid data analysis pipeline reproducibility, the analysis pipeline in the frameworks were saved as separate workflows in the Galaxy and Ruffus distributions installed on the HPC virtual environment.

### 3.4.1: Sample Data and Reference Genomic Data

Ten illumina NGS datasets from MTB (Tygerberg Medical School) were used in the SNP variant calling analysis pipeline. The MTB H37Rv strain (URL) was used as the reference genome for alignment and mapping in this study. The reference genome dataset was set to the instance of each workflow framework in the data libraries. The reference genome data was added to the instance using the administrative mode (Bretaudeau *et al.*, 2015) or admin write permission. The reference data for Galaxy analysis pipeline can be access from the data libraries menu on the Galaxy portal and imported to the histories for the downstream analysis. The reference data for Ruffus analysis pipeline was configured as list which formed part of the Ruffus configuration files and libraries.

### 3.4.2: Data Quality Assessment

Data quality control check and processes including data cleaning and formatting was seamlessly performed. Data quality analysis was performed using FASTQC on the short sequence reads and the subsequent results were evaluated prior to downstream analysis in page 92 (i.e., in Appendix E) (Pabinger *et al.*, 2014).

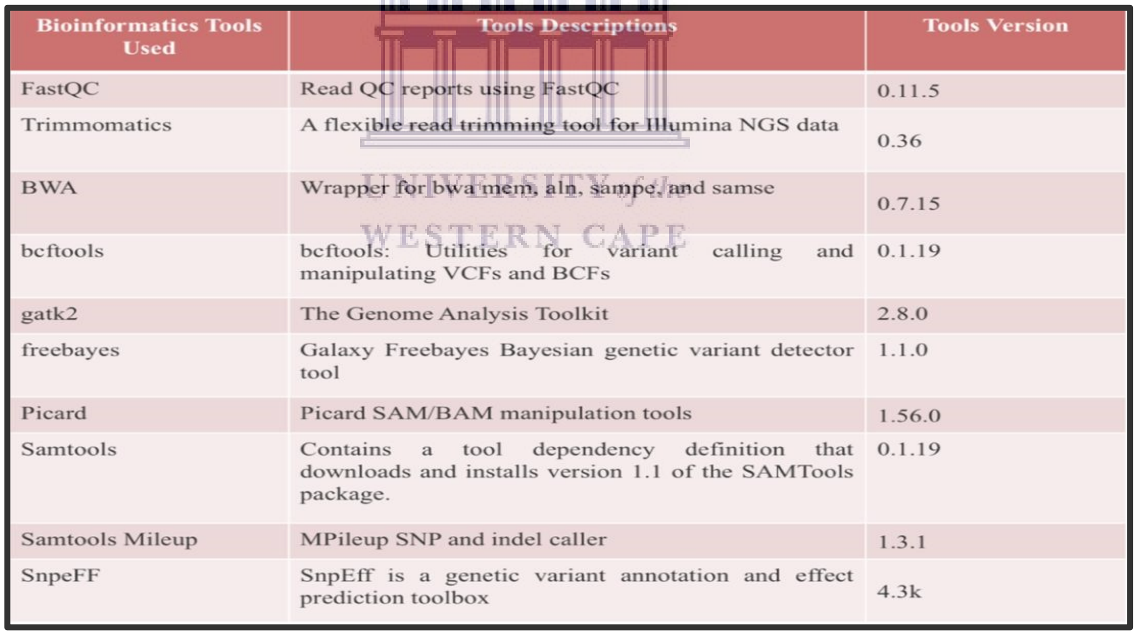
### 3.4.3: Secondary Analysis (Pre- and Post-Alignment)

Pre- and post-processing analyses were performed on per-sample data in three stages; 1) alignment to the known reference genome, 2) assembly, and 3) variant calling. The project use H37Rv decoy FASTQ dataset as reference file. The reference file (i.e., H37Rv) was indexed in order to ensure accurate alignment and mapping. Different mapping tools and algorithms (e.g., GATK best practice workflow) were used for different data types and results were captured and stored to files as variants calls file format (VCF) (Van Der Auwera *et al.*, 2013) (Appendix E).

## 3.5: Setting up Tools for the Galaxy and Ruffus Framework

The bioinformatics tools and versions integrated into the Galaxy and Ruffus framework are summarized in Table 5. The tools installation and the environment module files configurations were carried out within the cluster virtual environment. Python Conda modules were also setup in the virtual Python environment (Appendix A, B and C) (Sanner, 1999). The bioinformatics frameworks together with the integration of the data analysis pipeline consisted of additional external software and associated dependencies distributed within the cluster. Maintenance and updating of tools were performed and setup using the Git, a version control system to avoid conflict and out-of-date software issues when interrogating and manipulating the MTB datasets.

**Table 5: Tools used in the analysis pipeline**



| Bioinformatics Tools Used | Tools Descriptions | Tools Version |
|---|---|---|
| FastQC | Read QC reports using FastQC | 0.11.5 |
| Trimmomatics | A flexible read trimming tool for Illumina NGS data | 0.36 |
| BWA | Wrapper for bwa mem, aln, sampe, and samse | 0.7.15 |
| bcftools | bcftools: Utilities for variant calling and manipulating VCFs and BCFs | 0.1.19 |
| gatk2 | The Genome Analysis Toolkit | 2.8.0 |
| freebayes | Galaxy Freebayes Bayesian genetic variant detector tool | 1.1.0 |
| Picard | Picard SAM/BAM manipulation tools | 1.56.0 |
| Samtools | Contains a tool dependency definition that downloads and installs version 1.1 of the SAMTools package. | 0.1.19 |
| Samtools Mileup | MPileup SNP and indel caller | 1.3.1 |
| SnpeFF | SnpEff is a genetic variant annotation and effect prediction toolbox | 4.3k |

## 3.6: Setting up Module System Environment within the HPC

Installing, configuring, and maintaining environment modules package via modulefiles enables bioinformaticians to choose which bioinformatics software tools to use.

Moreover, from a technical point of view, packaging system modules is a time-consuming task which should not be a concern to biomedical researchers. Similarly, from a biomedical researcher's point of view, writing modules does not add any expertise to the biomedical software toolbox. An existing module environment was used in this study, and where required, missing modules were re-packaged and installed in the HPC cluster (i.e., the Python virtual environment on page 28).

## 3.7: Setting up the DRMAA to Interface with SGE on HPC

Figure 7 illustrates how the Sun Grid Engine (SGE) was setup for the host cluster to manage and control job submissions in the HPC environment (Appendix D). Using DRMAA, grid applications builders and portals, developers can use the same high-level API to link software with different cluster/resource management systems (Booth., 2013). The SGE-DRMAA software allows multi-user access and policy-based job control routines by the SGE queuing systems that manage the local computational resources (Deelman, 2010; Prajapati and Shah, 2014; Brown *et al.*, 2015). In this study, SGE + DRMAA usage provides an excellent tool for all the capabilities of the grid engine. The grid engine was administered via commands issued at the shell prompt and called within shell script. This was found to be a more flexible, rapid, and powerful strategy to change Grid Engine settings.

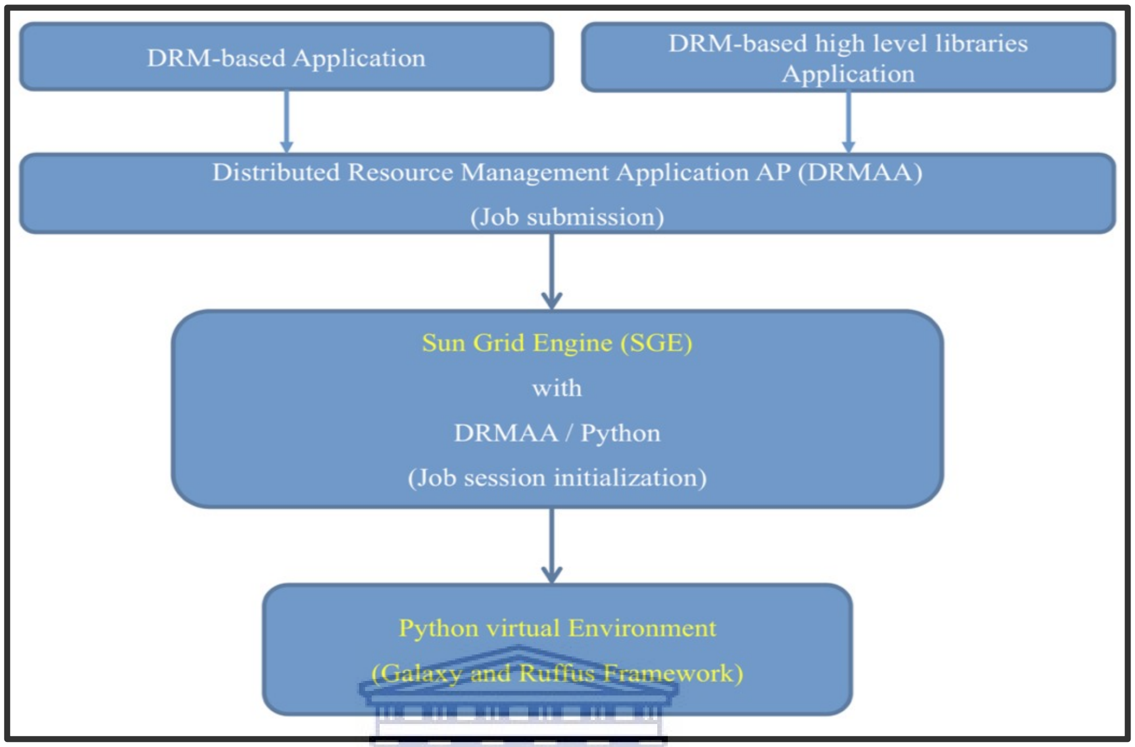
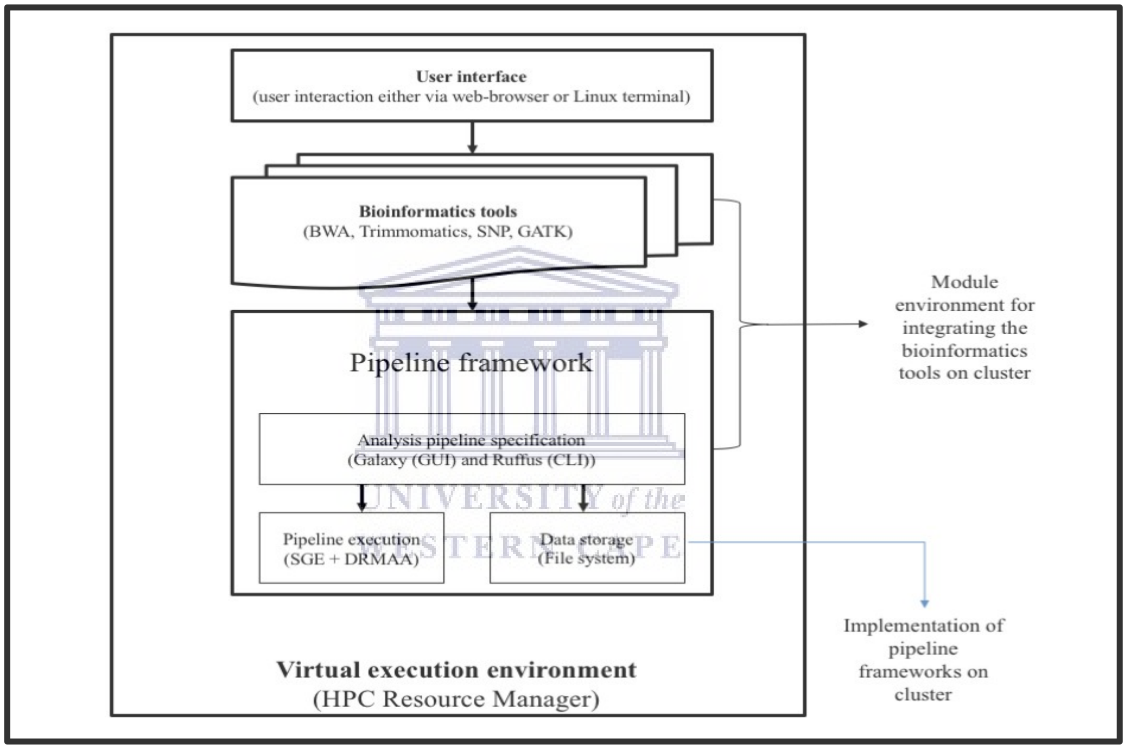

DRM-based Application
DRM-based high level libraries Application
Distributed Resource Management Application AP (DRMAA)
(Job submission)
Sun Grid Engine (SGE)
with
DRMAA / Python
(Job session initialization)
Python virtual Environment
(Galaxy and Ruffus Framework)

**Figure 7: Overview of the sun grid engine- distribution resource manager.**

We integrated some scripts detailing the SGE+DRMAA implementation within the Galaxy and Ruffus framework. The SGE-DRMAA control the analysis pipeline jobs submission as well as monitoring in the queue and reporting on both cluster usage and execution. The SGE+DRMAA manage the resource in the Python virtual environment and ensure resource and cluster management, profiling and tracing.

## 3.8: Service in the Pipeline Framework

Prior to commencing this study, building of an analysis pipeline involving several bioinformatics tools and pipeline frameworks was discussed. The analysis pipeline process (Figure 6 on page 31) involved six phases, including the quality control, alignment and format conversion, variant pre-processing, variant discovery or call sets and post-processing. The software programs that formed the bioinformatics toolkit would allow researchers to analyze and extract and/or transform the genomic data to glean information for the genetic study. The pipeline tools and their dependencies were specified using an integrated module system environment. The pipeline specifications consist of references to a range of software packages to be installed without specifying the execution environment (Möller *et al.*, 2017). In addition to Galaxy/Ruffus framework setup on the virtual Python environment and execution of SGE in combination with DRMAA, the pipeline framework stored the genomics data as files or objects, in data storage (on the SANBI HPC cluster). The overall virtual environment, the resource requirement for seamless running the pipeline framework, and the provisioning of user interface access for researchers to the analysis pipeline, is illustrated in figure 8.


User interface
(user interaction either via web-browser or Linux terminal)
Bioinformatics tools
(BWA, Trimmomatics, SNP, GATK)
Pipeline framework
Analysis pipeline specification
(Galaxy (GUI) and Ruffus (CLI))
Pipeline execution
(SGE + DRMAA)
Data storage
(File system)
Virtual execution environment
(HPC Resource Manager)
Module environment for integrating the bioinformatics tools on cluster
Implementation of pipeline frameworks on cluster

**<u>Figure 8: Overall pipeline framework and setup service.</u>**

This represents the pipeline framework and setup service components. The framework consists of third-party bioinformatics tools.

The analysis pipeline was formulated and established for downstream analysis, and jobs submission to the virtual cluster was monitored, and as such this established a process for accounting for the jobs profile. An accounting record for each job profile in the Galaxy and Ruffus workflow frameworks was set up and written to an SGE accounting file. The information in the accounting file included records that track analysis pipeline resource usage as well as the amount of data transferred in input and output pipeline operation (Booth., 2013; Prajapati and Shah, 2014). The SGE accounting parameters used in this study was the qacct command which enabled direct access to the complete resource usage information stored by the SGE (Oracle, 2017).

## 3.9: Benchmarking Criteria

As workflow features advanced, the difficulty in performance comparisons between the various workflow frameworks increased. In this study, we used tools to monitor the Galaxy and Ruffus workflow activities and the benchmarks used were Collectl-Util/Colplot pipeline response time and runtime execution. Furthermore, with the view to benchmark the cluster-based workflows in this study, the performance of the analysis pipelines in the two frameworks was conducted using standard tests (**such as real time testing, system time, and user time**). The benchmark process includes obtaining the total execution time (i.e., by considering the up-to-date completion time of the previous pipeline step to the latest completion time of the current pipeline step as described in Figure 6) for each pipeline step in the dataflow design during the implementation and execution of the DNA-seq analysis pipeline (Appendix E). Each stage in the implementation design provided steps enabling tool integration within the Galaxy and Ruffus frameworks and furthermore, each step allowed for both the processing and analysis of the MTB genomic dataset. Chapter 4 explains in detail the time measurement for each analysis pipeline step.

### 3.9.1: Performance Measurement

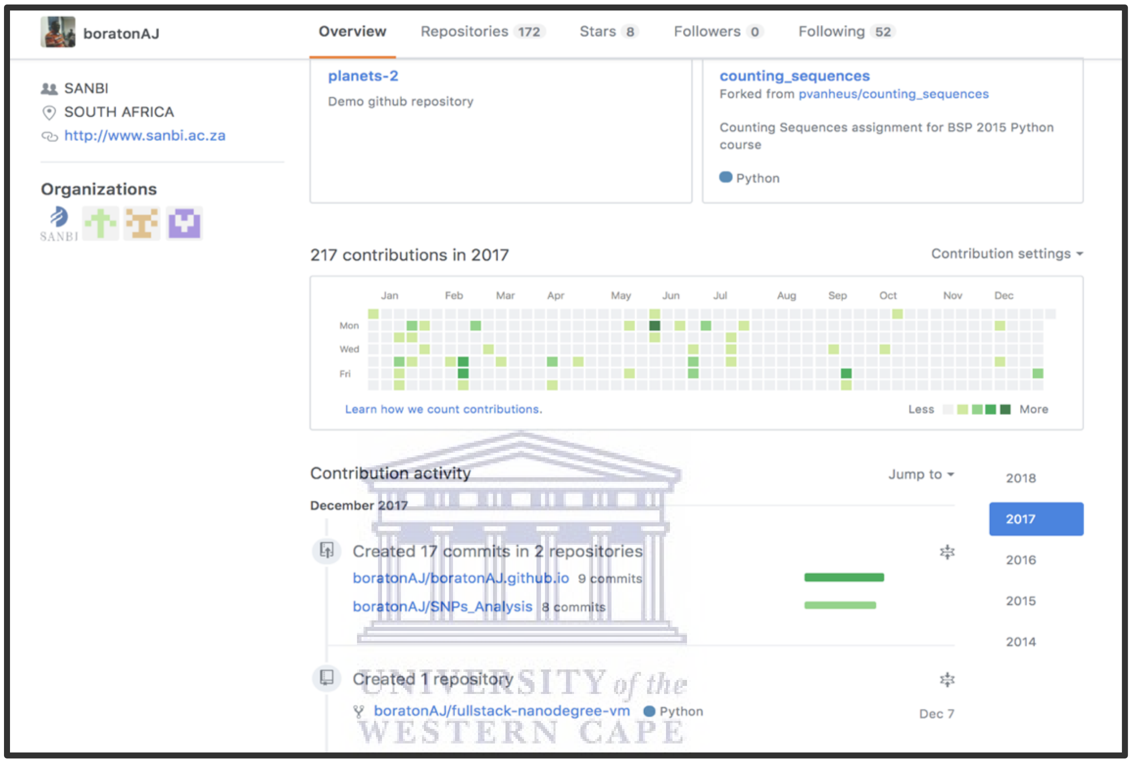
This study characterized each step of the analysis pipeline using Collectl-Utility, a tool used to measure the performance of a system, in order to create a pipeline profile that determines typical execution of tools within the workflow frameworks (Kelly *et al.*, 2015). The Collectl-Utility allows for transitory and/or comprehensive measurements for both Galaxy and Ruffus compute node. The transitory measurements allow for an aggregate view of the CPU usage in the system and the same techniques were employed to measure the disk performance and network performance. Comprehensive measurements provided further detail into the individual parts assessed (Layton, 2017). For example, CPU usage could be measured for each individual CPU using a comprehensive mode in Collectl. This analysis was also applied to disk and network performance and Colplot, and the addition with GNU plot, allowed for graphical representation of the findings (Appendix F).

## 3.10: Continuous Integration System

Galaxy and Ruffus frameworks components have software checks as quality control checks. It is therefore important to routinely run software update checks whenever changes are introduced into any of the framework's components. The rationale for introducing the continuous integration system is to certify that Galaxy and Ruffus workflow framework component modules continue functioning correctly after any developer has introduced changes (Sanner, 1999; Pabinger *et al.*, 2014). To this end, this study utilized buildbot (a Python-based approach) and Git for the software continuous integration and notification (Brian and Dustin, 2009; Gray *et al.*, 2010). Following the selected configuration in the Galaxy and Ruffus frameworks, the Git agent was used to monitor the remote repository, and as soon as changes were identified, Git agent fetched the changes and sent notification. Subsequently, the changes were evaluated in the software test suite in order to warn against potential software breaks (Blischak *et al.*, 2016).

### 3.10.1: Contributing Code on GitHub

Tracking changes made to the bioinformatics software tools utilized in this study was a vital component of the project success. To this end, software control versioning was used, which essentially allowed for the concurrent control the software versions, and the project source code. For sharing and collaboration amongst the open source community (Heller *et al.*, 2011), the source code for this project is made available on GitHub and can be found using the following link: **https://github.com/boratonAJ/SNPs_Analysis** (Figure 9) (Schall, 2015). Our contributing software tools utilized in this study can be found in Appendix A, B, C, D, and E.


boratonAJ

Overview | Repositories 172 | Stars 8 | Followers 0 | Following 52

SANBI
SOUTH AFRICA
http://www.sanbi.ac.za

Organizations

planets-2
Demo github repository

counting_sequences
Forked from pvanheus/counting_sequences
Counting Sequences assignment for BSP 2015 Python course
Python

217 contributions in 2017

Contribution settings

Learn how we count contributions.

Contribution activity

Jump to

December 2017

Created 17 commits in 2 repositories
boratonAJ/boratonAJ.github.io 9 commits
boratonAJ/SNPs_Analysis 8 commits

Created 1 repository
boratonAJ/fullstack-nanodegree-vm Python Dec 7

2018
2017
2016
2015
2014

**Figure 9: GitHub contribution and commit activity.**

The diagram represents the activity of our implementation at SANBI labs. The contribution timeline shows the way we contributed to the open source project.

## 3.11: Distributing SNP Analysis Packages on Python Package Website

SNP Analysis used Setup.py to setup the Python package from Python Package Index (Pypi). The Setup.py is a Python file that tells operating system the module to install with the assistance of Python distribution utilities (Distutils). The Distutils is the standard for distributing Python Module. The SNP Analysis project setup was as follows;

a) Package - A folder/directory that contains __init__.py file.

b) Module - A valid Python file with .py extension.

c) Distribution – Package that is related to the project.

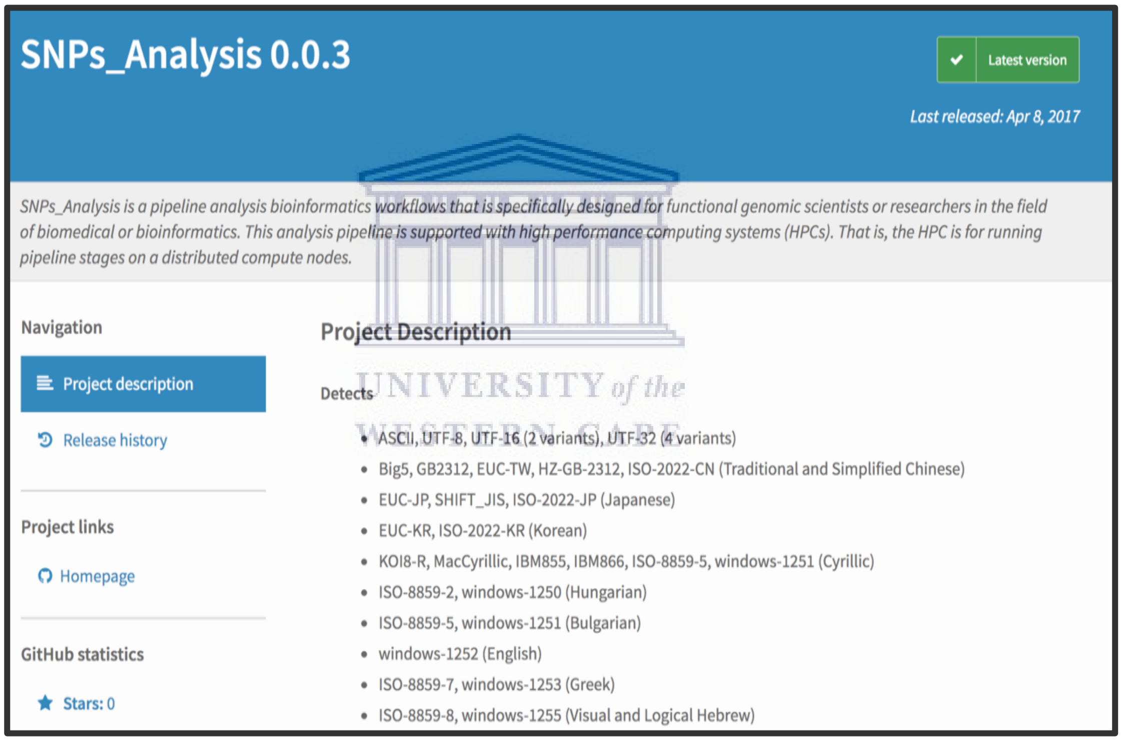
More so, the following steps highlighted how the package was built and distributed;

a) The layout of the project files

b) directory structure

c) creating the project distribution file and

d) the project package name was registered at the Python Package Index (PyPI).



SNPs_Analysis 0.0.3

Latest version

Last released: Apr 8, 2017

SNPs_Analysis is a pipeline analysis bioinformatics workflows that is specifically designed for functional genomic scientists or researchers in the field of biomedical or bioinformatics. This analysis pipeline is supported with high performance computing systems (HPCs). That is, the HPC is for running pipeline stages on a distributed compute nodes.

Navigation

Project description

Release history

Project links

Homepage

GitHub statistics

Stars: 0

Project Description

Detects

- ASCII, UTF-8, UTF-16 (2 variants), UTF-32 (4 variants)
- Big5, GB2312, EUC-TW, HZ-GB-2312, ISO-2022-CN (Traditional and Simplified Chinese)
- EUC-JP, SHIFT_JIS, ISO-2022-JP (Japanese)
- EUC-KR, ISO-2022-KR (Korean)
- KOI8-R, MacCyrillic, IBM855, IBM866, ISO-8859-5, windows-1251 (Cyrillic)
- ISO-8859-2, windows-1250 (Hungarian)
- ISO-8859-5, windows-1251 (Bulgarian)
- windows-1252 (English)
- ISO-8859-7, windows-1253 (Greek)
- ISO-8859-8, windows-1255 (Visual and Logical Hebrew)

**Figure 10: An illustration of the SNP Analysis pipeline published on Python packages with the Python package index.**

An account was created on (https://pypi.org/) and we publish the developed Python package with the Python Package index at (https://pypi.org/project/SNPs_Analysis/) for sharing the project package distribution. Figure 10 show the SNP Analysis package page on Pypi. This helps the biomedical researcher find and install the developed package.

# Chapter 4: Pipeline Integration and Benchmarking

Knowledge gained from the Chapter 2 and 3 was implemented in a practical case. As the need for access to an efficient workflow framework in high performance computing environment has increased, so has the requirement for the development of bioinformatics workflow systems in South Africa, across the Africa continent and on a global level. With an aim to contribute to the bioinformatics open-source community, the addition of new pipeline analyses to the bioinformatics workflow systems would also allow for biomedical researchers to perform tasks that are impossible. This means that bioinformaticians can use the workflow systems to build analysis pipelines, allowing biomedical researcher to run the pipeline, for example, without ever leaving the workflow framework environment. Galaxy and Ruffus integration with computing resources at SANBI were benchmarked. This benchmarking process was criterial to address factors such as pipeline flexibility, ease of use, execution time, processing time, solvability and reproducible and community support. It further motivated the relevance of the framework in the biomedical research community. The tabulated features demonstrated measurable performance and metrics. A short discussion on how the Distributed Resource Management Application API (DRMAA), a generalized resource manager, enabled higher levels of integration, and, our modifications to it. An evaluation of the management of the Galaxy and Ruffus framework on cluster resources (how the pipeline frameworks handle the basic computational units during execution of the analysis pipeline) was required to provide knowledge of how our setup works in terms of data storage, network capabilities and processing time and accuracy.

## 4.1: Genetic Data Processing

A total number of 10 MTB genomic datasets from Tygerberg Medical School, South Africa was used for the downstream analysis. The analysis includes data processing, manipulation, filtering, assembly and annotation using Galaxy and Ruffus workflow framework. In this study, we implemented analysis pipelines in the Galaxy and Ruffus framework, custom-made specifically for identification of a SNP (Figure 13) based on genetic variations. The genetic variation was detected by using reference sequences to identify variant at a given position in an individual genome or transcriptome. The variant were characterizing to be either synonymous or non-synonymous, together with insertions and deletions (Cohen *et al.*, 2015). The output result from the downstream analysis on SANBI HPC facility was stored for interpretations and future retrieval. The project metadata datasets were managed within the pipeline framework and both the input and output data was stored as parallel filesystems on the HPC environment for pre- and post- data analysis. The reference genomes of MTB (H37Rv) used for alignment and mapping were downloaded from the NCBI database (ftp://ftp.ncbi.nih.gov/genomes/Bacteria/) and a PERL script was used to convert from GenBank format - to FASTA format, and generic feature format (GFF). The reference genome index was also integrated as part of our analysis pipeline.

## 4.2: Galaxy Configuration

Galaxy is a web-browser software system application for accessible, reproducible, and collaborative analysis of high-throughput '-omics' data (Goecks et al., 2010). The Galaxy project aims to make computational analysis pipelines accessible to research scientists that do not have computer programming experience (Blankenberg et al., 2010, Atwood et al., 2015) and is widely supported by a large research community. Galaxy provides an intuitive user interface with which researchers can build pipelines or use existing pipelines to perform analysis on data such as genomic DNA sequences. In our Galaxy configuration, BWA-GATK and Freebaye-GATK for calling variants, the analysis pipeline was wrapped and configured based on the GATK recommended best practices, to demonstrate the SNP analysis pipeline on the HPC cluster. The SNP analysis pipeline selectively calls variants by grouping Synonymous and Non-Synonymous variant call sets. A concise description on source code structure is published as part of the official Galaxy documentation and can be found using the following URL: https://github.com/galaxyproject. The project structure overview is provided in appendix C.

## 4.3: Ruffus Configuration

The design and architectural goals of the Ruffus module is to be simple, intuitive, lightweight, powerful and flexible (Ruffus, 2016). Integrating and configuring the Ruffus framework on HPCs, provides a way to develop scripting data analysis pipelines that work together with a suite of predefined bioinformatics software tools, and are customized as modular Python scripts. The framework has enabled the building of an easily accessible and reproducible in-house SNP data analysis pipeline and dataflow management system. To this end, the Ruffus framework manages the computational analysis operations of each stage of the SNP analysis pipeline that are written in separate Python scripts. The analysis pipeline has input sample files that includes the pair-end read sample number (e.g. assigning "_R1" and "_R2" as prefix to the sample files) and the file extension “. fastq", all in lower-case. The input sample files to the workflow are gzip-compressed with the file extension “. fastq.gz". In addition, three simple phases were used to build this in-house Ruffus SNP analysis pipeline and include: 1) importing ruffus, 2) “Decorating” functions that are part of the pipeline, and 3) running the pipeline. With Ruffus framework, the process of executing the analysis pipeline is managed and ensure that the dependencies software and file names of the dataset as it flows across the analysis pipeline stages are specified in advance. The pipeline stage functions are specified in the correct order, with the precise parameters, running in parallel with the SGE + DRMAA that assist in splitting the HPC central processing units (CPUs) into several processes and jobs submission. The Ruffus environment which is utilized at SANBI uses a Python function in the script which performs the analysis staging. The source code for this project can be found using this URL (https://github.com/boratonAJ/SNPs_Analysis,) and the source code structure overview is provided in appendices A and B.

## 4.4: Galaxy and Ruffus deployment on HPC?

We deployed each workflow framework on an HPC cluster, in a virtual environment by following the instructions on the official Galaxy and Ruffus webpages to setup the framework instances. In addition to the setup processes and as outlined in section 3.3 and 3.4 (page 28 and 29), the following steps were also considered;

a) A clean VM environment and a dedicated user account was provided
b) A local SQLite database to a dedicated VM server instance (e.g., PostgreSQL) was set up to ensure for concurrent connections to stored metadata and to increase the response time of the Python virtual environment on the HPC.
c) SSH and FTP mechanisms to access and send data off-browser were enabled
d) Different performance tuning aimed at ensuring a better user experience was ensured with Collectl-Util, a resource monitor that dynamically monitor the performance of the Galaxy and Ruffus framework.

## 4.5: Implementation of the SNP Analysis pipeline in Galaxy and Ruffus

Following extensive considerations of the above-mentioned steps, the SNP analysis pipeline was tested and deployed. The analysis pipeline follows best practices as outlined by the Broad Institute (Mckenna *et al.*, 2010) (Appendix E). The downstream analysis tools used in the pipeline include quality data format tool (FastQC), aligners and sorting (BWA-MEM and SAM tools), mark and remove duplicate (Picard tools), variant callers (GATK Haplotype Caller) and SNP effect predictors (SNPEff). Genome Analysis Toolkit (GATK) is one of the most popular variant calling application tools, and together with BWA-MEM enabled the project to compose a data analysis pipeline focusing on SNP and insertions/deletions (INDELs) discovery (Mckenna *et al.*, 2010; Depristo *et al.*, 2011). In this project, we did not use one approach to configure all tools, but we utilized dual processes, the optimal configurations for each of the variant analysis tools and parameter. In appendix E, we demonstrate a simplified step of the variant call pipeline in which pre-processing alignment, post-processing alignment and variant discovery in the Galaxy and Ruffus frameworks were tested. The integration of the pipeline steps formed the SNP analysis pipeline implementation. From the overview above, the pipeline steps helped to format, convert, correct and identify the novel genetic variants that are associated with *Mycobacterium tuberculosis* drug resistance. The steps encompass phases that include quality control checks, alignment and mapping with reference genome *(*MTB-H37Rv), local realignment, discovery of single nucleotide polymorphisms (SNPs) and annotation of the variants using the GATK Haplotype Caller (Mckenna *et al.*, 2010) or Freebayes (Pabinger et al., 2014). The general flowchart for the analysis pipeline model in the workflow frameworks is represented in the figure below (Figure 11).

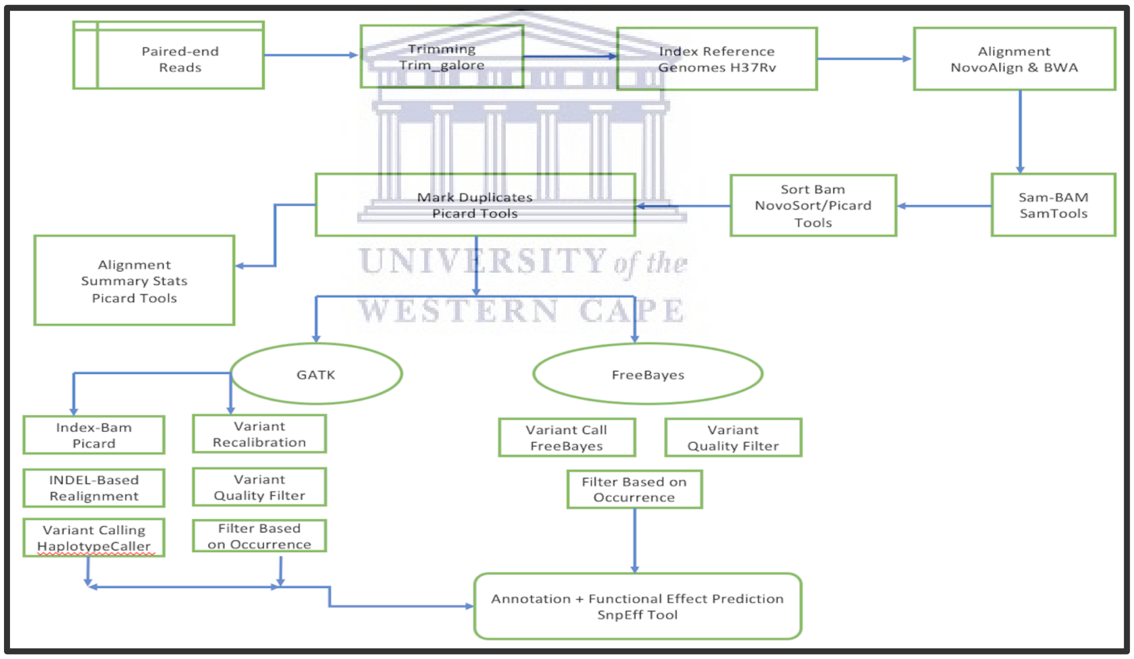

Paired-end Reads
Trimming Trim_galore
Index Reference Genomes H37Rv
Alignment NovoAlign & BWA
Mark Duplicates Picard Tools
Sort Bam NovoSort/Picard Tools
Sam-BAM SamTools
Alignment Summary Stats Picard Tools
GATK
FreeBayes
Index-Bam Picard
Variant Recalibration
INDEL-Based Realignment
Variant Quality Filter
Variant Calling HaplotypeCaller
Filter Based on Occurrence
Variant Call FreeBayes
Variant Quality Filter
Filter Based on Occurrence
Annotation + Functional Effect Prediction SnpEff Tool

**Figure 11: A generic unify flow model for SNP discovery analysis pipeline.**

The following diagram illustrated the strategy we used to set up the pipeline analysis in the pipeline frameworks. The flow model represents the general the analysis pipeline process we established for this project.Several bioinformatics tools were incorporated. https://doi.org/10.1371/journal.pone.0075619.g001.

## 4.5.1: SNP Analysis Pipeline Implementation in Galaxy

Galaxy SNP analysis pipeline and bioinformatics tools combines the power of genomics annotation catalogs with a web portal (Blankenberg *et al.*, 2010). A variety of bioinformatics tools and algorithms were implemented with the aim to enable researchers to search local and remote resources. Following this, the SNP analysis pipeline in the Galaxy framework was created and deployed (Figure 12). The advantage of the SNP analysis pipeline in Galaxy is that it enables researchers to perform an analysis on the genomic data using the large suite of available bioinformatics tools, from beginning to end, and, allows for interaction with selected genomic results through browsing Galaxy history. The history generated in Galaxy serves as an analysis record which can be used to demonstrate result reproducibility. Galaxy output files obtained were detailed PDF reports with results in the form of tables, text and graphic files. The Galaxy framework tracks an individual's job runs, along with features that enable the researchers to perform independent data queries, prepare, manipulate and visualize, share, publicly post, delete, or archive results. Galaxy uses a shared file system between its application server and the cluster nodes. This ensures that researchers can create and share pipelines/workflows of their analysis with each other. The Galaxy tool utilized additional scripts that allowed for the upload of the genomics data sets from the in-house storage server. In this study, not all parameters were set and as such the flexibility of Galaxy made it unique to deliver a highly automated solution. The workflows can be created in one of two ways; 1) Using the installed tools to create the required analysis pipeline prior to generating the grid flowchart workflows, or 2) Using the grid workspace to directly create and connect the GUI flowchart workflows steps (Appendix C). Although Galaxy is in late-stage beta testing, over 600 users have created almost 2500 workflows since August 2008 (Project, 2016). However, further testing of Galaxy is underway in order to address serious application issues, such as the simplest way to build and automate the Galaxy application tools (Piras *et al.*, 2017)

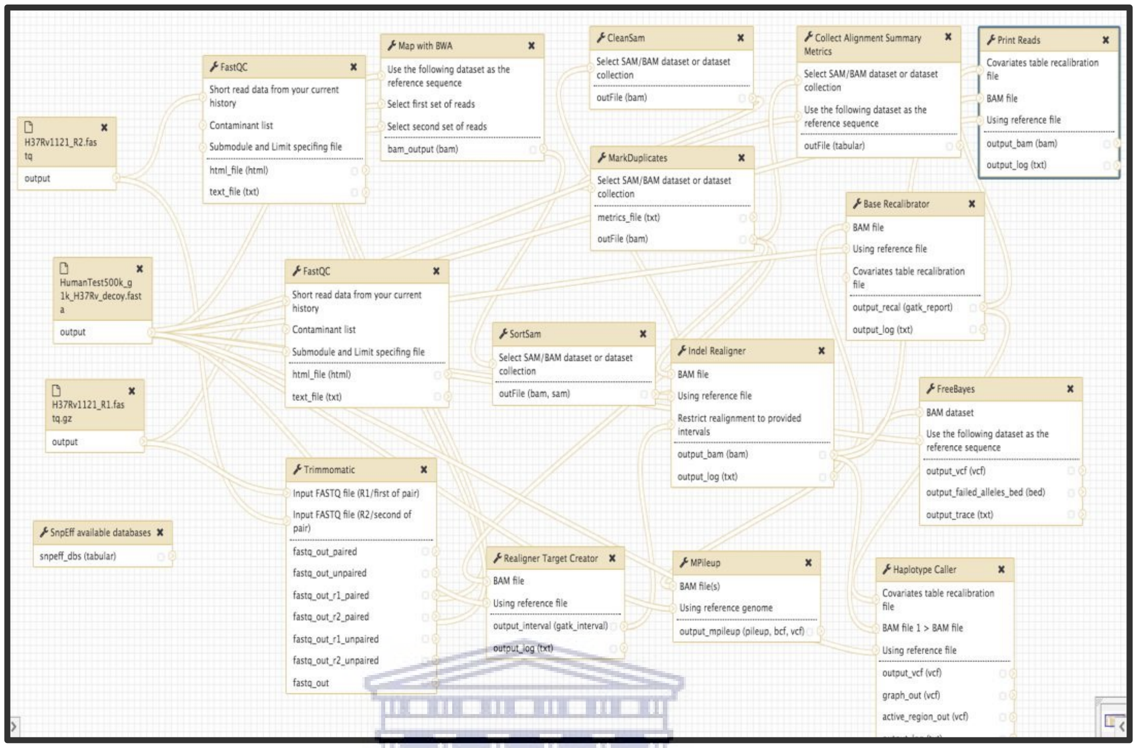

H37Rv1121_R2.fastq
output
FastQC
Short read data from your current history
Contaminant list
Submodule and Limit specifing file
html_file (html)
text_file (txt)
Map with BWA
Use the following dataset as the reference sequence
Select first set of reads
Select second set of reads
bam_output (bam)
CleanSam
Select SAM/BAM dataset or dataset collection
outFile (bam)
Collect Alignment Summary Metrics
Select SAM/BAM dataset or dataset collection
Use the following dataset as the reference sequence
outFile (tabular)
Print Reads
Covariates table recalibration file
BAM file
Using reference file
output_bam (bam)
output_log (txt)
MarkDuplicates
Select SAM/BAM dataset or dataset collection
metrics_file (txt)
outFile (bam)
Base Recalibrator
BAM file
Using reference file
Covariates table recalibration file
output_recal (gatk_report)
output_log (txt)
HumanTest500k_g1k_H37Rv_decoy.fasta
output
FastQC
Short read data from your current history
Contaminant list
Submodule and Limit specifing file
html_file (html)
text_file (txt)
SortSam
Select SAM/BAM dataset or dataset collection
outFile (bam, sam)
Indel Realigner
BAM file
Using reference file
Restrict realignment to provided intervals
output_bam (bam)
output_log (txt)
FreeBayes
BAM dataset
Use the following dataset as the reference sequence
output_vcf (vcf)
output_failed_alleles_bed (bed)
output_trace (txt)
H37Rv1121_R1.fastq.gz
output
Trimmomatic
Input FASTQ file (R1/first of pair)
Input FASTQ file (R2/second of pair)
fastq_out_paired
fastq_out_unpaired
fastq_out_r1_paired
fastq_out_r2_paired
fastq_out_r1_unpaired
fastq_out_r2_unpaired
fastq_out
SnpEff available databases
snpeff_dbs (tabular)
Realigner Target Creator
BAM file
Using reference file
output_interval (gatk_interval)
output_log (txt)
MPileup
BAM file(s)
Using reference genome
output_mpileup (pileup, bcf, vcf)
Haplotype Caller
Covariates table recalibration file
BAM file 1 > BAM file
Using reference file
output_vcf (vcf)
graph_out (vcf)
active_region_out (vcf)

**Figure 12: Diagram showing the Galaxy SNP analysis pipeline steps.**

A grid workflow diagram which represents an overview of the Galaxy SNP analysis pipeline employed in this study is provided below. The Print reads command option write out sequence read data from the BAM file after the BQSR and was used as part of BQSR for filtering, merging, and subsetting etc.

## 4.5.2: SNP Analysis Pipeline Implementation in Ruffus

SNP analysis pipelines using the Ruffus framework library allows a biomedical researcher to carry-out variant calling data analysis with specific sets of bioinformatics tools (Appendix A, B, and E) (Leipzig, 2016). Ruffus framework ensures that the correct data flows down the analysis pipeline in the correct way at the right time. With Ruffus library, the SNP analysis pipeline permits the automation of tasks in parallel, alongside management of task execution and visualization. The Python scripts at each stage of the SNP analysis pipeline implementation take single inputs at a time (except for pre-processing data analysis stage which takes paired-end input data) when configuring the pipeline. All jobs parallelism is handled by the integration of SGE + DRMAA with Ruffus framework. The Ruffus SNP analysis scripts enables discrepancy checking between tools and job checkpoints to ensure that tasks have been completed. In addition, the Ruffus SNP analysis pipeline provides several enhancements, including a convenient command-line syntax with configuration files that helps the biomedical researcher to describe the pipeline parameters, as well as the ability to run jobs either locally or on HPC cluster systems (Bao et al., 2014; Kurs et al., 2016). In this study, the SNP analysis pipeline was based entirely on standard Ruffus metadata libraries for variant call sets, SNP-transcripts, and genomic reference-based data. Ruffus libraries in combination with Graphviz software (i.e., *"a utility programs useful in graph visualization; and libraries for attributed graphs"*) were among the various software tools integrated in this project (Ellson *et al.*, 2001). The Graphviz software was utilized to generate an automatic analysis pipeline flowchart graph which provides an overview of the SNP analysis pipeline in the Ruffus framework (Figure 13).

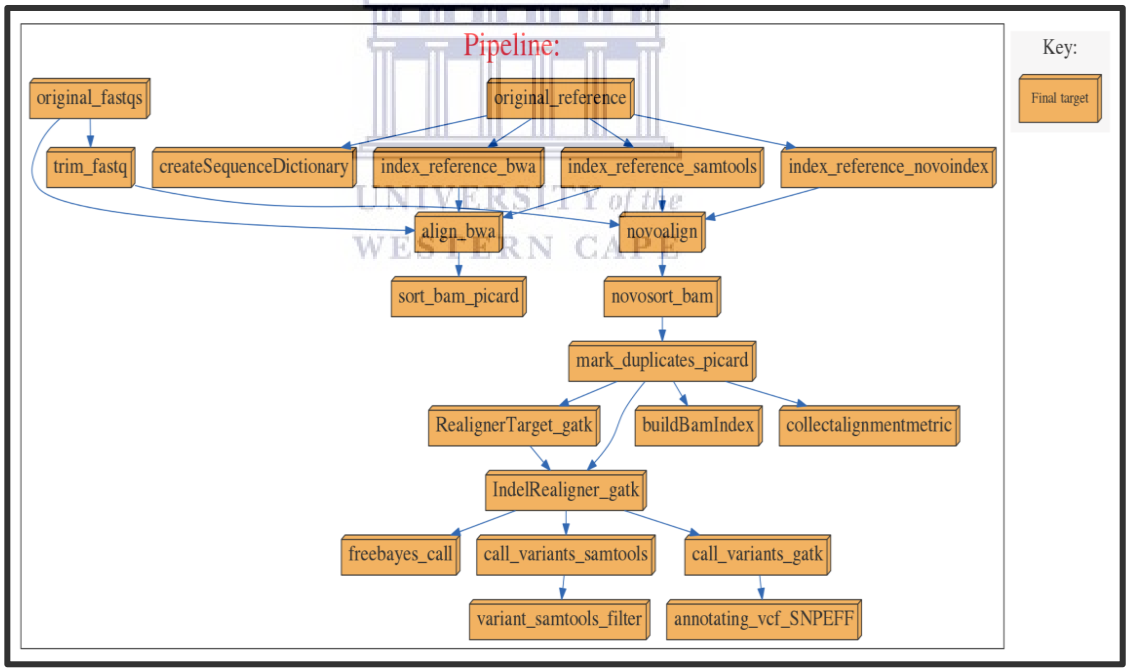


**Figure 13: Diagram showing the Ruffus SNP analysis pipeline steps.**

The graph illustrates the overview of the steps executed in the SNP analysis pipeline. The flow chart was generated using the pipeline_printout_graph function from Ruffus. Other tools, such as SNPEff automatically generated a statistical summary report in html format following annotation of the VCF file. The base quality recalibration (BQSR) was avoided in this project due to direct detection of haplotypes.

Therefore, the integration and implementation of Galaxy and Ruffus SNP analysis pipeline together with bioinformatics software tools allows this project to integrate and test the framework with the mycobacteria genomic datasets in an automatic, reliable, and disk space-saving way.

## 4.6: Comparative Analysis

In Chapter 3, the case study which involved a customized data analysis pipeline for calling variants (i.e. SNP) (with jobs running from start to end) in both Galaxy and Ruffus, was presented. The pipeline is customized around several bioinformatics tools (e.g., BWA, Freebaye and GATK) and is routinely utilized for analyzing and annotating the bacterial genome datasets (e.g., MTB). Logical techniques were applied when constructing the pipeline for discovery of variant call sets and the bioinformatics tools utilized on the cluster working environment were parallelized to speed up analyses in each framework virtual manager environment on page 30. In the Galaxy virtual environment, scripts such as XML, were incorporated and wrapped around the various bioinformatics tools in the virtual cluster manager. On the other hand, in the Ruffus virtual environment, the analysis pipeline was developed by integrating the Ruffus library together with configuration module files which called all the bioinformatics tools in the HPC cluster virtual environment. An advantage of using the Galaxy framework is that experimental biologists (i.e., naïve scientists) that have no knowledge of computer science and programming but want to develop a variant calling analysis pipeline to find genomic region targets for experimental validation can interact with Galaxy workflow with a focus on workflow reproducibility and collaboration between biomedical research labs or institutes that may experience difficulty in developing and/or building analysis pipelines. On the other hand, the advantages of using Ruffus analysis pipeline is that the framework can accommodate both basic Python scripts and production level data analysis pipelines, which includes features such as, serial and parallel steps, dependency checking, data transformation and good naming convention for input and output files, as well as user-defined parameters that are fixed and deliverable, and automatic failure recovery (Ruffus, 2016). The following sections give more insight on the comparative evaluation;

## 4.7: Galaxy Framework features versus Ruffus features

A unique feature of Galaxy is the large number of tools for operations on a number of DNA-seq files (*.fasta) (e.g., multiple sequence alignment) based on the bx-Python project, a Python library for manipulating biological data with associated set of scripts developed by the Galaxy Team for fast implementation of rapid genome scale analyses (Sinclair, 2010; Blankenberg *et al.*, 2011). These scripts include intersect, subtract, complement, merge, concatenate, cluster, coverage, base coverage, and join. Galaxy users may benefit from the Galaxy project in the cloud, particularly when 'on-demand', fast, and inexpensive resources are required. As with many other workflow servers (e.g., Taverna), there is no restriction on data file size, nor on the amount of storage space available for each user on Galaxy. There are however practical limitations for large file movement from one genome server to the next. Galaxy "history" tracks all analyses performed by a user and it is continually recorded, and never deleted (unless the user deletes the history). In a case where history has been deleted, records are retained for 60 days prior to permanent deletion from the main server. A disadvantage of Galaxy was illustrated by Hillman-Jackson and co-workers (2012). Here, the authors suggest that novice users may experience difficulty with creating and modifying workflows. Furthermore, libraries which need to be implemented in Galaxy to develop tool wrappers does require bioinformatics experience and as such, it is recommended that informative tools (such as instructional videos) are utilized by users to gain an understanding of Galaxy's features. Galaxy, as a web-based framework makes the analysis pipeline, tools and genomic data available to any biomedical researcher that has access to the internet. In contrast, Ruffus framework provides built-in features that supports and manage file naming as well as efficiently assist bioinformaticians to combine multiple bioinformatics tools together in an analysis chain. Ruffus framework uses standard Python syntax and decorators. As such, the SNP analysis pipeline as a series of Python scripts uses the Ruffus framework library for data extraction, manipulation, moving and transformation. The advantage of using Ruffus framework is having a consistent naming convention (i.e., input and output files) for the analysis pipeline. By using the Ruffus "@transform", the decorator enables us to specify the files moving through the SNP analysis pipeline. With Ruffus managing the SNP pipeline parameters, the following features were checked: 1) out-of-date parts of the pipeline were re-run 2) Multiple jobs were run in parallel (on different processors on the HPC cluster) 3) Pipeline stages were bound together automatically (i.e., apply the pipeline to more than 10 files at a time). A workflow framework specification requirement and comparative analysis summary based on customized analysis pipeline application tests and validations in this study, is represented in Table 6 and 7 below (Pabinger *et al.*, 2014; Leipzig, 2016; Ruffus, 2016).

**Table 6: Workflow Framework Summary**

The table shows Galaxy/Ruffus application differences.

| **Criteria** | **Galaxy** | **Ruffus** |
|---|---|---|
| Programming Language | Written in Python. | Written in Python. |
| Task Management | Pipeline task can be paused and restarted with the history refresh button. | Task cannot be stopped. Run from start to end. |
| DRM | Galaxy framework accommodates more than one engine e.g., Torque, Slurm, and DRMAA etc. | Only tested on SGE plus Ruffus drmaa wrapper for job submission. |
| Target Audience | Computational and Experimental Scientists. | Computational and Experimental Scientists. |
| Hardware | Windows, Mac, & Linux. | Linux Only. |

**Table 7: Comparative Feature Analysis Summary**

The table shows the analysis difference in the Galaxy and Ruffus features.

| **Criteria** | **Galaxy** | **Ruffus** |
|---|---|---|
| Analysis Pipeline | Good for beginners and advanced users. Possibility of integrating custom workflow solutions. | Complex to create (good for advance developers). Custom scripting workflow solution recommended with the use of Ruffus library (Python integration). |
| User Interface | Easy to use; Galaxy menus are clearer, designed to meets basic to advanced user expectations. | Complex to use (require large learning curve and extensive programming knowledge). Meets advanced user expectations. |
| Data Size | No restriction of data size of files. No limit to storage space. | No restriction of data size of files. No limit to storage space. |
| Accessibility | Doesn't support windows client download, offer as web-browser as service for all operating systems. Available on GitHub. | Ruffus framework application is strictly UNIX/Linux package, and is available as a pip or an easy-install. |
| Audit History | Tracks all analysis performed by user and is never deleted. | Tracks all analysis performed by user and is never deleted. |
| Information Managing | Custom generated workflows are shareable and can be published. | Not available. Only use shared memory for data/output share. |

## 4.8: Galaxy Implementation and deployment Pitfalls

In this project, the Galaxy framework does not support automation of the analysis pipeline and as such the workflow requires constant intervention where users have to restart the analysis pipeline. The Galaxy application programming interface (API) access requires expert knowledge of bioinformatics for installation, implementation, and deployment on HPC cluster for data-intensive analysis and may present a steep learning curve for novice users. More so, at the time writing this thesis, Galaxy does not specifically support SGE, but rather was design to support SLURM grid engine (i.e., a batch management system for jobs submission to the HPC cluster for data-intensive analysis)(Guimera, 2012; Reuther *et al.*, 2016). Other pitfalls we encountered include;

a) The host on which the Galaxy application server processes run can only be configured in the DRM as a submit host.
b) The use of Galaxy from basic to advanced developers must have a root or super permission for Galaxy API to write in the hosting virtual manager environment (i.e., /etc/passwd, LDAP).
c) The Galaxy virtual environment requires configuration of disabled shells like /bin/false in Debian/Ubuntu.
d) The Galaxy application server and worker nodes require the same version of Python
e) The Galaxy shared filesystem and absolute pathname also are limitations in this project since the project does not have full write permission, which however, delay the project software development process.
f) Host manager debugging and network latency limitation.

## 4.9: Ruffus Implementation and deployment Pitfalls

The pitfalls of implementing and deploying Ruffus on HPC cluster can be seen as the aspects related to the community support and understanding the workflow syntax and modules. Most Ruffus libraries are object-oriented decorator syntax which requires in-depth knowledge of Python programming language. The Ruffus framework does not provide customization of database but rather, provides support for only a single database called SQLite. There is no way to read and write directly from the database except through file configuration. Other pitfalls include extensive use of regular expression and wildcards for file matching (i.e., file naming convention), lack of file cleanup and preservation of history. In this project, we explicitly handle the restarting of failed jobs, hence, rebooting and/or restarting the entire pipeline again when tasks failed. Therefore, Ruffus library is case sensitive and are quick to both sematic and syntax errors when tasks encountered errors at any instance of ambiguity in analysis steps.

## 4.10: Benefit of Galaxy over Ruffus on HPC Cluster

In this project, the Galaxy framework rendered easy user interaction through the use of web browsers. Galaxy has a respectable community of users, developers and currently, several biomedical research labs have adopted this platform (Blankenberg *et al.*, 2010; Goecks *et al.*, 2010). On the other hand, Ruffus framework lacks the community supports and its uses in research or institute labs. Ruffus frameworks lacks a pipeline analysis progress bar and a way to query jobs that are being run on the HPC cluster. Another discrepancy in Ruffus is the lack of dynamic control (e.g., switching on/off the tasks, priorities changes, etc.) during the execution of the analysis pipeline. The use of Ruffus is regarded as running any other tool on the HPC cluster. Therefore, if the researchers have agreed with the term of service and have accepted responsibility and liability, the same rules apply to any other users in a cluster, willing to run any type of software. When running the analysis pipeline in Ruffus, there is a possibility to enable audit trails for logging the analysis pipeline history. This assist with controlling the pipeline bugs and the underlying method used by the HPC cluster facility. Consequently, Ruffus libraries have advantages, but do not offer an overall solution that allows a bioinformatics tool to be easily integrated in an analysis chain and run by biomedical researchers without programming experience.

## 4.11: Testing and Deployment of Workflow Frameworks

In this study, the framework efficiency and by extension, the possible relevance to the biomedical research field was tested. Test driven processes which included the analysis pipeline integration and implementation, and the logger, pipeline state, and tools integrated to run on the cluster, were reported. The benchmarking criteria which serve as building block for testing and deployment of the SNP analysis pipelines are explained and illustrated (Table 8). Scopes used to plan and assess the Galaxy and Ruffus framework include;

a) **Solvability**: analysis pipeline should have the ability to read data in a variety of different formats, and the support for data provenance, storage and file management systems that allowed the movement of both data input and output (Shannon *et al.*, 2006).
b) **Performance**: analysis pipeline should meet performance criterion. For instance, collectl-utilities evaluation on response time, runtime, and hardware usage (Furtaw, 2016).
c) **Scalability**: Analysis pipeline should be scalable. That is, evaluation based on jobs running on HPC cluster are scaled (Nishanth and Kihoon, 2015).
d) **Evolution**: Continuous software integration, usability of the analysis pipeline as well as community users and developers support.
e) **Reproducible**: Workflow framework should be able to reproduce previous analysis results when pipeline is re-running (Leipzig, 2016).
f) **Efficient**: Each step in the analysis pipeline should be as fast as possible, from data formatting, converting, and processing to discovery without interruption of any form. In other words, analysis pipeline should utilize the full HPC resources (Leipzig, 2016).
g) **Easy to use**: Researchers should be able to interact with the workflow framework easier in a spontaneous way (Blankenberg *et al.*, 2010).

**Table 8: Scope summary difference between the Galaxy and Ruffus framework**

| Benchmarks criteria | Galaxy | Ruffus |
|---|---|---|
| Solvability | Yes | Yes |
| Performance | Yes | Yes |
| Scalability | Yes | No |
| Evolution | Yes | No |
| Reproducible | Analysis pipeline can easily be reproducible. Suitable for beginners to advance user knowledge of programming. | Require good programing knowledge to reproduce pipeline. Not suitable for beginners. |
| Efficient | Very efficient for processing large dataset. Take long to process larger dataset. | Take long to process large dataset. Not as fast as expected. |
| Easy to use | Yes | Not for beginners. |

Table 8 summarizes the scope difference between the Galaxy framework and Ruffus libraries that aim to explain the problem of mining valuable scientific insights from large amounts of genomic data on next-generation sequencing experiments (i.e., DNA-seq experimentation).

## 4.12: Benchmarks Process

The following sections describe the benchmarking processes. In this project, we documented the CPU utilization profile as well as the performance of the base operating system (i.e., HPC Linux System) using the Collectl-Utility (White, 2016). During the execution of the analysis, the CPU, memory, disk, and network usage were measured at intervals of 30-seconds. The output result was then parsed and plotted using the Colplot, a simple web-based application tools for graph visualization (Awasthi *et al.*, 2015; Chhanga and Shukla, 2016) (Appendix F).

### 4.12.1: HPC Point of Reference

The basic points of reference evaluated included the number of cores (processors), disk, and memory. When researchers give reference to the size of a high-performance computing (HPC) cluster they are referring to how many processors or cores, it has. Each core needs memory attached with it, to provide a place for the processor to perform. The amount of memory on the cluster which was required for this study was driven by the requirements of the workflow frameworks and as such, had to be comprehensively benchmarked. Galaxy and Ruffus framework requirements varied with the variant calling pipeline. Each step in the SNP analysis pipeline step utilized memory that ranged from 1 gigabyte of RAM per core to 10 gigabytes (not processor) on the SANBI HPC cluster. The memory (RAM) on the cluster provided a temporary workspace for job execution on cluster core. Once the analysis jobs in both Galaxy and Ruffus (i.e., parallel jobs submission) on the HPC cluster were completed, the memory was permanently written on the HPC hard disks. The result of the analysis was then transferred to another shared storage on the HPC file system. Setting up the analysis pipeline architecture on HPC was a complex process, and as such, required the advice from the cluster administrators. Factors that influenced the SNP analysis pipelines included; 1) the number of memory (RAM) present on the cluster processors (CPU) to support each analysis step (one process or two processes at a time), 2) disk management requirement and 3) the framework application and research problem (Nishanth and Kihoon, 2015; Netto *et al.*, 2018).

### 4.12.2: Collectl-Utility in Practice (Parallel Benchmark)

The rationale for the use of collectl-utilities over other tools is due to its superior capabilities and usefulness for diagnosing or debugging (White, 2016). Collectl-Util/Colplot demonstrated how the Galaxy and Ruffus compute node operate by enabling the monitoring of components such as memory and CPU utilization (Kelly *et al.*, 2015). The Collectl-Util gathered the performance of the base operating system (UNIX/Linux OS), processors and the CPU utilization. The logged data from activities of the pipeline were then parsed and plotted. Figures 14, 15 and 16 are graphs plotted using the Colplot tool. From these graphs, we illustrate the manner in which Galaxy and Ruffus write to disk input/output (diskio), memory consumption and CPU utilization for total of 2 to 4 cores per analysis step, with CPU speed of 4527.066 MHz minimum for successive virtual cores (Kelly *et al.*, 2015). When we execute and test the SNP analysis pipeline, we captured the diskio metric and plotted the rate at which the compute node disk writes input and output. From the plot one can see that diskio is very high over the time.

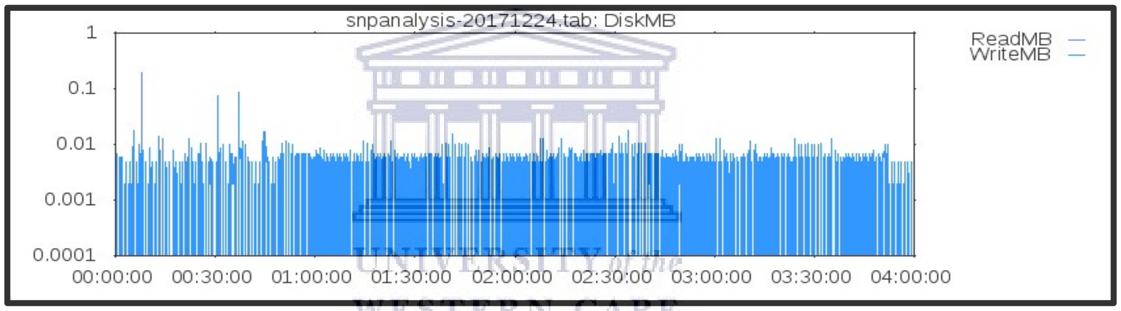


**Figure 14: Disk utilization summary.**

The plot shows the rate at which writing to disk increases over time. The more the input data supplied, the bigger the file generated.

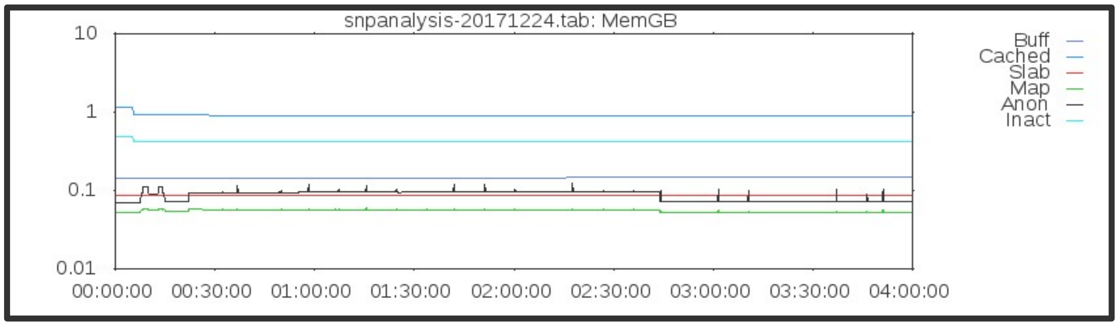


**Figure 15: Memory consumptions.**

The plot shows how the compute node memory was divided among the cached, buffered, dirty, and slab memory during SNP analysis pipelines execution on HPC cluster.

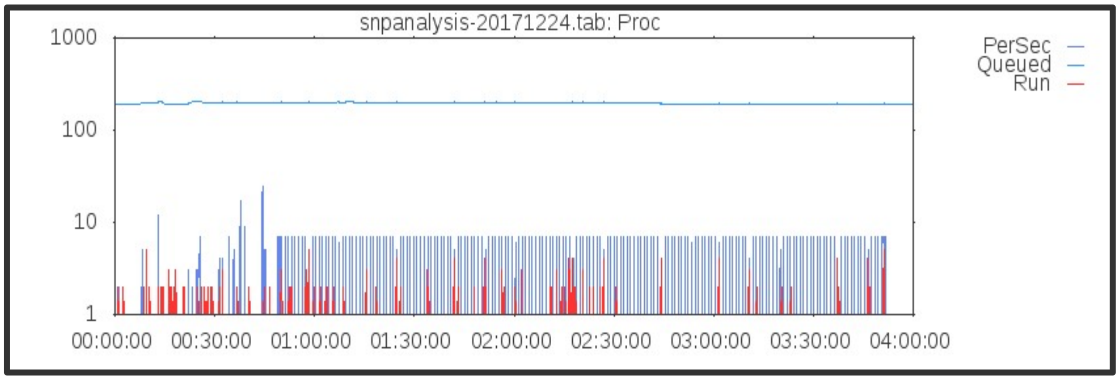


**Figure 16: CPU utilization.**

The plot shows the real time series of our compute node CPU utilization. Thus, both Galaxy and Ruffus framework CPU usage are monitored in both brief and detailed daemon mode.

### 4.12.3: Tools Runtime Measurement in Galaxy and Ruffus

The analysis pipeline total execution run time is the elapsed wall time from the initial start of the analysis steps (i.e., from the earliest start of Step 1 to the latest completion of Step 2 of the analysis on page 31) (Appendix E). To this end, time measurement for each SNP analysis pipeline (in the Galaxy and Ruffus framework) step is from the newest completion time of the earlier step to the latest completion time of the current step as described below (Figure 17) (Torri *et al.*, 2012; Spjuth *et al.*, 2015; Piras *et al.*, 2017).

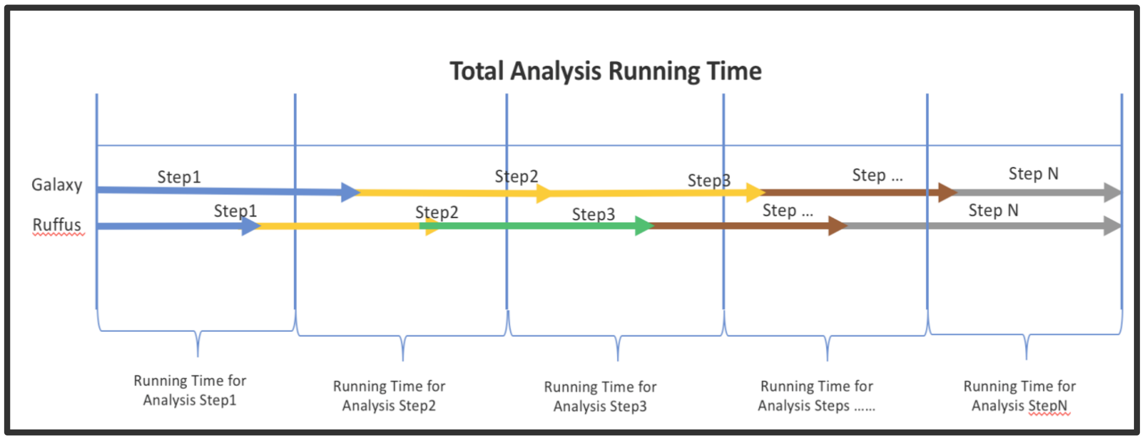
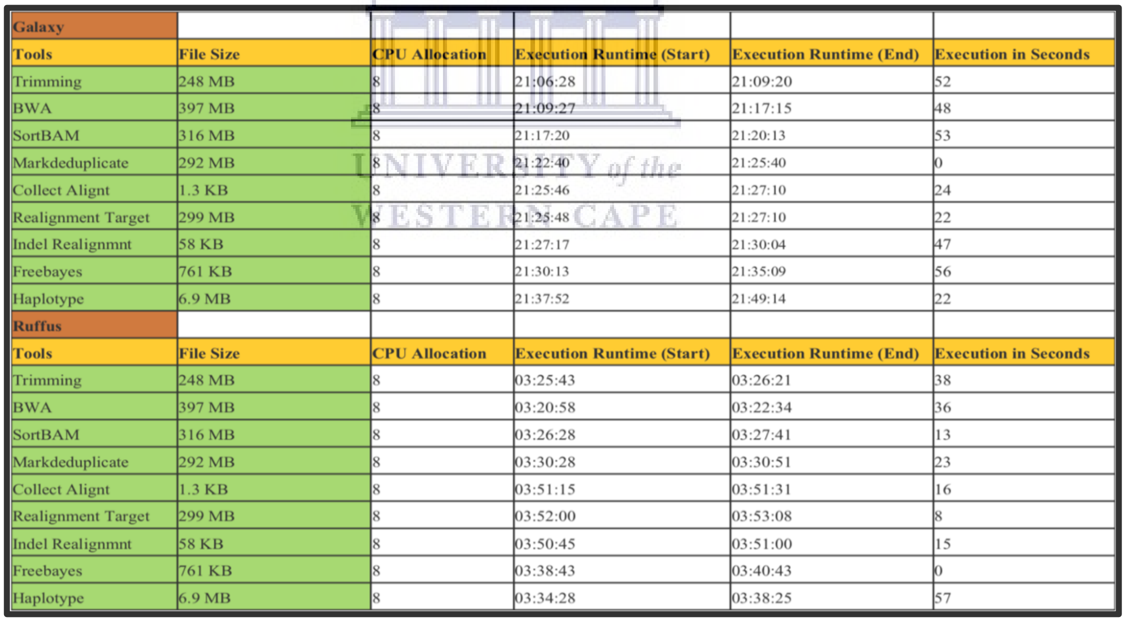

Total Analysis Running Time
Galaxy
Ruffus
Step1
Step2
Step3
Step ...
Step N
Step1
Step2
Step3
Step ...
Step N
Running Time for Analysis Step1
Running Time for Analysis Step2
Running Time for Analysis Step3
Running Time for Analysis Steps ......
Running Time for Analysis StepN

**Figure 17: Graph of Galaxy Real time against the Ruffus.**

The plot illustrates the total execution run time for each analysis pipeline step in the Galaxy and Ruffus framework.

**Table 9: A summary table showing the tools running time for each data set.**



| Galaxy | | | | | |
|---|---|---|---|---|---|
| **Tools** | **File Size** | **CPU Allocation** | **Execution Runtime (Start)** | **Execution Runtime (End)** | **Execution in Seconds** |
| Trimming | 248 MB | 8 | 21:06:28 | 21:09:20 | 52 |
| BWA | 397 MB | 8 | 21:09:27 | 21:17:15 | 48 |
| SortBAM | 316 MB | 8 | 21:17:20 | 21:20:13 | 53 |
| Markdeduplicate | 292 MB | 8 | 21:22:40 | 21:25:40 | 0 |
| Collect Alignt | 1.3 KB | 8 | 21:25:46 | 21:27:10 | 24 |
| Realignment Target | 299 MB | 8 | 21:25:48 | 21:27:10 | 22 |
| Indel Realignmnt | 58 KB | 8 | 21:27:17 | 21:30:04 | 47 |
| Freebayes | 761 KB | 8 | 21:30:13 | 21:35:09 | 56 |
| Haplotype | 6.9 MB | 8 | 21:37:52 | 21:49:14 | 22 |
| **Ruffus** | | | | | |
| **Tools** | **File Size** | **CPU Allocation** | **Execution Runtime (Start)** | **Execution Runtime (End)** | **Execution in Seconds** |
| Trimming | 248 MB | 8 | 03:25:43 | 03:26:21 | 38 |
| BWA | 397 MB | 8 | 03:20:58 | 03:22:34 | 36 |
| SortBAM | 316 MB | 8 | 03:26:28 | 03:27:41 | 13 |
| Markdeduplicate | 292 MB | 8 | 03:30:28 | 03:30:51 | 23 |
| Collect Alignt | 1.3 KB | 8 | 03:51:15 | 03:51:31 | 16 |
| Realignment Target | 299 MB | 8 | 03:52:00 | 03:53:08 | 8 |
| Indel Realignmnt | 58 KB | 8 | 03:50:45 | 03:51:00 | 15 |
| Freebayes | 761 KB | 8 | 03:38:43 | 03:40:43 | 0 |
| Haplotype | 6.9 MB | 8 | 03:34:28 | 03:38:25 | 57 |

In this study, the running time for each SNP analysis pipeline is summarized in Table 12. The data input size, size of DNA-sequence read files and the reference genomes are the most important factors affecting the analysis pipeline execution run time on the framework. For instance, during the mapping and alignment process/step on Galaxy and

Ruffus workflow framework, the reference genome size plays a major role while the GATK Haplotype caller step in the SNP analysis pipeline is affected by the size of the variant calls set.

### 4.12.4: Execution Time and Memory

The Galaxy/Ruffus framework computational time and the HPC resource manager (e.g., CPU, memory etc.) required to execute the SNP analysis pipeline are considered to be critical evaluators of efficiency. For example, if Galaxy/Ruffus framework's requirement for memory is high in the system requirement specification and the HPC resource (e.g., compute node available for implementing, testing and deploying the Galaxy and Ruffus) is low in memory for data intensive analysis, then the workflow framework will not be very useful. In addition, a workflow framework is not useful if it does not support multi-parallel for data processing and jobs handling and submission. Ideally, a workflow framework should be able to balance both CPU and memory usage during analysis pipeline execution. To this end, the computational response time of Galaxy and Ruffus during processing, analyzing and annotating MTB genomics data, was measured. Five runs were used to assess the impact of the analysis pipeline in Galaxy and Ruffus, to evaluate the response time, computational speed and memory usage. The response time is the total amount of time it takes Galaxy and Ruffus analysis pipelines to process and analyze the genomics data and is the sum of time it takes the cluster to respond to a request during the execution of the analysis pipeline on the computer node. The pipeline request includes; memory (RAM) request, number of cores (CPU), reading and writing to a disk input/output (diskio), data retrieval, and storing and database queries. Galaxy response time analysis pipeline execution includes loading a web browser that includes the HTTP GET and POST method (i.e., a process that enable communications between the client's computer and HPC servers), pipeline shared and executed with user-defined data and parameters (Blankenberg *et al.*, 2010; Hillman-Jackson *et al.*, 2012). On the other hand, Ruffus response time analysis pipeline execution includes building a flow chart of the pipeline tasks, beginning with the most recent ancestral pipeline task (i.e., with less dependencies) and calling each Python function in parallel to run multiple jobs (Ruffus, 2016). The response time for Galaxy analysis pipeline is compared to that of the Ruffus analysis pipeline as shown below in Figure 18 below.

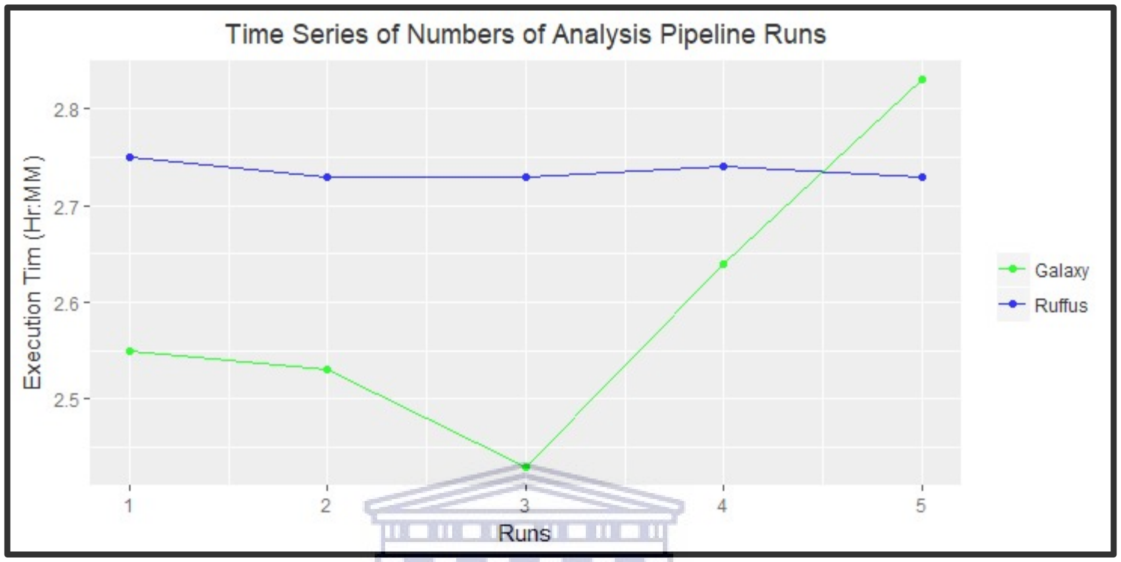
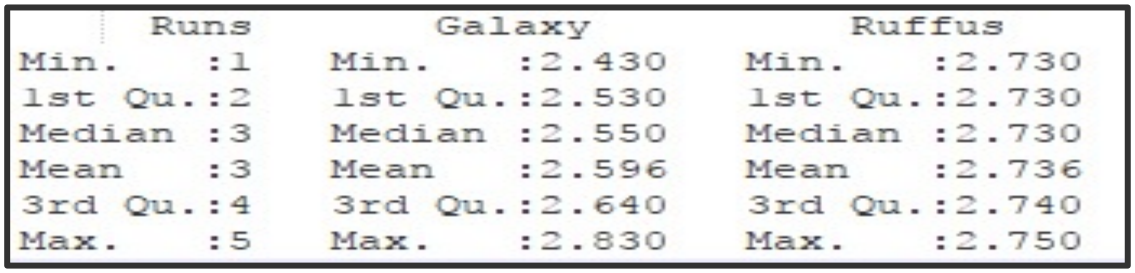


**Figure 18: Graph of Galaxy real time against the Ruffus analysis.**

The plot shows the response time of the pipeline analysis in the Galaxy and Ruffus framework (as discussed in section 4.7.4 above). The figure shows difference in execution run-time for each bioinformatics pipeline analysis.

From the graph, the Galaxy execution time (also referred to system real time) decreases over time per runs and superseded that of Ruffus runtime execution per run. Figure 19 shows the maximum (Max) coefficient and average (Mean) run time of Galaxy (as captured by the Colplot tool).



```
    Runs          Galaxy           Ruffus
Min.   :1   Min.   :2.430   Min.   :2.730
1st Qu.:2   1st Qu.:2.530   1st Qu.:2.730
Median :3   Median :2.550   Median :2.730
Mean   :3   Mean   :2.596   Mean   :2.736
3rd Qu.:4   3rd Qu.:2.640   3rd Qu.:2.740
Max.   :5   Max.   :2.830   Max.   :2.750
```

**Figure 19: Statistical Analysis Summary.**

The figure above shows the statistics summary of the execution time of the Galaxy and Ruffus analysis pipeline. The stats for Galaxy indicate more efficient values over that of Ruffus values.

Analysis in Galaxy framework utilized less time, on average, compared to the analysis pipeline in Ruffus framework. However, caution must be exercised, and it cannot be concluded that Galaxy out-performs Ruffus as both pipeline framework design is different in term of algorithm, concept, internal architecture and requirement specification for processing and analyzing genomic dataset, as well as for creating or composing analysis pipeline. Each framework has one or more limitations. As at the time writing this thesis, Galaxy was seen to be the preferred option by researchers to perform computational analysis.

### 4.12.5: Features Evaluation Matrix

We prioritized 7 criteria and a scale of 1 to 5 was used to summarize the Galaxy and Ruffus workflow frameworks and features during the framework deployment and testing (Table 10). An appropriate rule was set so that each rating criteria outline in the table below was used once. That is, the greater the importance or value of a criterion, the higher the value assigned. A "Very Good, Good, Poor, Fair, Not Applicable and Not Available" scale was used with numeric representation for each criterion (e.g., 0=Not Available, 1= Very Poor, 2 = Poor, 3 = Fair (or Available), 4 = Good, 5 = Very Good, N/A = Not Applicable). The evaluation matrix shows the ratings between the frameworks. Based on these features set, one can see that some workflow features and functionality scales well and fairly.  In the table, the active development status for

Galaxy framework was higher in scale (Very Good = 5) compare to Ruffus which was fair. This means that Galaxy framework have larger community support than Ruffus and are constantly changing in terms of software development. Ruffus, on the other hand, failed to meet active development status and are less supported. Extensibility criteria show that both frameworks can easily be extended. Installation and maintenance for both framework, each had scores of 4. Integration ease and usability for Galaxy had a significantly higher weighted score of 5 compare to Ruffus. This greater score indicates that the integration ease and usability for Galaxy fulfils more criteria that have been determined to be of greater importance. In the below table, the latter explanation applies. More so, since all features/functionality have been scored for all criteria, the individual feature score was summed by the appropriate criterion weighting. The total score was then calculated for Galaxy and Ruffus. The greater the score, the better the workflow frameworks satisfies the evaluation criteria. Our evaluation show that the two-workflow framework and features are different in design and usability. Based on these features, Galaxy is the preferred choice of workflow system that accommodate biomedical researchers with less programming knowledge. A summarize of their contribution to the workflow frameworks when building analysis pipeline and executing jobs can be seen in Table 9.

**Table 10: Evaluation Matrix**

| Evaluation Criteria (Features/Functionality) | Galaxy | Ruffus |
|---|---|---|
| Active Development Status | 5 | 3 |
| User base | 4 | 3 |
| Extensibility | 5 | 5 |
| Installation & Maintenance | 4 | 4 |
| Integration ease & Usability | 5 | 3 |
| Other features | 5 | 3 |
| Total Score | 28 | 21 |

## 4.13: Summary

In this chapter, a functional SNP analysis pipeline was built in the Galaxy and Ruffus frameworks to give an overview of biological data analysis. Each analysis pipeline was represented by a flowchart model. Each analysis pipeline executes and run jobs differently. For instance, the Galaxy used dynamic job configuration for configuring the pipeline execution, monitoring, and jobs scheduling. On the other hand, Ruffus uses a pipeline configuration file that interfaces with the Ruffus library for jobs runner configuration. We performed a performance evaluation of individual framework. The results show that in general, workflows tend to be CPU bound and memory intensive, and as such this study set up and utilized performance monitoring tools to assists in capturing the metrics and system logs. The logs were analyzed to determine system requirements or demonstrate the usefulness of the respective frameworks. The performance monitoring was an essential part of the process of system optimization, and if no testing was performed, pipeline framework bottlenecks would not have been identified. The Galaxy and Ruffus benchmark assessments were based on job submission and monitoring, parallelization of tasks, error logging and statistical summaries. Furthermore, SGE qccount and Collectl-Util/Colplot were used to create pipeline profiles which detailed information for file system temp space, diskio, memory and CPU utilization. In addition, the analysis pipeline response time and execution runtime of Galaxy was compared to that of Ruffus and allows us to identify the time when the analysis pipeline was ready to run and the time when the analysis finished its job. Logs collected gives more detailed information about the Galaxy and Ruffus framework.

# Chapter 5: Final Remarks

## 5.1: Conclusions

The use of bioinformatics workflows platforms has transformed biomedical research, by allowing a comprehensive analysis of NGS datasets. Choosing which computational workflow system to use to analyze the genomics data remains a challenge. Understanding bioinformatics workflow features can be helpful in addressing these challenges, providing a certain amount of computerization, and thus, enable advance more complex studies in the life sciences.

This thesis evaluated the theoretical and practical application of Galaxy and Ruffus workflow frameworks for annotation and analysis of MTB genomic datasets in an HPC facility. The Galaxy framework allows users with limited knowledge of bioinformatics and computational skills to set up and build an analysis pipeline. This thesis noted that Galaxy workflow execution and core requires Python and web programming language and tools to work. Galaxy project remains the preferred choice of workflow framework without biomedical researchers getting to know the major technical details of execution. In contrast, Ruffus requires intermediate, to advanced knowledge of Python programming language in order to use the framework library to carry out research in the genomics field. Ruffus workflow execution works well with environment module which handles the project details paths and the bioinformatics software package. The bioinformatics software tools used becomes explicit and were monitored with Linux Collectl Util and other job schedulers.

Furthermore, the use of the evaluation matrix in this study helps us to consider the most appropriate and feasible workflow features/functionality for questions identified in our aims and objectives. That is, the evaluation matrix provides an answer to the question and shows a reasonable comparison based on research finding, discussion and analysis in Chapter 3 and 4. The matrix table was systematically used to identify the workflow features by distinguishing the functionality between Galaxy and Ruffus. The workflow functionality selection of this matrix shows evaluation based on certain feature criteria for comparisons.

The workflow frameworks for building pipeline analysis requires adequate computing infrastructures and availabilities of resources to achieve satisfactory high performance and successful running of pipeline analysis from start to execution completion. Lack of computational resources and workflow frameworks logic and abstraction disrupt the building of pipeline analysis, causing a waste of time and money. Traditional local computing infrastructure and environment with limited resources are not well suited for building and running data-intensive analysis. Fully functional HPC or cloud computing is a very useful complement to the traditional local computing infrastructure and environment.

When building biological data analysis pipeline in Galaxy and Ruffus frameworks, we suggest that researchers ensure that the input genomic datasets are of high quality to facilitate the pipeline framework reliable for variant discovery and annotations. Furthermore, a higher storage capacity for further downstream analysis is recommended and will assist in faster, and more accurate variant detection and discovery.

To alleviate the workload of system administrators during the installation of a new update or developed bioinformatics software packages, tools sharing via GitHub has been set up at the organizational level. In this way, researchers, software developers and system managers can actively be contributing to the open source project and make it available to better a wider audience. Adding to the open source project is a great way to learn more about collaborative research on GitHub and as such, new genomics analysis pipelines are made available on the GitHub repository every day. In our particular case, we published the in-house tools and analysis pipeline which can be obtained using the URL; https://github.com/SANBI-SA. In doing so, the frameworks can be expanded and should be a consideration for future research. Our future work will examine the possible way of dockerizing the Galaxy and Ruffus frameworks to fit our development working environment.

## 5.2: Challenges and Limitations

During the development and implementation of the SNP analysis pipeline used in this study, computational challenges (such as high to low latency and workflow requirement inconsistency) were encountered. Furthermore, the following points describe the challenges encountered during the Galaxy/Ruffus frameworks implementation, deployment, as well as testing (i.e., the SNP analysis pipeline) on the HPC cluster:

a) Some sets of bioinformatics tools were problematic during the customization of the workflow frameworks in that they gave some programming syntax and semantics errors.
b) The benchmarking processes was not a straightforward one and often involved several iterative rounds to arrive at predictable and valuable conclusions.
c) Collecting the metric for the run time execution of the analysis pipeline in the Galaxy and Ruffus frameworks was not a straight forward process.
d) There existed a level of complexity in constant application debugging and pipeline profiling before capturing the performance of the workflow frameworks analysis.
e) Another complexity was that, Galaxy and Ruffus application utilized shared parallel filesystems on the HPC between their HPC compute nodes, and a head node that enable the submission of jobs to the HPC worker nodes (i.e., a multi-parallel interface (MPI) enabler). Hence, capturing and interpreting the workflows performance was a challenging exercise as there were other HPC applications utilizing the system resources. Other challenges were encountered during the process of integrating the analysis pipeline due to the limited administrator rights to the HPC facility.

f) Due to the requirements to satisfy best practices, the evaluation of Galaxy and Ruffus to other workflow systems was a rigorous process, and such, the evaluation was time consuming.
g) Other challenges encountered in this study included considerations with respect to the differences in Galaxy and Ruffus workflow features as well as the requirement for setting up the workflow platform. The Galaxy frameworks hardware installation requirements were completely different from that of the Ruffus requirements since each framework has different parameters and operates differently.
h) Both frameworks consist of multiple sub-layers of tasks that are not visible to end users and as such, require good programming knowledge to prevent unstructured objects, syntax and semantics errors when coding. In addition, when executing the analysis pipeline, some functions affect the system setup environment and files, and on occasion, may lead to system instability and breakdown.
i) Managing higher workflow layers such as workflow execution and management was not a completely solved problem.

## 5.3: Recommendations

In this thesis, we have evaluated the Galaxy and Ruffus framework by building biological data analysis pipelines using different bioinformatics tools and strategies to benchmark the frameworks. We have shown how different workflow features and functionalities impacted the frameworks, and the resource bottlenecks at runtime. To properly manage and decide which framework to use when a biomedical researcher try to build bioinformatics pipelines to carry-out omics analyses, this thesis recommends that an intuitive, a code-free workflow feature is needed to better understand the utilization of Galaxy and Ruffus framework. Furthermore, to understand the underlying infrastructure technologies and workflow abstractions, we suggest the implementation and configuration of Galaxy and Ruffus in Singularity environment. Singularity packages the workflow systems, the required dependencies and bioinformatics tools in a single Docker container. In so doing, it will assist biomedical researchers to have full control in pre-configured and ready to use workflow environment. In addition, it will reduce the turn-around time for installing and configuring bioinformatics software packages. Furthermore, emergence of newly developed workflow features may make it easier for a novice bioinformatics analyst to understand and acquire practical bioinformatics knowledge, thereby increasing a pool of expertise to further expand the field.

## 5.4: Future works

In the future, we intend to enhance our workflow features such as the jobs monitoring tools following good coding practices in Galaxy and Ruffus framework. The enhancement will allow a biomedical researcher to visualize workflow processes and then understand the automation of bioinformatics pipelines. Furthermore, we plan to enhance the Galaxy and Ruffus application programming interface (API) tools for seamless conditional execution of tasks. In doing so, it will help the biomedical researcher to stop the execution of a pipeline analysis and resume it later. We intend to enhance the integration of Galaxy and Ruffus for balance and performance (e.g., jobs submission and execution runtime) through tighter system level-integration, while maintaining workflow portability. Furthermore, we plan on upgrading our high-performance computing environment such as compute nodes, memory, and the bioinformatics tools, to allow us to overcome the barriers pertaining to workloads and deployments on cloud-based systems. We plan to make available the customized SNP analysis pipeline in the Galaxy and Ruffus to have its own Docker container that can be deploy on HPC or cloud-based system. Furthermore, the method utilized to customize the SNP analysis pipeline will be further expand for whole genome data analysis together with using the latest or newly created bioinformatics software tools. Other plans including; to explore other workflow management system (such as integration of bcbionextgen with Common Workflow Language (CWL) framework) and comparing their features to Galaxy and Ruffus framework. We also plan on creating a learning platform for novice biomedical researchers to learn Galaxy and Ruffus workflow engine and pipeline development.

# References

Abouelhoda, M., Issa, S. A. & Ghanem, M. 2012. Tavaxy: Integrating Taverna and Galaxy workflows with cloud computing support. *BMC Bioinformatics,* 13**,** 1.

Afgan, E., Baker, D., Coraor, N. *et al.* 2010. Galaxy CloudMan: delivering cloud compute clusters. *BMC Bioinformatics,* 11**,** S4.

Afgan, E., Baker, D., Nekrutenko, A. & Taylor, J. 2012. A reference model for deploying applications in virtualized environments. *Concurrency and Computation: Practice and Experience,* 24**,** 1349-1361.

Aldred, L. J. 2011. *Fundamentals of process integration.*

Altintas, I., Berkley, C., Jaeger, E. *et al.* Kepler: an extensible system for design and execution of scientific workflows. Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on, 2004. IEEE, 423-424.

Altintas, I., Wang, J., Crawl, D. & Li, W. Challenges and approaches for distributed workflow-driven analysis of large-scale biological data: vision paper. Proceedings of the 2012 Joint EDBT/ICDT Workshops, 2012. ACM, 73-78.

Alyssa, H. 2016. *High Performance Computing Cluster in a Cloud Environment* [Online]. Available: https://support.rackspace.com/whitepapers/ [Accessed].

Anderson, M. W. & Schrijver, I. 2010. Next generation DNA sequencing and the future of genomic medicine. *Genes (Basel),* 1**,** 38-69.

Armbrust, M., Fox, A., Griffith, R. *et al.* 2010. A view of cloud computing. *Communications of the ACM,* 53**,** 50-58.

Arvados. 2016. *Arvados | Open Source Big Data Processing and Bioinformatics* [Online]. Available: https://arvados.org/ [Accessed 14 June, 2016 2016].

Awasthi, M., Suri, T., Guz, Z. *et al.* System-level characterization of datacenter applications. Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering, 2015. ACM, 27-38.

Bao, R., Huang, L., Andrade, J. *et al.* 2014. Review of current methods, applications, and data management for the bioinformatics analysis of whole exome sequencing. *Cancer Inform.***,** 67-83.

Bartlett, J. C. & Toms, E. G. 2005. Developing a protocol for bioinformatics analysis: An integrated information behavior and task analysis approach. *Journal of the Association for Information Science and Technology,* 56**,** 469-482.

Bhagwanani, S. 2005. An evaluation of end-user interfaces of scientific workflow management systems.

Bhardwaj, S., Jain, L. & Jain, S. 2010. Cloud computing: A study of infrastructure as a service (IAAS). *International Journal of engineering and information Technology,* 2**,** 60-63.

Bianchi, V., Ceol, A., Ogier, A. G. *et al.* 2016. Integrated Systems for NGS Data Management and Analysis: Open Issues and Available Solutions. *Front Genet,* 7**,** 75.

Biostars. 2010. *How To Organize A Pipeline Of Small Scripts Together?* [Online]. Available: https://www.biostars.org/p/79/ [Accessed 14th March 2018].

Biostars. 2015. *Workflow management software for pipeline development in NGS* [Online]. Biostars. Available: https://www.biostars.org/p/115745/ [Accessed 14th March 2018].

Blankenberg, D., Taylor, J., Nekrutenko, A. & Team, G. 2011. Making whole genome multiple alignments usable for biologists. *Bioinformatics,* 27**,** 2426-2428.

Blankenberg, D., Von Kuster, G., Coraor, N. *et al.* 2010. Galaxy: a web-based genome analysis tool for experimentalists. *Curr. Protoc. Mol. Biol.,* Chapter 19**,** Unit 19 10 1-21.

Blischak, J. D., Davenport, E. R. & Wilson, G. 2016. A Quick Introduction to Version Control with Git and GitHub. *PLoS Comput. Biol.,* 12**,** e1004668.

Booth., G. 2013. *Open Grid Scheduler/Grid Engine* [Online]. 2012. Available: http://gridscheduler.sourceforge.net [Accessed June 6th 2017].

Bretaudeau, A., Monjeaud, C., Le Bras, Y., Legeai, F. & Collin, O. 2015. BioMAJ2Galaxy: automatic update of reference data in Galaxy using BioMAJ. *GigaScience,* 4**,** 22.

Brian, W. & Dustin, M. J. 2009. *Buildbot: The Continuous Integration Framework* [Online]. Available: http://buildbot.net/index.html#basics [Accessed 9th September 2017].

Brown, D. K., Musyoka, T. M., Penkler, D. L. & Bishop, Ö. T. 2015. JMS: A workflow management system and web-based cluster front-end for the Torque resource manager. *arXiv preprint arXiv:1501.06907*.

Calabrese, B. 2018. Cloud-Based Bioinformatics Platforms. *Reference Module in Life Sciences.* Elsevier.

Chhanga, D. & Shukla, X. 2016. FOSSICK: An Implementation of Federated Search Engine. *International Journal Of Computer Science Engineering And Information Technology Research (IJCSEITR),* 6**,** 69-78.

Chine, K. 2010. Open science in the cloud: towards a universal platform for scientific and statistical computing. *Handbook of cloud computing.* Springer.

Cohen, K. A., Abeel, T., Manson Mcguire, A. *et al.* 2015. Evolution of Extensively Drug-Resistant Tuberculosis over Four Decades: Whole Genome Sequencing and Dating Analysis of Mycobacterium tuberculosis Isolates from KwaZulu-Natal. *PLoS Med.,* 12**,** e1001880.

Curcin, V. & Ghanem, M. Scientific workflow systems-can one size fit all? 2008 Cairo International Biomedical Engineering Conference, 2008. IEEE, 1-9.

D'antonio, M., De Meo, P. D. O., Paoletti, D. *et al.* 2013. WEP: a high-performance analysis pipeline for whole-exome data. *BMC Bioinformatics,* 14**,** S11.

Dean, J. & Ghemawat, S. 2008. MapReduce: simplified data processing on large clusters. *Communications of the ACM,* 51**,** 107-113.

Deelman, E. 2010. Grids and clouds: Making workflow applications work in heterogeneous distributed environments. *The International Journal of High Performance Computing Applications,* 24**,** 284-298.

Deelman, E., Gannon, D., Shields, M. & Taylor, I. 2009. Workflows and e-Science: An overview of workflow system features and capabilities. *Future Generation Computer Systems,* 25**,** 528-540.

Depristo, M. A., Banks, E., Poplin, R. *et al.* 2011. A framework for variation discovery and genotyping using next-generation DNA sequencing data. *Nat. Genet.,* 43**,** 491-8.

Di Tommaso, P., Chatzou, M., Floden, E. W. *et al.* 2017. Nextflow enables reproducible computational workflows. *Nat. Biotechnol.,* 35**,** 316-319.

Doctorow, C. 2008. Big data: welcome to the petacentre. *Nature News,* 455**,** 16-21.

Ellson, J., Gansner, E., Koutsofios, L., North, S. C. & Woodhull, G. Graphviz—open source graph drawing tools. International Symposium on Graph Drawing, 2001. Springer, 483-484.

Emeakaroha, V. C., Maurer, M., Stern, P. *et al.* 2013. Managing and optimizing bioinformatics workflows for data analysis in clouds. *Journal of grid computing,* 11**,** 407-428.

Fisch, K. M., Meißner, T., Gioia, L. *et al.* 2015. Omics Pipe: a community-based framework for reproducible multi-omics data analysis. *Bioinformatics***,** btv061.

Foundation, P. S. 2016. *The Python Programming Languages* [Online]. Available: https://www.python.org/ [Accessed 14 June, 2016].

Furtaw, B. 2016. High performance data analytics in precision medicine using scale-up and hybrid supercomputing solutions.

Fusaro, V. A., Patil, P., Gafni, E., Wall, D. P. & Tonellato, P. J. 2011. Biomedical cloud computing with amazon web services. *PLoS Comput. Biol.,* 7**,** e1002147.

Garfinkel, S. 2007. An evaluation of amazon's grid computing services: EC2, S3, and SQS.

Glenn, T. C. 2011. Field guide to next-generation DNA sequencers. *Mol. Ecol. Resour.,* 11**,** 759-769.

Goble, C. & De Roure, D. 2009. The impact of workflow tools on data-centric research.

Goble, C. & Stevens, R. 2008. State of the nation in data integration for bioinformatics. *J. Biomed. Inf.,* 41**,** 687-693.

Goecks, J., Nekrutenko, A. & Taylor, J. 2010. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biol.,* 11**,** R86.

Goesmann, A., Linke, B., Rupp, O. *et al.* 2003. Building a BRIDGE for the integration of heterogeneous data from functional genomics into a platform for systems biology. *J. Biotechnol.,* 106**,** 157-167.

Gorelick, M. & Ozsvald, I. 2014. *High Performance Python: Practical Performant Programming for Humans*, " O'Reilly Media, Inc.".

Gray, J., Moore, K. T. & Naylor, B. A. OpenMDAO: An open source framework for multidisciplinary analysis and optimization. AIAA/ISSMO Multidisciplinary Analysis Optimization Conference Proceedings, 2010.

Guimera, R. V. 2012. *Enabling Automatic Data Analysis in Bioinformatics Core Facilities.*

Hale, K. S. & Stanney, K. M. 2014. *Handbook of virtual environments: Design, implementation, and applications*, CRC Press.

Heller, B., Marschner, E., Rosenfeld, E. & Heer, J. Visualizing collaboration and influence in the open-source software community. Proceedings of the 8th working conference on mining software repositories, 2011. ACM, 223-226.

Hillman-Jackson, J., Clements, D., Blankenberg, D. *et al.* 2012. Using galaxy to perform large-scale interactive data analyses. *Current protocols in bioinformatics*, 10.5. 1-10.5. 47.

Hinchcliffe, M., Le, H., Fimmel, A. *et al.* 2014. Diagnostic validation of a familial hypercholesterolaemia cohort provides a model for using targeted next generation DNA sequencing in the clinical setting. *Pathology,* 46**,** 60-8.

Huang, W., Liu, J., Abali, B. & Panda, D. K. A case for high performance computing with virtual machines. Proceedings of the 20th annual international conference on Supercomputing, 2006. ACM, 125-134.

Ison, J., Rapacki, K., Menager, H. *et al.* 2015. Tools and data services registry: a community effort to document bioinformatics resources. *Nucleic Acids Res.*

Jackson, K. R., Ramakrishnan, L., Muriki, K. *et al.* Performance analysis of high performance computing applications on the amazon web services cloud. Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on, 2010. IEEE, 159-168.

Jamalian, S. & Rajaei, H. ASETS: A SDN Empowered Task Scheduling System for HPCaaS on the Cloud. Cloud Engineering (IC2E), 2015 IEEE International Conference on, 2015. IEEE, 329-334.

Kang, M. H., Froscher, J. N., Sheth, A. P., Kochut, K. J. & Miller, J. A. A multilevel secure workflow management system. Advanced Information Systems Engineering, 1999. Springer, 271-285.

Kanwal, S., Khan, F. Z., Lonie, A. & Sinnott, R. O. 2017. Investigating reproducibility and tracking provenance–A genomic workflow case study. *BMC Bioinformatics,* 18**,** 337.

Kelly, B. J., Fitch, J. R., Hu, Y. *et al.* 2015. Churchill: an ultra-fast, deterministic, highly scalable and balanced parallelization strategy for the discovery of human genetic variation in clinical and population-scale genomics. *Genome Biol.,* 16**,** 6.

Kircher, M. & Kelso, J. 2010. High-throughput DNA sequencing–concepts and limitations. *Bioessays,* 32**,** 524-536.

Kodama, Y., Shumway, M. & Leinonen, R. 2012. The Sequence Read Archive: explosive growth of sequencing data. *Nucleic Acids Res.,* 40**,** D54-D56.

Korpelainen, E., Tuimala, J., Somervuo, P., Huss, M. & Wong, G. 2014. *RNA-seq Data Analysis: A Practical Approach*, CRC Press.

Koster, J. & Rahmann, S. 2012. Snakemake--a scalable bioinformatics workflow engine. *Bioinformatics,* 28**,** 2520-2.

Kurs, J. P., Simi, M. & Campagne, F. 2016. NextflowWorkbench: Reproducible and Reusable Workflows for Beginners and Experts. *bioRxiv*.

Lamprecht, A.-L. 2013. *User-Level Workflow Design: A Bioinformatics Perspective*, Springer.

Layton, J. 2017. *Monitor Your Nodes with collectl* [Online]. Available: http://www.admin-magazine.com/index.php/HPC/Articles/Monitor-Your-Nodes-with-collectl [Accessed 23rd October 2017].

Leading, D. C. S. 2016. it@ intel.

Lefkowitz, H. M. 2000. Graphical user interface. Google Patents.

Leipzig, J. 2016. A review of bioinformatic pipeline frameworks. *Brief Bioinform*.

Li, Y. & Chen, L. 2014. Big biological data: challenges and opportunities. *Genomics, proteomics & bioinformatics,* 12**,** 187-189.

Liu, B., Madduri, R. K., Sotomayor, B. *et al.* 2014. Cloud-based bioinformatics workflow platform for large-scale next-generation sequencing analyses. *J Biomed Inform,* 49**,** 119-33.

Liu, J. 20 Years of teaching parallel processing to computer science seniors. Proceedings of the Workshop on Education for High Performance Computing, 2016. IEEE Press, 7-13.

Loman, N. J., Misra, R. V., Dallman, T. J. *et al.* 2012. Performance comparison of benchtop high-throughput sequencing platforms. *Nat. Biotechnol.,* 30**,** 434-439.

Luna, D., Mayan, J., García, M., Almerares, A. & Househ, M. 2014. Challenges and potential solutions for big data implementations in developing countries. *Yearb. Med. Inform.,* 9**,** 36.

Marathe, A., Harris, R., Lowenthal, D. K. *et al.* A comparative study of high-performance computing on the cloud. Proceedings of the 22nd international symposium on High-performance parallel and distributed computing, 2013. ACM, 239-250.

Mcgough, A. S., Afzal, A., Darlington, J. *et al.* 2005. Making the grid predictable through reservations and performance modelling. *The Computer Journal,* 48**,** 358-368.

Mckenna, A., Hanna, M., Banks, E. *et al.* 2010. The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Res.,* 20**,** 1297-303.

Metzker, M. L. 2010. Sequencing technologies—the next generation. *Nature reviews genetics,* 11**,** 31-46.

Michael, M. & William, J. F. 2014. *Patent Application Publication* [Online]. San Francisco, CA (US) United States. Available: https://patents.google.com/patent/US20140136968 [Accessed 15th March 2018].

Möller, S., Prescott, S. W., Wirzenius, L. *et al.* 2017. Robust cross-platform workflows: how technical and scientific communities collaborate to develop, test and share best practices for data analysis. *Data Science and Engineering,* 2**,** 232-244.

Nagalakshmi, U., Waern, K. & Snyder, M. 2010. RNA-Seq: a method for comprehensive transcriptome analysis. *Curr. Protoc. Mol. Biol.,* Chapter 4**,** Unit 4 11 1-13.

Neron, B., Menager, H., Maufrais, C. *et al.* 2009. Mobyle: a new full web bioinformatics framework. *Bioinformatics,* 25**,** 3005-11.

Netto, M. A., Calheiros, R. N., Rodrigues, E. R., Cunha, R. L. & Buyya, R. 2018. HPC Cloud for Scientific and Business Applications: Taxonomy, Vision, and Research Challenges. *ACM Computing Surveys (CSUR),* 51**,** 8.

Nishanth, D. & Kihoon, Y. 2015. Dell HPC System for Genomics v2.0. *Eng. J.*

Nocq, J., Celton, M., Gendron, P., Lemieux, S. & Wilhelm, B. T. 2013. Harnessing virtual machines to simplify next-generation DNA sequencing analysis. *Bioinformatics,* 29**,** 2075-83.

Nyrönen, T. H., Laitinen, J., Tourunen, O. *et al.* Delivering ICT infrastructure for biomedical research. Proceedings of the WICSA/ECSA 2012 Companion Volume, 2012. ACM, 37-44.

O'sullivan, B. 2009. Making sense of revision-control systems. *Communications of the ACM,* 52**,** 56-62.

O'driscoll, A., Daugelaite, J. & Sleator, R. D. 2013. 'Big data', Hadoop and cloud computing in genomics. *J. Biomed. Inf.,* 46**,** 774-781.

Oinn, T., Addis, M., Ferris, J. *et al.* 2004. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics,* 20**,** 3045-54.

Oracle. 2017. *Solaris Advanced User's Guide* [Online]. Available: https://docs.oracle.com/cd/E19683-01/806-7612/startup-78447/index.html [Accessed 5th May 2017].

Pabinger, S., Dander, A., Fischer, M. *et al.* 2014. A survey of tools for variant analysis of next-generation genome sequencing data. *Brief Bioinform,* 15**,** 256-78.

Pepke, S., Wold, B. & Mortazavi, A. 2009. Computation for ChIP-seq and RNA-seq studies. *Nat. Methods,* 6**,** S22-32.

Perl. 2016. *The Perl Programming Language - www.perl.org* [Online]. Perl.org. Available: https://www.perl.org/ [Accessed 14 June, 2016 2016].

Piras, M. E., Pireddu, L. & Zanetti, G. 2017. wft4galaxy: a workflow testing tool for galaxy. *Bioinformatics,* 33**,** 3805-3807.

Prajapati, H. B. & Shah, V. A. Scheduling in grid computing environment. Advanced Computing & Communication Technologies (ACCT), 2014 Fourth International Conference on, 2014. IEEE, 315-324.

Project, G. 2016. *The Galaxy Project: Online bioinformatics analysis for everyone* [Online]. Available: https://galaxyproject.org/ [Accessed 14 June, 2016 2016].

Raman, K., Yeturu, K. & Chandra, N. 2008. targetTB: a target identification pipeline for Mycobacterium tuberculosis through an interactome, reactome and genome-scale structural analysis. *BMC Syst. Biol.,* 2**,** 1.

Reuther, A., Byun, C., Arcand, W. *et al.* Scheduler technologies in support of high performance data analysis. High Performance Extreme Computing Conference (HPEC), 2016 IEEE, 2016. IEEE, 1-6.

Romano, P. 2008. Automation of in-silico data analysis processes through workflow management systems. *Briefings in Bioinformatics,* 9**,** 57-68.

Rother, K., Potrzebowski, W., Puton, T. *et al.* 2011. A toolbox for developing bioinformatics software. *Briefings in bioinformatics,* 13**,** 244-257.

Ruffus. 2016. *Ruffus — ruffus 2.6.3 documentation* [Online]. Ruffus — ruffus 2.6.3 documentation. Available: http://www.ruffus.org.uk/ [Accessed 14 June, 2016 2016].

Sanner, M. F. 1999. Python: a programming language for software integration and development. *J. Mol. Graph. Model.,* 17**,** 57-61.

Santana-Perez, I. & Pérez-Hernández, M. S. 2015. Towards reproducibility in scientific workflows: An infrastructure-based approach. *Scientific Programming,* 2015.

Schall, D. 2015. *Social network-based recommender systems*, Springer.

Schindelin, J., Arganda-Carreras, I., Frise, E. *et al.* 2012. Fiji: an open-source platform for biological-image analysis. *Nat. Methods,* 9**,** 676-682.

Schulz, W. L., Durant, T. J., Siddon, A. J. & Torres, R. 2016. Use of application containers and workflows for genomic data analysis. *J. Pathol. Inform.,* 7.

Shannon, P. T., Reiss, D. J., Bonneau, R. & Baliga, N. S. 2006. The Gaggle: an open-source software system for integrating bioinformatics software and data sources. *BMC Bioinformatics,* 7**,** 176.

Sinclair, L. 2010. Development Of An Interactive Genome Browser To Visualize And Analyse Large Scale Genomic Data.

Spjuth, O., Bongcam-Rudloff, E., Hernandez, G. C. *et al.* 2015. Experiences with workflows for automating data-intensive bioinformatics. *Biol. Direct,* 10**,** 43.

Stein, L. 1996. How Perl saved the human genome project. *Dr Dobb's Journal (July 2001)*.

Stein, L. D. 2010. The case for cloud computing in genome informatics. *Genome Biol.,* 11**,** 1.

Stephens, Z. D., Lee, S. Y., Faghri, F. *et al.* 2015. Big Data: Astronomical or Genomical? *PLoS Biol.,* 13**,** e1002195.

Stevens, W. R. & Rago, S. A. 2013. *Advanced programming in the UNIX environment*, Addison-Wesley.

Sun, M. 2007. *Sun N1 Grid Engine 6.1 Administration Guide* [Online]. USA: Oracle. Available: https://docs.oracle.com/cd/E19957-01/820-0698/book-info/index.html [Accessed June 6th 2017].

Taura, K., Matsuzaki, T., Miwa, M. *et al.* 2013. Design and implementation of GXP make - A workflow system based on make. *Future Gener. Comput. Syst.,* 29**,** 662-672.

Tolvanen, J.-P. & Kelly, S. 2008. Domain-Specific Modeling: Enabling Full Code Generation. *Wiley-IEEE Computer Society,* 444**,** 231.

Torri, F., Dinov, I. D., Zamanyan, A. *et al.* 2012. Next generation sequence analysis and computational genomics using graphical pipeline workflows. *Genes (Basel),* 3**,** 545-75.

Truong, H.-L. & Dustdar, S. 2011. Cloud computing for small research groups in computational science and engineering: current status and outlook. *Computing,* 91**,** 75-91.

Van Der Aalst, W. & Van Hee, K. M. 2004. *Workflow management: models, methods, and systems*, MIT press.

Van Der Aalst, W. M. & Ter Hofstede, A. H. 2005. YAWL: yet another workflow language. *Information systems,* 30**,** 245-275.

Van Der Auwera, G. A., Carneiro, M. O., Hartl, C. *et al.* 2013. From FastQ data to high confidence variant calls: the Genome Analysis Toolkit best practices pipeline. *Curr Protoc Bioinformatics,* 43**,** 11 10 1-33.

Van Deventer, C. 2014. *Expressed sequence tag clustering using commercial gaming hardware.* University of Johannesburg.

Van Heusden, P., Yi, L. & Christoffels, A. An OpenNebula-based cloud computing environment for bioinformatics. 2012. SATNAC.

Vince, B. 2015. Remedial Unix Shell. *In:* JOLLYMORE, C. N. A. A. (ed.) *Bioinformatics Data Skills.* First Edition ed. 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reilly Media.

Wang, Y., Mehta, G., Mayani, R. *et al.* 2011. RseqFlow: workflows for RNA-Seq data analysis. *Bioinformatics,* 27**,** 2598-600.

White, P. 2016. Peer Review Publication: GenomeNext's NGS Analysis Engine *Per. Med.*

Wilke, A., Rückert, C., Bartels, D. *et al.* 2003. Bioinformatics support for high-throughput proteomics. *J. Biotechnol.,* 106**,** 147-156.

Williams, A. G., Thomas, S., Wyman, S. K. & Holloway, A. K. 2014. RNA-seq Data: Challenges in and Recommendations for Experimental Design and Analysis. *Curr Protoc Hum Genet,* 83**,** 11 13 1-20.

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S. & Stoica, I. 2010. Spark: Cluster computing with working sets. *HotCloud,* 10**,** 95.

Zhang, Q., Cheng, L. & Boutaba, R. 2010. Algorithms and architectures for parallel processing. *J. Int. Serv. Appl,* 1**,** 7-18.

Zhao, Z., Belloum, A., Wibisono, A. *et al.* Scientific workflow management: between generality and applicability. Quality Software, 2005.(QSIC 2005). Fifth International Conference on, 2005. IEEE, 357-364.

Zou, Q., Li, X.-B., Jiang, W.-R. *et al.* 2013. Survey of MapReduce frame operation in bioinformatics. *Briefings in bioinformatics,* 15**,** 637-647.

# Appendix A

## Pipeline Framework Configuration for Variant Calling Pipeline

The SNP analysis pipeline was designed for 100 base pair or greater Illumina short read MTB sequence data with Illumina 1.9 quality encoding and uses Illumina naming convention. The SNP Analysis pipeline is based on the Galaxy and Ruffus framework for building pipelines (Figure 22). The Python libraries allow the integration of several bioinformatics tools and its dependencies. The project source code is made available on public domain (i.e., open source platform) hosted on GitHub.

Pipeline features include:

- Job submission on a cluster using DRMAA (currently only tested with SLURM).
- Job dependency calculation and check pointing.
- Pipeline can be displayed as a flowchart.
- Re-running a pipeline will start from the most up-to-date stage. It will not redo previously completed tasks.

### License

3 Clause BSD License. See LICENSE.txt in source repository.

## Installation: External dependencies

SNP Analysis depends on the following programs and libraries:

- **Python** (version 2.7.5), Galaxy and Ruffus and pyYaML

- **DRMAA** for submitting jobs to the cluster. The pipeline uses libdrama.so by running Python-drmaa for either local or cluster job submission system.
- **BWA** for aligning reads to the reference genome (version 0.7.10)
- **NovoCraft**
- **GATK** Genome Analysis Toolkit (version 3.3-0)
- **SAMTOOLS** (version 0.1.2)
- **PICARD** (version 1.127)
- **FASTQC** version 0.10.1 (http://www.bioinformatics.babraham.ac.uk/projects/fastqc/)
- **VCFTOOLS** (http://vcftools.sourceforge.net/)
- **VIRTUAL** version 15.0.1 (2016-03-17))

## Input Data Source

Genomic Dataset used in this project were from the Tygerberg Hospital Group. 10 paired end (PE) MTB samples data were used as input datasets for the pipeline. E.g.;

- H37Rv1116_R1.fastq.gz
- H37Rv1116_R2.fastq.gz

## Reference Genome (MTB):

- human_g1k_v37_decoy. fasta

# Appendix B

## Ruffus Framework Implementation Steps

This section describes how we installed and configured the Ruffus framework in HPC Python virtual environment. We used the virtual Python environment to implement the SNP data analysis pipeline using our GitHub repository; the following steps illustrates the processes:

```
cd /place/to/install
virtualenv Ruffus_SNP_Analysis
source Ruffus_SNP_Analysis/bin/activate
pip install -U git+https://github.com/boratonAJ/ Ruffus_SNP_Analysis
```

If you don't want to use a virtual environment, then you can just install with pip:

```
pip install -U git+https://github.com/boratonAJ/ Ruffus_SNP_Analysis
```

## Cloned Work

The worked example directory in the source distribution contains the Mycobacterial dataset to illustrate the use of the pipeline.

Get a copy of the source distribution

```
cd /path/to/test/directory
git clone https://github.com/boratonAJ/Ruffus_SNP_Analysis.git
Install `Ruffus_SNP_Analysis` as described above
```

Get a reference genome.

```
cd Ruffus_SNP_Analysis/example
mkdir reference
```

copy your reference into this directory, or make a symbolic link call it reference/H37rv.fa

## DRMAA library

We tell Python where our DRMAA library is. This is will depend on your local settings):

```
export DRMAA_LIBRARY_PATH=/usr/local/slurm_drmaa/1.0.7/gcc/lib/libdrmaa.so
```

Run Ruffus_SNP_Analysis and ask it what it will do next

```
Ruffus_SNP_Analysis -n --verbose 3
```

**Generate a flowchart diagram**

```
Ruffus_SNP_Analysis--flowchart pipeline_flow.png --flowchart_format png
```

**Run the pipeline**

```
Ruffus_SNP_Analysis --use_threads --log_file pipeline.log --jobs 2 --verbose 3
```

**Usage**

You can get a summary of the command line arguments like so:

```
Ruffus_SNP_Analysis -h
usage: Ruffus_SNP_Analysis   [-h] [--verbose [VERBOSE]] [-L FILE] [-T JOBNAME]
                   [-j N] [--use_threads] [-n] [--touch_files_only]
                   [--recreate_database] [--checksum_file_name FILE]
                   [--flowchart FILE] [--key_legend_in_graph]
                   [--draw_graph_horizontally]
                   [--flowchart_format FORMAT] [--forced_tasks JOBNAME]
                   [--config CONFIG] [--jobscripts JOBSCRIPTS]
                   [--version]
```

**optional arguments:**

```
  -h, --help           show this help message and exit
  --config CONFIG       Pipeline configuration file in YAML format, defaults
                   to pipeline.config
  --jobscripts JOBSCRIPTS
                   Directory to store cluster job scripts created by the
                   pipeline, defaults to jobscripts
  --version            show program's version number and exit
```

**Common options:**

```
  --verbose [VERBOSE], -v [VERBOSE]
                   Print more verbose messages for each additional
                   verbose level.
```

```
 -L FILE, --log_file FILE
                    Name and path of log file
```

**pipeline arguments:**

```
 -T JOBNAME, --target_tasks JOBNAME Target task(s) of pipeline.
 -j N, --jobs N   Allow N jobs (commands) to run simultaneously.
  --use_threads  Use multiple threads rather than processes. Needs --jobs N with N > 1
  -n, --just_print  Don't actually run any commands; just print the pipeline.
 --touch_files_only    Don't actually run any commands; just 'touch' the
                    output for each task to make them appear up to date.
 --recreate_database   Don't actually run any commands; just recreate the
                    checksum database.
 --checksum_file_name FILE  Path of the checksum file.
--flowchart FILE      Don't run any commands; just print pipeline as a flowchart.
--key_legend_in_graph
                    Print out legend and key for dependency graph.
 --draw_graph_horizontally
                    Draw horizontal dependency graph.
 --flowchart_format FORMAT
                    format of dependency graph file. Can be 'svg', 'svgz',
                    'png', 'jpg', 'psd', 'tif', 'eps', 'pdf', or 'dot'.
                    Defaults to the file name extension of –flowchart FILE.
  --forced_tasks JOBNAME
                    Task(s) which will be included even if they are up to date.
```

**Configuration file:**

You must supply a configuration file for the pipeline in YAML format. Here is an example:

```
walltime: '10:00'
mem: 30
modules:
```

- 'snpeff/default'

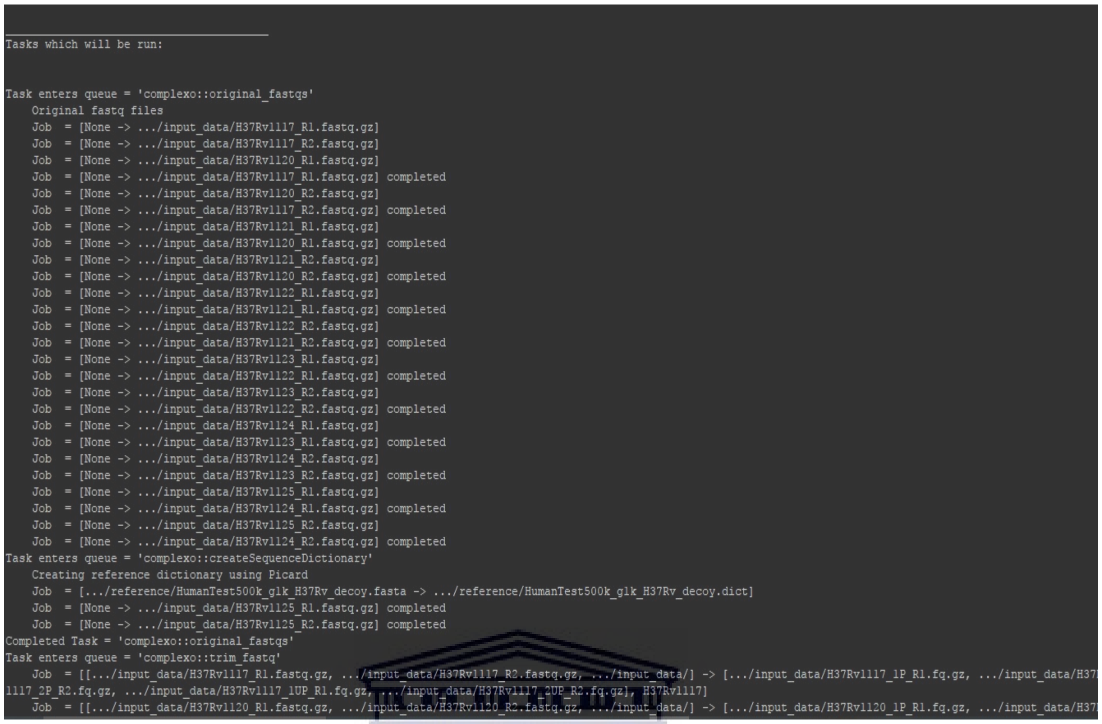
**Reference: The Human Genome in FASTA format.**

ref_grch37:/usr/people/ajayi/test/
Ruffus_SNP_Analysis/example/reference/HumanTest500k_g1k_H37Rv_decoy.fasta

index_file: /usr/people/ajayi/test/ Ruffus_SNP_Analysis /example/reference/*.nix

**The input FASTQ files.**

```
fastqs:
  - /cip0/research/scratch/ajayi/Input_fasta_files/H37Rv1117_R1.fastq.gz
  - /cip0/research/scratch/ajayi/Input_fasta_files/H37Rv1117_R2.fastq.gz
  - /cip0/research/scratch/ajayi/Input_fasta_files/H37Rv1118_R1.fastq.gz
  - /cip0/research/scratch/ajayi/Input_fasta_files/H37Rv1118_R2.fastq.gz
  - /cip0/research/scratch/ajayi/Input_fasta_files/H37Rv1119_R1.fastq.gz
  - /cip0/research/scratch/ajayi/Input_fasta_files/H37Rv1119_R2.fastq.gz
  - /cip0/research/scratch/ajayi/Input_fasta_files/H37Rv1120_R1.fastq.gz
  - /cip0/research/scratch/ajayi/Input_fasta_files/H37Rv1120_R2.fastq.gz
  - /cip0/research/scratch/ajayi/Input_fasta_files/H37Rv1121_R1.fastq.gz
  - /cip0/research/scratch/ajayi/Input_fasta_files/H37Rv1121_R2.fastq.g
pipeline_id: 'H37Rv'
```



```
________________________________________
Tasks which will be run:


Task enters queue = 'complexo::original_fastqs'
    Original fastq files
    Job  = [None -> .../input_data/H37Rv1117_R1.fastq.gz]
    Job  = [None -> .../input_data/H37Rv1117_R2.fastq.gz]
    Job  = [None -> .../input_data/H37Rv1120_R1.fastq.gz]
    Job  = [None -> .../input_data/H37Rv1117_R1.fastq.gz] completed
    Job  = [None -> .../input_data/H37Rv1120_R2.fastq.gz]
    Job  = [None -> .../input_data/H37Rv1117_R2.fastq.gz] completed
    Job  = [None -> .../input_data/H37Rv1121_R1.fastq.gz]
    Job  = [None -> .../input_data/H37Rv1120_R1.fastq.gz] completed
    Job  = [None -> .../input_data/H37Rv1121_R2.fastq.gz]
    Job  = [None -> .../input_data/H37Rv1120_R2.fastq.gz] completed
    Job  = [None -> .../input_data/H37Rv1122_R1.fastq.gz]
    Job  = [None -> .../input_data/H37Rv1121_R1.fastq.gz] completed
    Job  = [None -> .../input_data/H37Rv1122_R2.fastq.gz]
    Job  = [None -> .../input_data/H37Rv1121_R2.fastq.gz] completed
    Job  = [None -> .../input_data/H37Rv1123_R1.fastq.gz]
    Job  = [None -> .../input_data/H37Rv1122_R1.fastq.gz] completed
    Job  = [None -> .../input_data/H37Rv1123_R2.fastq.gz]
    Job  = [None -> .../input_data/H37Rv1122_R2.fastq.gz] completed
    Job  = [None -> .../input_data/H37Rv1124_R1.fastq.gz]
    Job  = [None -> .../input_data/H37Rv1123_R1.fastq.gz] completed
    Job  = [None -> .../input_data/H37Rv1124_R2.fastq.gz]
    Job  = [None -> .../input_data/H37Rv1123_R2.fastq.gz] completed
    Job  = [None -> .../input_data/H37Rv1125_R1.fastq.gz]
    Job  = [None -> .../input_data/H37Rv1124_R1.fastq.gz] completed
    Job  = [None -> .../input_data/H37Rv1125_R2.fastq.gz]
    Job  = [None -> .../input_data/H37Rv1124_R2.fastq.gz] completed
Task enters queue = 'complexo::createSequenceDictionary'
    Creating reference dictionary using Picard
    Job  = [.../reference/HumanTest500k_g1k_H37Rv_decoy.fasta -> .../reference/HumanTest500k_g1k_H37Rv_decoy.dict]
    Job  = [None -> .../input_data/H37Rv1125_R1.fastq.gz] completed
    Job  = [None -> .../input_data/H37Rv1125_R2.fastq.gz] completed
Completed Task = 'complexo::original_fastqs'
Task enters queue = 'complexo::trim_fastq'
    Job  = [[.../input_data/H37Rv1117_R1.fastq.gz, .../input_data/H37Rv1117_R2.fastq.gz, .../input_data/] -> [.../input_data/H37Rv1117_1P_R1.fq.gz, .../input_data/H37R
1117_2P_R2.fq.gz, .../input_data/H37Rv1117_1UP_R1.fq.gz, .../input_data/H37Rv1117_2UP_R2.fq.gz], H37Rv1117]
    Job  = [[.../input_data/H37Rv1120_R1.fastq.gz, .../input_data/H37Rv1120_R2.fastq.gz, .../input_data/] -> [.../input_data/H37Rv1120_1P_R1.fq.gz, .../input_data/H37R
```

**Figure 20: This illustrates the command line interface for executing the Ruffus Pipeline analysis.**

The command line interface illustrates the Ruffus framework, a rule-based framework enactment system that uses declarative specifications of data dependences between workflow steps to routinely order the execution of other steps.

# Appendix C

## Galaxy Framework Implementation Steps

Galaxy workflow framework application detail is provided below. Screen shots are noted to provide a visual outlook of the framework. Galaxy is an open source project, developed by the Center for Comparative Genomics & Bioinformatics at Peninsula State University (Figure 23). The Galaxy project was funded by NSF, Eberly College of Science, and the Huck Institutes for the Life Sciences.

## Python Support

The Galaxy framework supports Python 2.4 or higher. This is needed for biomedical researchers who manage/install the application. More so, Python is required in the virtual environment computer in order to support Galaxy.

## SNPs Analysis in Galaxy Virtual Environment

This Galaxy framework used a virtual Python environment to implement the SNPs data analysis pipeline from the GitHub repository;

```
cd /place/to/install

virtualenv Galaxy SNP_Analysis

source Galaxy_SNPs_Analysis/bin/activate

pip install -U git+https://github.com/galaxyproject/galaxy.git
```

If you don't want to use a virtual environment, then you can just install with pip:

```
pip install -U git+https://github.com/galaxyproject/galaxy.git
```

## Platform UNIX (Ubuntu)

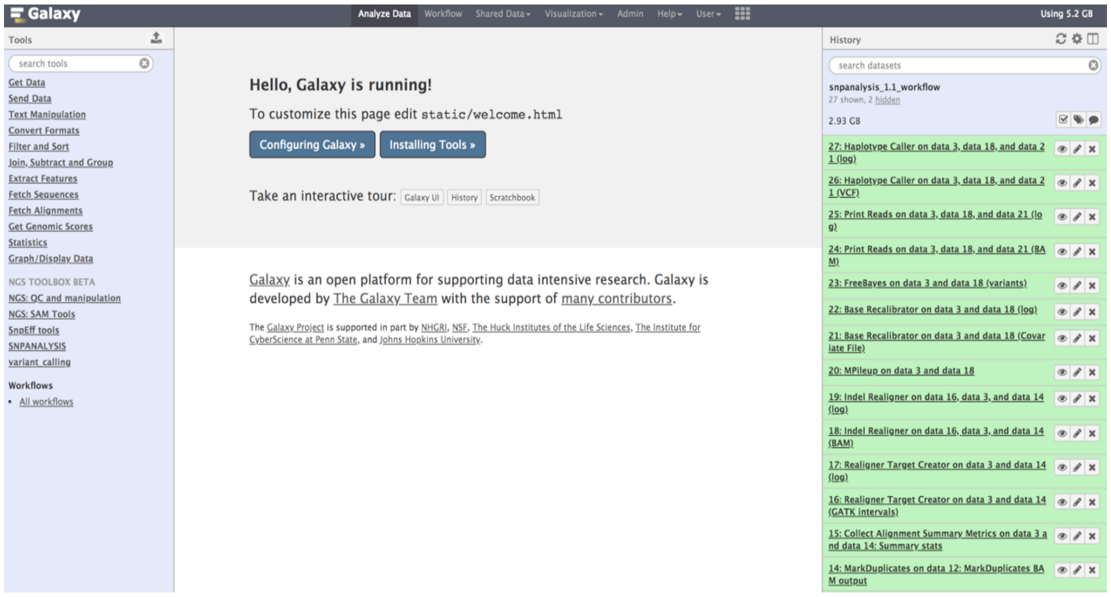
As at the time of writing this thesis, Galaxy framework can easily be downloadable for UNIX and MAC platforms. There is no support for Windows platform with distribution for building Python eggs. That is, modules specific to a Python version that have been compiled and packaged into a single file. Users of Microsoft windows can directly access Galaxy web application from web browsers without downloading the application.

## Data Formats

Galaxy framework accepts input data formats that follow to Browser Extensible Data format (or *.bed), Axt, fastqsolexa, fasta, gff3, gff, html, lav, maf, wiggle, tabular and interval and Other text (characterized by extension, *.txt) file, etc. Other data formats are accepted contingent upon changes of Galaxy framework source code for support of a new data type that is done by the application automatically via the format converters available in the application.

## Customized tools

In this project the customized tool was coded in Python and XML and were integrated in the Galaxy framework. Figure 24 and 25 steps shows the integrated tools with our local instance of Galaxy application. The analyses were created prior to generating the GUI workflows. Figure 24 shows an example of how we started the process that include "Get A File," or "upload a new file either from Hard Drive, Server Libraries or other browser data".



Hello, Galaxy is running!

To customize this page edit `static/welcome.html`

Take an interactive tour:

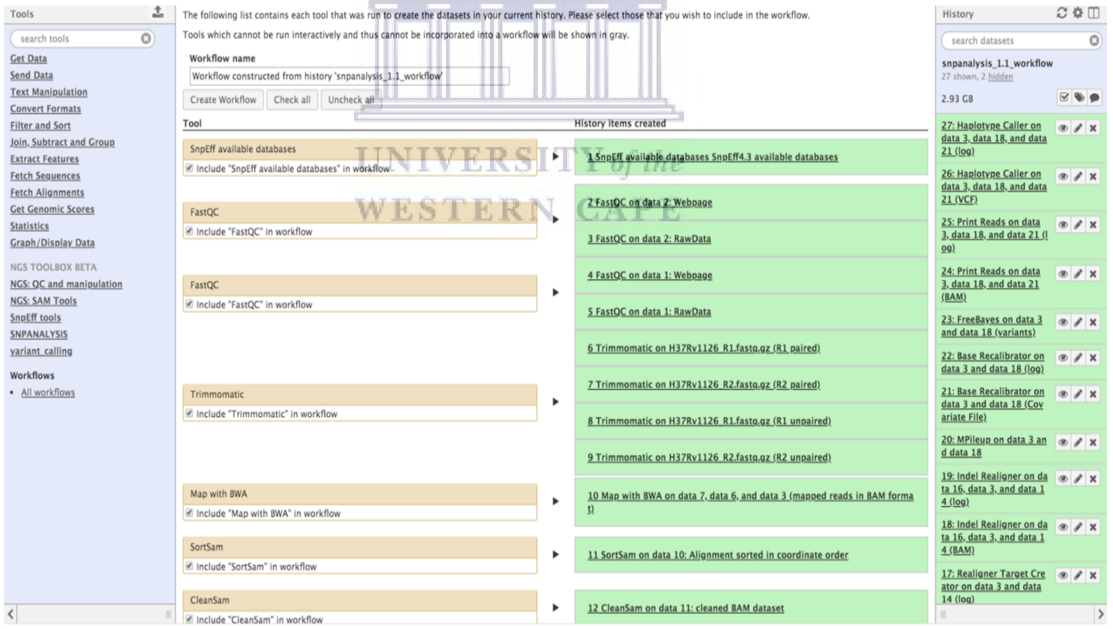
Galaxy is an open platform for supporting data intensive research. Galaxy is developed by The Galaxy Team with the support of many contributors.

The Galaxy Project is supported in part by NHGRI, NSF, The Huck Institutes of the Life Sciences, The Institute for CyberScience at Penn State, and Johns Hopkins University.

snpanalysis_1.1_workflow
27 shown, 2 hidden
2.93 GB

27: Haplotype Caller on data 3, data 18, and data 21 (log)
26: Haplotype Caller on data 3, data 18, and data 21 (VCF)
25: Print Reads on data 3, data 18, and data 21 (log)
24: Print Reads on data 3, data 18, and data 21 (BAM)
23: FreeBayes on data 3 and data 18 (variants)
22: Base Recalibrator on data 3 and data 18 (log)
21: Base Recalibrator on data 3 and data 18 (Covariate File)
20: MPileup on data 3 and data 18
19: Indel Realigner on data 16, data 3, and data 14 (log)
18: Indel Realigner on data 16, data 3, and data 14 (BAM)
17: Realigner Target Creator on data 3 and data 14 (log)
16: Realigner Target Creator on data 3 and data 14 (GATK intervals)
15: Collect Alignment Summary Metrics on data 3 and data 14: Summary stats
14: MarkDuplicates on data 12: MarkDuplicates BAM output

**Figure 21: Galaxy SNP analysis pipeline GUI Webpage.**



The following list contains each tool that was run to create the datasets in your current history. Please select those that you wish to include in the workflow.

Tools which cannot be run interactively and thus cannot be incorporated into a workflow will be shown in gray.

Workflow name: Workflow constructed from history 'snpanalysis_1.1_workflow'

| Tool | History items created |
|---|---|
| SnpEff available databases ✓ Include "SnpEff available databases" in workflow | 1 SnpEff available databases SnpEff4.3 available databases |
| FastQC ✓ Include "FastQC" in workflow | 2 FastQC on data 2: Webpage<br>3 FastQC on data 2: RawData |
| FastQC ✓ Include "FastQC" in workflow | 4 FastQC on data 1: Webpage<br>5 FastQC on data 1: RawData |
| Trimmomatic ✓ Include "Trimmomatic" in workflow | 6 Trimmomatic on H37Rv1126_R1.fastq.gz (R1 paired)<br>7 Trimmomatic on H37Rv1126_R2.fastq.gz (R2 paired)<br>8 Trimmomatic on H37Rv1126_R1.fastq.gz (R1 unpaired)<br>9 Trimmomatic on H37Rv1126_R2.fastq.gz (R2 unpaired) |
| Map with BWA ✓ Include "Map with BWA" in workflow | 10 Map with BWA on data 7, data 6, and data 3 (mapped reads in BAM format) |
| SortSam ✓ Include "SortSam" in workflow | 11 SortSam on data 10: Alignment sorted in coordinate order |
| CleanSam ✓ Include "CleanSam" in workflow | 12 CleanSam on data 11: cleaned BAM dataset |

snpanalysis_1.1_workflow
27 shown, 2 hidden
2.93 GB

27: Haplotype Caller on data 3, data 18, and data 21 (log)
26: Haplotype Caller on data 3, data 18, and data 21 (VCF)
25: Print Reads on data 3, data 18, and data 21 (log)
24: Print Reads on data 3, data 18, and data 21 (BAM)
23: FreeBayes on data 3 and data 18 (variants)
22: Base Recalibrator on data 3 and data 18 (log)
21: Base Recalibrator on data 3 and data 18 (Covariate File)
20: MPileup on data 3 and data 18
19: Indel Realigner on data 16, data 3, and data 14 (log)
18: Indel Realigner on data 16, data 3, and data 14 (BAM)
17: Realigner Target Creator on data 3 and data 14 (log)

**Figure 22: Datasets in the current history.**

The two figures above show the pipeline interface and history and the different stage of the file data formats. The SNPs analysis in Galaxy that run are managed in this interface and on the right show the status of the workflow tasks being run or in queue.

# Appendix D

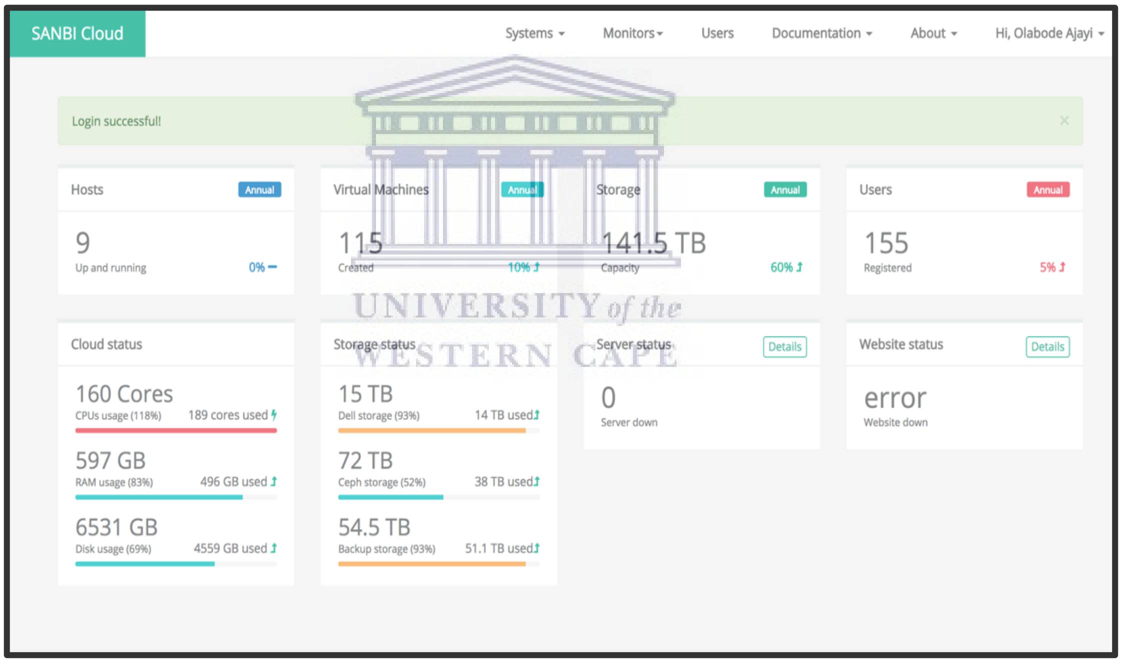
## Implementation of DRMAA for Ruffus and Galaxy

Tell the Python Virtual Environment where your DRMAA library is:

For example (this was depending on the HPC cloud settings):

## Export

```
export DRMAA_LIBRARY_PATH=/usr/local/slurm_drmaa/1.0.7gcc/lib/libdrmaa.so
```



Login successful!

Hosts (Annual): 9 — Up and running — 0%

Virtual Machines (Annual): 115 — Created — 10%

Storage (Annual): 141.5 TB — Capacity — 60%

Users (Annual): 155 — Registered — 5%

Cloud status
160 Cores — CPUs usage (118%) — 189 cores used
597 GB — RAM usage (83%) — 496 GB used
6531 GB — Disk usage (69%) — 4559 GB used

Storage status
15 TB — Dell storage (93%) — 14 TB used
72 TB — Ceph storage (52%) — 38 TB used
54.5 TB — Backup storage (93%) — 51.1 TB used

Server status (Details): 0 — Server down

Website status (Details): error — Website down

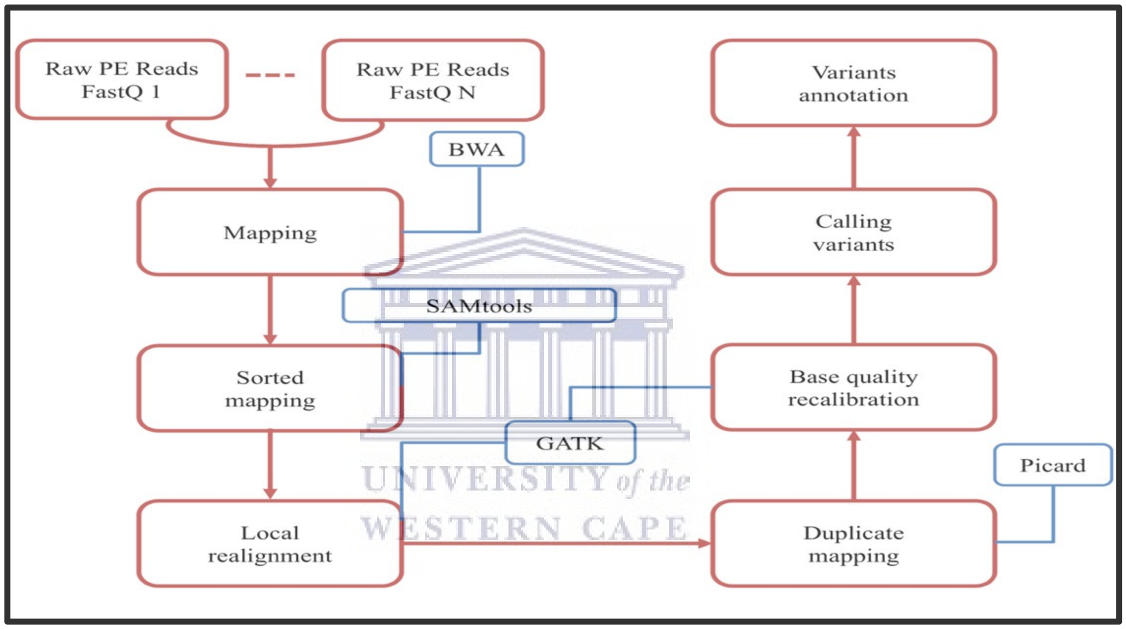
**Figure 23: This illustrates the cloud environment and VM configuration setting for genomic data storage, retrieval and analysis of genomics data.**

This project VM environment was managed by OpenNebula, and was partitioned to manage the operating system, code and database. The created virtual machines, with workloads ranging from web server to high performance computing nodes was also used to manage the workflow execution and management.

# Appendix E

## Simplify SNPs Pipeline Steps

A simplify walk-through steps for the SNPs Analysis Pipeline in Galaxy/Ruffus Framework. The following diagram illustrates the walk-through process for this project.



**<u>Figure 24: A simplify and generic flowchart representing the flow of analysis steps.</u>**

The diagram illustrates an optimized workflow step. Each tool and setting in the Galaxy and Ruffus were used to generate the variant calling pipeline. Galaxy/Ruffus allows an analysis to be started from any level of the process and with option of plugging virtually any bioinformatics tool or code.

## Aligning FASTQ files to reference genome with bwa and Sorting

Aligned_FASTQ.sam: Sort SAM by coordinate and convert to bam using Picard:

a. Use the Picard tool: aligned_FASTQ.sam → FastqtoSam → SortSam → sorted_file.bam

b. If read pairs, merge pairs sorted_file_R1.bam and sorted_file_R2.bam to file_R1_R2_sort.bam with Picard's MergeSamFiles class: Picard MergeBamAlignment → sorted_file_R1_R2_.bam

| **Step 1a.** | **Alignment – Map to Reference Genome** |
|---|---|
| Tool | BWA-MEM |
| Input | Fastq files, H37Rv Reference genome |
| Output | Aligned_Reads.sam |
| Command | bwa mem -M -R '@RG: Sample_1: Sample_1: ILLUMINA: HISEQ:Sample_1'human_g1k_v37_decoy.fasta Sample1_L1_R1. fq Sample1_L1_R2. fq \| samtools view -bSho BAM_FILE – > Aligned_Reads.sam |
| **Step 1b.** | **Sort SAM file by coordinate, convert to BAM** |
| Tool | Picard Tools |
| Input | Aligned_Reads.sam |
| Output | Sorted_Reads.bam |
| Command | java -jar picard.jar SortSam INPUT=Aligned_Reads.sam OUTPUT=Sorted_Reads.bam SORT_ORDER=coordinate |

## Mark and Remove duplicates & Collect Alignment Metrics

Picard MarkDeduplicates (sorted_file_R1_R2.bam) → bam without duplicates

| **Step 2a** | **Mark Duplicates** |
|---|---|
| Tool | Picard Tools |
| Input | Sorted_Reads.bam |
| Output | Dedup_Reads.bam, metrics.txt |
| Command | java -jar picard.jar MarkDuplicates INPUT=Sorted_Reads.bam OUTPUT=Dedup_Reads.bam METRICS_FILE=metrics.txt |

| **Step 2b** | **Collect Alignment Metrics** |
|---|---|
| Tool | Picard Tools, Samtools |
| Input | Sorted_Reads.bam, H37Rv Reference genome |
| Output | alignment_metrics.txt, insert_metrics.txt, insert_size_histogram.pdf |
| Command | java -jar picard.jar CollectAlignmentSummaryMetrics R= reference I=Sorted_Reads.bam O=alignment_metrics.txt |

## Generate Realigning Targets

This is the first step in a two-step process of realigning around indels.

RealignerTargetCreator: Input (bam without duplicates, reference file) → Output (target list file).

| **Step 3** | **Create Realignment Targets** |
|---|---|
| Tool | GATK |
| Input | Dedup_Reads.bam, H37Rv Reference genome |
| Output | Realignment_Targets. list |
| Command | java -jar GenomeAnalysisTK.jar -T RealignerTargetCreator -R reference -I Dedup_Reads.bam -o Realignment_Targets. list |

## Realigning around InDels:

IndelRealigner: Input (bam without duplicates, target list file, reference file) → Output (realigned bam)

| **Step 4** | **Realign Indels** |
|---|---|
| Tool | GATK |
| Input | Dedup_Reads.bam, Realignment_Targets.list, H37Rv Reference genome |
| Output | Realigned_Reads.bam |
| Command | java -jar GenomeAnalysisTK.jar -T IndelRealigner -R reference -I Dedup_Reads.bam -targetIntervals Realignment_Targets. list -o Realigned_Reads.bam |

## Base Recalibrate file

GATK BaseRecalibrator: Input (realigned bam, reference) → Output (recalibrated data table). The variants identified in this step will be filtered and provided as input for Base Quality Score Recalibration (BQSR). The BQSR is performed twice. The second pass is optional but is required to produce a recalibration report.

| **Step 5a** | **Base Quality Score Recalibration (BQSR) #1** |
|---|---|
| Tool | GATK |
| Input | Realigned_Reads.bam, filtered_snps.vcf, filtered_indels.vcf, H37Rv Reference genome |
| Output | Recal_Data.table* |
| Command | java -jar GenomeAnalysisTK.jar -T BaseRecalibrator -R reference -I Realigned_Reads.bam -knownSites filtered_snps.vcf -knownSites filtered_indels.vcf -o Recal_Data.table |

GATK -T CountCovariates → Input (recalibrated data table, reference file) → Output (post recalibrated data table → recalibration report → recalibration report. The second time BQSR is run, it takes the output from the first run (Recal_Data.table) as input

| **Step 5b** | **Base Quality Score Recalibration (BQSR) #2** |
|---|---|
| Tool | GATK |
| Input | Recal_Data.table, Realigned_Reads.bam, filtered_snps.vcf, filtered_indels.vcf, H37Rv Reference genome |
| Output | Post_Recal_Data.table |
| Command | java -jar GenomeAnalysisTK.jar -T BaseRecalibrator -R reference -I Realigned_Reads.bam -knownSites filtered_snps.vcf -knownSites filtered_indels.vcf -BQSR Recal_Data.table -o Post_Recal_Data.table |

Recalibration quality report (PDF and CSV). This step produces a recalibration report based on the output from the two BQSR runs

| **Step 5c** | **Analyze Covariates** |
|---|---|
| Tool | GATK |
| Input | Recal_Data.table, Post_Recal_Data.table, H37Rv Reference genome |
| Output | Recalibration_Plots.pdf |
| Command | java -jar GenomeAnalysisTK.jar -T AnalyzeCovariates -R reference before Recal_Data.table -after Post_Recal_Data.table -plots Recalibration_Plots.pdf |

## Variant Discovery – Calling variants:

Extract SNPs & Indels: This step separates SNPs and Indels so they can be processed and used independently.

| **Step 6a** | **Extract SNPs & Indels** |
|---|---|
| Tool | GATK |
| Input | Raw_Variants.vcf, H37Rv Reference genome |
| Output | Raw_Indels.vcf, Raw_Snps.vcf |
| Command | java -jar GenomeAnalysisTK.jar -T SelectVariants -R reference -V raw_variants.vcf -selectType SNP -o Raw_Snps.vcf java -jar GenomeAnalysisTK.jar -T SelectVariants -R reference -V Raw_Variants.vcf -selectType INDEL -o Raw_Indels.vcf |

GATK -T VariantFiltration → snp-filter.vcf. The SNPs which are 'filtered out' at this step will remain in the filtered_snps.vcf file, however they will be marked as 'basic_snp_filter', while SNPs which passed the filter will be marked as 'PASS'. The filtering criteria for SNPs are as follows: QD < 2.0, FS > 60.0, MQ < 40.0, MQRankSum < -12.5, ReadPosRankSum < -8.0, SOR > 4.0.

| **Step 6b** | **Filter SNPs** |
|---|---|
| Tool | GATK |
| Input | raw_snps.vcf, reference genome |
| Output | filtered_snps.vcf |
| Command | java -jar GenomeAnalysisTK.jar -T VariantFiltration -R reference -V raw_snps.vcf --filterExpression 'QD < 2.0 |

HaplotypeCaller and filter variants: GATK -T HaplotypeCaller → Input (recalibrated data table, reference file, recalibrated bam) → Output (variant call sets (snp.vcf)).

| **Step 6c** | **Call Variants** |
|---|---|
| Tool | GATK |
| Input | Realigned_Reads.bam, H37Rv Reference genome |
| Output | Raw_Variants.vcf |
| Command | java -jar GenomeAnalysisTK.jar -T HaplotypeCaller -R reference -I Realigned_Reads.bam -o Raw_Variants.vcf |

Evaluate Haplotype: GATK -T VariantEval → snpfilter.eval

Calculate variation effects: java -jar snpEff.jar → snp-filtereffects.tsv

# Appendix F

## Benchmarks: Collectl-Utility

Collectl-Utility is a Perl programming code that attracts as much detail as possible from the /proc filesystem. This project used Collectl-Util in daemon mode and modified one line in /etc/Collectl.conf by adding the following the default statistics monitored.

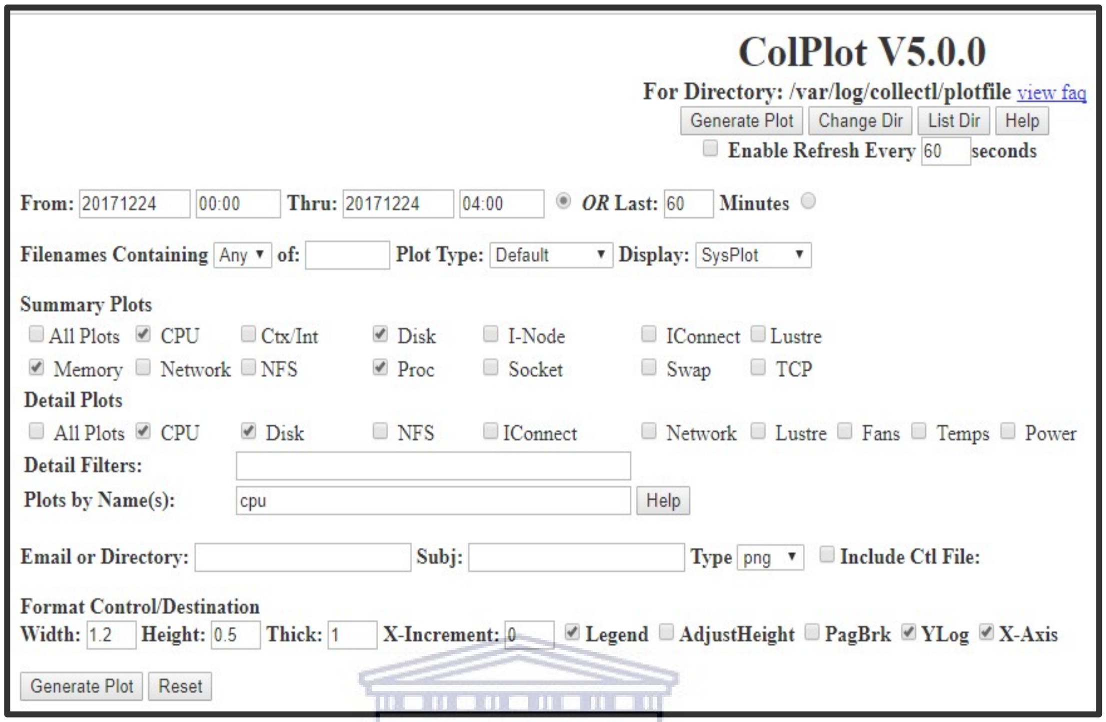
*"The line in /etc/Collectl.conf is:*

*DaemonCommands = -f /var/log/Collectl -r00:00,7 -m -F60 -s+YZCD –iosize"*

The above code options allow us to monitor the HPC virtual environment CPU, disk, and network in brief mode, and slab, processes, and disk in detailed mode. Furthermore, we added code to monitor disk input/outsize (iosizes). After the implementation of the workflow frameworks, the Collectt-Util testing was complete. We then grabbed the raw Collectl data file and copied it into a directory for post-processing. The file is named localhost-20120310133840.raw.gz. The data was processed with Collectl-ColPlot to create plot files for the various subsystems such as CPU, disk, and so on. The exact command is:

*"% Collectl -p localhost-20120310-133840.raw.gz -P -f ./PLOTFILES -ocz"*

The -p option tells Collectl to "play back" the data or, literally, to run the data back through Collectl, and it takes as an argument the name of the raw file. The -P option tells Collectl to create plot files. The -f option tells Collectl to use a specific directory in which to place the output (I created a subdirectory called PLOTFILES, where I stored the plot files). The option -ocz tells Collectl to open the plot files in create mode, which means it will overwrite existing files with the same name. The -z option tells Collectl not to compress the plot files.



ColPlot V5.0.0

For Directory: /var/log/collectl/plotfile view faq

[ ] Enable Refresh Every 60 seconds

From: 20171224 00:00 Thru: 20171224 04:00 (●) OR Last: 60 Minutes ( )

Filenames Containing Any of: Plot Type: Default Display: SysPlot

Summary Plots

[ ] All Plots [✓] CPU [ ] Ctx/Int [✓] Disk [ ] I-Node [ ] IConnect [ ] Lustre
[✓] Memory [ ] Network [ ] NFS [✓] Proc [ ] Socket [ ] Swap [ ] TCP

Detail Plots

[ ] All Plots [✓] CPU [✓] Disk [ ] NFS [ ] IConnect [ ] Network [ ] Lustre [ ] Fans [ ] Temps [ ] Power

Detail Filters:

Plots by Name(s): cpu

Email or Directory: Subj: Type png [ ] Include Ctl File:

Format Control/Destination

Width: 1.2 Height: 0.5 Thick: 1 X-Increment: 0 [✓] Legend [ ] AdjustHeight [ ] PagBrk [✓] YLog [✓] X-Axis

**<u>Figure 25: The diagram illustrates colplot.</u>**

We use the colplot to generate plots against captured logs files in our Galaxy/Ruffus directory that match the selected timeframe such as CPU, memory etc.