A Cloud-Based Testing Framework for Genomic Medicine Software

Michel Troup
SID: 312133227

Supervisor: Associate Professor Bing Zhou
External Supervisor: Dr Joshua Ho

This thesis is submitted in partial fulfilment of
the requirements for the degree of
Bachelor of Computer Science and Technology (Advanced) (Honours)

School of Information Technologies
The University of Sydney
Australia

3 November 2015

Student Plagiarism: Compliance Statement

I certify that:

I have read and understood the University of Sydney Student Plagiarism: Coursework Policy and Procedure;

I understand that failure to comply with the Student Plagiarism: Academic Board Policy: Academic Dishonesty and Plagiarism can lead to the University commencing proceedings against me for potential student misconduct under the 2012 Academic Dishonesty and Plagiarism in Coursework Policy.

This Work is substantially my own, and to the extent that any part of this Work is not my own I have indicated that it is not my own by Acknowledging the Source of that part or those parts of the Work.

Name: Michael Troup

Signature: Date:

Acknowledgement

I would like to acknowledge the support and guidance provided to me by the following people and organisations, to enable me to complete this research.

Thank you to my Sydney University supervisor, Associate Professor Bing Zhou, who has generously given his time to supervise me.

Much appreciation is given to Dr Joshua Ho for his generosity in supervising me as an Honours student, providing the research topic, and welcoming me into his bioinformatics lab at the Victor Chang Cardiac Research Institute (VCCRI). Thank you to the Victor Chang Cardiac Research Institute for supporting this research. Also at the VCCRI, I would like to thank UNSW Ph.D. candidate Andrian Yang, who has helped to supervise me on a day-to-day basis. Many other Ho lab members at VCCRI have also made themselves available to assist with the many bioinformatics issues related to this project.

Finally I thank my family for supporting me in this research, including my parents, who provided me with the educational foundation which made my contribution to this research possible.

Abstract

Genomic sequencing and subsequent comparison to a reference human genome is used to identify points of variation between individuals. Genomic sequencing has important applications in the field of targeted genomic medicine, where the importance of correctly functioning analysis software cannot be underestimated.

Testing of the software that analyses genomic sequencing data has lacked a readily accessible systematic approach to determining the correctness of the resulting output. In addition to this, smaller bioinformatics departments may also be challenged by the scale and complexity of implementing a suitable testing framework.

This research presents a novel approach to solving this problem, that utilises cloud computing resources, and both Metamorphic and traditional testing techniques, to test a supplied variant calling genomic sequencing software "pipeline". The testing framework has been applied to a particular genomic sequencing pipeline comprised of industry-standard open-source components, and demonstrates the utility of the tool to test such software in an automated fashion. The Metamorphic tests were successful in discovering anomalies in the tested pipeline.

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

Genomic medicine is a rapidly evolving field of medicine that relates to the use of genomic information to make decisions about a patient's treatment. A genome refers to an organism's full set of DNA (deoxyribonucleic acid) – which is used as a code to build and maintain that organism. Human DNA contains billions of pairs of four particular chemical bases, and there is only a small degree (< 1%) of variation in the sequence of DNA between humans [1].

Identifying variations in human DNA sequences is central to the field of genomic medicine, and particular "variants" have been shown to be the cause of certain diseases [2]. As is discussed in further detail in the following chapter, there is a "sequencing" process that takes a physical sample from a person, and produces an electronic data-file with a set of relatively short "reads" of the subject's DNA sequence. These sequencing reads are then used as input to a series of computational steps, whose aim is to produce as output, the variations – termed "variants", as compared to a reference genome. This reference genome is a representative human genome constructed from a large number of human subjects. The series of computations mentioned here are collectively referred to as a "variant calling genomic sequencing pipeline" (vGSP).

If such information is to be used in clinical medical decisions, it is critical that the vGSP be properly tested. Such testing has often relied on traditional testing techniques such as comparing a particular input with a known "gold standard" output. Some software also exists to simulate reads with mutations, and hence be able to be used to provide a large number of inputs and expected outputs with which the output of the vGSP under tests can be compared.

However, these methods are limited in their approach, not least because there are unlimited possibilities for inputs to a vGSP. Other current testing limitations include barriers such as knowledge of testing methodologies, technical computing expertise, and lack of computing resources. Cloud computing - which gives users internet access to shared computational resources using an on-demand, pay-as-you-go model - is helping to reduce the barrier of access to computational resources for practitioners such as bioinformaticians. However, there are significant learning curves for the practitioner in this area to configure the resources necessary to run a vGSP on the cloud.

Metamorphic Testing (MT), is a testing method that was introduced to deal with situations where, for a program under test, there is no easy way to determine the correct output for every possible set of inputs [3]. Instead of needing to know the expected output for a particular input, MT utilises known relationships that should hold between outputs, for given pairs of inputs. One simple example in this area of research is that if we run an input through a vGSP, then we would expect the same output if we used the same input data-set in a different order (e.g. randomly shuffled).

The general approach taken to improve the testing of vGSP's in this research, given the outline of the issues involved with current methodologies, has been to develop a proof-of-concept testing framework that the user can download to their own computing resource – in the form of a command line Linux software solution. This testing framework utilises both metamorphic and traditional tests, and gives the user control of selecting and paying for their own cloud resources. After configuration, where the user supplies their own cloud computing credentials, and access to files and scripts enabling installation and execution of their vGSP, the framework launches a set of automated tests that execute the user's vGSP on cloud computing resources. The results of testing are reported back to the user.

1.1 Contribution

The primary aim of this research is to improve software testing in the area of genomic medicine software. To demonstrate how this could be achieved, a proof-of-concept software testing framework for variant calling genomic sequencing pipelines (vGSP's) has been developed and applied to a particular vGSP pipeline.

One of the key contributions of this research has been the provision of a set of Metamorphic Relations (MR's) that can be applied to a vGSP as a whole entity. Most of these MR's have been adapted from previous research that looked primarily at only one component of the pipeline [4]. However, as the result of this research, a new MR was introduced that checks that the output from the vGSP is deterministic. Whilst this sounds like a trivial check, it may sometimes be overlooked, and in testing an industry standard vGSP as part of this research, the output was found not to be completely deterministic.

The testing framework produced as part of this research is the first known framework of this nature to handle data at scale. This has been achieved both through the use of cloud-based resources, which enables each test to be run on a separate cloud resource, and also by ensuring that the data manipulation for the MR's have been parallelised. The ability to handle large scale data is shown in the results chapter, where the total size of the input data is 30GB.

When testing the framework using real and simulated data, the utility of the MR's were demonstrated to discover anomalies in the industry-standard vGSP that was used for the test.

This research also combines the MT testing concepts with more traditional testing techniques through the use of open-source software that simulates reads. A particular test design that this research contributes, simulates reads directly from a reference genome – without any mutations. Running this simulated data (without mutations) through the vGSP should not produce any variants. This focus on false positive discovery was not found in the literature.

The rise of cloud computing has brought with it a range of complexities and learning curves that bioinformaticians working on their own software may have not had the time to master. The practitioner may also not be aware of the latest testing techniques. This research contributes a framework that takes care of the testing methodology, and most of the complexities in the cloud configuration. Once the user configures the framework, where the main configuration file is in the form of a simple text file, and supplies access to installation and execution scripts for their vGSP, the framework takes care of the rest. The tests are run automatically on created cloud resources, and the results are collated and reported back to the user.

One small example of the contribution to reduced complexity for the user is the feature of the framework to optionally use a cloud computing resource that can access a local (to the cloud resource) data storage that is a solid state drive. The use of such a drive can result in significantly increased performance, which is important when dealing with vGSP's, where a single test execution can take as much as 24 hours or more for a full dataset. Such a configuration can be too complex for a novice cloud user to complete for themselves manually.

1.1.1 Publications

An abstract for a poster describing this research was submitted by this author, and accepted for the 2015Australian Bioinformatics And Computational Biology Society (ABACBS) conference [5]. The poster was presented at this conference on 10 October 2015. ABACBS "*is focused on the science and profession of bioinformatics and computational biology in Australia*" [3].

1.2 Document Outline

This document includes a background on this research subject matter, including related works – in the Background chapter. Following this, there are chapters on Methodology and Implementation, with the Methodology taking a higher-level view of the approach taken, focusing more on design than implementation. Finally, there are chapters on results, discussion, and a conclusion. The Results chapter looks at the results of applying real and simulated data to an example of a vGSP that is used in practice. The Discussion chapter also includes a discussion on limitations and future work.

Chapter 2. Background

DNA contains the genomic blueprint for replication of cells in most living organisms. It is comprised of a string of chemical base pairs, with there being four bases: *cytosine (C)*, *thymine (T)*, *adenine (A)*, and *guanine (G)*. In humans, this string of DNA comprises around three billion pairs of such bases, and the order in which these occur determines vital information about the physical composition of an individual.

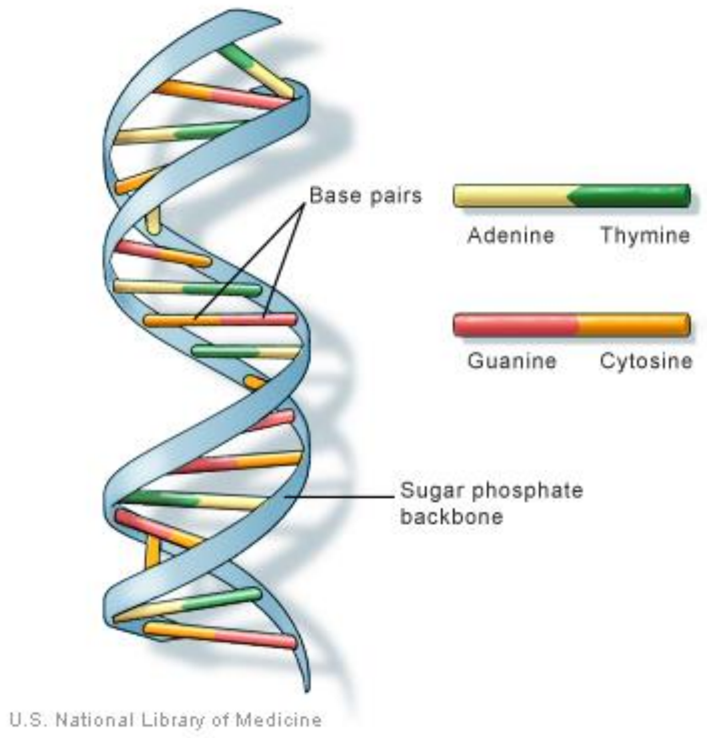


Figure 1 - DNA structure [6]

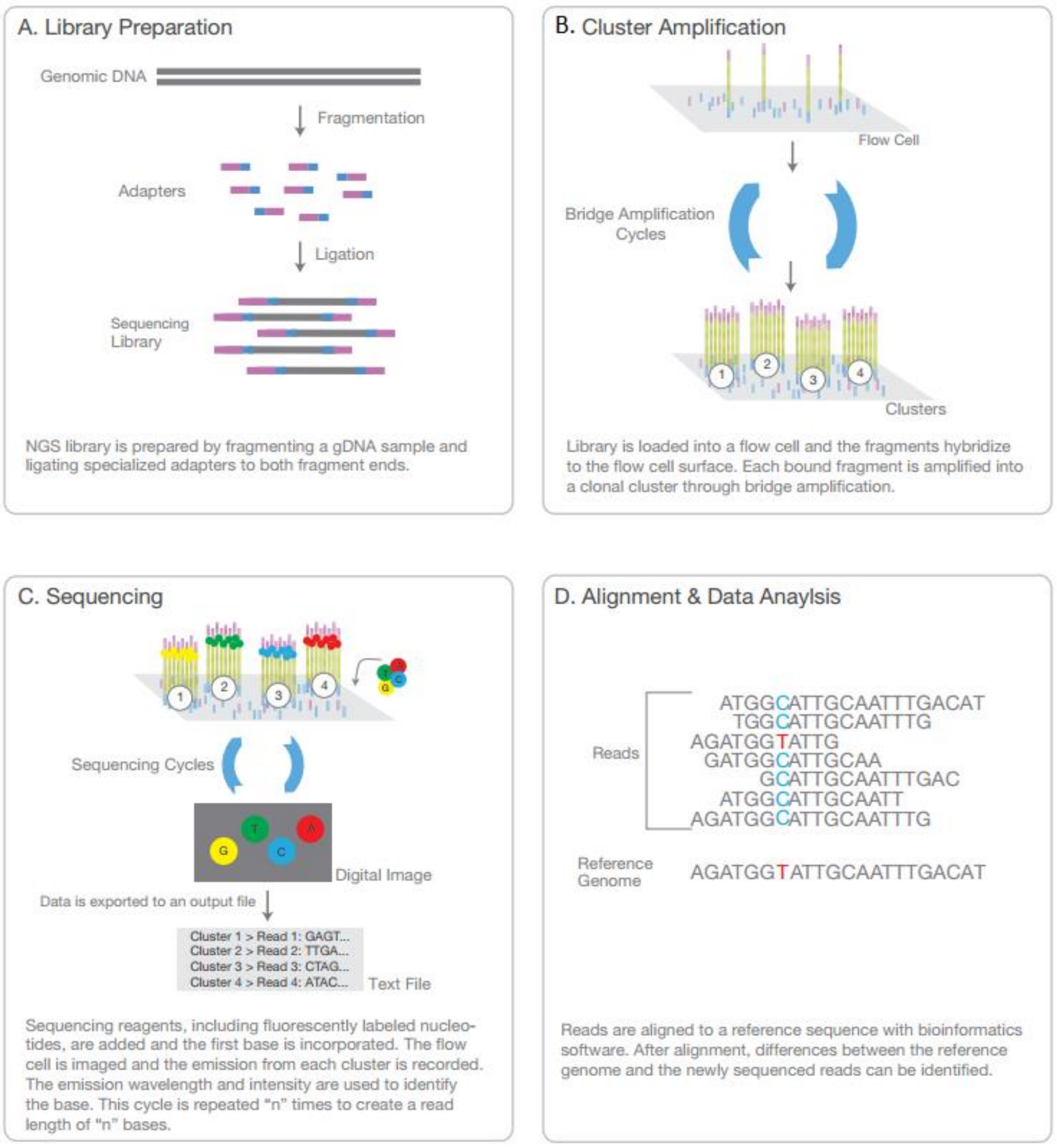
The complete set of DNA for an organism is called its genome, and hence genomic sequencing is concerned with finding the order in which base-pairs exist for a particular individual or organism. As will be discussed in the following section of this chapter, post-processing of the output from the application of a biological sample to a sequencing machine, requires a number of separate computational steps that form a so-called *genomic sequencing pipeline*. One very important high level use of genomic sequencing is to compare an individual's sequence with a reference genome, so as to be able to discover variations from this reference. This knowledge can then be used for disease diagnosis, tailored disease treatment plans, forensic identification, and anthropological uses, to name a few.

2.1 Sequencing

With technological advancements over recent decades, there have been many new sequencing methods introduced, many of which are collectively known as Next

Generation Sequencing (NGS) methods. A selection of NGS methods include: pyrosequencing, Massively Parallel Signature Sequencing (MPSS), sequencing by synthesis (Illumina), and sequencing by ligation (SOLiD).

The process of taking a DNA sample from the preparation phase, all the way through to data analysis, is shown diagrammatically below - using the Illumina sequencing by synthesis method.



NGS library is prepared by fragmenting a gDNA sample and ligating specialized adapters to both fragment ends.

Library is loaded into a flow cell and the fragments hybridize to the flow cell surface. Each bound fragment is amplified into a clonal cluster through bridge amplification.

```
Cluster 1 > Read 1: GAGT...
Cluster 2 > Read 2: TTGA...
Cluster 3 > Read 3: CTAG...
Cluster 4 > Read 4: ATAC...
```

Sequencing reagents, including fluorescently labeled nucleotides, are added and the first base is incorporated. The flow cell is imaged and the emission from each cluster is recorded. The emission wavelength and intensity are used to identify the base. This cycle is repeated "n" times to create a read length of "n" bases.

```
      ATGGCATTGCAATTTGACAT
       TGGCATTGCAATTTG
   AGATGGTATTG
     GATGGCATTGCAA
          GCATTGCAATTTGAC
      ATGGCATTGCAATT
   AGATGGCATTGCAATTTG

   AGATGGTATTGCAATTTGACAT
```

Reads are aligned to a reference sequence with bioinformatics software. After alignment, differences between the reference genome and the newly sequenced reads can be identified.

Figure 2 - Illumina NGS steps: preparation to data [7]

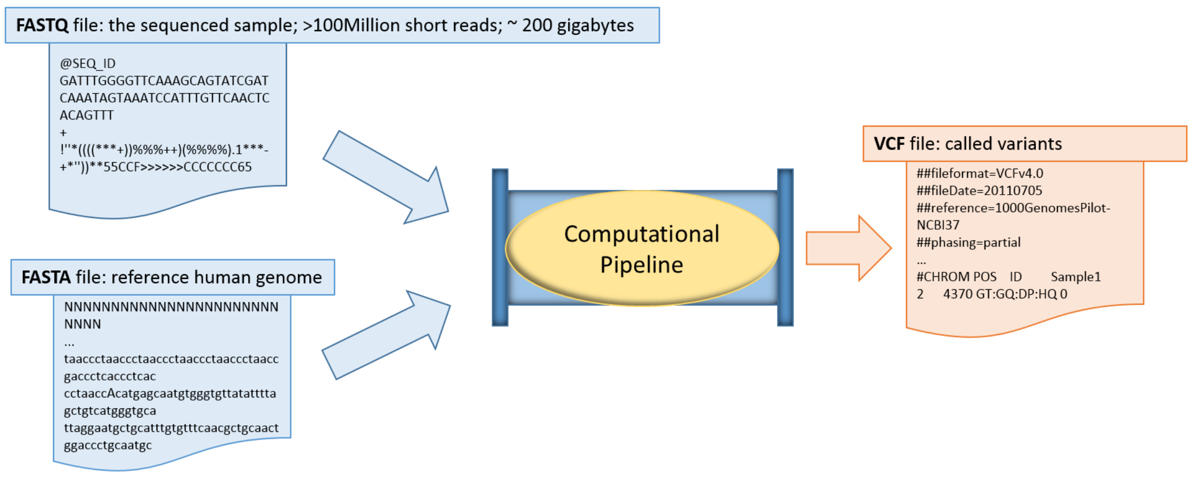
The output from a sequencing machine is normally a FASTQ file – which contains short reads (~70-300 base pairs long), as described separately in Appendix 1. The data is commonly produced as two separate FASTQ files, which are called paired-end reads. Paired-end reads are used to resolve ambiguous alignments.

One important concept relevant to vGSP's is sequencing coverage. Coverage is the average number of reads that align to ("cover") known reference nucleotides. The coverage level is a critical factor in determining the effectiveness of variant discovery. For detecting human mutations such as single base variations (Single Nucleotide Polymorphism – SNP) or INDELS (Insertions or Deletions), it is recommended to aim for 10x to 30x coverage [8].

Figure 3- A single Illumina sequencing machine in the Hiseq X series[9]

2.2 After Sequencing – The Computational Pipeline

Figure 4 below shows a high-level view of what happens after a sequencing machine outputs the sequence as a series of short reads in *FASTQ* format. This FASTQ file, along with the reference human genome are inputs into the computational pipeline, which produces an output file, listing the genomic variants detected, in *VCF* format.


FASTQ file: the sequenced sample; >100Million short reads; ~ 200 gigabytes
FASTA file: reference human genome
Computational Pipeline
VCF file: called variants

```
@SEQ_ID
GATTTGGGGTTCAAAGCAGTATCGAT
CAAATAGTAAATCCATTTGTTCAACTC
ACAGTTT
+
!''*((((***+))%%%++)(%%%%).1***-
+*''))**55CCF>>>>>>CCCCCCC65
```

```
NNNNNNNNNNNNNNNNNNNNNNN
NNNN
...
taaccctaaccctaaccctaaccctaaccctaacc
gaccctcaccctcac
cctaaccAcatgagcaatgtgggtgttatatttta
gctgtcatgggtgca
ttaggaatgctgcatttgtgtttcaacgctgcaact
ggaccctgcaatgc
```

```
##fileformat=VCFv4.0
##fileDate=20110705
##reference=1000GenomesPilot-
NCBI37
##phasing=partial
...
#CHROM POS    ID        Sample1
2     4370 GT:GQ:DP:HQ 0
```

Figure 4 - High-level view of computational pipeline process

Altmann et al. [11] does a good job at summarising the various stages of the genomic variant calling pipeline, breaking the pipeline into seven distinct phases. Firstly, the DNA sample undergoes a chemical process to cut it into small (in the order of 100's of base pairs) fragments, and via the use of a sequencing machine, output a file in FASTQ format - which shows the base sequence and quality score for each read. Partially due to an amplification process, the FASTQ file can contain hundreds of millions of bases of sequence information. Phase 2 is a quality phase where reads with unacceptable quality scores are removed. The filtered FASTQ file from phase 2 is used as input into phase 3, which is normally the first phase that we consider in our computational pipeline. Phase 3 aligns the reads to the reference human genome. Common open-source programs to perform this step include *BWA* [12] and *Bowtie* [13], and the output of phase 3 is a file in *SAM/BAM* format [14]. Phase 4 is alignment post-processing, using software such as *SAMtools* [14], *Picard* [15], or *GATK* [16], which will sort the alignments in chromosomal order. Phase 5 is quality score recalibration. Phase 6 calls the variants using a tool such as GATK, with the output being a VCF file. Finally, in phase 7, the variants are filtered to identify single nucleotide polymorphism (SNP) candidates (by reducing false-positives), also possibly using a software tool such as GATK.

Using the output from phase 2- the filtered FASTQ files - the user can use this as input into their chosen computational pipeline. Such a pipeline may be constructed by the user and run on the user's own computational resources - normally by combining a number of open-source components such as BWA, Bowtie, GATK, etc., into a script file. Alternatively, there are many online solutions available where the user can upload their data to some web/cloud based service. Examples of such online services include DNAnexus's Mercury [17], Atlas [18], and Illumina's BaseSpace [19]. One reason an organisation may not wish to use a cloud offering from another organisation could be related to security concerns with the data to be sequenced.

Figure 5 gives a summary of the inputs, intermediate and final outputs, and some common tools for a vGSP. Additional information relating to the input and output file formats relevant to vGSP's are included as an appendix.

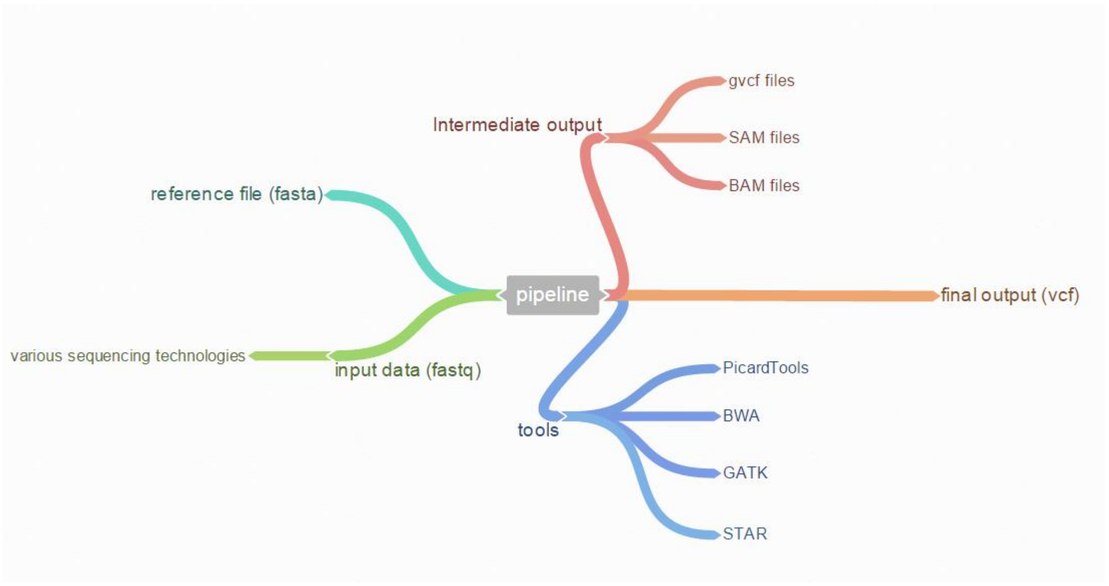


Figure 5 - Overview of pipeline inputs, outputs, and common tools

2.3 The need for testing in genomic medicine software

To illustrate how far the technology in this field has progressed over the last ten to fifteen years, consider the human genome project with its seminal paper *Initial sequencing and analysis of the human genome* [20] in 2001. The project was a collaboration by 20 groups from at least six different countries to create a map of the human genome. The paper states that the desire to map the human genome arose from the insight that "*to take global views of genomes could greatly accelerate biomedical research*". As a sign of the scale and complexity of the project, it was first launched in late 1990, leaving close to a decade for them to produce this 2001 paper. The budget for the project was three billion dollars [21]. There has been much work done in the field since this project, and in 2015 a whole genome can be sequenced in a few hours [22], whilst Kelly et al. [23] claim to be able to analyse the output of a high-coverage whole of genome sequencing in under 2 hours.

One major change within the last decade has been the significant increase in sequencing data available for analysis, at a cheaper cost [24]. *The National Human Genome Research Institute* [25] reports that the cost of sequencing per genome has reduced from $100M in 2001, to $10m in 2007, down to a figure of less than $10,000 in 2015. The introduction of technically advanced sequencing machines have helped coin the phrase *Next Generation Sequencing* (NGS), *high throughput sequencing,* and the associated *explosion of data*. The initial data that comes out of a NGS machine will be in the order of 200 gigabytes for a single human genome [26]. In 2013, an IEEE publication [27], highlighted the potential cumulative effect of this: "*The roughly 2000 sequencing instruments in labs and hospitals around the world can collectively sequence 15 quadrillion nucleotides per year, which equals about 15 petabytes of compressed genetic data.*".

With the vastly increased amount of data available, there is more opportunity to analyse this data, and apply the results in a related setting. As mentioned above, genomic sequencing has many important applications including forensics, disease identification, clinical treatment plans, and further genomic research. Ensuring that the computational pipeline is producing the correct results is critical in any of these practical settings, particularly so in clinical applications. The clinical need is discussed by Bennett et al. [28], who talks about moving from using NGS data in a research setting to using NGS in a clinical setting. In addition to this, the scale of the data involved, and computational complexity of the sequencing pipeline creates a need to ensure that there are robust and automated testing solutions in place.

To further illustrate the need for testing in this research area, Bennett et al. [28] reports on a study initiated by the *Boston Children's Hospital*. Twenty three international bioinformatics groups analysed NGS data relating to a group of participants with known genomic disorders. Only one third of the groups reported any of the known variants that the participants were known to contain. In another study, O'Rawe et al. [29] reports on the lack of agreement between a number of variant calling pipelines when analysing the same data.

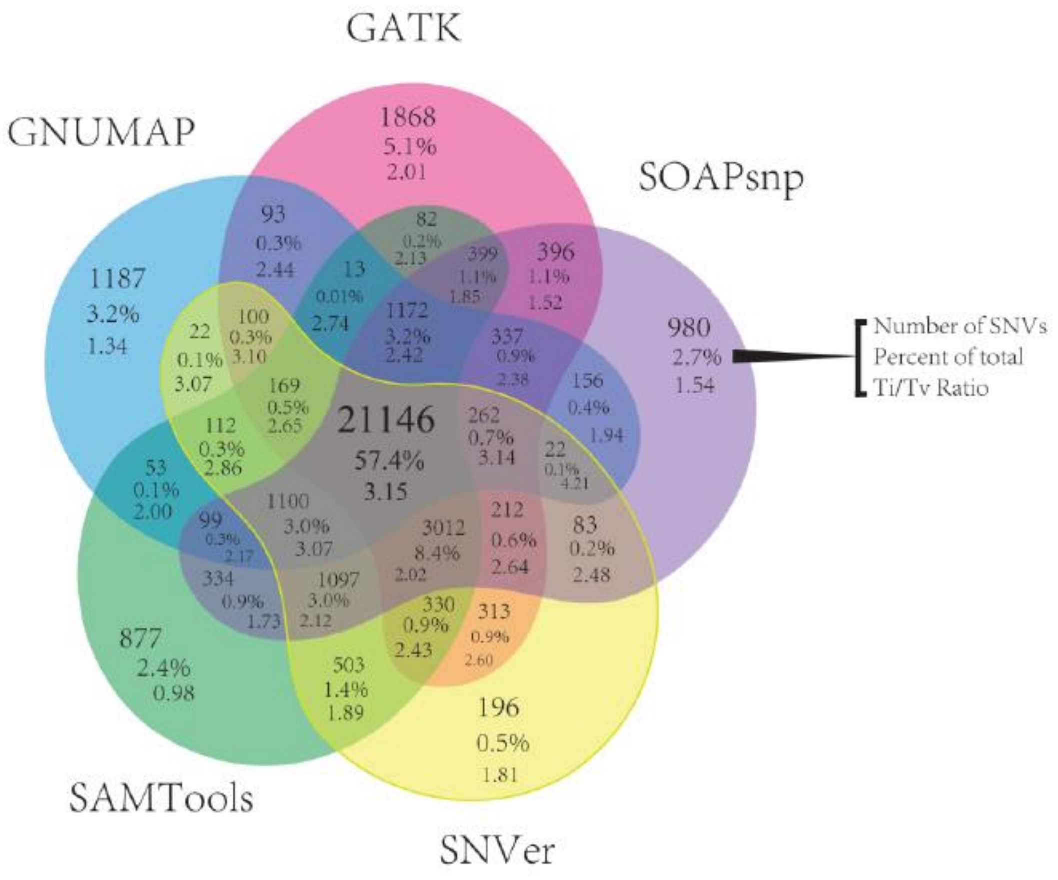


Figure 6 - Mean single-nucleotide variants (SNV) concordance between five variant-calling pipelines[29]

2.4 Software testing concepts and approaches

Ensuring that a software program is correct is simply not possible in many cases, even when the program is not complex. For example if we want to see that a program correctly outputs the sum of two integers, there is no way to verify the program output for the infinite set of possible inputs. However, we can test a range of inputs to see that the program is behaving as expected. According to Myers et al. [30], the goal of software testing in general is to ensure that a program works as intended or, perhaps more realistically, to identify as many errors as possible in the underlying software. There are testing techniques that identify problems by exploring the internal structure of the software, applying inputs based on the various logic built in to the system. Such a system of testing is termed *white-box* testing. *Black-box* testing is the process of examining the output or behaviour of a software system, without any examination of the internal workings of the system. The focus of this research is complex bioinformatics pipeline software that are comprised of many individually complex programs solving some subset of an overall task. Some of the individual components of this software application may include software for which the source code is not publicly available. Hence this research will be examining black-box testing techniques.

In many classes of software applications, it may be possible to verify the correctness of a small subset of output, but it is often impossible to obtain a systematic mechanism to verify all the output. The mechanism that enables any input to be verified is called an *oracle* [31], and is assumed to exist in many traditional testing techniques, such as: *random input testing* and *domain testing*. Random testing tests a program by supplying random independent test inputs, and compares the results with what is expected in these cases. Domain testing is a widely used software testing technique where a subset of test cases is selected from a possibly infinite range of possible test cases. As will be discussed in the next section, this is a popular method of testing in the bioinformatics field. Nonetheless, domain testing is not systematic enough, and assumes that the correctness of the untested domains is inferred from the test domains. In the case of a critical medical program, such as the genome variant calling pipeline, it is important to apply test cases much more systematically and comprehensively. We will discuss several software testing techniques that achieve this goal in the following sections.

2.5 Current testing approaches used in the genomics field

A common approach to testing a vGSP to is to run the pipeline using a reference input and compare the results (i.e. the variants called) with a well-established *gold standard* reference output. In the US, *The National Institute of Standards and Technology* (*NIST*), along with the *Genome in a Bottle Consortium* [32] and the *FDA*, is developing such reference material from whole human genomes. One of the goals of this partnership is to develop reference standards, methods and data for whole human genome sequencing. Reference data-sets can also be obtained from the manufacturer of the particular sequencing machine - e.g. Illumina’s *The BaseSpace Platinum Genomes Project* and the *Variant Calling Assessment Tool (VCAT)* [33]. Whilst this approach is a good starting point, it only verifies a select number of input/output combinations. All humans are different, and clinicians want to have confidence that the pipeline will work for every new case. For testing partial results, a smaller FASTQ input file can be used to either manually or automatically determine if the output is correct, or *Sanger* sequencing [34] can be used to verify a subset of the data. The Sanger sequencing method was first introduced in 1977, and has been replaced by more recent NGS techniques. However it is still remains in use, partly to validate NGS results.

Rehm et al. [35] summarises some of these above approaches in a 2013 paper that discusses the *American College of Medical Genetics and Genomics* clinical guidelines for NGS. Once a pipeline has been established, the results should be compared with a gold standard, as well as tested for repeatability - multiple runs should produce the same result. One particular validation technique that Rehm discusses, states: “*To determine the analytic validity of a test, the laboratory should utilize well-characterized reference samples for which reliable Sanger sequencing data exist*”. One should note that there is a limit to which such testing could be taken before either the availability of existing reference samples is exhausted, or the cost of comparative sequencing methods (e.g. Sanger) is prohibitive.

Related to the method of comparing the output with a known input and expected result, there are a number of simulation packages available that can simulate read data, and provide a "truth" VCF file for the expected output. Two such programs are known as ART [36], and VarSim [37]. Programs of these type allow for the generation of large amounts of data to be tested. Whilst partially addressing the "gold-standard" issue of lack of data, there are additional sources of uncertainty introduced depending upon the simulation process used.

As O'Rawe et al. [29] demonstrates, and as also commented on by Chen et al. [14], another popular testing method in the genomic sequencing bioinformatics area is that of *N-version programming*. Multiple versions of independently developed programs that are meant to solve the same problem are executed, and the output of these programs are compared to check if they are same. One obvious shortcoming of this is that if different results are obtained, then the tester may not be able to determine which program is correct.

2.6 State-of-the-art testing approaches

2.6.1 Adaptive Random Testing

Chen et al. [38] describes *adaptive random testing (ART)* as a method of random testing that takes into account the relationship between failure causing inputs. It is suggested that there exists empirical evidence that in some situations failures tend to cluster around similar inputs, and conversely that non-failure causing inputs can also cluster together. The idea is to maintain two sets of inputs - one that has been executed without failure, and another that have yet to be executed. Then choose the next input as the one that is *farthest away* from the non-failing inputs. The concept of distance would need to be determined in a domain specific setting. There have been a number of articles that have received a significant number of citations on the topic of ART [39][38][40][41].

However, it should be noted that this line of research has caused some controversy, as is evidenced by Arcuri et al. [21] in the paper entitled "*Adaptive Random Testing: An Illusion of Effectiveness?*". They caution that ART may not be effective in some situations, and that the previously published results were based either on simulations or case studies with unreasonably high failure rates.

No literature was found where ART had been applied to the bioinformatics area. Despite this, there could be scope to apply ART to the testing of genomics sequencing pipeline. Currently, one method of testing a pipeline is to input a very small subset of

FASTQ files as input. The problem with this being that the test does not deal with the diversity of the total input population. ART could be employed to enhance this method, by choosing more input that is "farthest away" from the previously validated input subset.

2.6.2 Metamorphic Testing

Metamorphic testing (MT) is a relatively new area of testing, first introduced by Chen et al. in 1998 [3], as a method to deal with situations where there is no oracle. Rather than verifying individual output values, multiple related input test data are executed by the same program under test, and their corresponding outputs are examined to determine if known relationships hold. Each particular domain-specific relationship is termed a *metamorphic relation (MR)*. Chen et al. [3] gives an example of this in a graph theory setting. Suppose there is a program, SP(G, a, b), that finds the shortest path from node *a* to node *b* in an undirected graph, *G*. One metamorphic testing "relation" would be: *SP(G, a, b) = SP(G, b, a)*. That is, the output by running the program to find the shortest path from *a* to *b* should be the same as running the program to find the shortest path from *b* to *a*. If a MR does not hold during testing, the program is incorrect.

Outside the field of bioinformatics, MT has been used in the testing of: web services [42], compilers [43], feature models [44], machine learning [45], and partial differential equations [46]. Whilst not directly related to this research, these contributions demonstrate the applicability of MT to a wide range of problems. In addition to this, Liu et al. [47] contribute to the field by determining that a smaller number of diverse MR's may be more helpful than a large number of less diverse MR's in identifying problems via MT.

Within the field of bioinformatics, there have been a number of studies that establish the case for MT. In 2009, Chen et al. [3] apply MT to two different bioinformatics problems: network simulation and, short sequence mapping - with the latter being more relevant to this area of research. The authors show that MT can be simple to apply, has the potential to be automated, and is applicable to a diverse range of programs. A key point of interest to bioinformaticians, is that the construction of relevant MRs draws on domain knowledge - which should come naturally to the thinking of someone working in this field. It is also made clear that MT also has its limitations, and just because all MR's are satisfied does not imply that the program is free of defects.

Another example of an application of MT to bioinformatics can be seen in research undertaken by Sadi et al. [48]. Here the topic is the verification of phylogenetic programs, an area where there is an oracle problem. The authors establish that both real and randomly generated inputs can be used to evaluate the effectiveness of MT. Whilst this area of bioinformatics is not directly transferrable to genomics pipelines, this research helps to validate some of the MT methodology and also highlights its utility in being able to be applied in diverse situations.

Finally, a 2014 paper by Giannoulatou et al. [4], actually applies MT to software components commonly used in the genomic sequencing pipeline - BWA and Bowtie (sequence alignment tools). Of particular interest in this research is the nature of the MR's for this domain. One example of a MR that was used is: "*MR1: Random permutation of reads. The reads in the FASTQ files are reshuffled… We expect the output mapping to be the same.*" The paper contributes both an example of how to create domain-specific MR's for computational components of the genomic pipeline, and also demonstrates how this can be used to highlight some shortcomings in these components.

The literature supports a case for MT in general, as well as for the specific area of genomic pipeline testing. However, there was no research to be found on how to make this accessible to practitioners in the area - or how to deal with the scalability issues that are often associated with a genomic pipeline. Also of note, there were no studies to be found in the literature that dealt with MT in relation to the genomics sequencing pipeline as a whole. These issues will be addressed in the coming sections of this report.

2.7 Cloud Computing

The US Department of Commerce's National Institute of Standards and Technology gives the following definition for cloud computing:
"*Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models.*"[49]

Essential Characteristics:

- On-demand self-service
- Broad network access
- Resource pooling
- Rapid elasticity
- Measured service

Service Models:

- Software as a Service
- Platform as a Service
- Infrastructure as a Service
- (Testing as a Service)

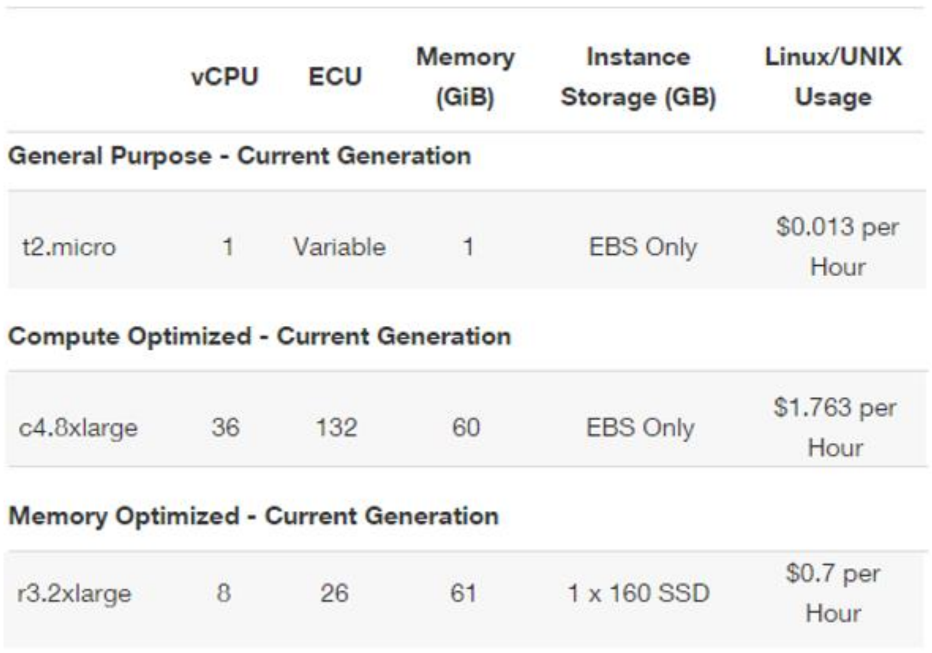
Deployment Models

- Private cloud

- Community cloud
- Public cloud
- Hybrid cloud

Cloud computing service providers maintain large-scale data-centres spread geographically throughout the world. These data-centres utilise virtualisation technology to share the same hardware with multiple users, and remove the need for the end-user to be concerned with physical hardware administration. Examples of cloud providers include: Amazon Web Services (AWS), Microsoft Azure, Google, Rackspace, Engine Yard, and Heroku.

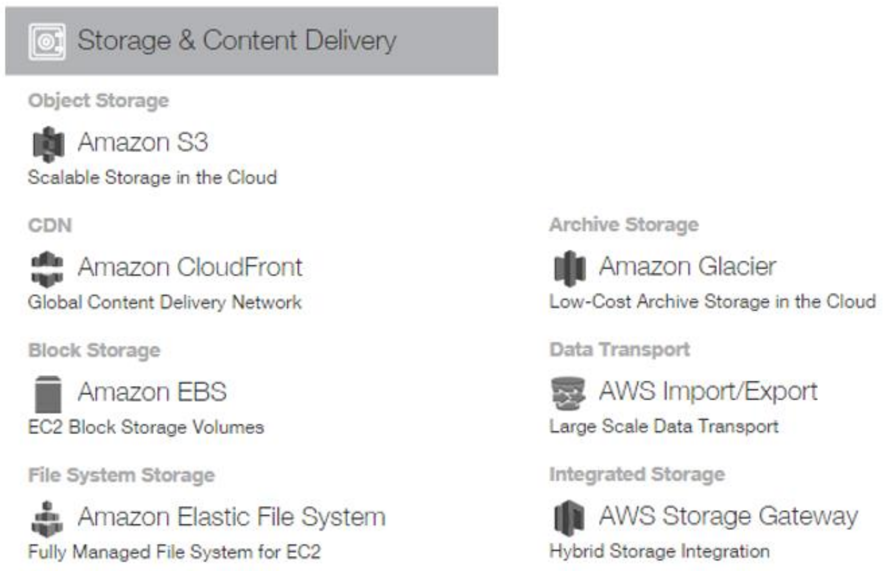
Cloud providers can offer access to "instances", which are the virtual machines for which a user can select from various operating systems to run on that instance. An instance typically starts within minutes, and the use may be charged an amount per hour to use such an instance.



	vCPU	ECU	Memory (GiB)	Instance Storage (GB)	Linux/UNIX Usage
General Purpose - Current Generation					
t2.micro	1	Variable	1	EBS Only	$0.013 per Hour
Compute Optimized - Current Generation					
c4.8xlarge	36	132	60	EBS Only	$1.763 per Hour
Memory Optimized - Current Generation					
r3.2xlarge	8	26	61	1 x 160 SSD	$0.7 per Hour

Figure 7 - Sample of AWS EC2 Instance Types

As well as traditional instances (servers or workstations in non-cloud terminology), cloud providers offer access to a range of other software and hardware offerings. Other offerings include compute facilities, storage and content delivery, database, networking, analytics, enterprise applications, mobile services, developer tools, management and security tools, and application services. Especially in a bioinformatics scenario, the user may have large amounts of data to store, and cloud providers have options available to cater for this storage.



Storage & Content Delivery

Object Storage
Amazon S3
Scalable Storage in the Cloud

CDN
Amazon CloudFront
Global Content Delivery Network

Archive Storage
Amazon Glacier
Low-Cost Archive Storage in the Cloud

Block Storage
Amazon EBS
EC2 Block Storage Volumes

Data Transport
AWS Import/Export
Large Scale Data Transport

File System Storage
Amazon Elastic File System
Fully Managed File System for EC2

Integrated Storage
AWS Storage Gateway
Hybrid Storage Integration

Figure 8 - Amazon AWS storage optionsSoftware testing in the cloud

Software testing, particularly in a bioinformatics environment, can be time and labour intensive. In addition to this, the necessary hardware and software to create a scalable testing environment could be prohibitive for some organisations. A software development organisation (or a bioinformatics department) may not have the resources at hand to test a new application in a distributed environment under different load or input scenarios. It makes sense to be able to have an automated environment in which a genomic sequencing pipeline could be tested. The cloud environment provides the necessary infrastructure to create such a testing service - due to its elasticity and pay-as-you-go user model. By utilising such cloud resources, the developer has access to greatly expanded testing resources.

Testing as a service (TaaS) on the cloud is described by Gao et al. [50] as a "s*ervice model, in which a provider undertakes software testing activities of a given application system in a cloud infrastructure for customers as a service based on their demands*". Gau goes on to describe how many of the cloud testing environments are concerned with testing in relation to load volumes, reliability, and performance. The testing solution in this research focuses on the verification and validation of the vGSP software.

As discussed by Riungu-Kalliosaari et al. [51], there are a number of organisations who have cloud-based testing offerings. Some of these commercial cloud-testing offerings include: *SOASTA* which provides a website testing product with load, performance and functional testing; *Sauce Labs* - tests web applications across multiple browsers, supports automation and, allows many tests to run simultaneously; and *BlazeMeter*, which is a self-service performance and load testing platform.

There seems to be little in the literature relating to any cloud-based service for testing bioinformatics software - despite there being many individual cloud-based solutions for genomic analysis, which one would classify as SaaS (software as a service) - as mentioned earlier in this document.

A rare exception to this lack of research in this area is from Highnam et al. [52] - who discusses a web-based testing solution that utilises the cloud to some extent. In the paper "*An analytical framework for optimizing variant discovery from personal genomes*" the authors present a genome comparison and analytic testing (*GCAT*) platform to enable performance testing and testing of analysis tools used in genomic sequencing pipelines. GCAT is a web-based tool where the user can compare the results of their own analysis pipeline using test data provided on the web site. The user downloads the data to their own system, runs the analysis, and then uploads the results back to the GCAT web site. GCAT then analyses the results on the cloud, and compares to known *gold standard* results for the given data set. Results can be visualised and compared with others who have also used the same data. Whilst this service is certainly novel in the bioinformatics space, it is still limited to cases where a *gold standard* is available.

Chapter 3. Aim

The previous chapter has shown that variant calling genomic sequencing pipelines (vGSP's) are complex systems that provide great utility for both researchers and society. Combined with the increasing amount of data that requires analysis, and the problems associated with establishing ways to test the software systems that are being created to analyse this data, there is a real need to improve testing solutions in this field. Very few papers in the related research focus on testing in this field.

The broad goal of this research is to contribute to the field of genomic medicine testing - to improve the testing of software in this area. This research aims to improve a number of areas of testing, including:

- Test design
- Accessibility of testing, and
- Technical simplification of the use of related computational resources

An automated cloud-based testing framework is a natural choice to provide a solution where bioinformaticians can test their own vGSP. Highnam et al. [52] describe a testing service where the users can input the results of their analysis and compare this with a reference set of results. There are several ways in which this approach can be improved.

Firstly, the testing framework should be fully self-contained, where a test run will automatically:

- Configure the cloud resources necessary for tests
- Install the user's vGSP on the cloud resources
- Execute the tests
- Report the results to the user

There is no need for the user to have a reliance on their own local computing resources to execute the tests. This will enable the testing to be carried out in a range of different situations - including on different operating systems, or by using different levels of computational power. This should also serve to reduce the barrier to testing for smaller organisations that may not have the necessary infrastructure or experience to create such an environment for themselves.

The second way in which we can improve on the Highnam approach is by using state of the art testing approaches such as Metamorphic Testing (MT) or Adaptive Random Testing (ART). MT will enable testing of input data without the reliance on a "gold-standard" set of inputs and outputs - although such a system could still provide this service. As MT and ART require a number of runs through the pipeline to establish final testing results, this is further reason to use cloud resources to host the testing - rather than performing this step locally. In the cloud environment, it will be possible to run many test cases in parallel.

The cloud-based approach as described above will also differentiate this research from prior research such as by Giannoulatou et al. [4], who do not provide a scalable implementation for their metamorphic testing. Nor do they focus their research on the whole of the vGSP, as this research does.

In addition to creating this automated cloud-based testing framework, this research aims to demonstrate its utility by testing a vGSP constructed from open-source software components that are considered to be “industry standards” in this area.

Chapter 4. Methodology

To achieve the goal of this research to provide the end user with an improved mechanism with which they can test their vGSP, this section outlines the design approach taken. The subsequent Implementation section describes in detail how these design concepts have been implemented.

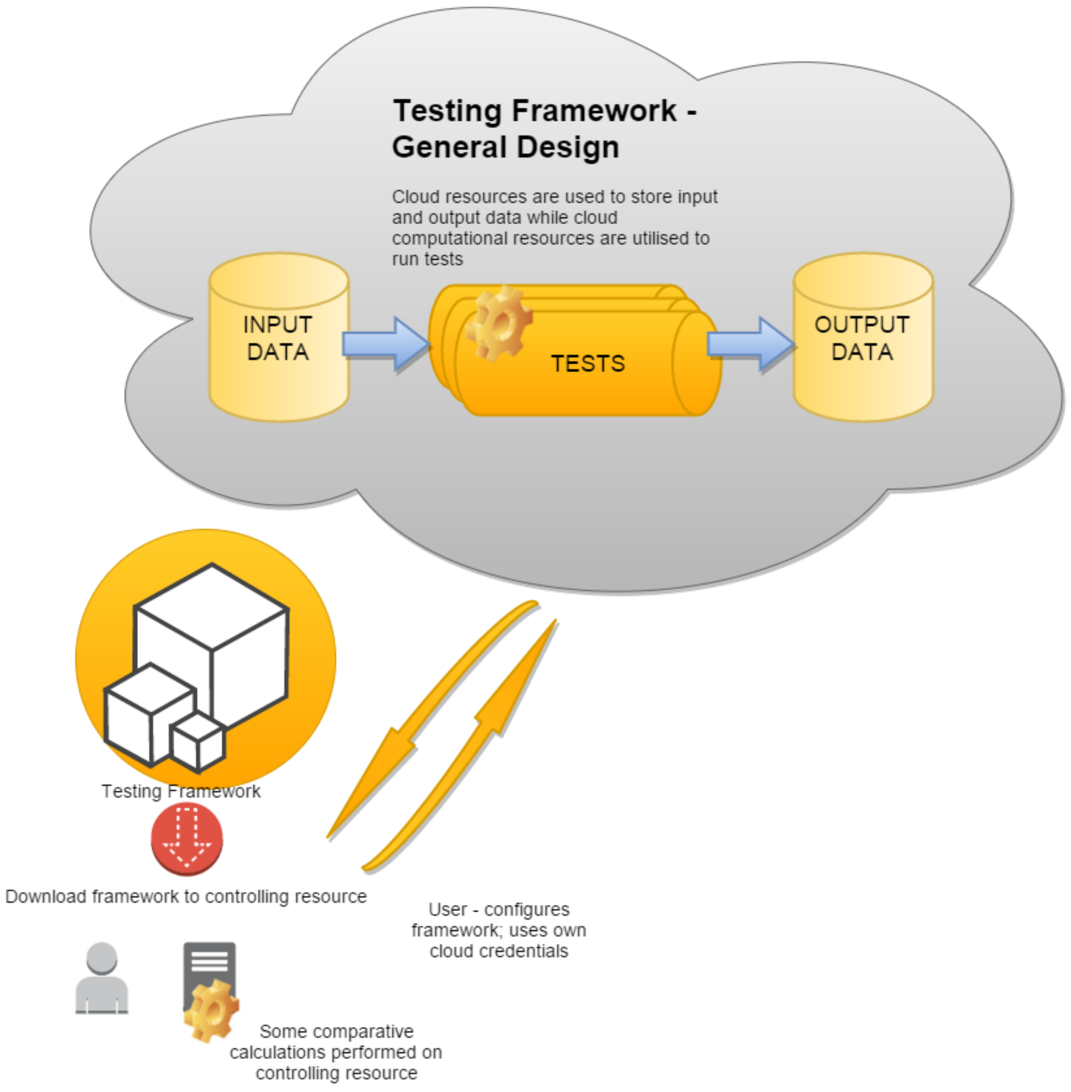

Testing Framework - General Design
Cloud resources are used to store input and output data while cloud computational resources are utilised to run tests
INPUT DATA
TESTS
OUTPUT DATA
Testing Framework
Download framework to controlling resource
User - configures framework; uses own cloud credentials
Some comparative calculations performed on controlling resource

Figure 9 - Testing framework – general design

4.1 Testing Framework

The testing framework is a software bundle that the user downloads to their local server or other computing resource – which becomes the controlling resource. A clear design decision was made to create this testing solution as a downloadable framework, as opposed to a fully cloud-based Testing-as-a-Service (TaaS) solution. One of the main reasons for this is that a TaaS solution raises issues with funding of such a solution, and how to create a payment model to which a user would likely subscribe. The chosen framework approach has the following advantages:

- The user uses their own cloud credentials and only pays the price that their cloud provider determines for the resources used
- Ability to fine-tune the level of security they require
- Maintain privacy of code in development
- Control over type and quantity of resources used
- Access to source code of the framework to make modifications possible
- Reduced complexity in maintaining cloud resources (a TaaS solution needs to allow for potentially large numbers of simultaneous users competing for the same resources)

An important design feature is the provision of a facility for the user to supply details on how to install and run their vGSP. This is necessary to install and run their vGSP on the cloud resources.

After initial configuration, the software is ready to perform its function of testing the user's vGSP. The testing process utilises cloud computing resources to perform the tests, store test data, and to store test results. The controlling resource co-ordinates all interactions with the cloud resources, and also performs comparative calculations to determine final test results.

Some key design concepts of the solution include:

- Making the framework flexible and configurable
- Shield the user from the complexity of technical concerns on either the controlling resource or any of the cloud resources
- Have control over cost/performance options
- Be able to change data inputs
- Have access to results and log files
- Provision of feedback to the user when the tests are running
- Be able to run multiple tests simultaneously

4.2 Tests

4.2.1 Metamorphic Testing

The core tests used in the testing framework are based on the Metamorphic Testing (MT) approach as described in the Background section. This approach is a key differentiator from traditional testing mechanisms for vGSP's which are based on comparing the output with limited "gold standard" data-sets. Specific Metamorphic Relations (MR's) have been developed to apply to the whole vGSP.

Most of the included Metamorphic Relations (MR's) have been adapted from previous research that has been performed in relation to one part of a vGSP – the alignment phase [4]. However, an additional independent contribution of this research has been to add a completely new "identity" MR, called MR0 in this implementation. The identity MR is one which checks for deterministic output from a vGSP. The list of MR's implemented in this research is provided in Table 1.

MR	Description
MR0	**Deterministic output.** The pipeline is run with the same FASTQ files on two separate occasions. The output should be the same.
MR1	**Random permutation of reads.** The reads in the FASTQ files are reshuffled. The permutation is the same for both files in paired-end files. We expect the output to be the same as the original output.
MR2	**Addition of reads.** The input reads in the FASTQ files are duplicated. We expect the output variants called to be a superset of the original output (at least the original variants should be called).
MR3	**Unmapped reads.** After initial alignment, only the unmapped reads are retained. We expect that no variants will be called.
MR4	**Mapped reads.** After initial alignment, only the mapped reads are retained. We expect the variants called will be the same as from the original file.

Table 1 - Summary of implemented Metamorphic Relations

4.2.2 Traditional / Simulated Testing

In addition to the MT approach, the framework also includes traditional tests where read data is generated through simulation based on a reference genome. The advantage of using simulated data is that there is no limitation based on the availability of known "real" data/output pairs – any number of simulated data sets can be introduced. With simulated data, two approaches are taken.

Firstly, reads are simulated from a reference genome without any mutations being introduced. After the vGSP under test processes this data, we expect no variants to be called.

Secondly, reads are simulated from the reference with mutations being introduced. Since the mutations that are introduced are known, we would hope that the output from the vGSP under test would call a very high percentage of these variants. Table 2 summarises the tests with using simulated data.

SI	Description (SI = Simulated Input)
SI0	**No mutations.** ART is used to generate simulated reads directly from a reference genome, without any mutations being introduced. The reads are used as input to the vGSP under test, producing a VCF file of variants called. We expect the output VCF file to be empty.
SI1	**Mutations.** VarSim is used to introduce mutations into a reference genome, producing a new "reference" genome. A "truth" VCF file is generated by VarSim. Reads are simulated from the mutated reference

	using ART. The reads are used as input to the vGSP under test, producing an output VCF file. We expect the output VCF to be close to the "truth" VCF

Table 2 - Summary of implemented tests based on Simulated Inputs

Chapter 5. Implementation

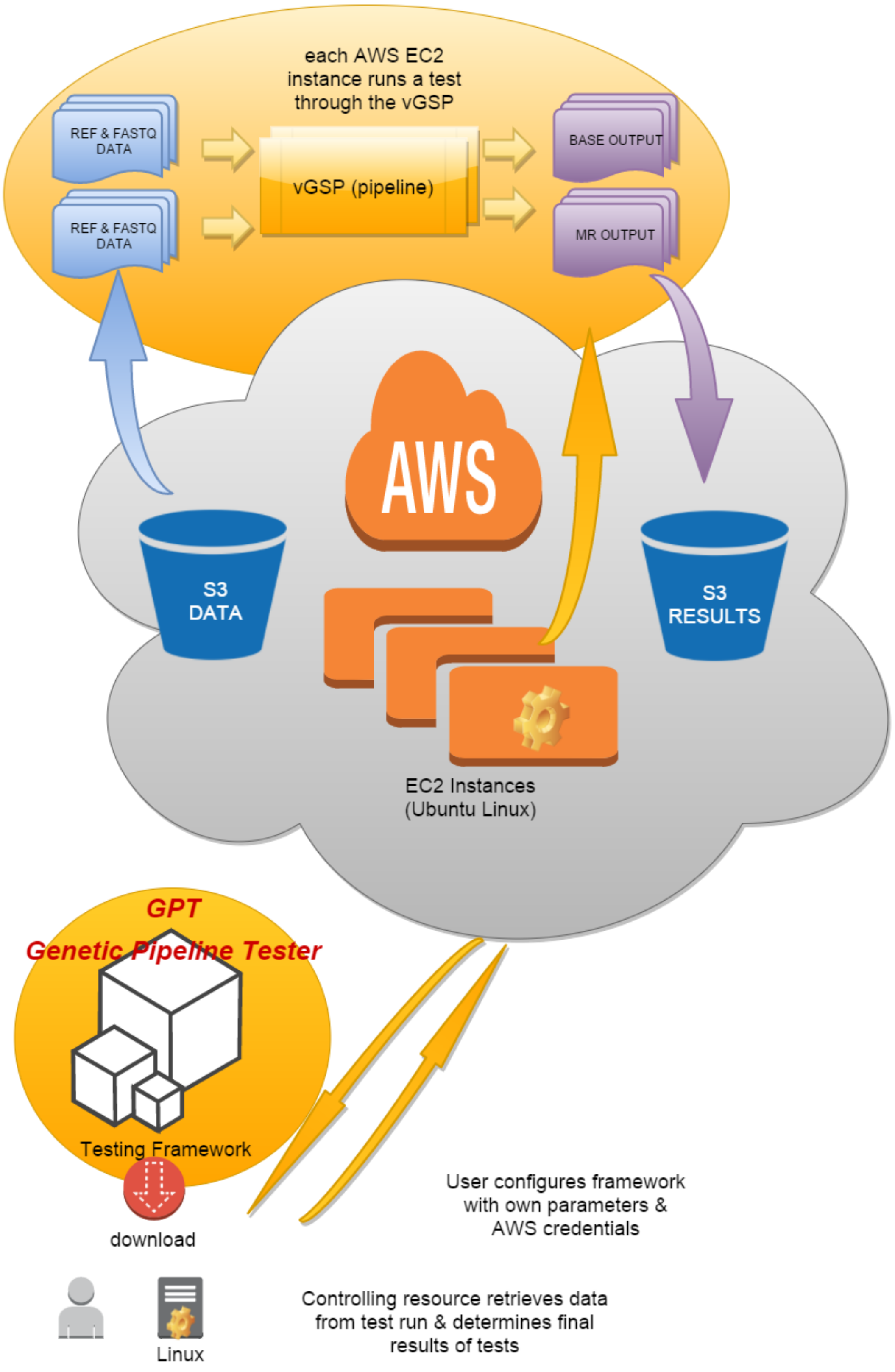


Figure 10 - Testing framework Implementation (GPT)

5.1 Overview

As a proof-of-concept, the framework, which has been named ***GPT*** (Genomic Pipeline Tester), has been implemented in a Linux environment, utilising Amazon Web Services (AWS) as the cloud provider. The basic premise of operation, as shown in Figure 10, starts with the user downloading the framework to a computing resource of their choice – which would most likely be the resource on which their vGSP is located. The user configures the framework to use their own AWS cloud credentials, and supplies configuration parameters and files that will enable the framework to be installed and executed on the cloud computing resources.

A range of metamorphic relations (MR's) have been developed to test the whole pipeline. These, along with the implemented simulated test data form the basis of testing which is implemented on the AWS cloud instances.

AWS cloud storage, called S3, has been selected to store both data for the tests, and the results and logs that are output from the tests. The controlling resource obtains these results from the S3 storage, and does any final processing to determine the test results. S3 is also used as a job management and control database for the GPT framework.

This chapter explains these implementation decisions in detail.

The code is currently under IP protection, as this work is part of an Amazon grant, and hence has not been publicly released. The code can be provided upon request.

5.2 Amazon Web Services – Relevant Background

Amazon Web Services (AWS) is a major cloud provider who offer reliable, scalable, low-cost infrastructure and services. Examples of large businesses using AWS infrastructure include Netflix, Philips, Spotify, Expedia, and many more. AWS cloud infrastructure and platform services include compute, storage and content, databases, networking, administration and security, analytics, application services, deployment and management, and mobile devices. This research primarily makes use of the compute, and storage services – in the form of EC2 compute resources, and the S3 storage service.

5.2.1 EC2

EC2 stands for Amazon Elastic Compute Cloud, and is a web service that enables a resizable (elastic) computing capacity on the cloud. EC2 provide users with access to varying levels of computing resources via a virtual instance, which a user can start with an operating system of choice in a matter of minutes. One or more instances can be started simultaneously, or as required depending on their computational needs.
Examples of instance types and their associated costs are given in the Background chapter.

A user can start an EC2 instance, using a particular operating system, and then configure that instance with any custom software or other configuration requirements. An image of this instance can then be created so that the whole configuration process does not need to be repeated each time the instance is restarted. Such an image is called an AMI (Amazon Machine Image).

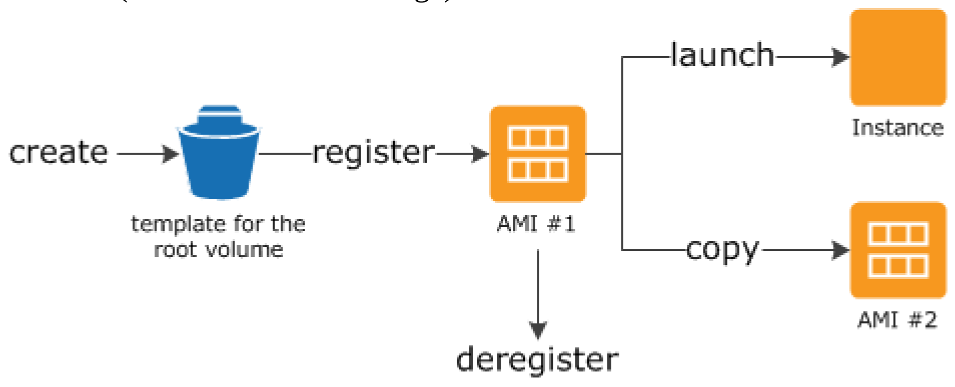


Figure 11 - Amazon Machine Image (AMI) lifecycle[53]

EC2 uses public-key cryptography to implement security for login information, including SSH (remote) access to Linux instances. Additional security for EC2 instances is provided in the form of a virtual firewall called a security group. The security group can control who can access an instance.

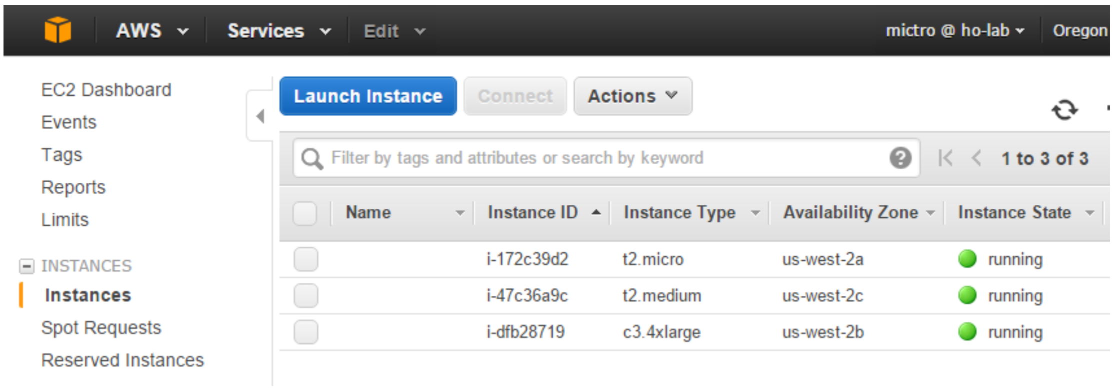
AWS EC2 provides a number of purchasing models for starting instances. The two models that are relevant to this implementation are *on-demand* instances, and *spot instances*. With on-demand instances, you pay a fixed amount per hour, and expect to receive a service that will remain running – unless there is some sort of hardware or software fault. Spot instances enable the user to bid a price for an instance – quite often at a price level far below that of an on-demand instance of the same type. However, the spot instance may be terminated if the current spot price in the AWS market exceeds the price limit for which the instance was originally purchased. Spot instances provide an attractive financial alternative for situations where the user could absorb the time cost of re-running their application should it be terminated.

5.2.2 S3

S3 is Amazon's Simple Storage Service, which provides secure, durable, and highly-scalable object storage. S3 enables users to retrieve large amounts of data from anywhere on the web via a URL address. The payment model is based on the amount of storage used and whether the storage is accessed from a location other than the location of the storage. A storage unit in S3 is called a bucket, and each file stored is referred to as an object. S3 is accessible via API's and high-level client utilities.

5.2.3 Management Console

AWS provides web-based graphical user interfaces to allow comprehensive monitoring and configuration of services.



Name	Instance ID	Instance Type	Availability Zone	Instance State
	i-172c39d2	t2.micro	us-west-2a	● running
	i-47c36a9c	t2.medium	us-west-2c	● running
	i-dfb28719	c3.4xlarge	us-west-2b	● running

Figure 12 - AWS EC2 management console

5.2.4 Client and API

AWS supplies a number of high-level clients (including a client for Linux EC2 instances, and an S3 client) that enable high-level programming of AWS services. These clients enable simple programming via common scripting methods for the platform of choice. Many of these high level commands provide an easy way to perform otherwise complex-to-program tasks such as transferring data to an S3 bucket in a parallel, fault-tolerant, and efficient manner.

In addition to this, AWS provides a lower-level, more powerful API (Application Programming Interface) – available in many common programming languages. For example the Boto API is provided for the Python programming language.

5.3 Platform and Programming Environment

The GPT testing framework has been implemented as a proof-of-concept using the Linux environment both for the controlling resource, and the cloud computational resources. The reason that the Linux platform has been chosen, is that many bioinformatics tools are designed specifically for this platform.

The Framework itself is written in Python v3.4, a choice that has been made due to the workflow required for the testing, the scripting nature required to tie together all the tasks that the framework is required to perform, ease of programming, and the fact that the performance of the framework itself is not under scrutiny – it is the user supplied program that is one of the major determiners for the overall time taken to complete the tests. In addition to this, many of the low-level automation tasks that are performed by the AWS service (like services provided by EC2) are themselves programmed using Python. The Boto3 Python API is used to integrate the python framework code with AWS services.

Linux bash scripts are also integrated into the framework – both to communicate between the controlling resource and the AWS instances, and also to integrate with the vGSP pipeline scripts – which are typically controlled by Linux bash scripts.

5.4 Tests

5.4.1 Metamorphic Tests

The actual implementation of the data manipulation required by the MR’s as described in the Methodology chapter, have been based on code already provided by [4]. However, as also discussed in the results section, modifications were necessary to enable large data-sets to be used for testing. One such modification related to MR1 – which is where the input reads are randomly sorted.

The read files were split into a number of smaller files, and these files were processed in parallel to be sorted as before – except with the temporary directory for sorting utilising the SSD drive. The smaller files were then recombined into one larger read file (for each of read1 and read2) – in a random order.

5.4.2 Tests Using Simulated Data

As explained in the previous chapter, part of the GPT framework design includes running tests with simulated data. To simulate reads from a reference genome, there are a number of open-source software programs that specialise in performing this task [36][54][37]. For this implementation, it was decided to use such an external open-source program to simulate reads because these specialised software have been developed with a large amount of consideration into simulated reads and mutations in such a way as to reproduce a read file that imitates what one may obtain in a real-world situation. The two related simulation packages selected were ART [36] (not to be confused with Adaptive Random Testing), a next-generation sequencing read simulator, and VarSim [37] – a simulation and validation tool for sequencing data. VarSim was used to generate mutations in the reference genome, and to validate output, while ART was used to generate the simulated reads.

ART simulates sequencing reads by utilising techniques that that aim to adhere closely to a real sequencing process, including a range of options relating to the production of sequencing errors for each individual read. ART can simulate reads for a range of commonly used NGS sequencing machines. The 1000 Genomes project [55], was the first project to analyse large amounts of human genomes (1000) to identify genomic variants in the sample population. One of the tools that was used by the project was ART.

VarSim is a tool-set whose purpose is to provide a mechanism for verifying output from a vGSP – by simulation of data with mutations. The user can utilise VarSim to introduce mutations into a reference genome, and the tool will generate a “truth” VCF to which the user can compare the output from their vGSP. The VarSim software has been used in this research to supplement tests from the MR's.

5.5 Specific Details of Framework

5.5.1 AWS Credentials and Security

The GPT framework enables the user to use their own AWS credentials to take ownership of costs and security for the testing that they complete. The user is required to have installed and configured the AWS Linux client, the Boto3 API, and created the following AWS security resources:

- An AWS security-group for access to EC2 instances
- An AWS security-key
- User credentials and policy to enable the created EC2 instances permissions to access AWS S3 resources. These credentials would normally be restricted to the specific S3 bucket that is to be used for the results and reporting.

To make these steps easier for the user, some of these steps could be provided in the form of automated scripts, or as an AMI image with the GPT framework pre-installed.

5.5.2 Instance Payment Model and Type

The GPT framework provides the user with the flexibility of choosing a payment model that is either spot instances or on-demand instances. The user edits the configuration file to select spot instances, and also provides a bid price for the spot instance. As discussed previously, the spot instance provides a financially attractive alternative to the on-demand instance types, with spot prices available at levels as low as ten percent of the on-demand price. The trade-off for the user is that their GPT run will be terminated if the market rate exceeds the spot price at which the instances were created.

The user also chooses the number and type of the instance – which determines the processor type, memory, and other computational characteristics associated with the resource.

5.5.3 AMI Image

When the GPT framework creates EC2 instances, it does so with a pre-configured AWS AMI, which contains a set of basic software components upon which a typical vGSP may depend. A summary of the details of the AMI are:

- Ubuntu 14.04, utilising hardware virtual machine virtualisation – for better performance, 64bit version
- Software: default-jdk, make, zip, unzip, g++, gcc, python-pip, awscli, boto3, dos2unix

5.5.4 SSD Option

AWS has different levels of storage for data that is required to be stored for an EC2 instance. The choice of the storage type will impact on the performance.

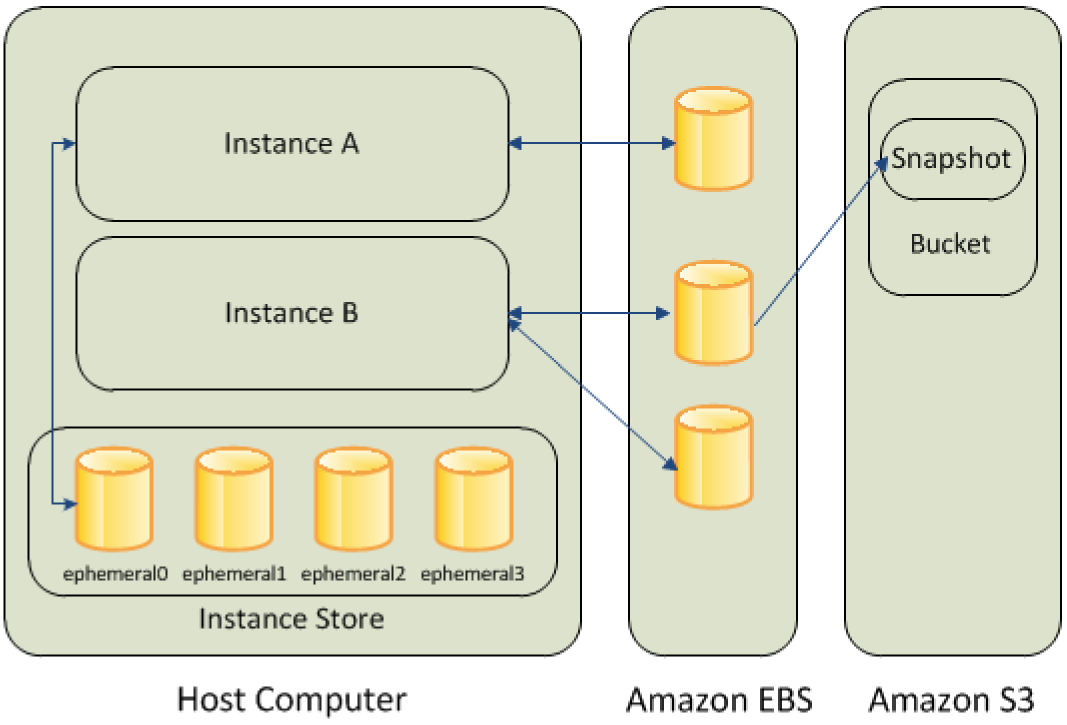


Figure 13 - AWS storage architecture [56]

Generally, the faster the I/O, the "closer" the storage device is to the instance. As a result of testing performed on an example vGSP, and varying the storage type, it was determined that it would be beneficial to give the user the option of selecting a local SSD device for use in executing the vGSP. This requires relatively difficult configuration for the novice, and the data on the SSD only lasts the lifespan on the instance (ephemeral). The GPT framework does this configuration for the user automatically.

5.5.5 Framework Configuration

Once installed, configuration of the GPT framework is controlled by a configuration file, called *gpt.cfg*, which the user can directly edit. The format of the configuration file is as specified in Python v3.4 *ConfigParser* specification. The style is similar to the Microsoft Windows *ini* format.

```
[aws]
region=us-west-2

[aws-instances]
# use spot instances - set to 1
use_spot=1
spot_price=1.60
instance-type=c3.8xlarge
count=9
user-data=gpt-system/user-data-ssd.sh
user-data-checker=gpt-system/user-data-checker.sh
dry-run=False
security-key=mictro-dev-key.pem

[install-pipeline]
pipe-install-script=install-pipeline-aws.sh

[run-pipeline]
run-script=pipe-exec.sh

[aws-s3]
script=copy-s3-data.sh
bucket=vccri.ho-lab.mictro.testing-project

[data]

```

Figure 14 - Beginning of GPT configuration file

There are currently six different sections in the configuration file for GPT. Three of these sections relate to AWS related configurations – where the user can supply details related to their own AWS preferences, such as their default region, the path to their AWS security key, or the name of the S3 bucket to be used for storage. Other AWS configurations include the important consideration of the type of EC2 instance used, the number of instances to be started, the choice of spot instances, and a spot bid price.

The data section is where the data and reference files are specified – that are required to run the supplied vGSP. These files include FASTQ paired-end read files, FASTA files, and related reference files required for calling variants. In addition to this, details of the simulated FASTQ read files, and truth VCF file are included.

Finally, there are sections in the configuration file relating to the installation and execution of the users' vGSP. The *[install-pipeline]* section is where the user specifies the script file that when executed, installs their vGSP on the AWS instances. The *[run-pipeline]* specifies the user script that executes the vGSP.

An example of a full GPT configuration file can be found in Appendix 2.

5.5.6 GPT Workflow

Figure 15 shows the workflow and key concepts involved when executing the GPT framework. Once a GPT session has been initiated, a unique session id - based on the current date and a sequential run number – is generated. This is referred to as the GPT id. The GPT id is then used to create a results directory on the controlling resource (the controlling resource is a Linux machine of the user's choice upon which they install the GPT system).

The GPT job manager then launches a synchronous job to start the AWS instances. The types and number of instances to start are as specified in the GPT configuration file. This includes the option of using either spot instances or on-demand instances. If the user has chosen an instance type that has a local SSD drive available (e.g. c3 & r3 instance types), this is configured for use. When all instances are ready for use, the GPT job manager switches to asynchronous mode – which means that jobs are launched, and the GPT manager is returned to processing immediately (the manager does not wait for the job to finish). This is necessary as there may be many instances that are ready to receive jobs, and each job may take in excess of five hours to run.

Particularly relevant to asynchronous jobs, the GPT system monitors job status via the S3 bucket. On each instance, each life-cycle stage of a job is recorded by sending a small log file to the S3 bucket – in a directory identified by the current GPT id. By examining the contents of this bucket, the GPT job manager is able to determine if a job has completed successfully or failed to execute correctly.

Each instance first receives jobs to install the vGSP, and to copy the necessary data from the S3 bucket to the instance. Following this the job manager assigns jobs relating to tests to each instance via a round-robin scheduling strategy, which is described separately in this chapter. This allows the user to select the number of instances less than the total amount of jobs.

When all jobs are complete, the controlling resource downloads the results – which comprise the output VCF files from the various tests. For each test, the output VCF is compared with the expected value, according to the particular MR or truth value from simulated data. The results of this comparison are printed to screen and saved to a report.

After a GPT run is complete, the instances are still running. At this stage, the user could potentially make changes to their code & re-run just the tests without having to restart the instances or copy the data from S3.

5.5.7 GPT Load Balancing and Scalability

The nature of the testing involved in this research leads to a primarily black-box approach to testing. The user supplies a piece of software about which we can make limited assumptions. The user supplies a vGSP which comes with its own performance issues, for which they may use some sort of parallelisation strategy. This limits what the GPT framework can do in terms of parallelising multiple tests. Despite this, there are a number of ways that the GPT system provides scalability.

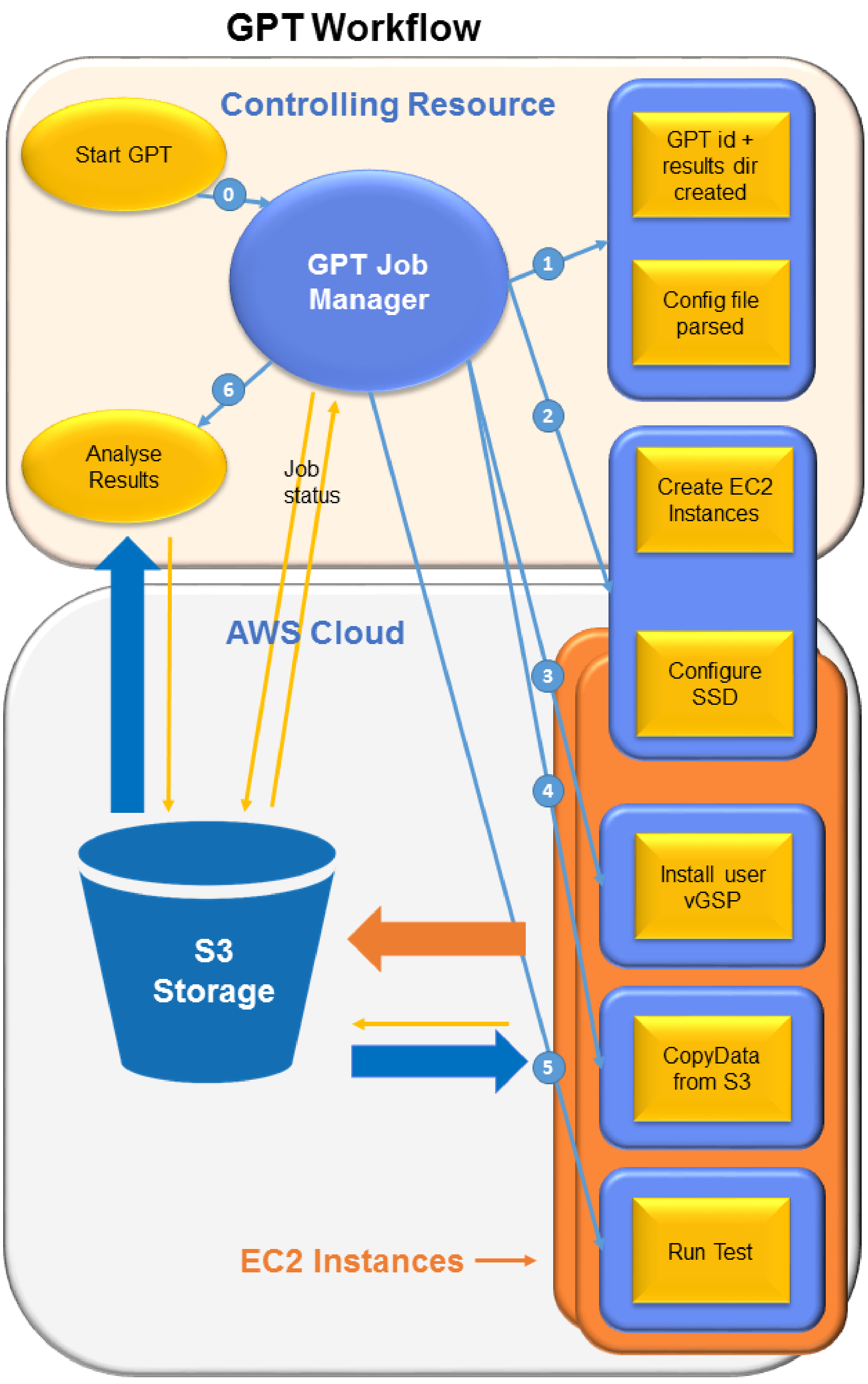


Figure 15 - GPT workflow

Firstly, the type of instance chosen can scale with the data size of the test. If the user decides to configure the test environment with a small data-set, both for reference and reads, then a smaller instance type may be sufficient. As the data increases in size, the GPT framework allows the user to select increasingly larger instance types.

The second way in which GPT can scale, is to increase the number of instances to run. For very large data sets, where each test can take five or more hours to run, the user may wish to select the number of instances to be equal to the number of tests (or more accurately – the number of times their vGSP is run). For example, if the user's pipeline takes five hours to run on a particular instance type, and there are ten runs required to satisfy the requirements of the set of tests, then the total test time would be at least fifty hours for a single instance, versus five hours with ten instances.

In terms of load balancing, the GPT framework utilises a round-robin job-scheduling approach for executing test jobs. As most tests are of a similar size, each instance is given one job at a time to put in its job queue, from the list of available jobs. This process is repeated until there are no jobs left to distribute. An instance is not sent a job to execute from its jobs list until it has completed any outstanding jobs.

Chapter 6. Results

Recall that the object of this research is to improve testing in genomic medicine, and in particular variant calling genomic sequencing pipelines (vGSP's). The developed proof-of-concept system – GPT – includes both traditional and metamorphic testing techniques, and the whole framework is designed to be flexible, cost-effective, and minimise the technological burden that is placed upon the practitioner that wishes to test their own vGSP.

To test the framework, a real-world, industry-standard pipeline has been selected as the vGSP to test. Installation and execution scripts were coded for this pipeline, and an AWS (AMI) machine image was built containing some basic programs that one may expect to find in the vGSP software environment. Real-world input data were select for inputs, and simulated datasets were also generated. The GPT program was then configured and executed.

This chapter describes the results from this process, including information on metamorphic and simulated tests, timing information, and costs involved. Note that the performance of this software is not directly compared to any other system – as there is not a system such as this to which a direct comparison can be made. Also note that, apart from the resource scalability on offer from this cloud framework, it is the user's vGSP (supplied to us as a virtual "black box") that will determine how long each test takes.

6.1 Test Configuration Details

The pipeline under test consists of a number of separate open-source components that are combined together in a Linux bash script - BWA, Picard Tools, SAMTools, and GATK. The reference data is the whole human genome reference hg19, and read data includes a barcode (terminology for a portion of) from a whole exome (the coding part of a genome) sequencing run for a particular individual. This data is actual read data that has been used in a familial study of congenital heart disease at the Victor Chang Cardiac Research Institute.

For simulated data, as described previously, the ART and VarSim programs have been used to generate the data. The read data has been generated at 30X coverage for only chromosome 11 in hg19.

A summary of the specifics of the test configuration is given in the Table 3 and Table 4.

6.1.1 AWS

Item	Description
AMI	Amazon Machine Image

	Id = ami-2221c511 Architecture = x86_64 Virtualization type = hvm Platform = Linux Linux version = 14.04 Root Device Type = ebs Based on ami from: https://cloud-images.ubuntu.com/releases/trusty/release-20150123/
AMI Software	The AMI was created after manually installing the following software: - Update of Ubuntu software - Java (default-jdk), gcc, g++, make, zip, unzip, dos2unix, awscli, python-pip, boto3
Payment Model	Spot instances Bid price = 1.60 per hour * (* the user is charged the market rate for this instance type, and the instance will be terminated if the market price exceeds the bid price)
Instance Details	Type = c3.8xlarge (32 CPU's, 60G RAM, 2x320G SSD) Number of instances = 9

Table 3 - AWS test configuration

Note that the AWS configuration utilises spot instances to minimise the cost of the testing process, and the instance type selected utilises solid-state drive (SSD) technology as discussed previously. The number of instances was chosen to be equal to the number of times that vGSP needs to be run to satisfy the set of tests.

6.1.2 Data

Table 4 summarises the data used in the tests. These data files are copied to each instance as part of the configuration process. Note the significant amount of total data required for these tests, which could be multiplied by a factor of 10 or more to process a full genome.

ITEM	Description	Size (GB)
FASTQ	Real sequencing data from the Illumina platform – paired-end reads (two files); one barcode from a whole exome	3.7 (R1) 3.7 (R2)
	Paired-end simulated reads from chromosome 11 of hg19 – no mutations – 30X coverage	1.8* (R1) 1.8* (R2)
	Paired-end simulated reads from chromosome 11 of hg19 – mutations – 30X coverage	2.3* (R1) 2.3* (R2)
FASTA	Hg19 reference genome	2.9

	Reference genome for chromosome 11 of hg19	0.2
VCF	Supporting VCF files with known locations of INDELs and SNPs	7
BWA	Associated index files needed for BWA – in relation to hg19 reference genome	5
TOTALS		31 GB

Table 4 - Data used in tests (compressed)*

6.1.3 Details of vGSP under Test

As mentioned earlier in this chapter, the pipeline chosen for use as the test subject for this research is one comprised of a number of industry standard tools. These tools include BWA for the alignment of reads to the reference genome, Picard Tools and SAMTools for the manipulation of SAM/BAM files, and GATK for pre-processing of BAM files and the calling of variants. The file is a Linux Bash script, and excerpts from this file are included as an appendix.

Along with this pipeline script file, there are a number of other user files supplied to fulfil the user part of the testing. All these files are copied to a user-files directory in the GPT framework. The extra files include:

- An installation script that will be used to install the user vGSP on the instances. Part of testing the GPT framework included building an installation file for this example vGSP
- A script that contains the commands to execute the supplied vGSP.
- Any supporting files required to complete installation. For example, this testing included some downloaded installation binaries, as well as script code that automatically downloads and installs the latest version of some of the pipeline components. The binary file may be necessary if such downloads require a registration process – as is the case with the GATK software.

6.2 Preliminary Testing

6.2.1 Instance Start-up Times

An important consideration in testing is the total time it takes to complete the tests. If this time is too large, then this can become a barrier to testing. One consideration when utilising a cloud-based framework is the extra overhead associated with starting such resources. This time would be an additional overhead when comparing some sort of local server that is always on.

However, it is important to keep in mind that a single run through a vGSP for a whole genome can take more than twenty-four hours or more – depending on the performance of the particular vGSP. To run a small test with a cut-down version of a reference genome, and very small read data, may takes only a few minutes to complete.

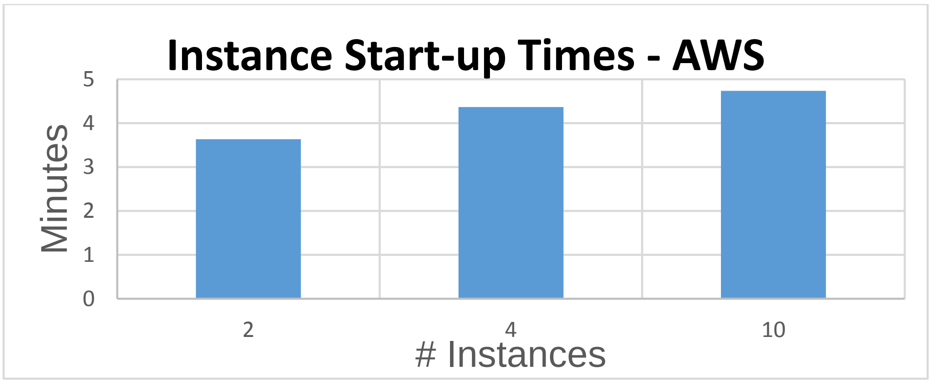


Figure 16 - Instance start-up times for varying number of instances (t1.micro)

Notwithstanding the abovementioned points, it is useful to demonstrate that the choice of AWS as a cloud provider will enable the number of instances to scale with the number of tests without causing a significant increase in the time to start these instances. Figure 16 shows the results of some basic testing that showed that instance creation overhead did not increase linearly with number of instances – rather it increased at a much slower rate. This is good for the testing environment.

It should be noted that the start-up time for nine instances of type c3.8xlarge, with SSD configuration, and spot instance payment model, was closer to ten minutes. This extra time would involve the longer start-up time required for more extensive hardware, the configuration of the SSD via a user-data script, and also the wait-time for the spot instance request to be fulfilled.

Certainly for tests taking hours, this timing information does not provide a barrier to use. However, if very small testing data was being used, the start-up time of the instances may represent a significant proportion of the total test time.

As an initial part of this project, the Microsoft Azure platform was considered, but ruled out due to the inability to reliably start multiple instances simultaneously in a reasonable amount of time. In addition, the azure process involved modifying a large JSON script (hundreds of lines), to be able to start the instances simultaneously. At the time Microsoft was in the process of trying to improve the process.

6.2.2 Accessing "local" SSD resources on AWS EC2

Part of the initial testing phase in the research was to determine which EC2 storage option was optimal for use during the computation phase of the vGSP. The concept of storage, and the ephemeral SSD was raised in the Implementation chapter. It was noticed that initial runs of the vGSP when using EBS (network) storage was significantly slower than experienced with local computing resources available at VCCRI. However, subsequent runs had improved performance.

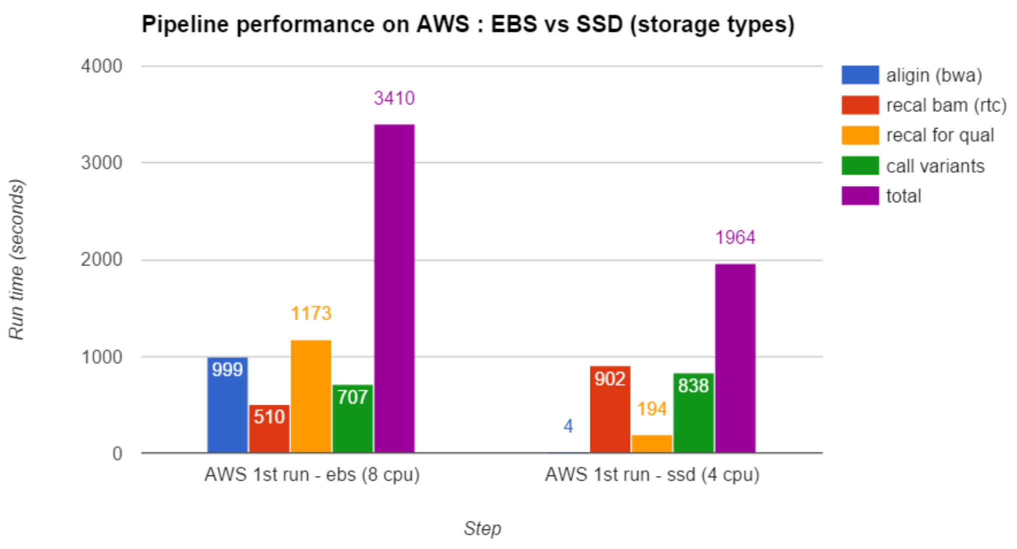


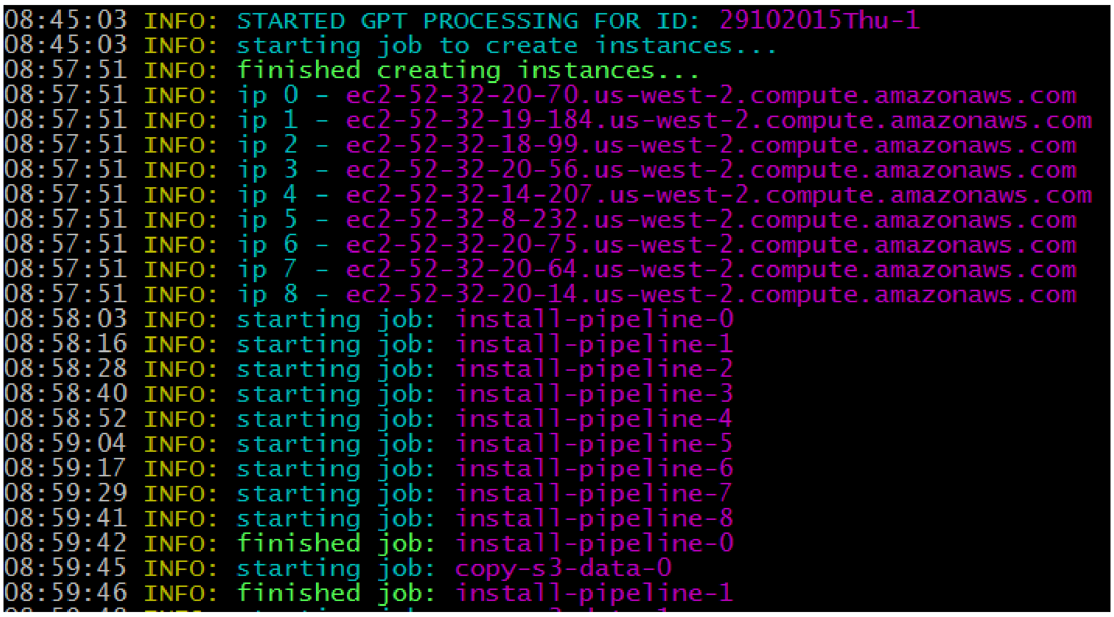
Figure 17 - AWS storage comparison – EBS vs SSD

To investigate this phenomenon, a test was devised to compare the use of an EC2 instance type that has an SSD drive available, and compare runs of the vGSP on this instance – using SSD or EBS. The results can be seen in Figure 17, which shows the time taken to complete various components of the vGSP, using a small data-set. The data on the left of the figure is the EBS data, and the right cluster is the SSD data. This shows a startling increase in performance by moving from an EBS to a SSD drive. This is compounded by the fact that the EBS run was done on a machine with double the specs of the SSD machine (EBS used r3.2xlarge – 8 CPU's, while SSD used r3.large – 4 CPU's). Note the time taken for the BWA step on the EBS instance was 999 seconds compared to 4 seconds for the same step on the SSD. Subsequent runs showed the SSD to have a performance advantage over the EBS drive, but the performance was much closer to that of the SSD than the first run.

With this evidence, it was thought beneficial to the user to include an option for them to have an SSD automatically configured for use in testing.

6.3 Main GPT Test

The GPT framework was used to test real and simulated data as described earlier in this chapter. Once the framework is installed and configured, a test run can be started at the Linux command-line with a python 3 command that includes the name of the GPT program and the desired option. In this case, the GPT command was *"gpt –all"*, which instructs GPT that a full test run is required – including starting the instances. As the testing progresses, logging output is printed to the screen, and also to the report file, as is shown in Figure 18. A full version of the report file generated whilst producing these results can be found in the related appendix.



```
08:45:03 INFO: STARTED GPT PROCESSING FOR ID: 29102015Thu-1
08:45:03 INFO: starting job to create instances...
08:57:51 INFO: finished creating instances...
08:57:51 INFO: ip 0 - ec2-52-32-20-70.us-west-2.compute.amazonaws.com
08:57:51 INFO: ip 1 - ec2-52-32-19-184.us-west-2.compute.amazonaws.com
08:57:51 INFO: ip 2 - ec2-52-32-18-99.us-west-2.compute.amazonaws.com
08:57:51 INFO: ip 3 - ec2-52-32-20-56.us-west-2.compute.amazonaws.com
08:57:51 INFO: ip 4 - ec2-52-32-14-207.us-west-2.compute.amazonaws.com
08:57:51 INFO: ip 5 - ec2-52-32-8-232.us-west-2.compute.amazonaws.com
08:57:51 INFO: ip 6 - ec2-52-32-20-75.us-west-2.compute.amazonaws.com
08:57:51 INFO: ip 7 - ec2-52-32-20-64.us-west-2.compute.amazonaws.com
08:57:51 INFO: ip 8 - ec2-52-32-20-14.us-west-2.compute.amazonaws.com
08:58:03 INFO: starting job: install-pipeline-0
08:58:16 INFO: starting job: install-pipeline-1
08:58:28 INFO: starting job: install-pipeline-2
08:58:40 INFO: starting job: install-pipeline-3
08:58:52 INFO: starting job: install-pipeline-4
08:59:04 INFO: starting job: install-pipeline-5
08:59:17 INFO: starting job: install-pipeline-6
08:59:29 INFO: starting job: install-pipeline-7
08:59:41 INFO: starting job: install-pipeline-8
08:59:42 INFO: finished job: install-pipeline-0
08:59:45 INFO: starting job: copy-s3-data-0
08:59:46 INFO: finished job: install-pipeline-1
```

Figure 18 - GPT log output

6.3.1 Timing Information

The timing details for the full GPT run are shown in Figure 19. The total time for GPT to complete all tests for this data configuration is close to five and one half hours. This time is determined primarily for the vGSP to complete a single test. This is the case for this GPT configuration, because nine instances were created – one instance for each job (as required to satisfy MR0-0 through SI1-1).

MR3 takes significantly less time than the other runs through the vGSP – twenty four minutes compared to around five hours for MR1-1. This expected as MR3 discards any mapped reads. MR3 is not expected to call any variants. MR2 is also shorter in duration to MR0, MR1, and MR4 due to the fact that it runs a version of the user's vGSP that ignores duplicates, and this removes several processing steps. Finally SI0 and SI1 are quicker to complete because they only operate on one chromosome of data – but still take a significant amount of time to process because of the 30X coverage of reads.

Another observation in relation to timing is that the analysis task is finished within two minutes, and this is an indication that this task should be well within the capabilities of the user's local resource.

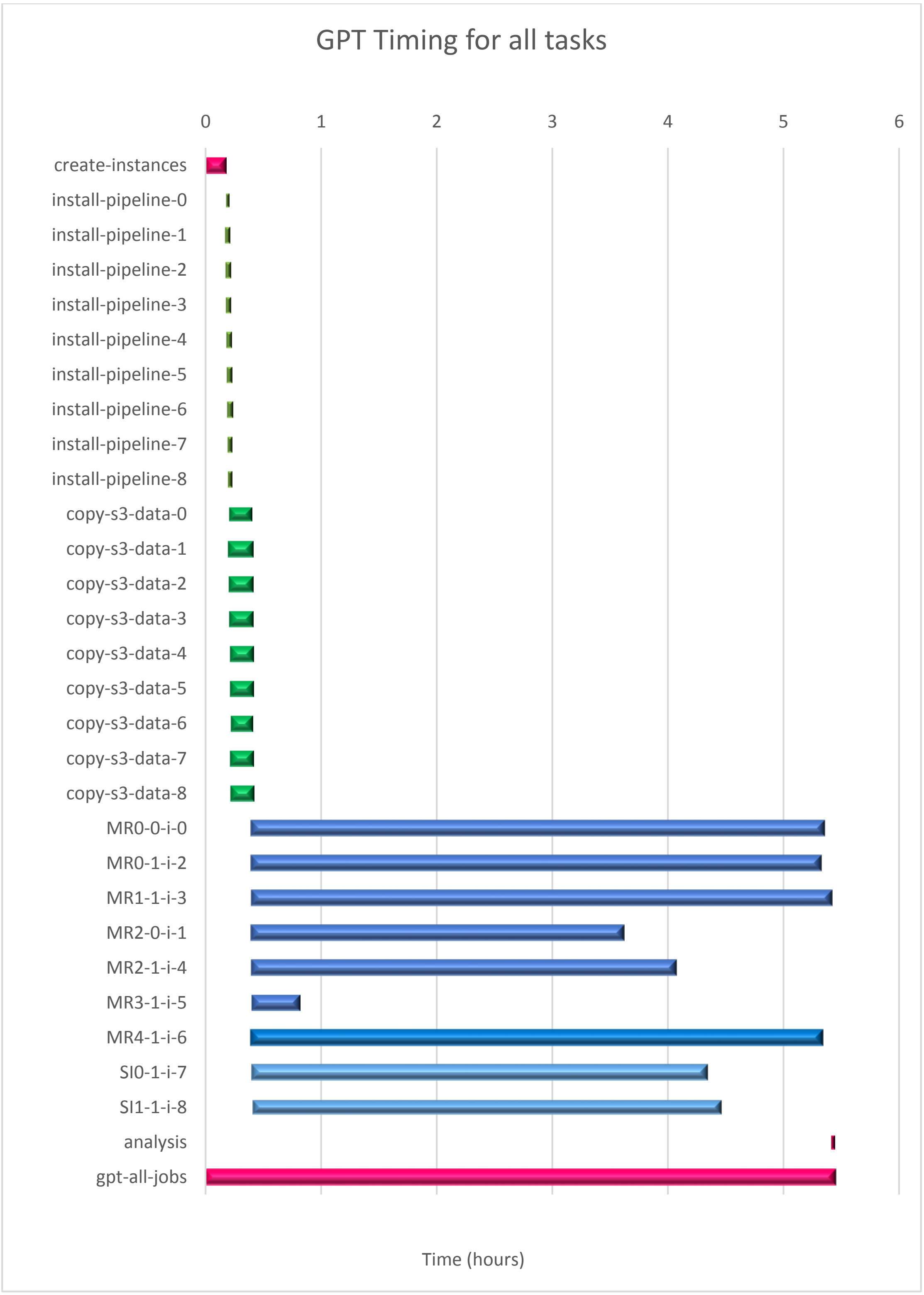


Figure 19 - Timing information from a full GPT run. Note the final digit in the task name refers to the instance id; e.g. install-pipeline-0 refers to install-pipeline on instance 0. Utilising nine instances – one for each vGSP task.

If one assumes that the user's vGSP is designed to take advantage of local resources in a parallel manner – and by extension that multiple instances of the vGSP cannot run simultaneously on the same local resource efficiently – one can make an observation about the time-savings provided by the cloud-based framework compared to using a local resource. It is further assumed that the local resource has the same computing power as one cloud instance as configured in this test. Figure 20 shows that, in the scenario described, GPT shows a speedup of close to seven, as compared to the local resource

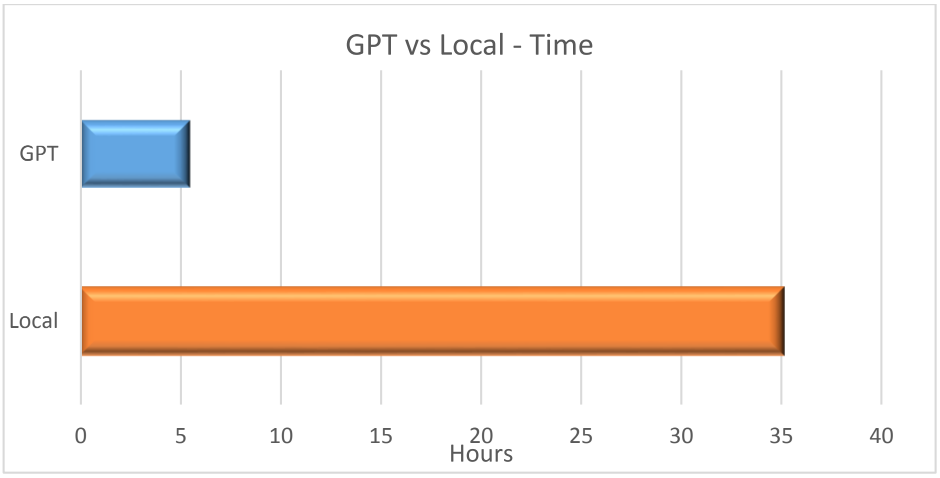


Figure 20 - GPT vs Single local resource. Comparison of total time taken to complete all the jobs as configured here – GPT vs single resource. GPT using 9 instances; local resource same specification as single instance; local instance does not include instance start-up time or time to copy test data

Initial testing of GPT was completed utilising a small data-set with only about 10,000 reads in each read file. This meant that, for the main read data FASTQ files, they were both 2.5MB in size compared to the full data-set size of 3.7GB each for the read1 and read2 paired-end files. When testing with this small data size, which was convenient for fast runs, there were no issues in relation to performance for the manipulation of data for the various MR's. For example, in MR1, the data is randomly sorted – remembering that this is not as straight-forward a task as it would first appear because each record is four lines, and both the read files must be sorted in a synchronised fashion. However testing on the complete data revealed that there was indeed a performance issue with the manipulation of data for MR1.

Using the code as adapted from [4] to randomly sort the FASTQ files resulted in an addition of two hours and twenty minutes to the run-time of GPT. In fact, without modification, the code failed to run, as the Linux sort command defaults to use the */tmp* directory for sorting – and this directory is an EBS drive of size 8GB – the sort was causing the root EBS volume to run out of storage space.

A solution was found by changing the MR1 code so that it utilises the larger SSD volume for sorting, and also to parallelise the code with the use of the GNU Parallel program[57]. After the introduced optimisation, the total time to randomly sort the FASTQ files reduces from two hours twenty minutes to approximately ten minutes. This represents a speedup factor of fourteen. Figure 21 summarises the effectiveness of this improvement.

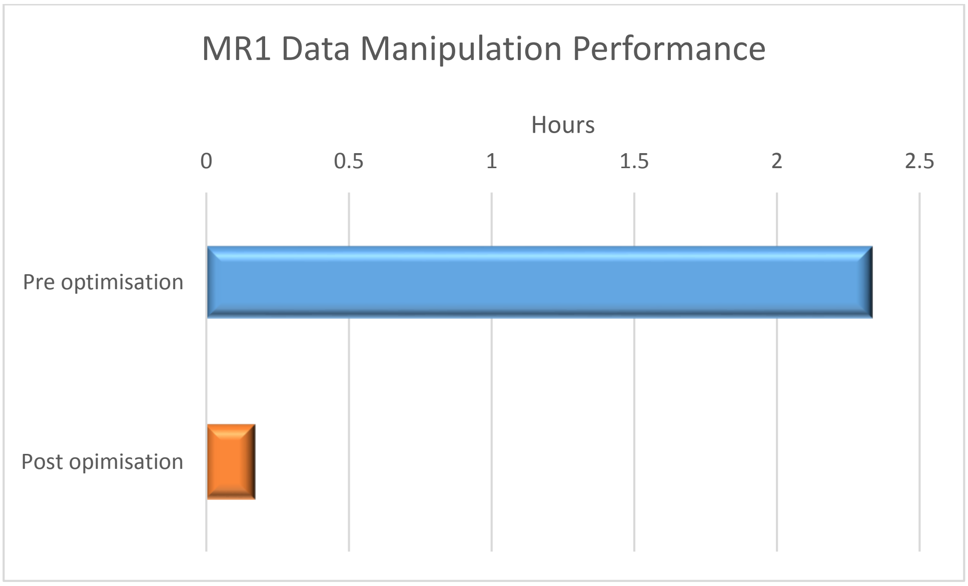


Figure 21 - MR1 Data Manipulation. Time taken to randomly sort the FASTQ input files for MR1 – pre & post optimisation. The optimisation technique splits the input files into smaller files and processes them using GNU Parallel.

Timing results are summarised in the generated report, which is stored in the results directory associated with the GPT id for that particular run.

6.3.2 Cost

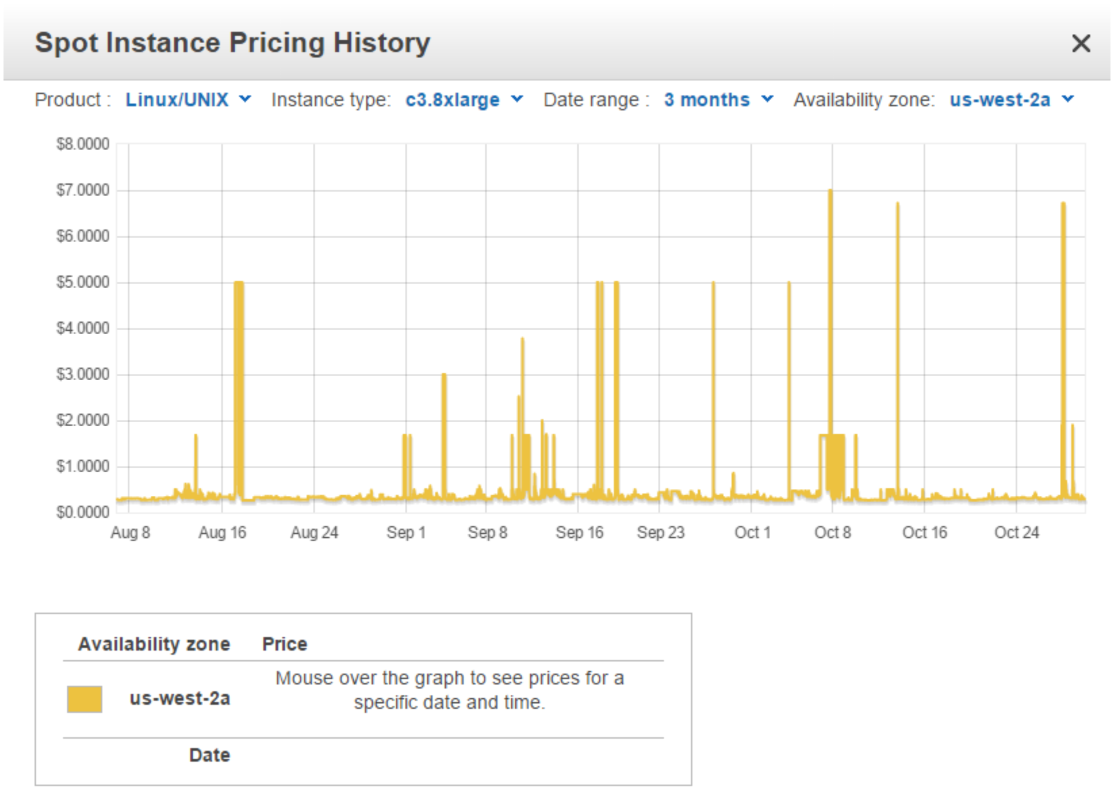
To give the potential user of GPT an indication of the cost involved for usage of cloud resources, this section details the costs for a configuration as described earlier in this chapter. One of the advantages of using a cloud-based framework is the pay-as-you-go model for payment. There are no up-front charges, and the user only has to have registered for an account with the cloud provider – AWS in this instance. This framework utilises two main cloud resources: EC2 instances, and S3 storage.

For EC2 instances, the on-demand price for one instance of type c3.8xlarge in the *us-west-2* region is 1.68 USD per hour [58]– as of the time of writing. In comparison, the spot instance price was less than 0.40 USD per hour during the testing phase. Table 5 shows the total cost for a full GPT that actually took place in testing. The results show how the use of spot instances reduced the cost of a run from $90 to $21, a reduction of approximately 76%.

Payment Model	Cost of EC2
On-demand	$90.72
Spot	$21.60

Table 5 - Cost of EC2 instances in a GPT run – On-demand vs Spot. As per configuration described here with nine instances, and rounding up to the next hour for the total run-time. Based on an on-demand hourly rate of $1.68, and a spot rate of $0.40.

Figure 22 shows the spot price history over the last three months. Note there are some spikes in the spot price that well exceed the on-demand price for some short periods of time. For example, there was some time in early October where the spot price reached $7.00 for a short period of time. This test configuration used a spot bid price of $1.60 to perform all testing, and in the many test runs that were started, none were terminated due to price. This would not have been the case should the price have spiked above $1.60 during testing.


Spot Instance Pricing History

Product : Linux/UNIX Instance type: c3.8xlarge Date range : 3 months Availability zone: us-west-2a

Availability zone	Price
🟨 us-west-2a	Mouse over the graph to see prices for a specific date and time.
Date	

Figure 22 - Spot price 3 month history – 2015 (AWS console)

There are also costs associated with the storage and transfer of data in relation to the AWS S3 storage service. The cost model includes a charge for storage, transfer of data to another region, and for transfer of data from S3 to the internet. These charges are $0.03, $0.02, and $0.09 per GB respectively [59]. The storage charge is a per month charge. Unless the user is regularly transferring hundreds of GB of data to or from the S3 storage, the storage cost will be negligible.

The main storage cost to a user to set-up and run the tests here would be the monthly storage cost – which is less than $1.00 for 30GB. The only transfer fee is the fee to transfer results from S3 to the local resource over the internet. The sum of the sizes of all the results files (VCF) for a run using this data is in the order of are in the order of 356MB, and this results in a transfer fee of around $0.03. It is sufficient to say, that in the case of this test scenario, the cost for the GPT run is determined by the EC2 instances.

6.3.3 Test Results

When the GPT framework runs, it completes tests MR0 through MR4, SI0 and SI1 – as described previously. A summary of the results is shown in Table 6.

Test	Result
MR0	Passed *
MR1	Failed
MR2	Failed
MR3	Passed
MR4	Passed
SI0	Passed
SI1	Failed

*Table 6 - Test Results. * Variants called are the same. However some information in the two VCF files differ.*

Before discussing these results, it is instructive to examine a portion of the output from the report file generated for this test run – shown in Table 7.

```
RESULTS
14:12:54 INFO:     MR0: PASSED
14:12:54 WARN:     MR0: variants same, but some other vcf values different
14:13:07 INFO:     MR1: FAILED
14:13:07 INFO:     MR1:
                   results/29102015Thu-1/MR1_FN.vcf 48
                   results/29102015Thu-1/MR1_FP.vcf 58
                   results/29102015Thu-1/MR1_report.json 0
                   results/29102015Thu-1/MR1_TP.vcf 196929
                   results/29102015Thu-1/MR1_unknown_FP.vcf 0
                   results/29102015Thu-1/MR1_unknown_TP.vcf 0
                   total 197035
14:13:21 INFO:     MR2: FAILED
14:13:21 INFO:     MR2:
                   results/29102015Thu-1/MR2_FN.vcf 149
                   results/29102015Thu-1/MR2_FP.vcf 638834
                   results/29102015Thu-1/MR2_report.json 0
                   results/29102015Thu-1/MR2_TP.vcf 213000
                   results/29102015Thu-1/MR2_unknown_FP.vcf 0
                   results/29102015Thu-1/MR2_unknown_TP.vcf 0
                   total 851983
14:13:21 INFO:     MR3: PASSED
14:13:34 INFO:     MR4: PASSED
14:13:34 WARN:     MR4: variants same, but some other vcf values different
14:13:34 INFO:     SI0: PASSED
14:13:36 INFO:     SI1: FAILED
```

14:13:36 INFO:	SI1: results/29102015Thu-1/SI1_FN.vcf 72 results/29102015Thu-1/SI1_FP.vcf 5 results/29102015Thu-1/SI1_report.json 0 results/29102015Thu-1/SI1_TP.vcf 6140 results/29102015Thu-1/SI1_unknown_FP.vcf 0 results/29102015Thu-1/SI1_unknown_TP.vcf 0 total 6217
14:13:37 INFO:	FINISHED GPT PROCESSING FOR ID: 29102015Thu-1

Table 7 - Information from results section of report for GPT run.

6.3.3.1 MR0

The test MR0 is reported to have passed. Recall that this MR checks for deterministic output by passing the same data through the vGSP twice, and comparing the output to check that it is the same in both runs. The test pass indicates that the two VCF files, minus their headers, are identical in their first five columns of the VCF file - that identify the variant type and position.

Despite this indication of a passed test, the GPT framework indicates a warning that some variants called have different information reported in other columns. The difference is found in a column of the VCF file called “INFO” – which provides additional information relating to the call of the variant. Table 8 shows a list of the info fields and their descriptions for cases where differences have been identified between the two MR0 VCF files.

Field ID	Description
BaseQRankSum	Z-score from Wilcoxon rank sum test of Alt Vs. Ref base qualities
ClippingRankSum	Z-score From Wilcoxon rank sum test of Alt vs. Ref number of hard clipped bases
MQ	RMS Mapping Quality
MQRankSum	Z-score From Wilcoxon rank sum test of Alt vs. Ref read mapping qualities
ReadPosRankSum	Z-score from Wilcoxon rank sum test of Alt vs. Ref read position bias
QD	Variant Confidence/Quality by Depth

Table 8 - INFO fields for VCF data anomalies

Of these fields that show differences between the two MR0 VCF files for the same data, QD (Variant Confidence/Quality by Depth) appears to be a field that could potentially make a difference for a borderline variant call. Analysing the VCF files further reveals that 25% of the 196,977 variants called show some differences in the INFO fields, and between 3% and 4% of the variants called show a difference in the QD INFO field. A specific example of differing QD values that occurred in this test run occurred at position 16173 in chromosome M. The two different values were 34.60 and 25.26 – representing an increase of 37% from the smaller to the larger value.

6.3.3.2 MR1

MR1, which randomly sorts the input data, fails. The detailed results show a very small percentage of false positives (FP) and false negatives (FN). The false positives indicating that with the modified data, the vGSP has called variants that were not called with the unmodified data. False negatives indicate that variants that were called with unmodified data, are no longer being called with the modified data.

There were 48 FN and 58 FP out of the 196,977 variants called. This represents a very small percentage (~0.03%) of the total number of variants called. Despite this small number of FN and FP results, it should not be expected functionality for a vGSP to behave in this manner. This result confirms research completed by Giannoulatou et al. [4], who reported a similar result for an MR as applied to a single part of a vGSP – BWA. BWA is part of the example pipeline used in this study.

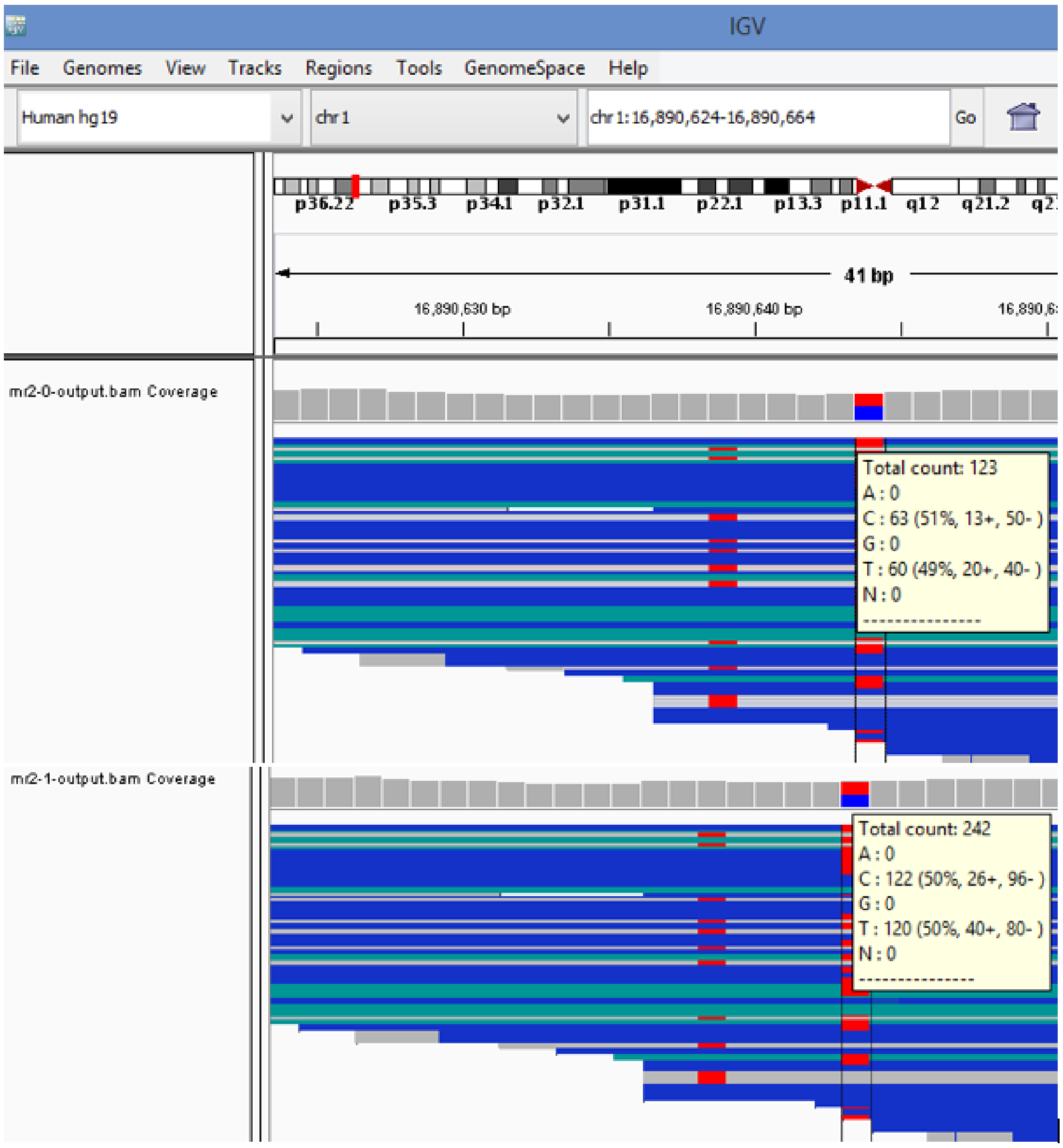
6.3.3.3 MR2

MR2 is another test that fails. It duplicates the input data, and so the not all of the steps of the pipeline are run – otherwise the duplicated data would be rejected. The analysis show that there are 638,834 FP calls, which is not unexpected, as when the data is duplicated, it will give the vGSP false confidence in calling variants that would otherwise not have been called. However, the reason the test fails is that there are 149 FN results. Again this is a very small percentage of the total variants called in the original run (which also skips the same steps in the pipeline as are skipped in the run with the duplicated data).

In this case, further analysis was done to determine why these FN events may be occurring. To investigate the data, the final BAM files were examined for MR2-0 (original data) and MR2-1 (duplicated data). The tool used to compare these two BAM files was the Integrative Genomics Viewer (IGV) browser from the Broad Institute [60] – a high performance genomics data visualisation and exploration tool.

A selection of FN cases were selected for further analysis, and a number of these cases appeared to be situations where the call of the variant was a borderline case with the original data. As an example, Figure 23 shows the same section of the two BAM files. The location is chromosome 1 at position 16,890,644. The reference has base C in this position, and the vGSP originally calls a variant – base T, but with double the data, no longer calls the variant. IGV shows that this position is in a genomic coding region – the gene NBPF1 – and that the variant can be characterised as heterozygous, which can be seen from the close to even split between the number of reads for C and T.

The surprising information that IGV gives us is that there are 4 reads that "disappear" when the data is doubled. The "Total count" field for MR2-0 shows that that BAM file has 123 reads covering the position being examined. When the data is duplicated, we expect that the coverage will double to 246 – which is not the case and the coverage is instead 242. With the split between reads registered as either C or T being so close to 50%, this small change appears to have been the determining factor in the variant no longer being called.



Total count: 123
A : 0
C : 63 (51%, 13+, 50-)
G : 0
T : 60 (49%, 20+, 40-)
N : 0

Total count: 242
A : 0
C : 122 (50%, 26+, 96-)
G : 0
T : 120 (50%, 40+, 80-)
N : 0

Figure 23 - IGV BAM visualisation

6.3.3.4 MR3

MR3 removes any reads that were mapped to the reference, and the remaining unmapped reads are passed through the vGSP to call variants. None are called in this case, which is expected behaviour. As mentioned in the timing section, this task only takes a fraction of the time of a normal run to complete – less than thirty minutes compared to around five hours for a full run. This is because there is minimal if any data on which to call variants.

6.3.3.5 MR4

The test MR4 passed. This test discards unmapped reads.

6.3.3.6 SI0

SIO successfully passed. The reads at 30X coverage were generated from chromosome 11 of hg19 – without any mutations. No variants were called, as expected.

6.3.3.7 SI1

SI1, simulated reads with mutations, failed. The truth VCF contained 6,212 variants, and the result of running the vGSP with this simulated data was to call all but 72 of these, plus 5 FP calls. This 1.2% error rate is quite low given the findings of O'Rawe et al. [29], and can be explained by the fact that the data has been simulated with each read given the highest possible quality score. Real-world data has a range of quality values, and had such quality values been included, one would expect the result to have a higher error rate.

Chapter 7. Discussion

Recall that the aim of this research is to improve testing in the genomic medicine software field - by improving test design, accessibility to testing, and by simplifying the use of related computational resources. The chosen method to achieve this goal was to create a cloud-based framework for testing variant calling genomic sequencing pipelines.

A proof-of-concept framework has been developed to operate in a Linux environment using AWS cloud resources. The user is required to provide the following before the test suite can be executed:

- Their own AWS credentials
- Configuration details such as type, number and payment model for cloud resources
- An installation script and any other necessary files to enable their vGSP to be installed on a cloud resource
- Other configuration details, including the names of data and reference files to be used in the execution of their pipeline

Once this information is provided, the process of testing is fully automated, including creation and configuration of any cloud resources, installation of the vGSP under test on the cloud resources, execution and co-ordination of the tests, and finally reporting back to the user.

In addition to this, metamorphic testing techniques have been incorporated in the framework, along-side a facility that allows testing of outputs from known inputs – in the form of simulated data.

The aim of improved test design in genomic medicine is being met with the use of metamorphic testing, and this has been implemented by adapting existing metamorphic relations that were applied by Giannaulatou et al. [4] to a single part of a vGSP. An additional MR was also generated – to check for deterministic output.

The results identified real issues with an industry standard vGSP, constructed from well-known open-source software components. Apart from confirming a result that was observed by Giannaulatou et al. [4] in relation to randomly sorting input data when using the BWA alignment software, this new testing framework identified issues with determinism in output. This result was made possible by the introduction of the new MR - MR0. MR2, duplication of data, produced results that suggest there is most likely another problem with this vGSP, in relation to inconsistent treatment of reads.

The use of simulated data, provided a check that the vGSP did not produce any false positives when presented with simulated reads from a data source without any mutations. The failure result from the simulated reads with mutations is no real surprise given O'Rawe's [29] findings of low concordance of vGSP's generally.

The features of the cloud-based testing framework produced as part of this research provide advantages and advancements over other offerings found in the literature. Highnam et al.[52] provide a web interface to analyse the results of a vGSP analysis – but does not provide the framework to facilitate the initial processing through the vGSP, nor do they utilise MT testing techniques. Giannaulatou et al. [4] applies MT to a single part of a vGSP, whilst this research extends and adds to the MR's and provides a cloud-based framework. In addition, this framework is designed to handle test cases using real-world computational loads. Finally, offerings such as ART [36] and VarSim [37] provide a mechanism to produce simulated data and associated "truth" VCF files. Again, whilst useful, this offering does not provide a computational framework with which to perform and co-ordinate testing, nor does it incorporate other testing techniques such as MT.

Not only does this research contribute in the areas discussed here, it also applies this framework to a vGSP constructed with industry standard components, and discovers issues that have not been seen raised in the relevant literature. It should be noted that these discoveries were made via the use of MR's, including the newly created MR0 – which checks for deterministic output.

Despite this progress, further work could be done to improve existing MR's and create new ones. The current MR's are quite simple in nature, and future work could devote more time to develop more involved relations. One area that could be improved in MT is the number of follow-up tests cases for each MR. For example, in MR1 where the input data is randomly permuted, this test could be changed so that this occurs a number of times instead of just once. Another approach could be to combine several MR's into a new MR. One example of this could be to randomly sort the input data & then duplicate this data.

Another extremely important area where the testing approach can be improved is to attempt to classify the failure-causing inputs, or regions in the reference genome which may be more likely than others to produce an unstable result. An unstable result could be associated with a region in the reference genome where false positives or false negatives are more likely to occur. Alternatively, an unstable result may be associated with the nature of the input reads if the reads in a particular position are clearly heterozygous, or heavily biased towards one of the two DNA strands.

The identification of such anomalies could be made with the development of classification software that is trained with the aid of large amounts of real (not simulated) sequencing data. Classification parameters could include read depth, the reference base in the position being examined, percentage of reads at each of the bases, and the strand bias – to name a few examples.

The introduction of such a classification feature could bridge the current gap that exists between identifying a problem with a vGSP, and being able to fix that problem. The options for addressing a known problem with vGSP software could include trying to identify the area in the code that has caused the problem and suggesting a patch for this problem (in the case of open-source software). Alternatively, a separate software program could be created that the vGSP builder could include in their pipeline to warn them when input or output is of a nature that has been classified as potentially problematic.

The testing suite will also be improved with the introduction of other state-of-the-art testing techniques, such as adaptive random testing – as was discussed in the background chapter. The testing framework would then include:

- Metamorphic Testing
- Adaptive Random Testing
- Traditional testing of known output for a given input (including simulated data)
- A classification feature for failure causing inputs or identification of known problematic cases

In terms of the aims of accessibility to testing, and reduced technical barriers, it is clear that the proof-of-concept development in this research has met both of these goals. The testing framework provides a self-contained testing environment that automates the process of provisioning cloud resources and distributing and executing tests on these resources. Once the user chooses the type and number of resources, all the configuration and copying of data is taken care of by the framework. The timing and costing results demonstrate a low barrier to testing with respect to these items, with the use of cloud resources potentially reducing testing time by a factor of seven in the given scenario as compared to a single local resource. The framework also provides automatic configuration of access to “spot” instances – something that reduced the price by 73% compared even to normal on-demand cloud resources.

At this stage, the framework has a number of limitations that could be improved on in future work. Firstly, the framework is built in a Linux operating system environment. To allow testing of vGSP’s that may depend on other operating system environments, a framework that is platform independent is needed.

Another limitation is that the user is required to provide an installation script for their vGSP. Some pipelines used in practice have many components that have been put together over a number of years. In these cases, it may be easier to provide the user with a mechanism whereby they could reproduce their vGSP environment on a cloud image, such as an AWS AMI. This would enable them to potentially copy or restore their vGSP to an image and then specify this image for use in testing.

A third way that the framework could be improved is to provide a rich user interface, accessible via a browser, that would simplify the process of configuration, monitoring, and viewing of reports.

Finally, the whole approach to testing in the field of genomic medicine software is critical in nature, especially in a clinical setting. As such, it would be desirable to consult widely in the industry with practitioners and researchers to further develop testing methodology. At the least, this research could be improved by involving other bioinformaticians in the field to give feedback on a version of this software that has been improved with the above suggestions.

Chapter 8. Conclusion

This research has created an automated cloud-based framework to test variant calling genomic sequencing pipelines. This tool improves testing in the field of genomic medicine software by implementing metamorphic testing techniques alongside the more traditional method of comparing output from a vGSP for specific inputs with “known” outputs – whether this be some “gold standard” or from simulated data. This is the first known testing framework of this nature that is designed to complete metamorphic testing of a vGSP on a realistic scale.

Barriers to testing are reduced by providing an automated framework that includes a suite of tests that, once an initial configuration process is completed, proceed in an automated fashion. The framework has been implemented in a Linux environment, utilising cloud resources from Amazon, including an option to use attractively priced “spot” instances.

Testing on real exome data and UCSC reference hg19 was performed, on a vGSP constructed from well-established software in the field – considered to be industry standard. Timing results demonstrate the utility of the framework to run data through the vGSP by scaling the size and number of computing instances to the size of the data and number of tests respectively. For the particular scenario given for testing, this was shown to potentially give the user a seven-fold saving in time – which represented a saving of close to thirty hours under the given assumptions.

As mentioned at the beginning of this conclusion, this research importantly contributes the adaptation of existing metamorphic testing relations from the alignment part of a vGSP to the vGSP as a whole – something that is not found in the literature. In addition to this, a new MR has been developed to check for deterministic output from a vGSP, given the same input.

The application of the new framework, including the metamorphic tests, resulted in the discovery of potential problems with the vGSP used for testing in this research. The two most interesting results were from MR’s that did not pass the test for this vGSP. Firstly, the test for deterministic output indicated that the output of the vGSP was not completely deterministic. Whilst the variants called were the same for the data used in this test, the parameters that express the confidence in these values were different in around 3% of all cases. Secondly, the MR dealing with duplication of data indicated a problem with the software “losing” a small amount of reads that covered a particular position in the genome – leading to some variants not being called with the duplicated data that were called with the normal data. This result was not expected.

This research could be continued to improve the framework by adding more testing techniques such as Adaptive Random Testing, by providing a browser-based interface, and importantly with the provision of a classifier for failure-causing inputs and identification of possible unstable outputs. Consultation within the field of genomic medicine will help to improve the outcomes in this area of testing.

Appendix 1. File Type Descriptions

This appendix describes the key input, intermediate, and output data files involved in a variant calling genomic sequencing pipeline (vGSP).

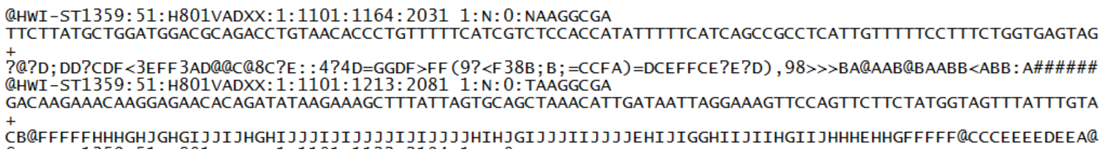
8.1 Input Data – FASTQ files

Along with the reference genome - the FASTA file - the reads that are output from a sequencing machine are used as input into a variant calling genomic sequencing pipeline (vGSP). These reads are contained in a human readable (when uncompressed) text file of format FASTQ [10], which contains quality scores, in addition to the actual sequence that was read. Due to the large size of FASTQ files, which can be around 220GB for a single human genome with 30X coverage, these files are commonly stored using the GNU zip format. Each entry of a FASTQ file contains data relating to one read from the sequencing machine, and the contents of each line are described in Table 9.

Line 1	Sequence identifier
Line 2	Sequence
Line 3	Quality score identifier (+)
Line 4	Quality score

Table 9 - The four lines of a single FASTQ record

Figure 24 gives a sample of the contents of a FASTQ file that was generated from an Illumina sequencing platform.



```
@HWI-ST1359:51:H801VADXX:1:1101:1164:2031 1:N:0:NAAGGCGA
TTCTTATGCTGGATGGACGCAGACCTGTAACACCCTGTTTTTCATCGTCTCCACCATATTTTTCATCAGCCGCCTCATTGTTTTTCCTTTCTGGTGAGTAG
+
?@?D;DD?CDF<3EFF3AD@@C@8C?E::4?4D=GGDF>FF(9?<F38B;B;=CCFA)=DCEFFCE?E?D),98>>>BA@AAB@BAABB<ABB:A######
@HWI-ST1359:51:H801VADXX:1:1101:1213:2081 1:N:0:TAAGGCGA
GACAAGAAACAAGGAGAACACAGATATAAGAAAGCTTTATTAGTGCAGCTAAACATTGATAATTAGGAAAGTTCCAGTTCTTCTATGGTAGTTTATTTGTA
+
CB@FFFFFHHHGHJGHGIJJIJHGHIJJJIJIJJJJIJIJJJJHIHJGIJJJIIJJJJEHIJIGGHIIJIIHGIIJHHHEHHGFFFFF@CCCEEEEDEEA@
```

Figure 24 - two records from a FASTQ file

The sequence identifier line is in the following format, and describes details of the sequencing run that produced the read, including instrument, run number, lane, etc.:

@<instrument>:<run number>:<flowcell ID>:<lane>:<tile>:<x-pos>:<y-pos> <read>:<is filtered>:<control number>:<index sequence>

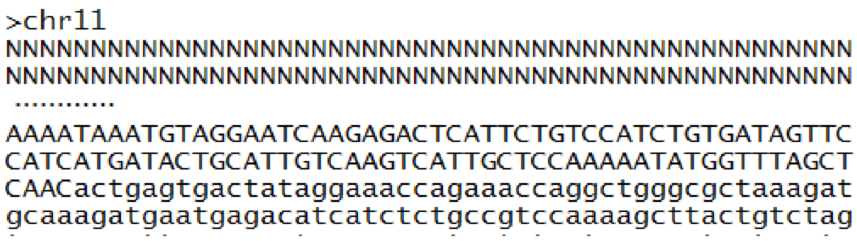
The Sequence itself is described in line 2, and is a series of characters representing the base that was read (A, C, G, or T). For each character in the sequence line, there is a corresponding character in the quality line (line 4) that indicates the probability that the base call is correct. The quality score is indicated by an ASCII character ranging from "!" (lowest) to "~" (highest).

8.2 Input Data – Reference Genome – FASTA files

Along with the FASTQ read data, the other input to a vGSP is the reference genome, in the form of a FASTA file. A human reference genome is a representative sequence of the human DNA, constructed from the DNA data from a number of different donors. The FASTQ reads are aligned to the reference, and variants are called by comparing the aligned FASTQ data to the reference data.

The Genome Reference Consortium [61], which is an initiative of The National Center for Biotechnology Information (NCBI) is responsible for creating reference genomes. The latest release is called GRCh38, sometime referred to as hg38, with the previous release being named either as hg37 or hg19.

For DNA data, FASTA format is a text based format representing nucleotide sequences, split by chromosome. The sequence is preceded by a description line that begins with “>”. Repeats are shown in lower-case, with non-repeats in upper-case. The character “N” in the sequence represents an unknown base (A, C, G, or T).

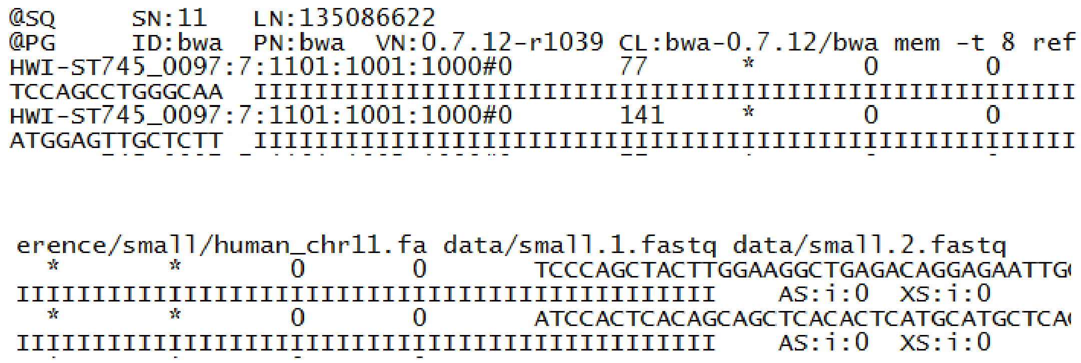


```
>chr11
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
 ............
AAAATAAATGTAGGAATCAAGAGACTCATTCTGTCCATCTGTGATAGTTC
CATCATGATACTGCATTGTCAAGTCATTGCTCCAAAAATATGGTTTAGCT
CAACactgagtgactataggaaaccagaaaccaggctgggcgctaaagat
gcaaagatgaatgagacatcatctctgccgtccaaaagcttactgtctag
```

Figure 25 - sample fasta file contents (edited to show beginning & middle)

8.3 Intermediate output – SAM/BAM files

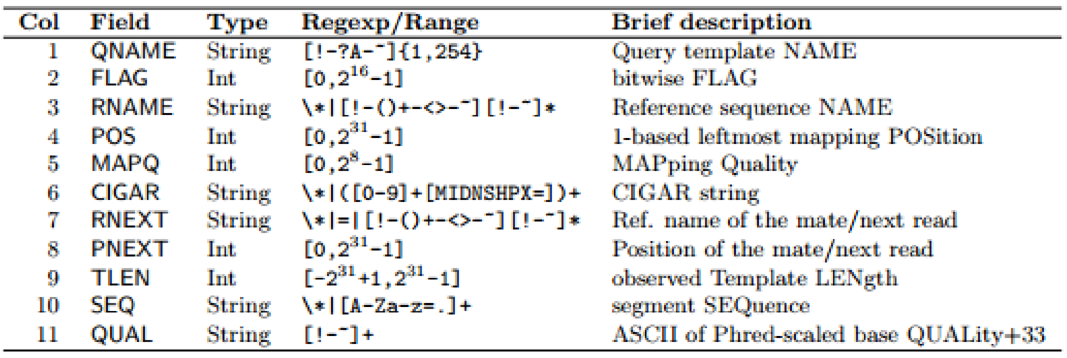
The first step in a vGSP is to align the read data to the reference. The industry standard alignment file that is produced is called a SAM file, with the binary version a BAM file. SAM stands for Sequence Alignment/Map format, which is maintained by the SAM/BAM Format Specification Working Group [14][62]. A SAM file is a TAB-delimited text file with a header sections, and an alignment section. Header lines start with an “@” character.



```
@SQ     SN:11   LN:135086622
@PG     ID:bwa  PN:bwa  VN:0.7.12-r1039 CL:bwa-0.7.12/bwa mem -t 8 ref
HWI-ST745_0097:7:1101:1001:1000#0       77      *       0       0
TCCAGCCTGGGCAA  IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
HWI-ST745_0097:7:1101:1001:1000#0       141     *       0       0
ATGGAGTTGCTCTT  IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII


erence/small/human_chr11.fa data/small.1.fastq data/small.2.fastq
  *       *       0       0       TCCCAGCTACTTGGAAGGCTGAGACAGGAGAATTG
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    AS:i:0  XS:i:0
  *       *       0       0       ATCCACTCACAGCAGCTCACACTCATGCATGCTCA
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    AS:i:0  XS:i:0
```

Figure 26 - SAM file (image split into two horizontal sections)

Each alignment in a SAM file has 11 mandatory fields.



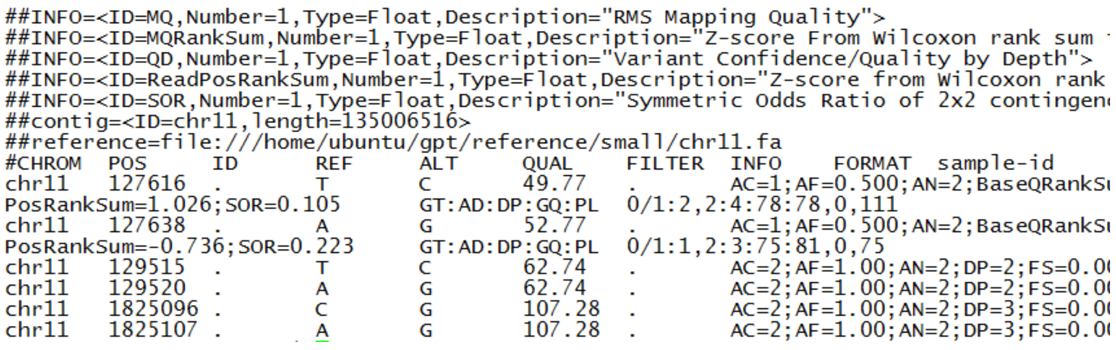
Col	Field	Type	Regexp/Range	Brief description
1	QNAME	String	[!-?A-~]{1,254}	Query template NAME
2	FLAG	Int	[0,2^{16}-1]	bitwise FLAG
3	RNAME	String	*|[!-()+-<>-~][!-~]*	Reference sequence NAME
4	POS	Int	[0,2^{31}-1]	1-based leftmost mapping POSition
5	MAPQ	Int	[0,2^{8}-1]	MAPping Quality
6	CIGAR	String	*|([0-9]+[MIDNSHPX=])+	CIGAR string
7	RNEXT	String	*|=|[!-()+-<>-~][!-~]*	Ref. name of the mate/next read
8	PNEXT	Int	[0,2^{31}-1]	Position of the mate/next read
9	TLEN	Int	[-2^{31}+1,2^{31}-1]	observed Template LENgth
10	SEQ	String	*|[A-Za-z=.]+	segment SEQuence
11	QUAL	String	[!-~]+	ASCII of Phred-scaled base QUALity+33

Figure 27 - SAM mandatory alignment fields[62]

A BAM file is a binary version (compressed) of a SAM file. It uses the BGZF compression format. Separate BAM index files are generated to achieve fast retrieval of alignments. A BAM index file will normally have a file extension of *".bai"*.

8.4 Output file – VCF

The output of a vGSP is a VCF file, which stands for Variant Call Format. The format was created for the 1000 genomes project [63], and describes the structural variants such as SNP's and INDELS that occur in the reads – giving the position in the reference genome at which they occur. A VCF file can be stored in a compressed format, and can also have an index file with which it is associated.



```
##INFO=<ID=MQ,Number=1,Type=Float,Description="RMS Mapping Quality">
##INFO=<ID=MQRankSum,Number=1,Type=Float,Description="Z-score From Wilcoxon rank sum
##INFO=<ID=QD,Number=1,Type=Float,Description="Variant Confidence/Quality by Depth">
##INFO=<ID=ReadPosRankSum,Number=1,Type=Float,Description="Z-score from Wilcoxon rank
##INFO=<ID=SOR,Number=1,Type=Float,Description="Symmetric Odds Ratio of 2x2 contingen
##contig=<ID=chr11,length=135006516>
##reference=file:///home/ubuntu/gpt/reference/small/chr11.fa
#CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO    FORMAT  sample-id
chr11   127616  .       T       C       49.77   .       AC=1;AF=0.500;AN=2;BaseQRankS
PosRankSum=1.026;SOR=0.105      GT:AD:DP:GQ:PL  0/1:2,2:4:78:78,0,111
chr11   127638  .       A       G       52.77   .       AC=1;AF=0.500;AN=2;BaseQRankS
PosRankSum=-0.736;SOR=0.223     GT:AD:DP:GQ:PL  0/1:1,2:3:75:81,0,75
chr11   129515  .       T       C       62.74   .       AC=2;AF=1.00;AN=2;DP=2;FS=0.0
chr11   129520  .       A       G       62.74   .       AC=2;AF=1.00;AN=2;DP=2;FS=0.0
chr11   1825096 .       C       G       107.28  .       AC=2;AF=1.00;AN=2;DP=3;FS=0.0
chr11   1825107 .       A       G       107.28  .       AC=2;AF=1.00;AN=2;DP=3;FS=0.0
```

Figure 28 - sample contents of a VCF file

Appendix 2. GPT Configuration File

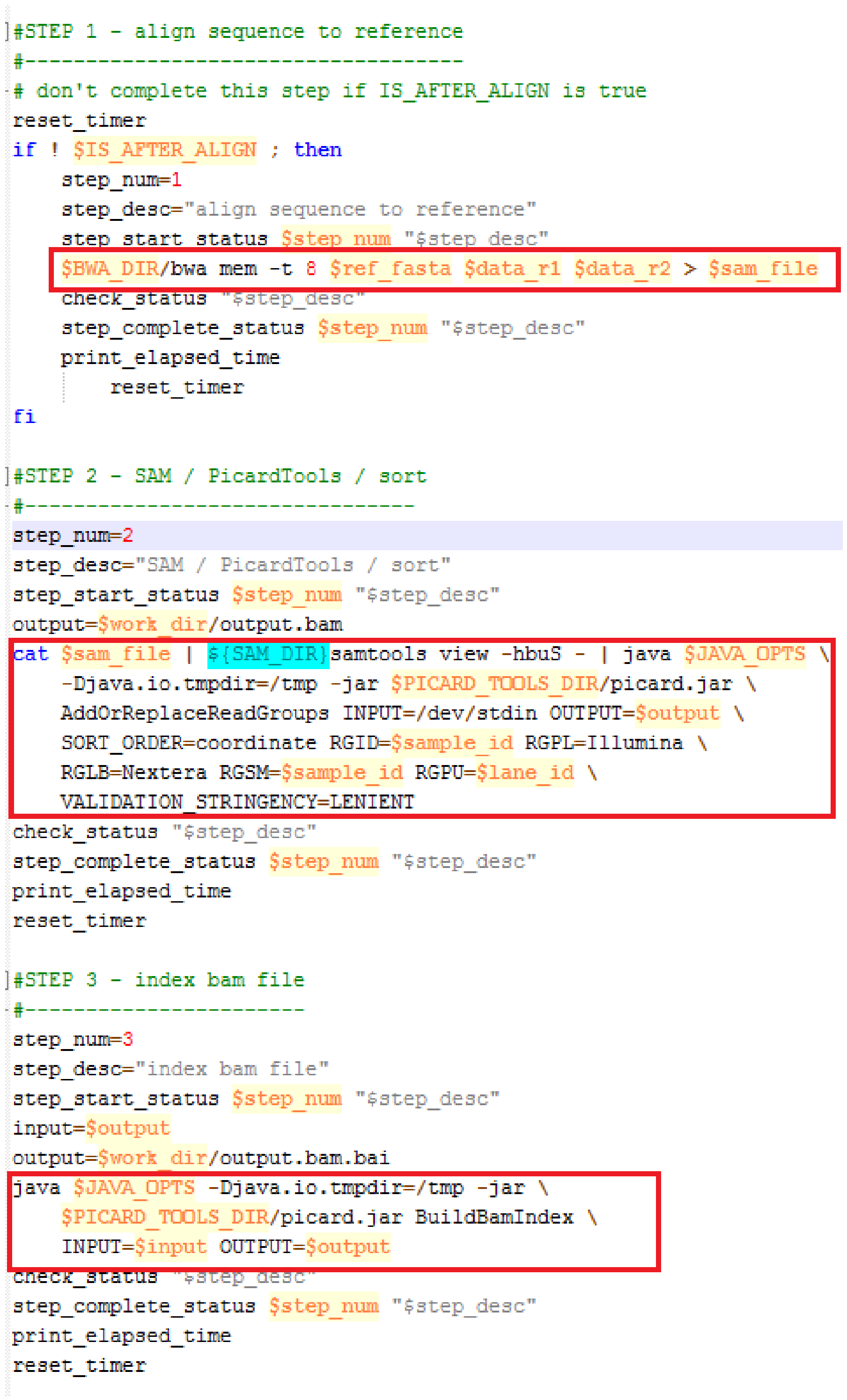
The following figure gives a full example of a GPT configuration file that was used in testing. The format is similar to the Microsoft Windows *ini* format and is consistent with the Python 3.4 *ConfigParser* specification.

```
[aws]
region=us-west-2

[aws-instances]
# use spot instances - set to 1
use_spot=1
spot_price=1.60
instance-type=c3.8xlarge
count=9
user-data=gpt-system/user-data-ssd.sh
user-data-checker=gpt-system/user-data-checker.sh
dry-run=False
security-key=mictro-dev-key.pem

[install-pipeline]
pipe-install-script=install-pipeline-aws.sh

[run-pipeline]
run-script=pipe-exec.sh

[aws-s3]
script=copy-s3-data.sh
bucket=vccri.ho-lab.mictro.testing-project

[data]
base-dir=/ssd
data-dir=data
ref-dir=reference
# FASTQ data
data-1=H801VADXX-1-701-501_TAAGGCGA_L001_R1_001.fastq
data-2=H801VADXX-1-701-501_TAAGGCGA_L001_R2_001.fastq
# SI0 data - simulated reads - no mutations
sim-ref-data-1=sim-no-mut-chr11-r1.fq.gz
sim-ref-data-2=sim-no-mut-chr11-r2.fq.gz
# SI1 data - simulated reads - mutations
sim-mut-ref-data-1=sim-mut-chr11-read1.fq.gz
sim-mut-ref-data-2=sim-mut-chr11-read2.fq.gz
# reference data
ref-fasta=hg19/working/ucsc.hg19.fasta
ref-1000G-phase1-indels=hg19/1000G_phase1.indels.hg19.sites.vcf
ref-mills-1000G-indels=hg19/Mills_and_1000G_gold_standard.indels.hg19.sites.vcf
current-db-snp=hg19/1000G_phase1.snps.high_confidence.hg19.sites.vcf
# truth vcf for simulated data
sim-mut-truth-vcf=varsim_run/out/simu.truth.vcf
```

Figure 29 - GPT Configuration – full example

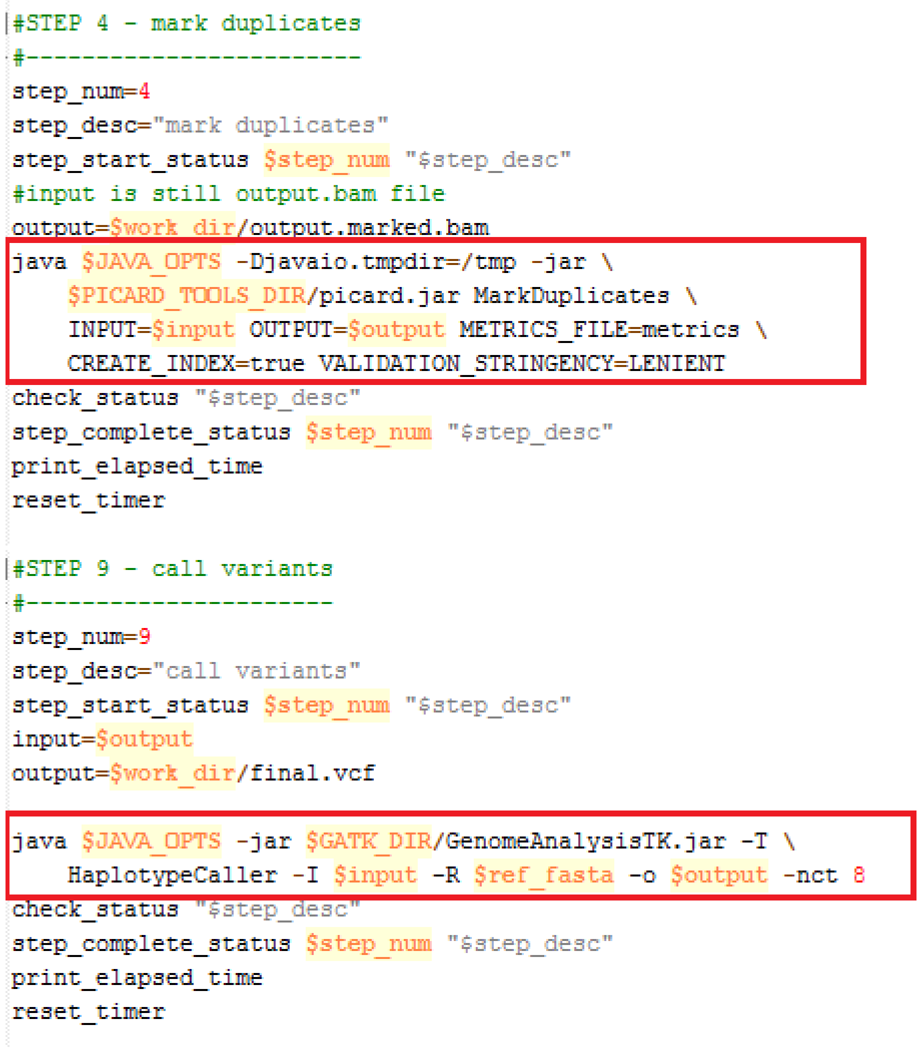
Appendix 3. Example vGSP Code

This appendix shows a portion of the contents of an example variant-calling genomic sequencing pipeline (vGSP). This pipeline uses the BWA software for the alignment step, and the GATK software for the remainder of the pipeline steps – including the variant calling steps. The vGSP is written as a Linux bash script.



```
#STEP 1 - align sequence to reference
#-------------------------------------
# don't complete this step if IS_AFTER_ALIGN is true
reset_timer
if ! $IS_AFTER_ALIGN ; then
    step_num=1
    step_desc="align sequence to reference"
    step_start_status $step_num "$step_desc"
    $BWA_DIR/bwa mem -t 8 $ref_fasta $data_r1 $data_r2 > $sam_file
    check_status "$step_desc"
    step_complete_status $step_num "$step_desc"
    print_elapsed_time
        reset_timer
fi

#STEP 2 - SAM / PicardTools / sort
#--------------------------------
step_num=2
step_desc="SAM / PicardTools / sort"
step_start_status $step_num "$step_desc"
output=$work_dir/output.bam
cat $sam_file | ${SAM_DIR}samtools view -hbuS - | java $JAVA_OPTS \
    -Djava.io.tmpdir=/tmp -jar $PICARD_TOOLS_DIR/picard.jar \
    AddOrReplaceReadGroups INPUT=/dev/stdin OUTPUT=$output \
    SORT_ORDER=coordinate RGID=$sample_id RGPL=Illumina \
    RGLB=Nextera RGSM=$sample_id RGPU=$lane_id \
    VALIDATION_STRINGENCY=LENIENT
check_status "$step_desc"
step_complete_status $step_num "$step_desc"
print_elapsed_time
reset_timer

#STEP 3 - index bam file
#-----------------------
step_num=3
step_desc="index bam file"
step_start_status $step_num "$step_desc"
input=$output
output=$work_dir/output.bam.bai
java $JAVA_OPTS -Djava.io.tmpdir=/tmp -jar \
    $PICARD_TOOLS_DIR/picard.jar BuildBamIndex \
    INPUT=$input OUTPUT=$output
check_status "$step_desc"
step_complete_status $step_num "$step_desc"
print_elapsed_time
reset_timer
```

Figure 30 - Example vGSP – BWA/GATK – part 1



```
#STEP 4 - mark duplicates
#------------------------
step_num=4
step_desc="mark duplicates"
step_start_status $step_num "$step_desc"
#input is still output.bam file
output=$work_dir/output.marked.bam
java $JAVA_OPTS -Djavaio.tmpdir=/tmp -jar \
    $PICARD_TOOLS_DIR/picard.jar MarkDuplicates \
    INPUT=$input OUTPUT=$output METRICS_FILE=metrics \
    CREATE_INDEX=true VALIDATION_STRINGENCY=LENIENT
check_status "$step_desc"
step_complete_status $step_num "$step_desc"
print_elapsed_time
reset_timer

#STEP 9 - call variants
#----------------------
step_num=9
step_desc="call variants"
step_start_status $step_num "$step_desc"
input=$output
output=$work_dir/final.vcf

java $JAVA_OPTS -jar $GATK_DIR/GenomeAnalysisTK.jar -T \
    HaplotypeCaller -I $input -R $ref_fasta -o $output -nct 8
check_status "$step_desc"
step_complete_status $step_num "$step_desc"
print_elapsed_time
reset_timer
```

Figure 31 - Example vGSP – BWA/GATK – part 2

Appendix 4. Report File from a GPT Run

Following is the report file from the GPT test run that was used to generate the results as per the Results chapter.

```
08:45:03 INFO: STARTED GPT PROCESSING FOR ID: 29102015Thu-1
08:45:03 INFO: starting job to create instances...
08:57:51 INFO: finished creating instances...
08:57:51 INFO: ip 0 - ec2-52-32-20-70.us-west-2.compute.amazonaws.com
08:57:51 INFO: ip 1 - ec2-52-32-19-184.us-west-2.compute.amazonaws.com
08:57:51 INFO: ip 2 - ec2-52-32-18-99.us-west-2.compute.amazonaws.com
08:57:51 INFO: ip 3 - ec2-52-32-20-56.us-west-2.compute.amazonaws.com
08:57:51 INFO: ip 4 - ec2-52-32-14-207.us-west-2.compute.amazonaws.com
08:57:51 INFO: ip 5 - ec2-52-32-8-232.us-west-2.compute.amazonaws.com
08:57:51 INFO: ip 6 - ec2-52-32-20-75.us-west-2.compute.amazonaws.com
08:57:51 INFO: ip 7 - ec2-52-32-20-64.us-west-2.compute.amazonaws.com
08:57:51 INFO: ip 8 - ec2-52-32-20-14.us-west-2.compute.amazonaws.com
08:58:03 INFO: starting job: install-pipeline-0
08:58:16 INFO: starting job: install-pipeline-1
08:58:28 INFO: starting job: install-pipeline-2
08:58:40 INFO: starting job: install-pipeline-3
08:58:52 INFO: starting job: install-pipeline-4
08:59:04 INFO: starting job: install-pipeline-5
08:59:17 INFO: starting job: install-pipeline-6
08:59:29 INFO: starting job: install-pipeline-7
08:59:41 INFO: starting job: install-pipeline-8
08:59:42 INFO: finished job: install-pipeline-0
08:59:45 INFO: starting job: copy-s3-data-0
08:59:46 INFO: finished job: install-pipeline-1
08:59:48 INFO: starting job: copy-s3-data-1
08:59:50 INFO: finished job: install-pipeline-2
08:59:52 INFO: starting job: copy-s3-data-2
08:59:54 INFO: finished job: install-pipeline-3
08:59:56 INFO: starting job: copy-s3-data-3
09:00:15 INFO: finished job: install-pipeline-4
09:00:17 INFO: starting job: copy-s3-data-4
09:00:40 INFO: finished job: install-pipeline-5
09:00:42 INFO: starting job: copy-s3-data-5
09:00:43 INFO: finished job: install-pipeline-6
09:00:45 INFO: starting job: copy-s3-data-6
09:00:47 INFO: finished job: install-pipeline-7
09:00:49 INFO: starting job: copy-s3-data-7
```

```
09:01:10 INFO: finished job: install-pipeline-8
09:01:12 INFO: starting job: copy-s3-data-8
09:10:20 INFO: finished job: copy-s3-data-0
09:10:22 INFO: starting job: MR0-0-i-0
09:10:24 INFO: finished job: copy-s3-data-1
09:10:26 INFO: starting job: MR2-0-i-1
09:10:27 INFO: finished job: copy-s3-data-2
09:10:29 INFO: starting job: MR0-1-i-2
09:10:31 INFO: finished job: copy-s3-data-3
09:10:33 INFO: starting job: MR1-1-i-3
09:10:34 INFO: finished job: copy-s3-data-4
09:10:36 INFO: starting job: MR2-1-i-4
09:10:39 INFO: finished job: copy-s3-data-6
09:10:41 INFO: starting job: MR4-1-i-6
09:10:43 INFO: finished job: copy-s3-data-7
09:10:45 INFO: starting job: SI0-1-i-7
09:10:46 INFO: finished job: copy-s3-data-8
09:10:48 INFO: starting job: SI1-1-i-8
09:11:00 INFO: finished job: copy-s3-data-5
09:11:02 INFO: starting job: MR3-1-i-5
09:34:19 INFO: finished job: MR3-1-i-5
12:19:47 INFO: finished job: MR2-0-i-1
12:49:39 INFO: finished job: MR2-1-i-4
13:06:43 INFO: finished job: SI0-1-i-7
13:13:58 INFO: finished job: SI1-1-i-8
13:59:56 INFO: finished job: MR0-0-i-0
14:03:05 INFO: finished job: MR0-1-i-2
14:11:18 INFO: finished job: MR1-1-i-3
14:11:31 INFO: finished job: MR4-1-i-6
14:12:41 INFO: RESULTS:
14:12:54 INFO: MR0: PASSED
14:12:54 WARN: MR0: variants same, but some other vcf values different
14:13:07 INFO: MR1: FAILED
14:13:07 INFO: MR1:
results/29102015Thu-1/MR1_FN.vcf 48
results/29102015Thu-1/MR1_FP.vcf 58
results/29102015Thu-1/MR1_report.json 0
results/29102015Thu-1/MR1_TP.vcf 196929
results/29102015Thu-1/MR1_unknown_FP.vcf 0
results/29102015Thu-1/MR1_unknown_TP.vcf 0
total 197035
14:13:21 INFO: MR2: FAILED
14:13:21 INFO: MR2:
results/29102015Thu-1/MR2_FN.vcf 149
```

```
results/29102015Thu-1/MR2_FP.vcf 638834
results/29102015Thu-1/MR2_report.json 0
results/29102015Thu-1/MR2_TP.vcf 213000
results/29102015Thu-1/MR2_unknown_FP.vcf 0
results/29102015Thu-1/MR2_unknown_TP.vcf 0
total 851983
14:13:21 INFO: MR3: PASSED
14:13:34 INFO: MR4: PASSED
14:13:34 WARN: MR4: variants same, but some other vcf values different
14:13:34 INFO: SI0: PASSED
14:13:36 INFO: SI1: FAILED
14:13:36 INFO: SI1:
results/29102015Thu-1/SI1_FN.vcf 72
results/29102015Thu-1/SI1_FP.vcf 5
results/29102015Thu-1/SI1_report.json 0
results/29102015Thu-1/SI1_TP.vcf 6140
results/29102015Thu-1/SI1_unknown_FP.vcf 0
results/29102015Thu-1/SI1_unknown_TP.vcf 0
total 6217
14:13:37 INFO: FINISHED GPT PROCESSING FOR ID: 29102015Thu-1

TIMING INFORMATION
------------------
run-instances-task        0:12:46
install-pipeline-0-task    0:01:24
install-pipeline-1-task    0:01:15
install-pipeline-2-task    0:01:11
install-pipeline-3-task    0:01:12
install-pipeline-4-task    0:01:15
install-pipeline-6-task    0:01:13
install-pipeline-5-task    0:01:30
install-pipeline-7-task    0:01:13
install-pipeline-8-task    0:01:21
copy-s3-data-4-task        0:09:54
copy-s3-data-3-task        0:10:14
copy-s3-data-2-task        0:10:18
copy-s3-data-8-task        0:09:00
copy-s3-data-7-task        0:09:22
copy-s3-data-0-task        0:10:27
copy-s3-data-6-task        0:09:27
copy-s3-data-1-task        0:10:25
copy-s3-data-5-task        0:10:01
MR3-1-i-5-task            0:23:49
MR2-0-i-1-task            3:09:43
```

```
MR2-1-i-4-task            3:39:33
SI0-1-i-7-task            3:56:25
SI1-1-i-8-task            4:03:41
MR0-0-i-0-task            4:50:08
MR0-1-i-2-task            4:53:09
MR1-1-i-3-task            5:01:16
MR4-1-i-6-task            5:01:19
results-analysis-task     0:01:54
gpt-task                  5:29:11

CONFIGURATION INFORMATION
----------------------
[aws]
region=us-west-2

[aws-instances]
# use spot instances - set to 1
use_spot=1
spot_price=1.60
instance-type=c3.8xlarge
count=9
user-data=gpt-system/user-data-ssd.sh
user-data-checker=gpt-system/user-data-checker.sh
dry-run=False
security-key=mictro-dev-key.pem

[install-pipeline]
pipe-install-script=install-pipeline-aws.sh

[run-pipeline]
run-script=pipe-exec.sh

[aws-s3]
script=copy-s3-data.sh
bucket=vccri.ho-lab.mictro.testing-project

[data]
base-dir=/ssd
data-dir=data
ref-dir=reference
#
# FASTQ data
#
# small data set - 10000 reads
```

```
#data-1=H801VADXX-1-701-
501_TAAGGCGA_L001_10000READS_R1_001.fastq
#data-2=H801VADXX-1-701-
501_TAAGGCGA_L001_10000READS_R2_001.fastq
# full data-set : fastq uncompressed is ~ 3.6G
data-1=H801VADXX-1-701-501_TAAGGCGA_L001_R1_001.fastq
data-2=H801VADXX-1-701-501_TAAGGCGA_L001_R2_001.fastq
#
# SI0 data - simulated reads - no mutations
#
# 10000 reads from simulated chr11 - no mutations
#sim-ref-data-1=art.10000.1.fastq
#sim-ref-data-2=art.10000.2.fastq
# all reads from simulated chr11 - no mutations - 30x coverage
sim-ref-data-1=sim-no-mut-chr11-r1.fq.gz
sim-ref-data-2=sim-no-mut-chr11-r2.fq.gz
#
# SI1 data - simulated reads - mutations
#
# whole of chr11 simulated reads - 30x coverage
sim-mut-ref-data-1=sim-mut-chr11-read1.fq.gz
sim-mut-ref-data-2=sim-mut-chr11-read2.fq.gz
# 10000 reads from - whole of chr11 simulated reads - 30x coverage
#sim-mut-ref-data-1=si1-10000-read1.fq
#sim-mut-ref-data-2=si1-10000-read2.fq
#data-1=small.1.fastq
#data-2=small.2.fastq
#
# reference data
#
#small reference - chr11 from hg19
#ref-fasta=small/chr11.fa
ref-fasta=hg19/working/ucsc.hg19.fasta
ref-1000G-phase1-indels=hg19/1000G_phase1.indels.hg19.sites.vcf
ref-mills-1000G-indels=hg19/Mills_and_1000G_gold_standard.indels.hg19.sites.vcf
current-db-snp=hg19/1000G_phase1.snps.high_confidence.hg19.sites.vcf
# truth vcf for simulated data
sim-mut-truth-vcf=varsim_run/out/simu.truth.vcf
```

Bibliography

[1] National Human Genome Research Institute, "Frequently Asked Questions About Genetic and Genomic Science." [Online]. Available: http://www.genome.gov/19016904. [Accessed: 01-Sep-2015].

[2] L. G. Biesecker and R. C. Green, "Diagnostic Clinical Genome and Exome Sequencing," *N. Engl. J. Med.*, vol. 370, no. 25, pp. 2418–2425, Jun. 2014.

[3] T. Y. Chen, S. C. Cheung, and S. M. Yiu, "Metamorphic testing: a new approach for generating next test cases." Technical Report HKUST-CS98-01, Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong, 1998.

[4] E. Giannoulatou, S.-H. Park, D. T. Humphreys, and J. W. Ho, "Verification and validation of bioinformatics software without a gold standard: a case study of BWA and Bowtie," *BMC Bioinformatics*, vol. 15, no. Suppl 16, p. S15, Dec. 2014.

[5] "Conference," *ABACBS*. [Online]. Available: http://www.abacbs.org/conference/. [Accessed: 20-Oct-2015].

[6] "What is DNA?," *Genetics Home Reference*, 12-Oct-2015. [Online]. Available: http://ghr.nlm.nih.gov/handbook/basics/dna. [Accessed: 19-Oct-2015].

[7] Illumina, "An Introduction to Next Generation Sequencing Technology." [Online]. Available: http://www.illumina.com/content/dam/illumina-marketing/documents/products/illumina_sequencing_introduction.pdf. [Accessed: 12-Oct-2015].

[8] Illumina, "Sequencing Coverage." [Online]. Available: http://www.illumina.com/science/education/sequencing-coverage.html. [Accessed: 13-Oct-2015].

[9] Illumina, "HiSeq X Series of Sequencing Systems." [Online]. Available: http://www.illumina.com/content/dam/illumina-marketing/documents/products/datasheets/datasheet-hiseq-x-ten.pdf. [Accessed: 21-Oct-2015].

[10] P. J. A. Cock, C. J. Fields, N. Goto, M. L. Heuer, and P. M. Rice, "The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants," *Nucleic Acids Res.*, vol. 38, no. 6, pp. 1767–1771, Apr. 2010.

[11] A. Altmann, P. Weber, D. Bader, M. Preuß, E. B. Binder, and B. Müller-Myhsok, "A beginners guide to SNP calling from high-throughput DNA-sequencing data," *Hum. Genet.*, vol. 131, no. 10, pp. 1541–1554, Aug. 2012.

[12] H. Li and R. Durbin, "Fast and accurate short read alignment with Burrows–Wheeler transform," *Bioinformatics*, vol. 25, no. 14, pp. 1754–1760, Jul. 2009.

[13] B. Langmead, C. Trapnell, M. Pop, and S. L. Salzberg, "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome," *Genome Biol.*, vol. 10, no. 3, p. R25, Mar. 2009.

[14] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, R. Durbin, and 1000 Genome Project Data Processing Subgroup, "The

Sequence Alignment/Map format and SAMtools," *Bioinformatics*, vol. 25, no. 16, pp. 2078–2079, Aug. 2009.

[15] "Picard." [Online]. Available: http://broadinstitute.github.io/picard/. [Accessed: 05-Jan-2015].

[16] A. McKenna, M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis, A. Kernytsky, K. Garimella, D. Altshuler, S. Gabriel, M. Daly, and M. A. DePristo, "The Genome Analysis Toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data," *Genome Res.*, vol. 20, no. 9, pp. 1297–1303, Sep. 2010.

[17] J. G. Reid, A. Carroll, N. Veeraraghavan, M. Dahdouli, A. Sundquist, A. English, M. Bainbridge, S. White, W. Salerno, C. Buhay, F. Yu, D. Muzny, R. Daly, G. Duyk, R. A. Gibbs, and E. Boerwinkle, "Launching genomics into the cloud: deployment of Mercury, a next generation sequence analysis pipeline," *BMC Bioinformatics*, vol. 15, no. 1, p. 30, Jan. 2014.

[18] U. S. Evani, D. Challis, J. Yu, A. R. Jackson, S. Paithankar, M. N. Bainbridge, A. Jakkamsetti, P. Pham, C. Coarfa, A. Milosavljevic, and F. Yu, "Atlas2 Cloud: a framework for personal genome analysis in the cloud," *BMC Genomics*, vol. 13, no. Suppl 6, p. S19, Oct. 2012.

[19] "BaseSpace: Genomics Cloud Computing." [Online]. Available: https://basespace.illumina.com/home/sequence. [Accessed: 01-May-2015].

[20] E. S. Lander, et al., "Initial sequencing and analysis of the human genome," *Nature*, vol. 409, no. 6822, pp. 860–921, Feb. 2001.

[21] "National Human Genome Research Institute." [Online]. Available: https://www.genome.gov/11006943. [Accessed: 01-May-2015].

[22] "China's genomics success shows big data challenges." [Online]. Available: http://www.cnbc.com/id/101712807. [Accessed: 01-May-2015].

[23] B. J. Kelly, J. R. Fitch, Y. Hu, D. J. Corsmeier, H. Zhong, A. N. Wetzel, R. D. Nordquist, D. L. Newsom, and P. White, "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.*, vol. 16, no. 1, 2015.

[24] M. Baker, "Next-generation sequencing: adjusting to data overload," *Nat. Methods*, vol. 7, no. 7, pp. 495–499, Jul. 2010.

[25] "National Human Genome Research Institute: Cost per Genome." [Online]. Available: https://www.genome.gov/images/content/cost_genome.jpg. [Accessed: 01-May-2015].

[26] R. J. Robinson, "How big is the human genome?" [Online]. Available: https://medium.com/precision-medicine/how-big-is-the-human-genome-e90caa3409b0. [Accessed: 01-May-2015].

[27] IEEE Spectrum, "The DNA Data Deluge." [Online]. Available: http://spectrum.ieee.org/biomedical/devices/the-dna-data-deluge. [Accessed: 01-May-2015].

[28] N. C. Bennett and C. S. Farah, “Next-Generation Sequencing in Clinical Oncology: Next Steps Towards Clinical Validation,” *Cancers*, vol. 6, no. 4, pp. 2296–2312, Nov. 2014.

[29] J. O’Rawe, T. Jiang, G. Sun, Y. Wu, W. Wang, J. Hu, P. Bodily, L. Tian, H. Hakonarson, W. E. Johnson, Z. Wei, K. Wang, and G. J. Lyon, “Low concordance of multiple variant-calling pipelines: practical implications for exome and genome sequencing,” *Genome Med.*, vol. 5, no. 3, p. 28, Mar. 2013.

[30] G. J. Myers, C. Sandler, and T. Badgett, *The Art of Software Testing*. John Wiley & Sons, 2011.

[31] E. J. Weyuker, “On Testing Non-Testable Programs,” *Comput. J.*, vol. 25, no. 4, pp. 465–470, Nov. 1982.

[32] “Genome in a Bottle Consortium.” [Online]. Available: http://www.genomeinabottle.org/. [Accessed: 01-May-2015].

[33] E. Allen, “Variant calling assessment using Platinum Genomes, NIST Genome in a Bottle, and VCAT 2.0,” *BaseSpace Blog*. [Online]. Available:http://blog.basespace.illumina.com/2015/01/14/variant-calling-assessment-using-platinum-genomes-nist-genome-in-a-bottle-and-vcat-2-0/. [Accessed: 01-May-2015].

[34] F. Sanger, S. Nicklen, and A. R. Coulson, “DNA sequencing with chain-terminating inhibitors,” *Proc. Natl. Acad. Sci.*, vol. 74, no. 12, pp. 5463–5467, Dec. 1977.

[35] H. L. Rehm, S. J. Bale, P. Bayrak-Toydemir, J. S. Berg, K. K. Brown, J. L. Deignan, M. J. Friez, B. H. Funke, M. R. Hegde, and E. Lyon, “ACMG clinical laboratory standards for next-generation sequencing,” *Genet. Med. Off. J. Am. Coll. Med. Genet.*, vol. 15, no. 9, pp. 733–747, Sep. 2013.

[36] W. Huang, L. Li, J. R. Myers, and G. T. Marth, “ART: a next-generation sequencing read simulator,” *Bioinformatics*, vol. 28, no. 4, pp. 593–594, Feb. 2012.

[37] J. C. Mu, M. Mohiyuddin, J. Li, N. B. Asadi, M. B. Gerstein, A. Abyzov, W. H. Wong, and H. Y. K. Lam, “VarSim: a high-fidelity simulation and validation framework for high-throughput genome sequencing with cancer applications,” *Bioinformatics*, vol. 31, no. 9, pp. 1469–1471, May 2015.

[38] T. Y. Chen, F.-C. Kuo, R. G. Merkel, and T. H. Tse, “Adaptive Random Testing: The ART of test case diversity,” *J. Syst. Softw.*, vol. 83, no. 1, pp. 60–66, Jan. 2010.

[39] T. Y. Chen, H. Leung, and I. K. Mak, “Adaptive Random Testing,” in *Advances in Computer Science - ASIAN 2004. Higher-Level Decision Making*, M. J. Maher, Ed. Springer Berlin Heidelberg, 2004, pp. 320–329.

[40] J. Mayer, “Lattice-based Adaptive Random Testing,” in *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, New York, NY, USA, 2005, pp. 333–336.

[41] I. Ciupa, A. Leitner, M. Oriol, and B. Meyer, “Object Distance and Its Application to Adaptive Random Testing of Object-oriented Programs,” in

Proceedings of the 1st International Workshop on Random Testing, New York, NY, USA, 2006, pp. 55–63.

[42] C.-A. Sun, G. Wang, B. Mu, H. Liu, Z. Wang, and T. . Chen, “A metamorphic relation-based approach to testing web services without oracles,” *Int. J. Web Serv. Res.*, vol. 9, no. 1, pp. 51 – 73, Jan. 2012.

[43] Q. Tao, W. Wu, C. Zhao, and W. Shen, “An Automatic Testing Approach for Compiler Based on Metamorphic Testing Technique,” in *Software Engineering Conference (APSEC), 2010 17th Asia Pacific*, 2010, pp. 270–279.

[44] S. Segura, R. M. Hierons, D. Benavides, and A. Ruiz-Cortés, “Automated metamorphic testing on the analyses of feature models,” *Inf. Softw. Technol.*, vol. 53, no. 3, pp. 245–258, Mar. 2011.

[45] X. Xie, J. W. K. Ho, C. Murphy, G. Kaiser, B. Xu, and T. Y. Chen, “Testing and validating machine learning classifiers by metamorphic testing,” *J. Syst. Softw.*, vol. 84, no. 4, pp. 544–558, Apr. 2011.

[46] T. Y. Chen, J. Feng, and T. H. Tse, “Metamorphic testing of programs on partial differential equations: a case study,” in *Computer Software and Applications Conference, 2002. COMPSAC 2002. Proceedings. 26th Annual International*, 2002, pp. 327–333.

[47] H. Liu, F.-C. Kuo, D. Towey, and T. Y. Chen, “How Effectively Does Metamorphic Testing Alleviate the Oracle Problem?,” *IEEE Trans. Softw. Eng.*, vol. 40, no. 1, pp. 4–22, Jan. 2014.

[48] M. S. Sadi, F.-C. Kuo, J. W. K. Ho, M. A. Charleston, and T. Y. Chen, “Verification of phylogenetic inference programs using metamorphic testing,” *J. Bioinform. Comput. Biol.*, vol. 09, no. 06, pp. 729–747, Dec. 2011.

[49] P. Mell and T. Grance, “The NIST Definition of Cloud Computing,” 01-Sep-2011. [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf. [Accessed: 13-Oct-2015].

[50] J. Gao, X. Bai, W. T. Tsai, and T. Uehara, “SaaS Testing on Clouds - Issues, Challenges and Needs,” in *2013 IEEE 7th International Symposium on Service Oriented System Engineering (SOSE)*, 2013, pp. 409–415.

[51] L. Riungu-Kalliosaari, O. Taipale, and K. Smolander, “Testing in the Cloud: Exploring the Practice,” *IEEE Softw.*, vol. 29, no. 2, pp. 46–51, Mar. 2012.

[52] G. Highnam, J. J. Wang, D. Kusler, J. Zook, V. Vijayan, N. Leibovich, and D. Mittelman, “An analytical framework for optimizing variant discovery from personal genomes,” *Nat. Commun.*, vol. 6, Feb. 2015.

[53] Amazon, “Amazon Machine Images (AMI).” [Online]. Available: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html. [Accessed: 15-Oct-2015].

[54] M. Frampton and R. Houlston, “Generation of Artificial FASTQ Files to Evaluate the Performance of Next-Generation Sequencing Pipelines,” *PLoS ONE*, vol. 7, no. 11, p. e49110, Nov. 2012.

[55] T. 1000 G. P. Consortium, “A map of human genome variation from population-scale sequencing,” *Nature*, vol. 467, no. 7319, pp. 1061–1073, Oct. 2010.

[56] Amazon, “AWS EC2 storage archictecture.” [Online]. Available: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/Storage.html. [Accessed: 16-Oct-2015].

[57] O. Tange, “GNU Parallel - The Command-Line Power Tool,” *USENIX Mag.*, vol. 36, no. 1, pp. 42–47, Feb. 2011.

[58] AWS, “Amazon EC2 Pricing.” [Online]. Available: https://aws.amazon.com/ec2/pricing/. [Accessed: 30-Oct-2015].

[59] AWS, “Amazon S3 Pricing.” [Online]. Available: https://aws.amazon.com/s3/pricing/. [Accessed: 30-Oct-2015].

[60] H. Thorvaldsdóttir, J. T. Robinson, and J. P. Mesirov, “Integrative Genomics Viewer (IGV): high-performance genomics data visualization and exploration,” *Brief. Bioinform.*, p. bbs017, Apr. 2012.

[61] NCBI, “Genome Reference Consortium.” [Online]. Available: http://www.ncbi.nlm.nih.gov/projects/genome/assembly/grc/human/. [Accessed: 13-Oct-2015].

[62] “Sequence Alignment/Map Specification.” [Online]. Available: https://samtools.github.io/hts-specs/SAMv1.pdf. [Accessed: 13-Oct-2015].

[63] P. Danecek, A. Auton, G. Abecasis, C. A. Albers, E. Banks, M. A. DePristo, R. E. Handsaker, G. Lunter, G. T. Marth, S. T. Sherry, G. McVean, R. Durbin, and 1000 Genomes Project Analysis Group, “The variant call format and VCFtools,” *Bioinformatics*, vol. 27, no. 15, pp. 2156–2158, Aug. 2011.