Automating installation, testing and development of bcbio-nextgen pipeline

GUILLERMO CARRASCO HERNÁNDEZ

guillermo.carrasco@scilifelab.se

June 2013

Final project at Barcelona School of Informatics (FIB)

Subject: Computer Science

Company: Science for Life Laboratory

Supervisor at Science for Life Laboratory: Roman Valls Guimera

Supervisor FIB: Daniel Jiménez González

Abstract

In the recent years, the costs of obtaining biological data have been drastically reduced. This has lead into an exponential growth of the available data. Having such growth of data to analyze sometimes results in very platform-dependent and difficult to scale software solutions.

This final project tries to provide a solution to those problems in a real bioinformatics core facility in the *Science For Life Laboratory*. *Science For Life Laboratory* is a center for large-scale biosciences with the focus in health and environmental research. It is located in Stockholm, Sweden. This laboratory has 15 next generation sequencing instruments at present, with a combined capacity for DNA sequencing equal to several hundreds of complete human genomes per year. This implies a massive amount of data to be managed and analyzed.

This data is analyzed using *bcbio-nextgen*. *bcbio-nextgen* is an in-house maintained genomics pipeline, originally developed by Brad Chapman at Harvard School of Public Health [Rom12].

The first goal of this project is to automate the installation, deployment and testing of the aforementioned pipeline. On the other hand, the alignment[1] step of the analysis will be modified to use Seal, a Hadoop based aligner. This will allow us to check that all automations are working properly, as the pipeline will have to be installed and tested in several nodes.

[1] A sequence alignment is a way of arranging the sequences of DNA, RNA, or protein to identify regions of similarity that may be a consequence of functional, structural, or evolutionary relationships between the sequences

Acknowledgements

Roman Valls Guimerà, my supervisor at *Science For Life Laboratory*, for giving me the opportunity to carry out this project. His advice, support and knowledge during the whole project have made possible its completion.

Daniel Jimenez Gonzalez, my supervisor in Barcelona, for accepting the task of supervising this project from the first moment, even the given distance between us. Thanks a lot for all the insights during this project, and for the incredibly accurate reviews of this report on every single chapter.

Adrià Casas Escoda for his invaluable help on writing this report. His consideration and reviews of this report, together with his experience, have improved my writing skills.

Thomas Svensson, Per Kraulis and Joakim Lundeberg for trusting me and giving me the means to fulfill this project, from economical support to making me feel at home in a foreign country.

The team in Biomedinfra, in Finland, specially Jarno Laitinen, for his incredibly efficient technical support.

Luca Pireddu, author of Seal, for all his support while answering my questions with very clear explanations and giving me very valuable hints and recommendations on how to work with Seal and Hadoop.

Pontus Larsson, Valentine Svensson, Per Uneberg and, in general, all the people in *Science For Life Laboratory* that at some point have shown interest for my work and have offered his/her help.

Thank you everybody.

Contents

Nomenclature

BAM	Binary Alignment Map
CI	Continuous Integration
DFS	Distributed File System
DVCS	Distributed Version Control System
HDFS	Hadoop Distributed File System
HPC	High Performance Computing
NAS	Network Attached Storage
NGS	Next Generation Sequencing
OS	Operating System
PPA	Personal Package Archive
SAM	Sequence Alignment Map
SNP	Single Nucleotide Polimorphism
TDD	Test Driven Development
VM	Virtual Machine

Chapter 1

Introduction

Bioinformatics is an interdisciplinary field that aims to develop and improve methods to retrieve, store, organize and analyze biological data. It encompasses an extensive number of research areas, which makes it difficult to precisely define it. In experimental molecular biology, bioinformatics techniques such as image and signal processing allow extraction of useful results from large amounts of raw data. In the field of genetics and genomics, it aids in sequencing and annotating genomes and their observed mutations. Basically, it has become of crucial importance in biology, as it is the key to understand the meaning of biological data [Les02].

The necessity of understanding the DNA structure has driven the development of Bioinformatics. Nowadays, week-long sequencing runs on a single machine can produce as much data as entire genome centers did a few years ago. This translates into an exponential growth of the information available. This growth can be observed in figure 1.1.

As a result, the need of processing terabytes of information has become *de rigueur* for many laboratories engaged in genomic research. Fast and scalable computing solutions are constantly needed.

Together with the need of fast solutions that deal with huge amount of data, there is another problem. Sometimes in this kind of scientific environments, the structure of the software development platform is not very well defined. The merit of really good algorithms is occasionally over-casted by the difficulty on scaling the software, or by a tedious installation procedure of this one. The over-fitting to an specific platform is also a common problem in bioinformatic core facilities.

1.1 Motivation

This work originates from the needs of a concrete genome sequencing and analysis center: Science For Life Laboratory, in Stockholm, Sweden[1].

[1] `www.scilifelab.se`

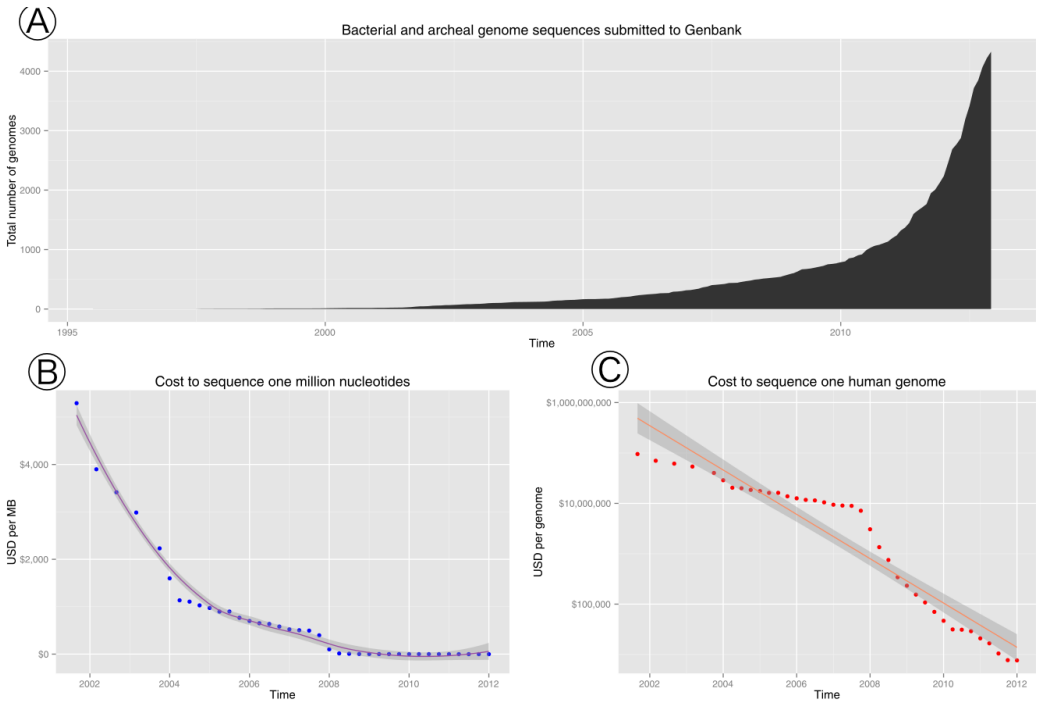

Figure 1.1. (A) Exponential growth of genome sequence databases since 1995. (B) The cost in US Dollars (USD) to sequence one million bases. (C) The cost in USD to sequence a 3,000 Mb (human-sized) genome on a log-transformed scale.

Science For Life Laboratory is a center for large-scale biosciences with the focus in health and environmental research. It is comprised of multiple research platforms: *Proteomics, Genomics, Bioimaging and Functional Biology, Bioinformatics and Comparative Genetics*. The Genomics platform is the largest technical platform and provides access to technology for massively parallel/next generation DNA sequencing and bioinformatics support. Altogether, 15 next generation sequencing instruments are available at present, with a combined capacity for DNA sequencing equal to several hundreds of complete human genomes per year. However, there is a negative aspect in this capacity of generating big amounts of data. The techniques and tools used to analyze this data have had to evolve and adapt to that growth equally faster. In fact, the lack of time to accurately plan and design software solutions has turned out to be a problem when trying to scale or modify them. Note that software development is not only writing code. It is also part of the development process to plan and set up a convenient working environment, satisfy third-party dependencies, procure a replicable and reliable installation procedure, and continuously test the product.

If none or few of those processes are automated, the developer will spend a

considerable amount of time setting up things, rather than implementing new ideas. Though not everything can be automated, the *idyllic* situation is that in which the developer is totally focused on the software *per se*.

Automating is not just a matter of saving time, but also of reliability. Manual and repetitive steps in processes usually means a high chance of errors. Experience tells us again and again that trying to run a "simple" process by implementing a series of "simple" manual steps, backed by "simple" written instructions, doesn't work well. It is very common to just forget, skip or bungle a step.

It is usual not to pay much attention to automation in scientific environments. Usually, the rush due to the need to publish makes the software development process to focus exclusively in immediate results [Cla05]. Another impediment is the lack of dedicated funding for software development. This usually results in a software that is difficult to install, scale or modify.

In this project we analyze which processes can be automated in Science For Life Laboratory's bioinformatics core facility. We also integrate and test a highly scalable sequence aligner that will help to test all the automations. This integration will also show how reproducible research should be carried out [Bro12].

1.2 Objectives

This project has two main goals. The first one is to standardize and automate several procedures carried out in Science For Life Laboratory when it comes to *bcbio-nextgen* pipeline installation and testing. The second objective is to evaluate a highly scalable and reliable sequence alignment software, Seal [Pir11]. This software, together with the pipeline, will be installed in a bunch of machines, which will be a real test for the automations done previously. Taking advantage of the situation, as we integrate and test Seal, we will also show how to test new software solutions in a reproducible way.

To accomplish those objectives we will perform a deep analysis of the current structure of *bcbio-nextgen* pipeline. This will help us not just to determine what can be automated, but also to understand the pipeline wokflow.

To automate the installation of the pipeline, we will package all its external dependencies in a standard way. After that, we will implement an automatic procedure to install the pipeline. Also, in order to automate the testing of the pipeline, we will integrate its development within a Continuous Integration (CI) system. Doing so, the developer will not have to take care of running the test suite each time a change is done in the code, as this will be done automatically. The user, on the other side, will be aware of the state of the current release, as the information about the tests results will be public and continuously updated.

To evaluate the sequence aligner, the installation of this software in a cluster of computers needs to be planned and structured. This should be done in such a way

that is as much reproducible as possible, so other scientists can perform the same tests that we do with minimal effort.

1.2.1 Specific objectives

The main specific objectives for this project follow:

1. Study the structure and processes carried on by *bcbio-nextgen* genomics pipeline.
2. Automate the installation of *bcbio-nextgen* pipeline:
   - Package external dependences to ease the pipeline installation.
   - Automate the setting up of the working environment.
   - Procure an easy way for new users to test the pipeline.
3. Automate the pipeline testing:
   - Integrate the pipeline development within a CI system.
4. Test Seal:
   - Plan the structure of the testing cluster and the tests to be performed.
   - Implement a set of recipes to automatically configure the nodes in the cluster.
   - Test seal with different datasets and cluster configurations. Compare performance against BWA, the most widely used sequence alignment tool [LD09], and the one used in Science For Life Laboratory.
   - Integrate Seal into the current pipeline and perform a complete analysis.

1.3 Methodology

The general methodology followed to accomplish the goals of this project is described below.

First, we have been working in our own copy of the pipeline's repository. With the aim of getting familiar with the installation procedure, and also with the software used by the pipeline, it is installed and tested as it is as a first approach. This allow us to detect automatable steps in the installation. A deeper study of the pipeline workflow is done after the installation.

After having a clear idea on how the pipeline should be installed to work properly, we proceed to implement the automatic installation process. To do so, we perform the manual installation step by step, building progressively a system to install the pipeline. To ensure reliability, the tests are executed in clean virtual

machines, so no previous installation can interfere. This ensures that the installation will work in a raw system.

Once the automatic installation has been implemented, we test it thoroughly. To do so, we integrate the pipeline deployment and testing in a CI system. Thanks to the CI, on each deployment the test suite of the pipeline is ran. This becomes an implicit checking of the correctness of the pipeline installation. This allows us to check that the deployment of the pipeline is working properly.

The next step is to integrate the Hadoop-based sequence aligner Seal into the pipeline. Before the integration, we plan the structure of the testing cluster. We decide to test it against different cluster configurations, as this is also a mesure of performance. The next step is to prepare data sets to test against Seal. Once the data sets have been prepared, we test Seal in all the different cluster configurations. We compare the results and performance with the currently used aligner.

1.4 Outline of the report

The document is organized as follows:

Chapter 2 presents a general background about DNA sequencing and analysis. It explains the process of obtaining the data and the most common steps in an analysis. It introduces the concept of genomics pipeline.

Chapter 3 describes *bcbio-nextgen* pipeline and its workflow. The automatic installation implementation and the CI integration is explained in chapter 4.

In Chapter 5 we describe the Hadoop architecture and how we test Seal. We also explain how we have adapted the pipeline to work with Seal. Results are shown in Chapter 6.

Chapters 7 and 8 show a summary of costs that this project has carried and final remarks and conclusions of this project.

Chapter 2

Background on DNA sequencing and analysis

In the introduction, we talked about the huge increment of data generation in laboratories involved in genetics research due to high-throughput sequencing machines. In this chapter, we present a brief background on what is DNA and DNA sequencing. This background is important in order to understand the data we are trying to analyze. It also helps us to understand the kind of results produced by the sequencing machines.

2.1 DNA sequencing

2.1.1 What is DNA?

DNA, or deoxyribonucleic acid, is the hereditary material in humans and almost all other organisms. Nearly every cell in a person's body has the same DNA. The information in DNA is stored as a code made up of four chemical bases: adenine (A), guanine (G), cytosine (C), and thymine (T) [Les02]. Human DNA consists of about 3 billion bases, and more than 99 percent of those bases are the same in all humans. The order, or sequence, of these bases determines the information available for building and maintaining an organism, similar to the way in which letters of the alphabet appear in a certain order to form words and sentences.

DNA bases pair up with each other, A with T and C with G, to form units called base pairs. Each base is also attached to a sugar molecule and a phosphate molecule. Together, a base, a sugar molecule, and a phosphate molecule are called a nucleotide. Nucleotides are arranged in two long strands that form a spiral called a double helix. Figure 2.1 shows the well known double helix shape of the DNA.

The double-helix model of DNA structure was first published in the journal Nature by James D.Watson and Francis Crick in 1953 [CW53]. Already in that time, molecular biologists knew that DNA contained the hereditary information.

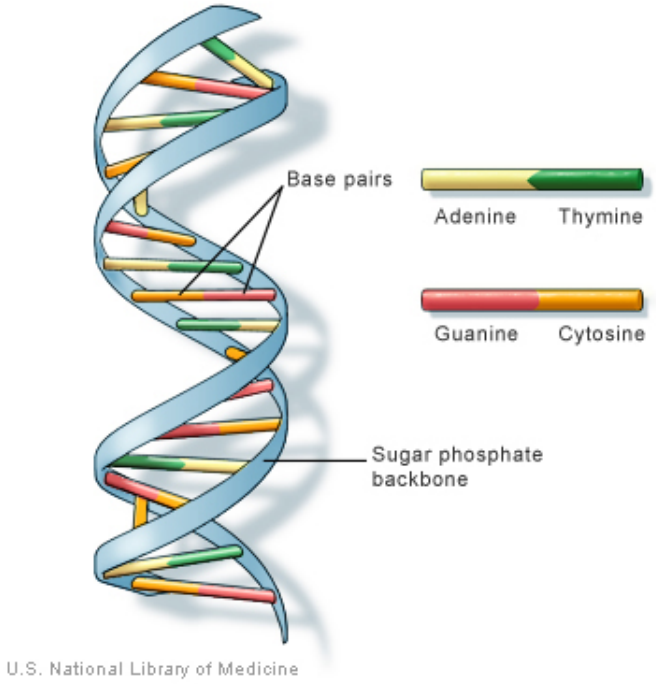

Figure 2.1. Double-helix shape of DNA

However, several decades passed before fragments of DNA could be reliably analyzed for their sequence in the laboratory.

Fredrick Sanger, Cambridge biochemist who devoted his career to sequence determination of biological macromolecules, determined for first time the amino acid sequence of the protein insulin in 1958 [Les12]. His work was rewarded with the Nobel Prize in Chemistry in that year. He next turned his attention into RNA, and finally into DNA. His methods for sequencing of DNA won a second Nobel Prize in 1980.

Based on Sanger's studies, Allan Maxam and Walter Gilbert published a DNA sequencing method in 1977 based on chemical modification of DNA and subsequent split at specific bases. However, this method's use of radioactive labeling and its technical complexity discouraged its extensive use [MG77]. Though it was not very extended, it can be considered as the first DNA sequencing method.

2.1.2 Sanger sequencing method

Fredrik Sanger proposed a method that used fluorescent dyes as reporters, rather that radioactivity, in the same year. This was an important technical advance in sequencing. The chain-termination method developed by Frederick Sanger and

coworkers soon became the method of choice, owing to its relative ease and reliability. The chain-terminator method uses fewer toxic chemicals and lower amounts of radioactivity than the Maxam and Gilbert method. Because of its comparative ease, the Sanger method was soon automated and was the method used by the first generation of DNA sequencers [SC75]. As a very short summary, this method consists on the following steps:

1. To decode the sequence of A, T, C and G nucleotides in a piece of DNA, the template DNA (the sample) is copied repeatedly.
2. The copying reactions stop when modified nucleotides called "chain terminators", which are fluorescently labeled, are added.
3. This is repeated many times, generating a large number of fragments of different lengths that end in fluorescently labeled bases. By analyzing these fragments the original sequence can be read.

The automated Sanger method is considered as a "first generation" technology, and newer methods are referred to as next-generation sequencing (NGS).

2.1.3 High-throughput sequencing machines

The high demand for low-cost sequencing has driven the development of high-throughput sequencing (or next-generation sequencing) technologies that **parallelize** the sequencing process, producing thousands or millions of sequences at once. High-throughput sequencing technologies are intended to lower the cost of DNA sequencing beyond what is possible.

To begin with, a sample of DNA is acquired from living tissue. The sampled cells are then broken apart, first mechanically and then chemically, so that the DNA material can be extracted and isolated from other cell material. Once the DNA is isolated, the biochemical procedure may vary for different sequencing technologies [Met09]. Figure 2.2 shows the processes from the biological sample to the files that are being computationally analyzed. Here we treat briefly how to sequence with the Illumina technology, as is the technology used in *Science for Life Laboratory*, and thus we can explain how we obtain the data that will be analyzed later on.

In these platform sequences are bound to a glass surface (flow cell). In this flowcell, several million DNA fragments are generated in each of the eight channels (or lanes) of the flow cell. These fragments form distinct clusters on the surface. Each fragment is first read from one end and then from the other, producing a total of two reads. This two reads are called a paired reads. With the HiSeq 2000, one of the newest models of Illumina sequencers at the time of writing this report, the read length is typically about 100 bases, meaning that for each DNA fragment of 300–500 nucleotides we have read the first and the last 100 bases, leaving an unread section in between.

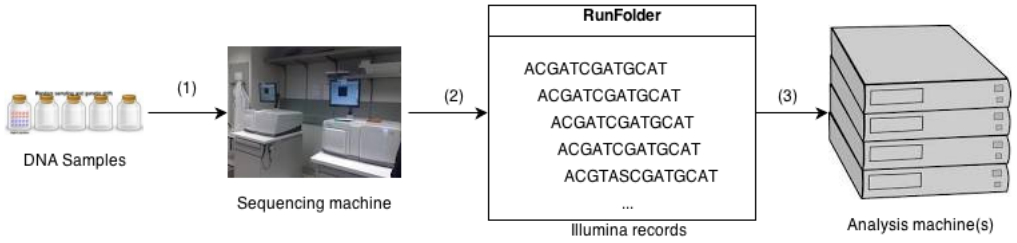

Figure 2.2. Steps to obtain the data to analyze. (1) DNA samples are prepared in the laboratory from biological tissue. The samples are introduced in the sequencing machine, where the're sequenced. (2) During the sequencing process, the DNA samples are broken into fragments. The machine's output consists on a set of records representing this fragments. (3) The computational analysis can start.

From a single run, the sequencing machine can produce about 800 GB of data. The amount of data generated in *Science For Life Laboratory* within the past year is shown in figure 2.3. The data generated consists on billions of records, two for each DNA fragment generated in the sequencing process. Each one of this records contains four pieces of information: a key identifying the DNA fragment, the read number (one or two), the DNA sequence of the fragment and the quality score for each base in the sequence, which estimates the probability of a reading error at each base. The output in Illumina's format is later converted to fastq format, the standard format for representing DNA reads. In figure 2.4, we can see a sequence in FASTQ format.

Here is where the work in the "wet lab"[1] ends, and where the bioinformatics analysis starts.

2.2 Data analysis - A genomics pipeline

Once the data has been generated by the sequencing machines, it is time to analyze this data. There are a lot of tasks that needs to be done in a bioinformatics core facility.

A genomics pipeline is basically a software to manage the experimental data flow. Its main task is to automate as much as possible the processes carried out with the data. From the moment it is produced by the sequencers to the moment when the analysis results are presented to the customer, the information flow should be handled by a pipeline. The kind of analysis that are performed in a genomic center is a choice of the center itself. This means that even if a general solution of pipeline can be provided, it always needs some adjustments to fit the particular needs of the center where it is running. However, there are some tasks that are common in any laboratory.

[1]Wet laboratories are laboratories where chemicals, drugs, or other material are handled in liquid solutions or volatile phases, requiring direct ventilation, and specialized piped utilities.

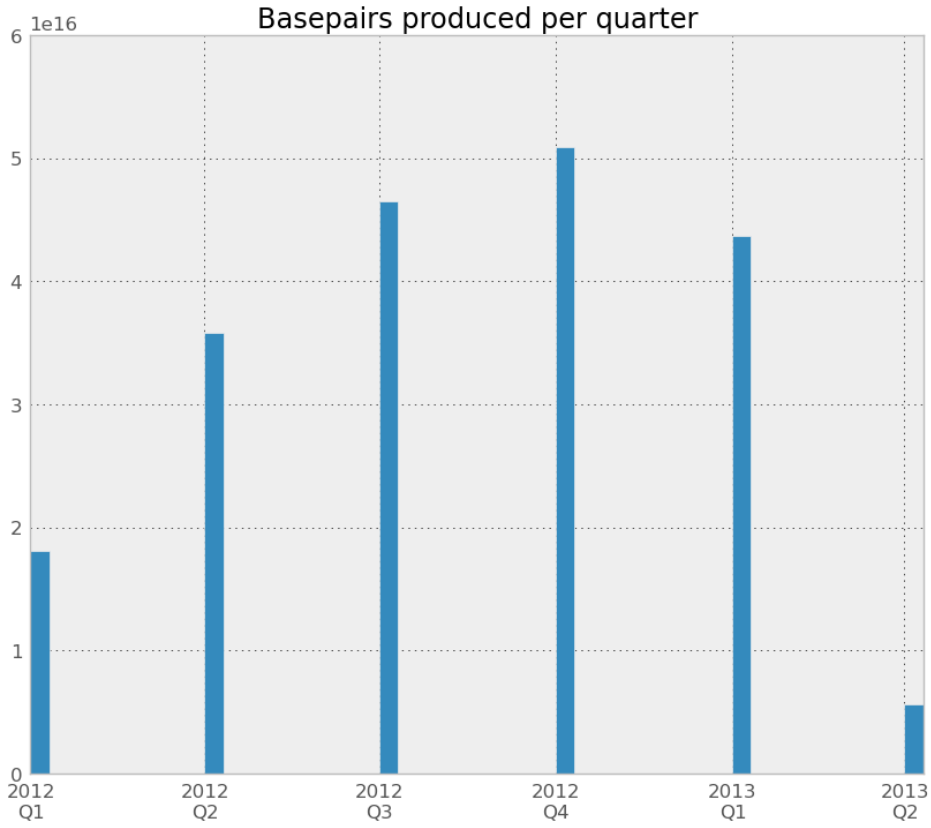

Figure 2.3. Data produced in Science For Life Laboratory within the year 2012 expressed in base pairs produced by quarter.

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

Figure 2.4. Record in FASTQ format. First line starts with @ and is followed by the sequence id. It may contain other information like flowcell lane, coordinates in the corresponding fragment, etc. Second line is the raw sequence letters. Line 3 starts with +, is a separator. Line 4 encodes the quality values for the sequence in Line 2, and must contain the same number of symbols as letters in the sequence.

```
First string:  ABCDE
Second string: ACDEF
Possible alignment 1:    ABCDE-     Possible alignment 2: ABCDE-
                         A-CDEF                           -ACDEF
```

Figure 2.5. Possible alignments of two strings. In the second case, we allow a mismatching.

First of all, the sequencing machines are connected either to a Network Attached Storage (NAS) or to a computer where the output data is stored. As one can imagine, the storage capacity of this devices is limited. It is very common to have a dedicated machine or disk for the storage, as this incurs a very high number of I/O operations. For the analysis to be possible, the data should be prepared (converted to FASTQ format) and moved to the analysis machine. This is the first task of a genomics pipeline. The following tasks depend on the protocols defined by the laboratories. One of the first steps is usually the sequence alignment.

A sequence alignment is a way of arranging the sequences of DNA to identify regions of similarity that may be a consequence of functional, structural, or evolutionary relationships between the sequences [Les02]. Computationally speaking, the sequence alignment is the process of matching two or more strings in such a way that the edit distance is minimal. Any assignment of correspondences that preserves the order of the bases within the sequences is an alignment, so several alignments are possible. Figure 2.5 shows a possible alignment for the strings “ABCDE” and “ACDEF”.

When aligning, one can be more or less permissive, allowing gaps, mismatches or substitutions. All alignment tools take care of this possibilities, allowing the user to tune the parameters of the alignment to make it more or less precise. Given the volume of data generated by the sequencers, this step in the analysis is one of the most computationally intense (if not the most). This is why we decided to adapt a highly scalable sequence alignment into *bcbio-nextgen* pipeline.

On the next chapter, we detail the protocol followed in *Science For Life Laboratory* for the analysis of the data. This protocol is carried out by *bcbio-nextgen* pipeline.

Chapter 3

Current Pipeline: *bcbio-nextgen* Pipeline

In this chapter, we describe the structure of *bcbio-nextgen* pipeline and the analysis carried on by this pipeline.

Bcbio-nextgen is entirely written in Python. It was originally implemented by Brad Chapman at Harvard School of Public Health, and introduced and adapted to *Science For Life Laboratory* by Roman Valls Guimerà in 2011 [Rom12]. Since then, *bcbio-nextgen* has been the pipeline used for the automation of the analysis in the genomics core.

3.1 How does *bcbio-nextgen* work?

Bcbio-nextgen is composed of a set of scripts that automate the flow of the analysis and watch for event files from the sequencing machines.

For each sequencing experiment, the sequencing machines create a folder with a unique name in which they store the sequencing data, this directory is called run directory. They also have a mechanism to indicate their progress. This mechanism consists on the creation of "status" files that they store in the run directory. These files are checkpoints that indicate things like when a sequencing has started, when it has finished or the state of key steps in the sequencing. *Bcbio-nextgen* consumes this files in order to trigger the proper actions. In figure 3.1 a general overview of the whole system is showed. A complex communication scene is handled automatically by *bcbio-nextgen*.

3.1.1 *Bcbio-nextgen* work flow

The first step in the pipeline work flow is the data conversion and demultiplexing.

The sequencing machines produce files that cannot be directly analyzed without a previous conversion. As explained in section 2.1.3, the standard format

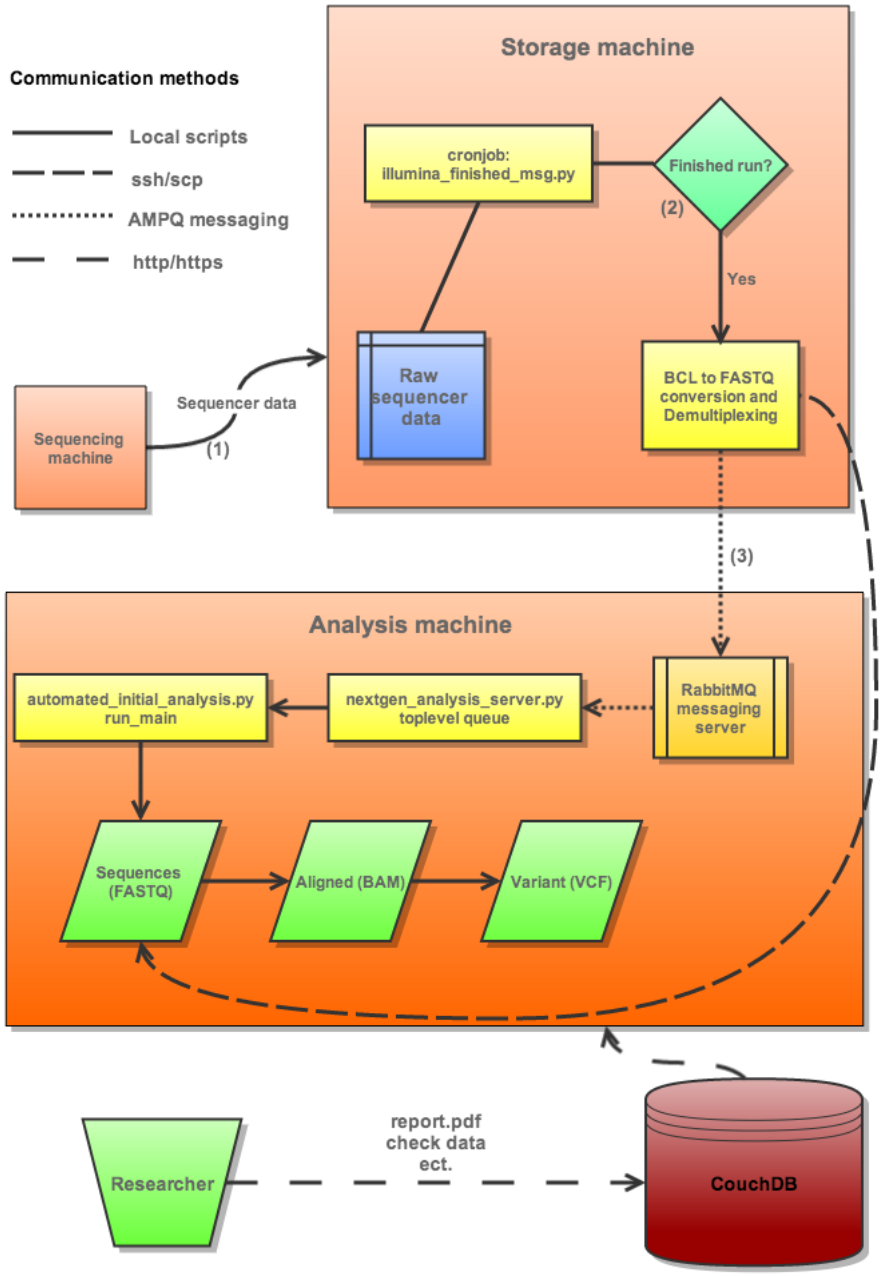

Figure 3.1. General overview of the interaction between *bcbio-nextgen* and the rest of the system. (1) The sequencer machine outputs the data into the storage machine. (2) The storage machine checks if the run has finished and acts accordingly. (3) The storage machine sends a message to the analysis machine to start the processing.

for sequence files is FASTQ. Illumina machines like the ones running in *Science for Life Laboratory* produce BCL files[1] that need to be converted to FASTQ. Also, in the laboratory it is common to put different biological samples in the same lane of the sequencing machine. The data generated for these samples needs to be separated. This process is called **demultiplexing**.

The BCL conversion and demultiplexing is triggered by the pipeline when it detects that a sequencing run has finished (or when the first read ends in a paired-end run). For both processes, the pipeline invokes the proprietary Illumina's software CASAVA[2].

Once the data has been converted and demultiplexed, the proper analysis of the produced data starts. To that purpose, the data is transferred from the NAS to the analysis machine(s), in our case, to UPPMAX. Figure 3.2 shows the work flow of *bcbio-nextgen* for the first steps in the analysis:

1. The first step is called a PhiX filtering[3]. This is a quality control, and its aim is to remove reads that pertain to the virus Phi X 174. Phi X is regularly used as a control in DNA sequencing due to its relatively small genome size in comparison to other organisms (5.4 KB in FASTA format) and the extensive work that has been done on it.
2. The next step is to remove contaminants. Sequences obtained from impure nucleic acid preparations may contain DNA from sources other than the sample. This process aims to remove this foreign DNA [SE11].
3. After the previous quality controls, the third step is the sequence alignment, which involves several subprocesses:
   a) A configuration file in the pipeline allows the user to select the program to perform the alignment. The most commonly used is BWA. This process produces a Sequence Alignment Map (SAM) file(s) as a result [LHW$^+$09].
   b) After performing the alignment, the resulting SAM file is converted to Binary Alignment Map (BAM) format, which is a binary compressed (and therefore lighter) version of the SAM file.
   c) Finally, the resulting BAM files from the different sequence files are merged and sorted for better post alignment processing.
4. Following the sequence alignment, the next steps in the analysis are intended to give an overall quality and sample information. The flow of the pipeline is guided by configuration files, as expressed in the flowchart in figure 3.3, where can be seen that different actions are performed depending on the configurations. A short description of those processes follows:

[1] BCL files are binary files that contain base call and quality for each tile in each cycle of the run

[2] Illumina's Consensus Assessment of Sequence and Variation (CASAVA)

[3] `http://en.wikipedia.org/wiki/Phi_X_174`

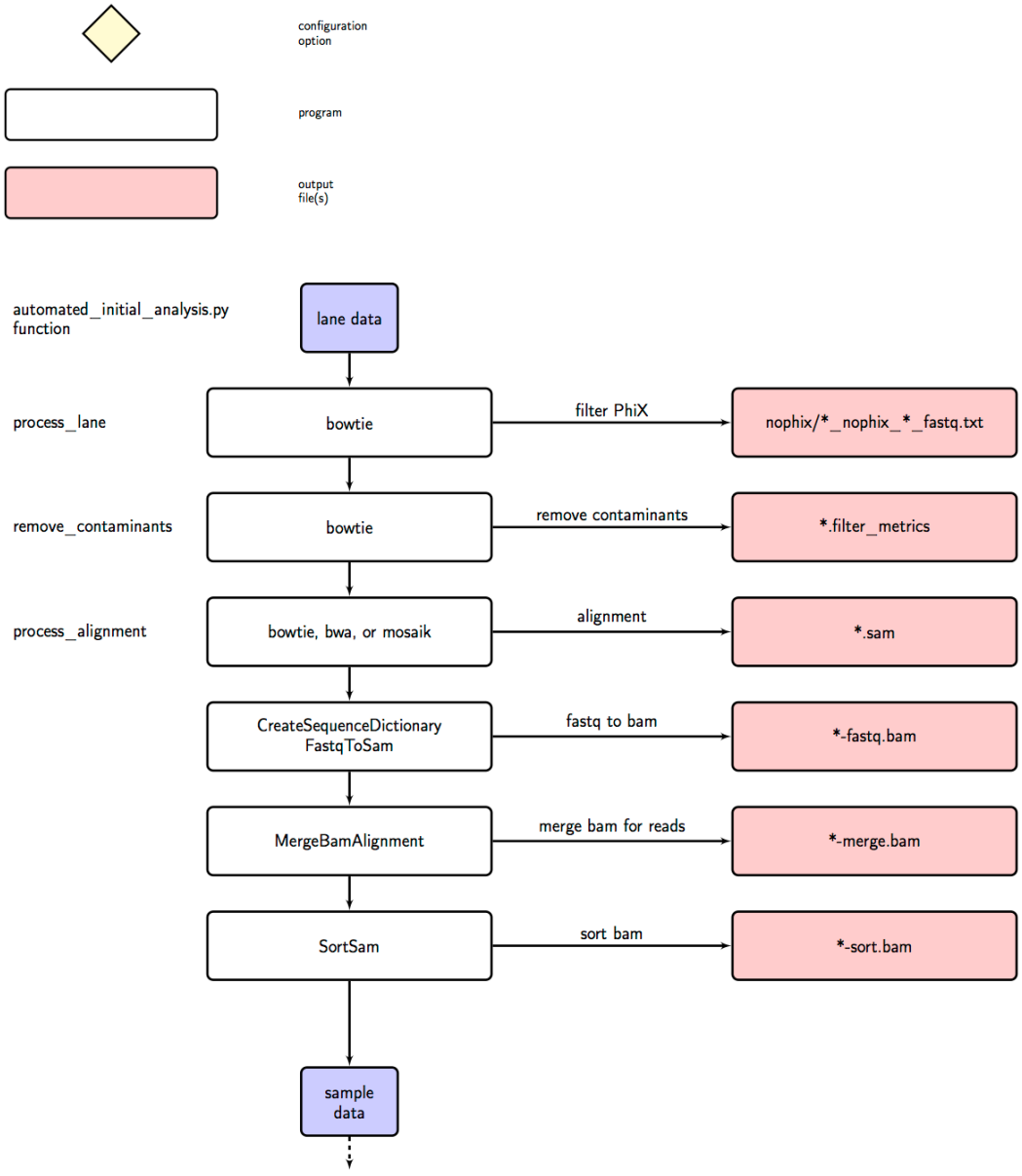

Figure 3.2. First steps in *bcbio-nextgen* pipeline: PhiX filtering, contaminants removal and sequence alignment

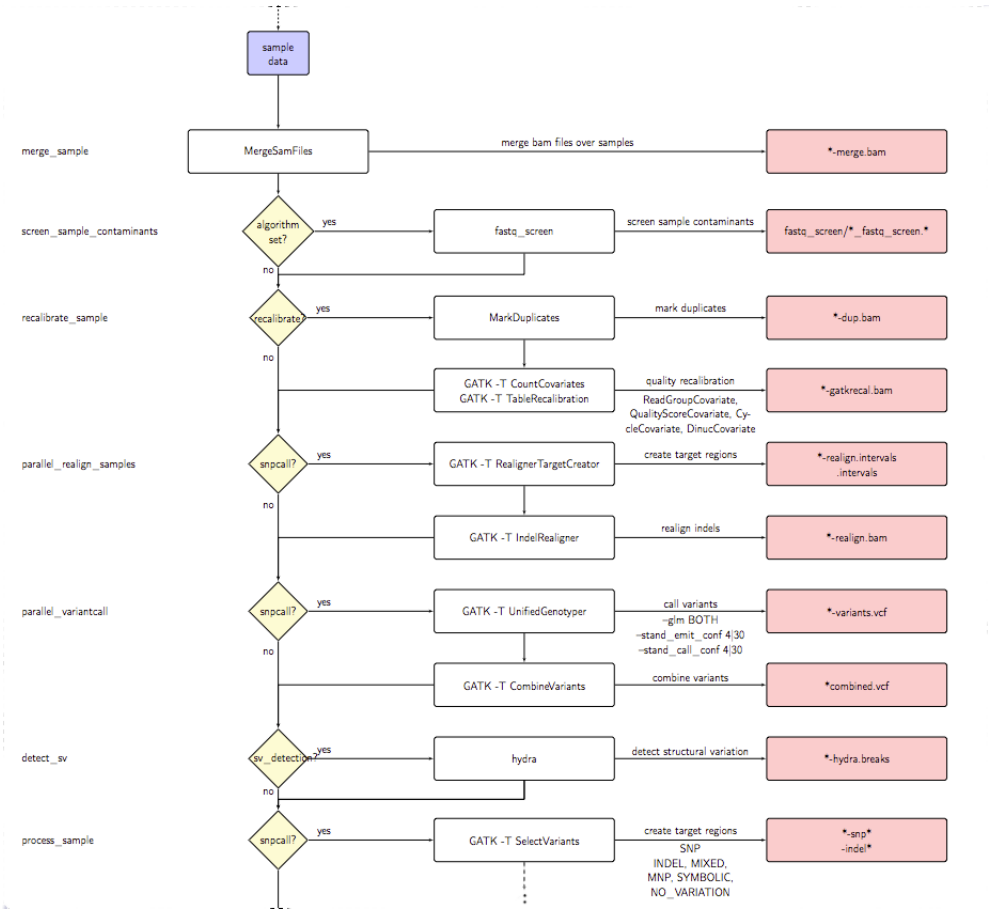

Figure 3.3. After-alignment steps in *bcbio-nextgen* pipeline: Fastq screening, mark duplicates and SNP calling

- **Fastq screening** is the process of mapping your reads against various reference genomes, in order to see how good your reads are. For example, if you're working with mouse reads, you expect these reads to map mainly to the mouse genome. Figure 3.4 illustrates precisely this case, where a set of mouse reads have been aligned against several reference genomes, and indeed about the 70% of them match with mouse genome.
- **Mark duplicates** is the process of removing possible duplicate reads from a sequence, one of the major Illumina concerns due to its sequencing technology [KNQ+09].
- **Single-nucleotide polymorphism (SNP) calling and Variant Calling** are intended to detect variations in the DNA. Figure 3.5 illustrates graphically what is a SNP. This process exposes the differences between the sequenced sample and the standardized reference genome.

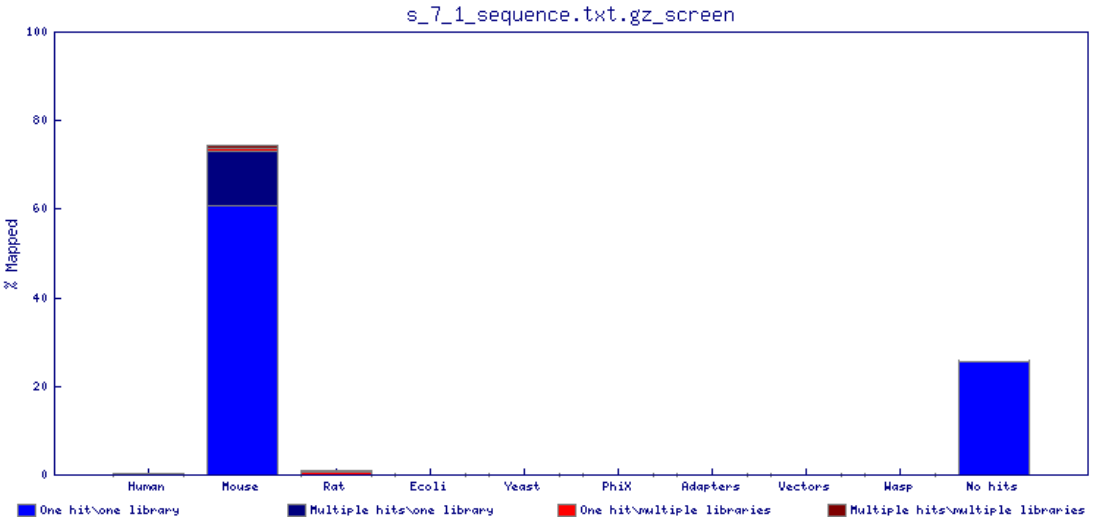

Figure 3.4. Expected screening results for a set of mouse reads. Source: `http://www.bioinformatics.babraham.ac.uk/projects/fastq_screen/`

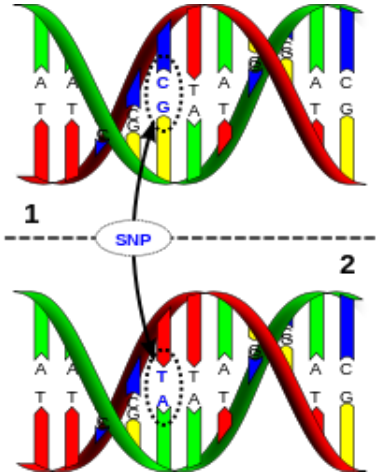

Figure 3.5. Single Nucleotide Polymorphism example

- **Transcriptome assembly**: The pipeline can be configured to do other kind of sequence analysis, like RNA-seq. RNA-seq uses high-throughput sequencing technologies to get information about the RNA content of a sample. This produces a set of transcripts (transcriptrome[4]) that need to be assembled. This process[WGS09] is out of the scope of this project. Figure 3.6 shows the schema for these last steps in the analysis.

[4]The transcriptome is the set of all RNA molecules produced in one or a population of cells

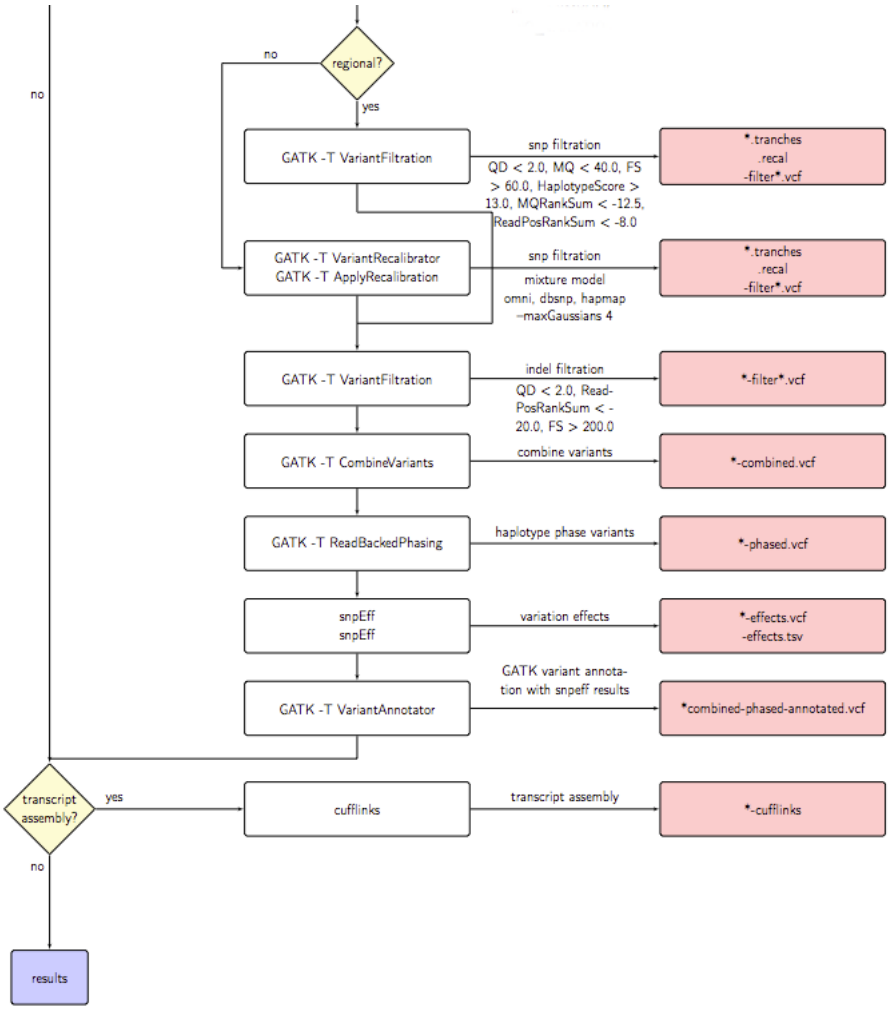

Figure 3.6. Last steps in *bcbio-nextgen* pipeline: Recalibrating and transcriptome assembly.

During the analysis, the relevant results are sent to a local database running CouchDB for future study [Len09]. When the analysis finishes, the researcher can generate a final summary report and get it from the database.

The pipeline installation

As said before, the analysis is guided by configuration files. These files, apart from specifying which parts of the analysis have to be performed and which skipped, allow the user to select between different tools to perform each task. For each one of these programs, there is a "wrapper" module in Python. This wrapper allows the use of this programs (written in many programming languages) to be called within

the pipeline. This modularity eases the integration of new tools, for which the developer only needs to write a new wrapper. However, installing all these modules and tools is not straightforward for non-experienced users.

At the time of starting this project, the installation process of *bcbio-nextgen pipeline* was mainly manual. The user had to follow a list of steps in order to have the pipeline running on their system. In the next chapter, we describe the installation process at the moment of starting this project and the methodology followed for its automation.

Chapter 4

Automatic pipeline Installation: Design and Implementation

Regardless of the complexity of gluing and synchronizing all the steps required for a genomic analysis, a Genomics Pipeline, explained in chapter 3, is no more than a set of tools that interact together.

Those tools have not necessarily been created by the same developer, nor even in the same OS or programming language. In fact, software heterogeneity is usually high[HMS+09] in scientific community, and sometimes, very research group dependent. In some contexts, bioinformatic research is performed by individuals writing private *ad-hoc* software to quickly resolve specific needs [Rom12]. Those facts become an issue when trying to re-use some existing tools, having to deal with problems like:

- Lack of documentation.
- A lot of manual steps: Users following steps from one or more manuals.
- Pre-compiled packages without the source code available that doesn't work in all systems (i.e. 64 bits compilations in 32 bits systems).

Bcbio-nextgen pipeline is a perfect example of that heterogeneity. It uses a wide set of tools to perform different types of analysis.

We can distinguish between two different parts of the pipeline: One one hand, we have the server scripts that check the status of the sequencing machines, look for new data available, etc. Those are scripts that need to be placed manually in their correct places (for example in the NAS attached to the sequencing machine). As the configuration of the hardware infrastructure is inevitably dependent on the laboratory, this part of the installation is hardly automatable. On the other hand, we have the set of python modules and tools that, all together, manage the flow of the analysis, and this part should be installed without any user intervention.

The process of installation of these python modules was completely manual and not very straightforward before this project. As a summary, the steps that a user had to do in order to have the pipeline installed and configured:

1. Set up a Python virtualenv[1]:
   - Edit the bashrc configuration file to correctly set up PYTHONPATH and read from other configuration files
   - Install virtualenvwrapper[2]
   - Create a virtual environment for the pipeline
   - Edit this virtualenv to use the user's python library instead of the system's.
2. Set up the pipeline code:
   - Remove any existing copy of the pipeline.
   - Clone the code of the pipeline from SciLifeLab GitHub repository.
   - Install the pipeline into the previously created virtual environment.
3. Set up custom modules:
   - Clone the scilifelab-specific module definitions from SciLifeLab Github repository.
   - Point the module system environment variables to the user's custom location by editing again the bashrc file.
4. Set up SciLifeLab utility scripts:
   - Remove any pre-existing versions of the utility scripts.
   - Clone the code from the GitHub repository.
   - Install the scripts into the virtual environment.

Furthermore, those steps assume that all the pipeline dependencies are already installed in the system. For a minimal sequence analysis, the dependencies are[Gui12]:

- BWA aligner
- Picard-tools
- Bowtie
- Freebayes
- snpEff
- Fastqc
- GATK 1.6
- Tophat
- Samtools
- Cufflinks
- WigToBigWig

[1] virtualenv is a tool to create isolated Python environments

[2] `https://bitbucket.org/dhellmann/virtualenvwrapper`

4.1 Creation of packages for external software dependencies

As corresponds to the concept of pipeline, *bcbio-nextgen* pipeline has a high number of software dependencies. Most of them do not have an automatic installation procedure. For this reason, a good approach is to "pack" all the pipeline dependencies in a common way.

In particular, we decided to follow the Ubuntu/Debian official packaging guide and build distributable packages for each dependency. That guide is a specification of how the software needs to be packaged following Ubuntu/Debian standards. The intention choosing this Linux distribution is to reach the maximum number of users. Ubuntu and Debian are widely used distributions maintained by a huge community of active developers.

Debian and Ubuntu use special software repositories for uploading source packages to be built and published. These repositories are called Personal Package Archives (PPA)[Bla00]. Using PPAs, the developers can distribute software and updates directly to Ubuntu and Debian users. When creating a software package and uploading it to a PPA, users can install it in the same way as they install standard Ubuntu packages, and will automatically receive updates when the developer makes them. To install the packages in any computer, the PPA needs to be added to the system.

In short, the processes of packaging and distributing consist on the following steps:

1. Create a public PPA.
2. Package the source code (or the binaries if the source code is not available). Basically this is done by setting up the corresponding files that tell the building system how to compile and/or install the software.
3. Locally test that the package works, simulating the exact environment that Ubuntu and Debian use to build the packages.
4. Publish the package in the PPA.

However, not all the software required for the pipeline needs to be packaged. Some of the tools were already present as official packages. Even for those dependencies not present as official packages, we searched for third-party PPAs that could provide these tools (we do not want reinvent the wheel!).

We found out that there is a team of developers working in a medical research based distribution of Debian. This distribution is called Debian-Med, and it aims to adapt Debian into an operating system that is particularly suited for the requirements of medical practice and biomedical research. Debian-Med has its public PPA, and there we could find packages for some of the dependences of the pipeline. For the rest of dependencies, we could not find any public PPA or

something similar that suited our needs. Therefore, we decided to package and publish into a new PPA the rest of the dependencies.

We packaged and published the rest of the dependencied in our *Science for Life Laboratory* PPA[3]. In some cases, we contacted with the authors of the respective tools to inform them of the availability of their software as Ubuntu packages, getting back very positive responses.

4.2 Automation of the installation step

Once the dependences have been packaged, the next step is to automate the installation of the pipeline itself. The installation procedure should take into account the kind of users that will use this package.

In general, the users are scientists doing research in *Science For Life Laboratory* that want to perform some kind of analysis. They usually have access to UPPMAX[4] [upp08], an academic HPC (High Performance Computing) environment. In this kind of environments, the permissions policies are usually restrictive, and users do not have root permissions.

Another common case is scientists working in other laboratories that are looking for new tools to improve their research. In that case, they usually do not want to deal with tedious installation procedures or configurations, but just see what the software can do for their research.

Having this in mind, the system to automate the installation of the pipeline has been thought in two ways:

1. **Local installation** in the machine: The pipeline is fully installed in the hosting OS.
2. **Virtual machine installation**: The pipeline is installed in one or several virtual machines created by the system itself.

Science For Life Laboratory installation

This is the most common way of installation. As most of the users of the pipeline are individuals with access to some kind of academic HPC like UPPMAX, the installation procedure **must** have this present and act accordingly to the peculiarities of these systems [Rom12].

This installation option uses the environment variables present in the HPC to install the pipeline in harmony with the execution environment.

It is important to note that no root permissions are required to execute it. If it is being executed in an HPC, the installation system tries to load the software

[3] `https://launchpad.net/~scilifelab/+archive/scilifelab`

[4] Uppsala University's resource of high-performance Computing: `http://www.uppmax.uu.se/`

dependencies of the pipeline through the module system[Fur91], and aborts the installation if any dependency is not found.

This mode can also install the pipeline in a personal computer. In this case, it assumes that all the dependencies are installed: the user most probably have root access and can install the dependences through the Ubuntu/Debian packages if they are not already installed.

In any case, the installation of the pipeline is done following the steps described at the beginning of this chapter. In figure 4.1, the usage instructions of this script are shown. Basically, it can install or remove the pipeline form a system, as well as update an existent installation.

```
usage: deploy_non_root.py [-h] [-v VERSION] [-d DIRECTORY] [--no-tests]
                          {install,uninstall,update} ...

Script to automatically install and configure bcbio-nextgen pipeline

positional arguments:
  {install,uninstall,update}
    install             Install the pipeline within a python virtual
                        environment (also created with this command)
    uninstall           Removes the pipeline and unisntall the created
        virtual
                        environment (master)
    update              Update the code of an existing installation

optional arguments:
  -h, --help            show this help message and exit
  -v VERSION, --version VERSION
                        Choose a version of the pipeline to pull. Use a
                            commit
                        hash as version
  -d DIRECTORY, --directory DIRECTORY
                        Directory of the pipeline installation (~/opt/bcbb by
                        default)
  --no-tests            If set, the test suite is not run
```

Figure 4.1. Installation system functionalities. It can install and uninstall *bcbio-nextgen*, as well as update an existing installation. Several additional options like installation directory or running or not the pipeline tests after the installation can also be provided

Virtual machine installation

To install the pipeline within a virtual machine, the installation system uses Vagrant[5] to create a virtual machine with a single command. Next, it uses the

[5]Vagrant uses Oracle's VirtualBox to build configurable, lightweight, and portable virtual machines dynamically

Python module Fabric[6] to install the pipeline within the virtual machine.

The script adds the *Science For Life Laboratory* PPA to the system and installs all the dependencies. Then it installs the Pipeline in the same way as it is done in a local installation and, if specified, executes the pipeline test suite.

There exists the option of installing simultaneously the pipeline in several virtual machines. These virtual machines are created in a way that they are connected in local network. Therefore this option is very appropriate to experiment with different kind of parallelizations: both multi-core and multi-machine, like Hadoop or MPI.

4.3 Automating testing

Software testing is often under-valued by organizations and even by software developers. This is sometimes due to a lack of understanding of its purpose within the development life cycle and the benefits it can bring. The time investment required to write good tests is another common objection.

Testing is one of the most important parts of the development process. It helps to ensure the correctness and the stability of the software. There are different levels of testing:

- **Unit testing** refers to tests that verify the functionality of a specific section of code, usually at the function level.
- **Integration testing** is any type of software testing that seeks to verify the interfaces between components against a software design.
- **System testing** tests a completely integrated system to verify that it satisfies its requirements.

Software testing can be implemented at any time in the development process, but we will focus in the concept of **Test Driven Development** (TDD).

TDD is an advanced technique of using automated unit tests to drive the design of software and force decoupling of dependencies. The programmer writes unit tests *before* implementing the functionality that wants to test. Then writes the code of the function and tests it. The idea is to use the tests as a specification. Optimization of the function to test is done once the test passes [HE11].

A scheme of the life cycle of a software developed using this methodology can be seen in figure 4.2. On the first phase (red), the developer(s) write a test that fails (because the functionality under testing is not yet implemented). Then the functionality is developed and the test passes (green phase). The next step is to improve the performance of the algorithm (refactor phase). This refactoring may result in the test failing again, which brings us back to the first phase.

[6]Fabric is a Python library and command-line tool for streamlining the use of SSH for application deployment or systems administration tasks

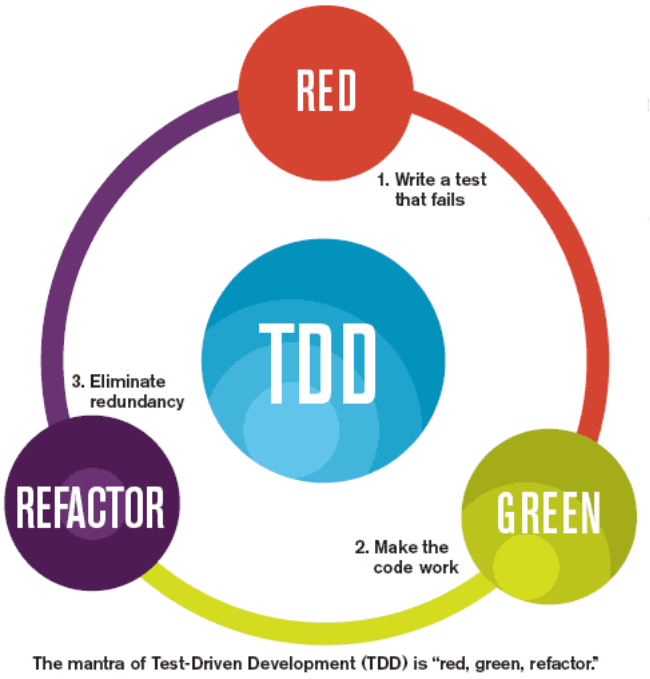

Figure 4.2. Life cycle of TDD

The result of using this methodology is a comprehensive suite of unit tests that can be run at any time to ensure that the software is still working. It helps the developer to be more focused on specific tasks, and therefore be more productive. In fact, TDD offers more than just simple validation of correctness, but also drives the design of a software.

4.3.1 Continuous Integration System

The advantages of making good tests is sometimes a clash point against the concept of not doing things manually. For the tests to be useful, they have to be run on every code change, so that possible regression or integration problems are detected immediately.

Executing all tests on every code change manually is not only time consuming for the programmer, but almost impossible to achieve in a collaborative project. If there is a group of programmers working in the same project, each one of them working on different functionalities, it is very likely that even having tested all functionalities separately, when merging all of them, something is going to fail.

The term *Continuous Integration* refers precisely to the process of assembling software every time the code changes. Continuous integration brings a lot of advantages to the software development process:

- Replace big (and long) integration phases by small and frequent ones.
- Minimize integration efforts.
- Helps to produce *CRISP* (Complete, Repeatable, Informative, Schedulable, Portable) builds.
- Generate deployment package at any time.
- Establish greater confidence within the development teams.
- Centralization of the building and testing processes.
- etc.

There are only three requirements for CI:

- A reliable and fully automatic building process.
- A version control system.
- A continuous integration server.

First and second requirements are covered by the automated installation implemented before (chapter 4) and by the use of Git as Distributed Version Control System (DVCS) and GitHub [7] as code repository. It remains to set up a CI server to have a completely automated testing phase.

There are a lot of options in the market of CI servers, but we tried to find a CI server that met the following requirements:

- Free of charge.
- Open Source, so we can collaborate and/or study the project.
- Integrated with GitHub.

After some research we found a CI server that completely fulfilled the previous requirements: Travis-CI.

Travis-CI

Travis-CI is a hosted continuous integration service for the open source community. It is integrated with GitHub and offers native support for many languages, including Python, the language used in *bcbio-nextgen* pipeline.

One of the main reasons for choosing Travis-CI among other similar (and also very good) options, like for example Jenkins[8], is that Travis-CI is **freely hosted**. This means that we don't need a dedicated server to host Travis, they do it for us. This is a great advantage for us, because we don't have to worry about any kind of maintenance.

[7] GitHub is a web-based hosting service for software development projects that use the Git revision control system

[8] `http://jenkins-ci.org/`

Travis-CI is perfectly integrated with GitHub, therefore activate the integration with Travis-CI is as easy as authorize Travis-CI through the GitHub web interface. Once this is done, the project is integrated with Travis and it will trigger a new build on every push to GitHub.

To configure the builds you just need to tell Travis-CI how to build your projects providing the following information:

- Which language use to build your project
- What to execute before the tests
- How to run the tests

Another advantage of Travis-CI is its public web interface, where you can see the state of your project with other useful information like the complete list of builds, the complete log of all the builds, who triggered the builds and with which commit, etc. This eases a lot the maintenance and releasing of a project. It has also a very good integration with GitHub pull requests system, allowing you to see the result of a merge even before merging it, so collaborative projects become really easy to maintain. Figure 4.3 shows an example of this integration.

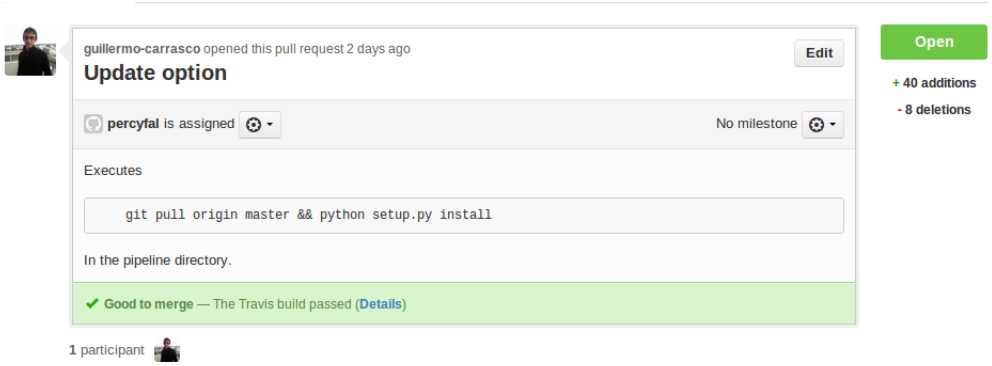

Figure 4.3. Travis-CI integration with GitHub

Chapter 5

Hadoop-based Sequence Aligner Evaluation

As shown in chapters 2 and 3, in every genomics analysis, besides other in-house defined quality control protocols, the alignment step is common in every laboratory. Due to the amount of data to align (also described in chapter 2), this is one of the most time demanding processes.

In order to improve the performance and reliability of this important step in the analysis, we evaluate and test a Hadoop based sequence aligner, Seal.

Seal is a suite of distributed applications for aligning, manipulating and analyzing short DNA reads. Seal applications run on Hadoop, and therefore they're made for the treatment of big amounts of data, like those produced by whole genome sequencing runs.

However, before start talking about a Seal, we need to explain some basic concepts about Hadoop and MapReduce.

5.1 MapReduce

MapReduce is a programming model for data processing. The MapReduce model simplifies parallel processing by abstracting away the complexities involved in working with distributed systems, such as computational parallelization, work distribution, and dealing with unreliable hardware and software. With this abstraction, MapReduce allows the programmer to focus on addressing business needs, rather than getting stuck in distributed systems complexities [Hol12].

The model is simple: MapReduce works by breaking the processing of a task into two phases: the map phase and the reduce phase. The role of the programmer is to define map and reduce functions. The map function outputs key/value tuples, which are processed by reduce functions to produce the final output. To illustrate this with an example, code 5.1 shows a basic "word count" MapReduce program

written in Python.

```
def map(text):
    for line in text:
        for word in line.split():
            yield (word, 1)

def reduce(word, partialCounts):
    occurences = 0
    for pc in partialCounts:
        occurences += pc
    yield (word, occurences)
```

Code 5.1. Word count MapReduce program in Python

In this example, the map function just emits a 1 for each word in the text, resulting in the key-value pair (word, 1) being emitted. The corresponding framework (like Hadoop) puts together all the pairs with the same key, and send them as input to the reduce function, that only has to sum all the partial counts (in this case all of them will be 1).

Despite everything cannot be solved with MapReduce, it is easy to applicate the pattern to Big Data problems [DG08]. However, to run MapReduce programs, a framework that communicates and coordinates the mappers and the reducers is needed. This framework must implement the core components of the MapReduce architecture, namely:

- **JobTracker**: The JobTracker coordinates activities across the TaskTracker processes. It accepts MapReduce jobs from client applications and schedules map and reduce tasks on TaskTrackers.
- **TaskTracker**: The TaskTracker is a daemon process that spawns child processes to perform the actual map and reduce tasks. It also sends progress reports to the JobTracker, which keeps a record of the overall progress of each job.

Figure 5.1 shows the basic MapReduce architecture scheme. In the figure we see how the client connects to the JobTracke, which has all the information about the active jobs, and which TaskTrackers are running which tasks.

5.2 Hadoop

Apache Hadoop is a software framework that allows the distributed processing of large data sets across clusters of computers using MapReduce. It is designed to scale up from a single server to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so it delivers

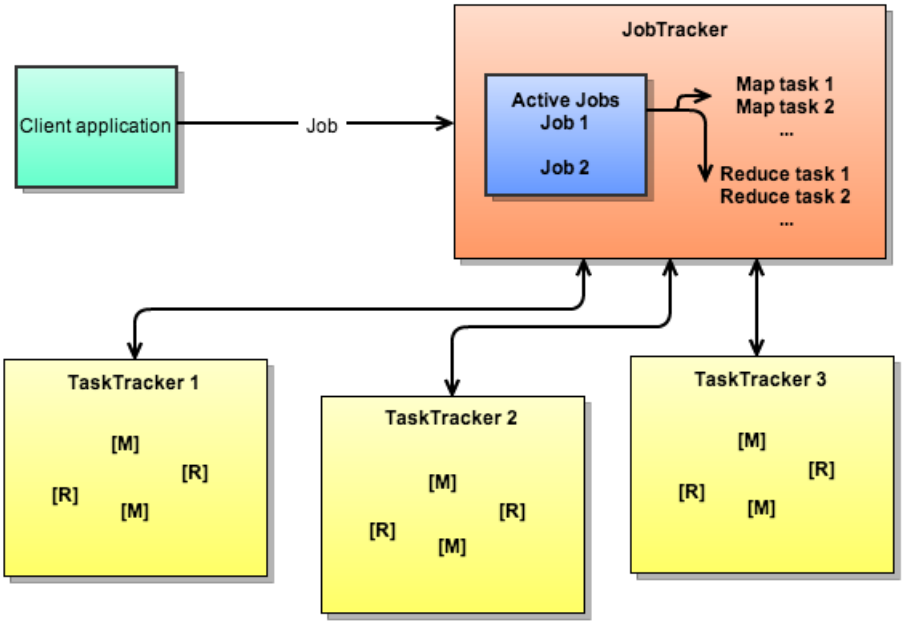

Figure 5.1. Core components of the MapReduce architecture

a highly-available service on top of a cluster of computers, each of which may be prone to failures [Sof12].

Hadoop clusters are usually composed of commodity hardware, meaning that no expensive and specialized hardware is required to run a Hadoop cluster. This is a big advantage in terms of usability. It is easy to create a small cluster using a bunch of machines for testing purposes, and even a single machine can be configured to run a complete Hadoop "cluster". However, in order to achieve a good performance, a production cluster must be composed of several machines.

On the next sections we describe the main components of the Hadoop framework.

5.2.1 Hadoop Distributed File System (HDFS)

Hadoop design has a general-purpose filesystem abstraction. This means that it can be configured to run, for example, using the local file system. However, when a dataset outgrows the storage capacity of a single machine, it becomes necessary to partition it across a number of separate machines. Filesystems that manage the storage across a network are called Distributed File Systems (DFS). Hadoop comes with its own DFS, the Hadoop Distributed File System (HDFS).

HDFS is a filesystem designed for storing very large files with streaming data access patterns[1]. It performs very well for large datasets for which several analysis

[1]Write once - Read many

needs to be performed. Each analysis will use a large portion, if not all, of the dataset. In order to distribute the data across several machines, HDFS splits the data into blocks.

HDFS blocks follow the same concept of common disks blocks[2], however, a block in HDFS is much larger that a block in a common filesystem, typically 64MB. The reason for this bigger size is to minimize the cost of seeks. By making a block large enough, the time to transfer the data from the disk can be significantly longer than the time to seek to the start of the block. Thus the time to transfer a large file made of several blocks operates at the disk transfer rate [Whi12].

Having the concept of block instead of file in a DFS brings several benefits:

- A file can be larger than any single disk in the network. There's nothing that requires the blocks of the same file to be stored in the same disk, thus they can be stored in many disks in the cluster.

- It simplifies the storage subsystem. The storage subsystem deals with blocks, which have a fixed size, so is easy to calculate the number of blocks that a storage unit can store. It also relieves the filesystem of handling metadata such as file permissions information, because blocks are just chunks of data, and the file's metadata can be handled by a separate system.

- Blocks fit well with replication. In order to ensure fault tolerance and data availability, blocks can be replicated in several nodes. In case of block corruption in one node, the filesystem can replace it for a copy of the same block in another node of the system.

This kind of filesystems suit very well in a bioinformatics core facility:

- The large amount of data is automatically replicated across several machines, ensuring data availability and reliability.

- As explained in chapter 3, the pattern of a genomics analysis is precisely write one - read many: Several quality controls and analysis needs to be performed over the same dataset.

5.2.2 Namenodes and datanodes

There are two basic entities in a Hadoop cluster: A namenode and several datanodes.

[2]Data is stored in blocks, which is the minimum amount of data that can be read or written form/to a disk

The namenode

The namenode is the centerpiece of the Hadoop file system. It keeps the directory tree of all files in the file system, and tracks where the data is kept across the cluster. It does not store the data of the files itself. Client applications talk to the namenode whenever they want to know the location of a file. The information of the filesystem is stored in the namenode permanently on the local disk in the form of two files: The namespace image (called *fsimage*), and the edit log.

As explained in the previous section, HDFS splits files across several nodes in the system. The namenode also keeps that information (which node stores which blocks). However, this information is not stored permanently, as it is reconstructed from the nodes when the system starts. The namenode usually runs the JobTracker daemon, though it can run on a separate machine.

The namenode is a single point of failure in a HDFS cluster, because is the only node that have all the information of the filesystem. Because of this, its important to provide some failure resilience to the namenode. Hadoop provides two mechanisms for this:

1. The first option is to backup all the metadata that the namenode holds about the filesystem. The namenode can be configured to periodically write its state to multiple locations.

2. It is also possible to run a *secondary namenode*. Despite its name, the secondary namenode is not a backup of the namenode, but it's more like a checkpoint node [gee12].

   The namenode keeps all the filesystem state in a snapshot in memory, the *fsimage*. It also writes every operation done in the filesystem since the start of the namenode to log files. The namenode only applies the changes written in the logs to the *fsimage* in every restart, so after a while, the *fsimage* diverges from the actual filesystem state. As in production clusters the restarts of the namenode are very sporadic, these log files can become really large for systems with thousands of files and nodes, and if the namenode crashes, on the next start it has to apply all the changes on the logs to the *fsimage*, which can make the startup time be very high.

   The secondary namenode, periodically takes the information in the logs of the namenode, as well as the *fsimage*. It applies the changes to the *fsimage*, and finally copies back the *fsimage* to the namenode. This will reduce the startup time of the namenode in case of crash.

In recent versions of Hadoop, several approaches have been made in order to ensure the reliability of the namenode. Interested readers can refer to [Whi12] and [Hol12] for detailed information.

The datanodes

The datanodes are the nodes that store the data and perform the calculations. They act as TaskTrackers in the MapReduce architecture. A functional cluster has several datanodes, with data replicated across them.

In contrast with the namenode, and thanks to the data replication scheme provided by DFS, if a datanode crashes while running a job, the job can continue. What will happen if a datanode crashes is that the namenode will re-schedule the task that the datanode was running to another datanode having the same blocks of data needed to that task.

5.3 Seal

After describing what is Hadoop and how it works, now we describe Seal and its functionalities.

Seal provides a number of several important features that other alignment tools cannot provide:

- **Scalability**: Thanks to Hadoop, it is inherently scalable. Hadoop applications are able to easily scale from single to thousands of computational nodes [DG08].
- **Robustness**: Hadoop also provides Seal with a resistance to node failure. Together with HDFS, Hadoop also ensures data availability.
- **Memory efficiency**: In order to perform an alignment, the reference genome needs to be sent to all the datanodes. However, Seal is implemented in a shared-memory architecture. Despite the fact that all datanodes need the reference genome, all TaskTrackers in the same datanode share the reference genome.

5.3.1 Seal applications

Seal consist on five main tools:

1. **Demux**, used for the process of demultiplexing, explained in Current Pipeline: *bcbio-nextgen* Pipeline.
2. **PairReadsQSeq (PRQ)** is a Hadoop utility to convert Illumina FASTQ files into the Seal specific format, PRQ.
3. **Seqal** Seqal is a distributed short read mapping and duplicate removal tool. It implements a distributed version of the BWA[3] aligner, currently used at *Science For Life Laboratory*.

[3] `http://bio-bwa.sourceforge.net/`

4. **ReadSort** is a Hadoop utility to sort read alignments.
5. **RecapTable** is a Hadoop program to calculate a table of base qualities for all values of a given set of factors.

The tools directly involved with the sequence alignment are *PairReadsQSeq* and *Seqal*. As the scope of this project is to analyze the performance of the sequence alignment, we describe in more detail how these tools work.

PairReadsQSeq

For a paired-end read, the sequencing machine produces two sequences: the forward read (F) and the reverse read (R). These two reads, belonging to the same DNA strand, should align against the reference genome in order, that is FR [VM12]. Figure 5.2 illustrate this with an example.

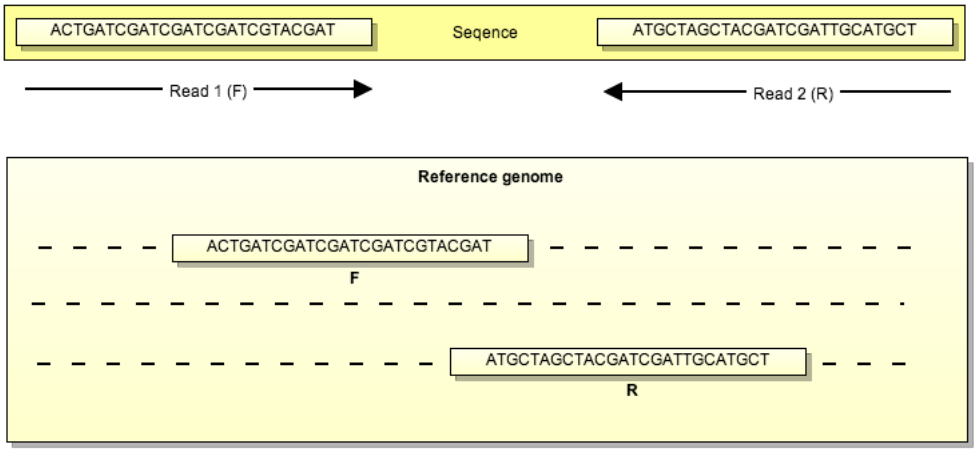

Figure 5.2. Paired-end read alignment against a reference genome

However, if we put the separate read files in the HDFS, it may happen that they're split in such a way that the two reads within a pair end in different datanodes. If that happens, when aligning against a reference genome, the aligner will only have one read of the pair, and therefore will not be able to do the alignment.

To avoid this problem, Seal introduces the PRQ[4] format for the sequencing data. PRQ is a line-oriented format, and each line contains a read pair with the following tab-separated fields: *Id, read sequence 1, quality 1, read sequence 2 and quality 2.* In plain words, PRQ just puts together both reads of a paired-end read in the same line. Doing so, Seal ensures that the datanode that is going to align these short reads, will receive both forward and reverse.

[4]PRQ is the extension format that generates the application PairReadsQSeq

This FASTQ-PRQ conversion can easily be done sequentially, but due to the size of the data, this would be too costly. This data conversion easily fits to a MapReduce pattern, and PairReadsQSeq takes advantage of this.

Figure 5.3 shows the whole MapReduce FASTQ-PRQ conversion process in Seal. For the map task, PairReadsQSeq takes the read ID as key, and the sequence and quality of the read as value. It emits a pair *(readID, sequence+quality)*. The reduce tasks only have to output all the values from the received key-value pairs in the same line. In this case, the reducers will receive exactly two values, one per read in the pair.

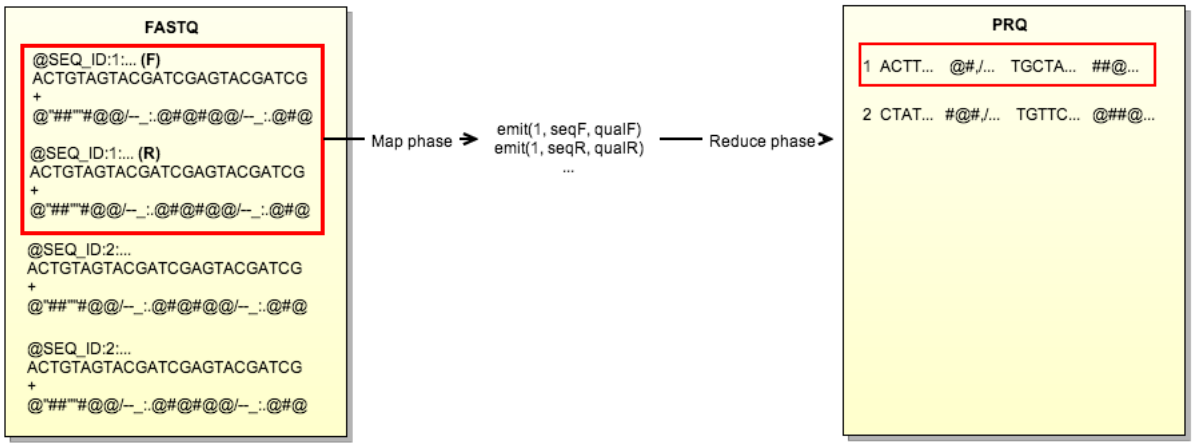

Figure 5.3. FASTQ to PRQ conversion diagram

Seqal

Seqal performs the proper sequence alignment. Like BWA, Seqal takes as input read pairs to be aligned, and a reference sequence. The input read pairs need to be in PRQ format in this case.

As previously said, the reference genome is transferred to each node in the cluster, but thanks to the Hadoop distributed cache mechanism, the whole reference genome is shared by all the TaskTrackers in the same node, efficiently using the available memory in the nodes. Rather than implementing a new aligner, Seal uses the same algorithm than BWA, and therefore the results obtained are comparable [Pir11].

The read alignment is implemented in the map function. If the user only wants to perform an alignment, the reads are directly outputted to SAM files after the map phase. Otherwise, the user can choose to remove duplicate reads. In such a case, the outputs from the map phase are put through a reduce phase where duplicates are removed. In our case, we are only interested in the alignment.

5.3.2 Seal integration with *bcbio-nextgen* pipeline

As we explained in Chapter 3, *bcbio-nextgen* is developed in a very modular way. This means that in order to integrate a new tool, "the only thing that needs to be done" is to write a Python module that calls that tool with the correct command line arguments. In this case, in order to integrate Seal, the module to write needs to take care of moving the data to and from HDFS, as well as performing the proper analysis.

The integration with *bcbio-nextgen* is independent of the aligning evaluation. In fact, in order to evaluate the performance of BWA and Seal, we have to run them isolated from other programs. For this reason, we have decided to do the integration with the pipeline after the evaluation of Seal.

Chapter 6

Tests and results

In this chapter, we describe the tests performed in this project and the conclusions that we can extract from the results.

Firstly, we describe the working methodology in *Science For Life Laboratory* after the implementation of the automated installation system and the integration in Travis-CI.

Secondly, we describe the procedure followed to evaluate the performance of Seal, as well as a comparison with the currently used genome aligner, BWA. We also implement the necessary changes in *bcbio-nextgen* to integrate Seal as an eligible option to perform the alignment.

6.1 Automated installation and CI

Before the implementation of the automated installation, users that wanted to install or test *bcbio-nextgen*, had to deal with several manual steps, as described in chapter 4. Once the automated system was implemented and properly tested, it bacame the main deployment method in production.

From the user's point of view, this has been a really big improvement: now, users do not need to read a large manual on "how to install *bcbio-nextgen*", as all the installation is carried on automatically. They just need to execute one single command.

From the developer's point of view, this has also been a big improvement:

- The fact that the installation is automated, without user intervention, makes the installation straightforward, and prevents non-experienced users to perform a step in a wrong way. Actually, that wrong way may make the installation to fail or, in a worst case, *bcbio-nextgen* to behave unexpectedly. With the new system, the number of emails addressed to the developers regarding the pipeline installation have been drastically reduced.

- Even for experienced developers, the automatic installation implies more efficiency in terms of time needed to configure new machines. It also avoids human errors when configuring the pipeline (even experienced developers make mistakes when repeating several times the same procedure).

The automatic installation has also allowed us to integrate the pipeline in a CI system, as described in section 4.3.1. Without an automatic installation procedure, the pipeline could not have been integrated in Travis-CI. In order to test not only the software, but also its deployment, Travis-CI spawns a new virtual machine (VM) on every test. In this VM the software is deployed and tested with the test suite that the software must provide. All this process is completely automated, without even the possibility of human intervention, and therefore the automatic deployment method is required.

The fact that Travis-CI deploys, installs and configures the pipeline on each test makes the process to take more time, but on the other hand, it ensures that both the deployment system and the pipeline itself works. Code 6.1 shows the instructions that we provide to Travis-CI so it can correctly deploy and tests *bcbio-nextgen*.

```
language: python

install:
    - sudo apt-get update
    - sudo apt-get install -q -y python-software-properties
(1) - sudo add-apt-repository -y ppa:scilifelab/scilifelab
(2) - sudo add-apt-repository -y ppa:debian-med/ppa
    - sudo apt-get update
    - sudo apt-get purge python-paramiko
(3) - sudo apt-get install -q -y git-core gcc libsam-java picard-tools
    bowtie bwa freebayes snpeff-2 fastqc gatk r-base tophat openjdk-6-
    jre samtools unzip lftp cufflinks wigtools python-dev
    #Download snpeff genome database
    - lftp -e 'pget -n 8 http://downloads.sourceforge.net/project/
        snpeff/databases/v2_0_5/snpEff_v2_0_5_GRCh37.63.zip; quit'
    - sudo unzip snpEff_v2_0_5_GRCh37.63.zip -d /usr/share/snpEff/ &&
        rm snpEff_v2_0_5_GRCh37.63.zip
    - cd nextgen && python setup.py install

notifications:
    email: false

before_script:
    - cd tests
    #Export some environment variables
    - export PICARD_HOME=/usr/share/java/picard
    - export SNPEFF_HOME=/usr/share/java/snpeff
    - export GATK_HOME=/usr/share/java/gatk

(4) script: nosetests -s -v --with-xunit -a standard
```

Code 6.1. Instructions given to Travis-CI to build de pipeline. In (1) and (2), we add Debian-Med and SciLifeLab custom PPAs. In (3), install all pipeline dependencies. In (4), execute the tests.

Before the integration of *bcbio-nextgen* in Travis-CI, the tests of the pipeline needed to be run manually by the developers on every change in the code. Merging the changes with the master branch of the pipeline's code was also risky. Even if the developer executed the test suite without errors, it could happen that on merging her changes with changes made by other developers, the pipeline could fail. Thanks to the great integration between Travis-CI and GitHub, this is no longer a problem. Figure 6.1 shows how Travis-CI helps us to determine if we can merge a change to the master branch in *bcio-nextgen*:

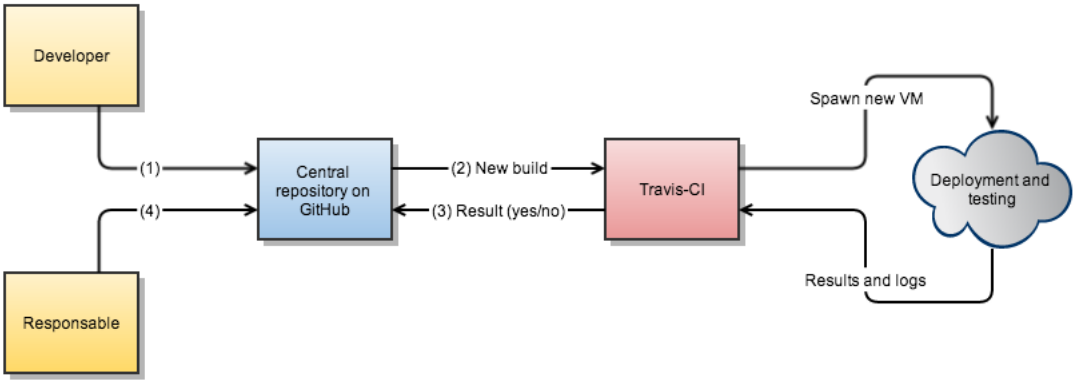

Figure 6.1. Pull request flow in *Science For Life Laboratory*.

1. After changing the code, the developer performs a pull request[1] to the master branch.
2. This pull request triggers a new build in Travis-CI, spawning a new VM. In this new build, Travis-CI merges the changes to the master branch in the VM, deploys the pipeline and runs the tests.
3. The results from the tests are sent back to GitHub, indicating if everything was fine or there was any error.
4. In accordance to the result, the developers can choose to merge the pull request or fix any exposed error.

At any time, developers can access the full log of every build from the moment in which the project is integrated with Travis-CI, improving traceability of issues

[1]Pull requests let you tell others about changes you've pushed to a GitHub repository. Once a pull request is sent, interested parties can review the set of changes, discuss potential modifications, and even push follow-up commits if necessary.

and changes. Actually, the possibility to check the logs has helped us in more than one occasion to refear known issues between developers.

6.2 Seal evaluation

6.2.1 Testing environment

In order to evaluate the performance of Seal, we need to evaluate it in a cluster of machines. It has no sense at all to evaluate Seal in a single machine, even configuring a virtualized cluster, which may be good for proof of concept tests, but not for testing on real data. The overhead of synchronizing all Hadoop and MapReduce tasks, the management of the nodes and the data transfers would make this evaluation worthless.

There are many HPC centers that offer hosting for research projects like this one. We have considered several possibilities for this project, but three of them seemed to be the more plausible:

1. **UPPMAX** was the first option. All the heavy calculations in *Science For Life Laboratory* are done in UMMPAX. They store all the data to be analyzed from the moment when the analysis starts, until the last step in the analysis, when the data is sent to a long-term storage facility. The structure of the cluster is a SLURM queue system that distribute the resources in a set of physical nodes [Lln11]. This use of physical hardware makes the usage of the resources more efficient, but also requires a more strict security policy, as the users could potentially access the system files. This stiffness in the rules, together with the coexistence with SLURM, makes impossible to freely configure a Hadoop cluster without having to trick the system.

2. **Amazon Elastic MapReduce (EMR)** is part of the Amazon Web Services [2]. Amazon Web Services offers a complete set of infrastructure and application services that enable the user to run everything in the cloud. Amazon EMR utilizes a hosted Hadoop framework running on the web-scale infrastructure of Amazon Elastic Compute Cloud (Amazon EC2) and Amazon Simple Storage Service (Amazon S3). They provide all the flexibility of a cloud service. Something very convenient is that the user can create his own OS images, with all the necessary software included. This images can be used later on to spawn several nodes and form a cluster.

   Despite the advantages of Amazon EMR, we realized that it really doesn't fit with NGS data analysis. The amount of data to analyze makes the usage of the resources almost constant, which will make the costs of this "pay for what you use" service to increase too much. Even for the tests of this project, the

[2] `http://aws.amazon.com/`

amount of data to transfer to Amazon, and the expenses that these transfers entails, makes that option non-viable for this project.

In any case, an Amazon image was created with all the software to run the tests ready in case we could not find another cluster to run the tests.

3. **ELIXIR project**: ELIXIR is an international project which aim is to construct and operate a sustainable infrastructure for biological information in Europe to support life science research. In addition to a central hub placed at the European Bioinformatics Institute (EMBL - EBI) near Cambridge, UK, ELIXIR project has national and regional nodes across Europe [Eur13]. One of these nodes is located in Finland, in the Biomedinfra[3] research center, and members in *Science For Life Laboratory* had previous collaborations there. We established contact with them, asking if they would like to host this project, and they agreed.

The main difference between UPPMAX and Biomedinfra is that, in Biomedinfra, they use Opennebula, instead of SLURM, to manage computing resources. Opennebula is an open-source project for cluster virtualization [SMLF08]. It permits the creation of isolated virtual machines on the top of the physical nodes. This has the disadvantage that the virtual machines will not be as efficient as direct physical software, but on the other hand, it has the big advantage of isolating the user within his virtual machines. This permits the user to have full root access to the system, without compromising security of the whole system.

With this Infrastructure as a Service (IaaS), we can easily create different cluster configurations. To add a new machine to the cluster, the user needs to specify the disk space, OS image, memory and CPU for the machine. The OS image can be created using a raw system image as a template. In our case we've created an OS image from Ubuntu 12.04, installing all the necessary software to run the tests.

This method allowed us to rapidly spawn new virtual machines within a cluster, without having to re-install and configure the whole software suite. However, this contradicts the reproducibility principle, as the tests will be very difficult to reproduce outside this particular environment.

In order to avoid this dependency, a set of recipes to install all the software have been implemented. This recipes use the automation platform Chef to install the software.

Chef relies on reusable definitions known as cookbooks and recipes that are written using the the Ruby programming language [NS11]. Cookbooks and recipes automate common infrastructure tasks. Their definitions describe what the infrastructure consists of and how each part of the infrastructure should be

[3] `http://www.biomedinfra.fi/`

deployed, configured and managed. Chef applies those definitions to servers to produce an automated infrastructure [Eur13].

These Chef recipes allow for the installation of the software independently of the platform in where they are executed. With this solution, not only the tests for this project are easily reproducible, but also eases the deployment of Hadoop-Seal clusters in production environments.

These recipes were used to configure the OS image used in the Hadoop clusters.

6.2.2 Cluster configurations

We want to evaluate the performance of Seal and BWA in different cluster configurations. This will determine the best combination of nodes that makes the investment in a Hadoop cluster worthy. It also will help to determine the amount of data necessary to bypass the Hadoop overhead.

Three cluster configurations have been used to run the tests, each one trying to pinpoint the weaknesses and strengths of Seal and BWA.

1. 22 datanodes, each one with:
   - 4GB of RAM
   - 1 CPU at 2.66GHz
   - 140GB of local storage + 1TB NFS shared partition
2. 11 datanodes, each one with:
   - 8GB of RAM
   - 4 CPU at 2.66GHz
   - 140GB of local storage + 1TB NFS shared partition
3. 2 datanodes, each one with:
   - 46GB of RAM
   - 24 CPU at 2.66GHz
   - 140GB of local storage + 1TB NFS shared partition

With the first cluster configuration, we want to demonstrate how Seal can be run in a cluster of commodity machines. These machines are affordable and easily acquirable. However, individually they don't provide enough computational power to run a genomics analysis, as we expect to see when executing BWA alone in a single node.

The second cluster configuration aims to demonstrate how BWA can take profit of multi-core CPUs, giving it a very good performance for not very large volumes of data. We expect Seal to be faster for large amount of data, as it can take profit not only of multi-core CPUs, but also of multiple machines.

The third cluster configuration tries to demonstrate that, even the benefits of Hadoop, it is not worth to use it without a fairly large number of datanodes. With only two nodes, even if they have high specifications, the overhead of managing Hadoop processes and writing in an out the HDFS is expected to cut down Seal's performance.

All the three clusters share the same namenode configuration:

- 1 namenode: 1 CPU 2.66GHz, 2GB of RAM, 20GB local sotrage.

In our HDFS, we're not managing huge amounts of data. We will store only the sets of test data and the results of the analysis. These sets of test data can easily be replicated, as well as the results. Also, this will not be a production cluster and therefore doesn't need to ensure high availability. Therefore, we have decided to don't configure a secondary namenode.

6.2.3 Test data

SimNGS

A realistic dataset is needed to test the performance of Seal. For confidentiality and reproducibility reasons, we cannot use real laboratory data. The datasets need to be general enough to ensure data independence for the tests. It may happen that using real data, this data could be biased or have a bad quality due to experiment characteristics or sequencing errors. Furthermore, using real sequencing data would require us to send the data to those who want to reproduce our tests, which is not feasible given the amount of data we are working with. We have to generate random data to align against a reference genome.

SimNGS is a software for simulating observations from Illumina sequencing machines using complex statistical models. By default, observations only incorporate noise due to sequencing and do not incorporate effects from more esoteric sources of noise that may be present in real data ("dust", bubbles, etc). Many of these additional sources may optionally applied. SimNGS takes FASTA format sequences and a file describing the covariance of noise between bases and cycles observed in an actual run of the machine, randomly generates noisy intensities representing the signals for the sequence at each cycle and calculates likelihoods for all possible base calls, generating a real-like final FASTQ file.

At the time of writing, simNGS does not generate Illumina standard reads, but instead it generates slightly different FASTQ records. SimNGS adds some extra information to the FASTQ records at the end of the first line of the record.

Running the first attempt of aligning we realized that Seal's FASTQ parser is not very flexible about the format of the input data, and this extra information that simNGS was introducing to the FASTQ record was stopping the execution due to parsing errors.

Another problem we found is that SimNGS was not generating unique read identifiers for the generated read records. This was also giving problems with Seal in the FASTQ to PRQ conversion process, as it relies on the read id to generate MapReduce keys.

For both of the problems, we searched around the Internet and contacted the developers of SimNGS and Seal. It resulted that more people were having these problems, so we decided to fix SimNGS and propose the changes to the authors.

A patch was implemented that allows SimNGS to generate Illumina-like reads with unique identifiers. The patch allows these functionalities to be chosen from the command line, coexisting with the already available options.

These changes were proposed to the authors of SimNGS, who merged them to the official release.

Generating data sets

The time and resources required by the alignment depends mainly on two factors: The size of the input data, and the size of the reference genome.

Obviously, the larger the input dataset is, the longer takes the alignment step, because more reads need to be aligned. On the other side, the size of the space the reads are mapped against is determined by the size of the reference genome. Bigger reference genomes imply a slower alignment process.

For this reasons, different datasets for different organisms have been created depending on the size of its complete genome:

- Small-size reference genome: Escherichia Coli (ecoli) - 4.6MB.
- Medium-size reference genome: Drosophila Melanogaster (dm3) - 165MB.
- Large-size reference genome: Human genome (hg19) - 3GB.

For each one of the chosen organisms, two datasets of sizes 5GB and 50GB of paired-end reads have been generated using SimNGS.

6.2.4 Results

Running the tests

To run the tests, an automated script was implemented that performs the following actions:

1. Runs Seal tests:
    a) Copy the dataset to analyze from the local file system to HDFS.

b) Runs PairReadsQSeq to convert FASTQ records to PRQ records.

c) Runs Seqal alignment.

2. Runs BWA tests:

a) Perform the alignment using all available threads in the datanode where it is running.

All the steps are timed using the linux standard timing tool. The resources that Biomedinfra could allocate for us are limited, and therefore the script could not be fully automated. Due to disk space problems, we had to clean the tests results after the execution of the tests for each reference genome. This is reflected in the script, which accepts three parameters, each one of them determines the data and reference genome to use: -e for Escherichia Coli, -d for Drosophila Melanogaster and -h for Human.

The tests were automated using the process scheduling tool *at*[4].

Per-cluster results

The results for the tests for the three cluster configurations follows with its analysis and conclusions.

Cluster configuration 1
Tables 6.1 and 6.2 show the timing results for this cluster configuration and both datasets.

		Ecoli	**Dm3**	**Hg19**
Seal times (hh:mm:ss)				
	Transfer	00:02:52	00:07:24	00:02:37
	FASTQ - PRQ conversion	00:06:47	00:13:24	00:08:03
	Alignment	00:11:31	00:17:49	–:–:–
BWA times (hh:mm:ss)				
	Alignment	00:20:15	01:17:48	28:04:28

Table 6.1. Summary of the results for cluster configuration 1 and 5GB dataset

Firstly, one have to notice that, when using Seal with HDFS, an extra time is required to move the files in an out HDFS. This is something to have in mind, as can be the bottleneck if the improvement on the alignment time is not big enough to compensate this transfer time. The speed of the connection between the datanodes affects directly to this process, as well as disk I/O performance.

[4] **at** reads commands from standard input or a specified file which are to be executed at a later time

		Ecoli	**Dm3**	**Hg19**
Seal times (hh:mm:ss)				
	Transfer	00:36:08	00:24:56	00:23:52
	FASTQ - PRQ conversion	01:26:22	01:23:51	01:13:20
	Alignment	01:46:37	02:19:18	–:–:–
BWA times (hh:mm:ss)				
	Alignment	03:25:47	12:08:10	28:04:28

Table 6.2. Summary of the results for cluster configuration 1 and 50GB dataset

Secondly, when using Seal you also need an extra time to convert the input data to Seal's format, PRQ. In this case, for Ecoli reads, the time to transfer the files, plus the FASTQ - PRQ conversion, plus the alignment with Seal, is approximately equal to the time of aligning with BWA, which is approximately 20 minutes. In the case of the alignment against Drosophila's genome, Seal has a very similar performance, while BWA's time increases in about an hour. Notice that in this first cluster configuration, BWA can not be parallelized, as the datanodes only have one core, therefore Seal is running the alignment in 22 parallel processes, whilst BWA is using just 1 process or core. We can expect BWA to be faster with more cores for this data size.

Finally, the results tables show an empty gap in the results for Seal's alignment against the human genome. The size of the uncompressed human genome is 3GB. When aligning, the whole reference genome is loaded into memory. In the case of the alignment with Seal, a certain amount of memory is also needed for the MapReduce daemons. With this cluster configuration, in which the datanodes have only 4GB of memory, loading the reference genome, together with the overhead of MapReduce and Hadoop daemons, sums more than 4GB. This situation makes the datanodes to run out of memory during the alignment, and therefore the alignment against the human genome cannot be performed using Seal.

We can extract several conclusions with this cluster configuration:

- Hadoop can be ran in a cluster composed of commodity hardware, increasing the total computational power of the cluster. This allows a bunch of commodity machines to run heavy computational tasks. However, more powerful machines are needed to run realistic analysis, as we saw with the alignments against the human genome with Seal.

- The use of a Hadoop-based sequence aligner implies the addition of some extra steps: I/O into HDFS and PRQ conversion. This musk be taken into account.

- BWA does not introduce a memory overhead other than loading the reference genome into memory, and therefore it can perform memory

demanding calculations that cannot be performed with Seal on the same type of machines.

This cluster configuration was meant demonstrate how Hadoop doesn't require high specifications machines to be run. However, given the computational needs of a Genomics center, and the results obtained, would not make sense to configure a cluster with this kind of machines.

Cluster configuration 2
Tables 6.3 and 6.4 show the timing results for this cluster configuration and both datasets.

		Ecoli	**Dm3**	**Hg19**
Seal times (hh:mm:ss)				
	Transfer	00:01:06	00:01:38	00:01:30
	FASTQ - PRQ conversion	00:02:12	00:02:47	00:02:55
	Alignment	00:04:59	00:08:10	00:14:33
BWA times (hh:mm:ss)				
	Alignment	00:05:35	00:19:28	00:46:10

Table 6.3. Summary of the results for cluster configuration 2 and 5GB dataset

		Ecoli	**Dm3**	**Hg19**
Seal times (hh:mm:ss)				
	Transfer	00:10:31	00:24:56	00:16:43
	FASTQ - PRQ conversion	00:22:51	00:22:58	00:21:58
	Alignment	00:46:07	01:15:35	02:07:43
BWA times (hh:mm:ss)				
	Alignment	00:55:48	03:14:26	07:31:46

Table 6.4. Summary of the results for cluster configuration 2 and 50GB dataset

This time, as the datanodes had 8GB of RAM, all alignments could be done with both aligners.

On the 5GB dataset, and aligning against the smaller genome, Escherichia Coli, we can see that Seal in slightly better that BWA: both programs take about 5 minutes. However, this is the time of the alignment. As explained before, with Seal we have to perform additional steps. If we count the time required for this steps, the whole process takes aproximatelly 9 minutes with Seal. We can consider that BWA performs better for the smaller genome is this case, because no pre-processing is needed.

This situation changes for bigger genomes like Drosophila or Human, where even summing the time required to transfer the data into HDFS and converting it to PRQ format, Seal performs better.

These type of datanodes are more realistic, in terms that these specifications can easily be provided by HPCs. If we compare the results of BWA with the previous cluster configuration, we can see that increasing the number of cores has lead into a high increment on the performance. In the case of Seal, this increment has not been that big, though it has also doubled the performance respect the previous cluster, and we have been able to calculate the alignment against the human genome.

These results demonstrate that even the advantages of Hadoop, it is not appropriate to small datasets. Though given the amount of data generated by NGS sequencers, this seems to be a good configuration.

Cluster configuration 3
Tables 6.5 and 6.6 show the timing results for this cluster configuration and both datasets.

		Ecoli	**Dm3**	**Hg19**
Seal times (hh:mm:ss)				
	Transfer	00:01:37	00:01:43	00:03:20
	FASTQ - PRQ conversion	00:05:23	00:08:04	00:08:14
	Alignment	00:23:53	00:41:35	01:10:41
BWA times (hh:mm:ss)				
	Alignment	00:04:17	00:07:57	00:14:03

Table 6.5. Summary of the results for cluster configuration 3 and 5GB dataset

		Ecoli	**Dm3**	**Hg19**
Seal times (hh:mm:ss)				
	Transfer	00:40:04	00:15:59	00:23:33
	FASTQ - PRQ conversion	01:08:11	00:59:11	01:03:21
	Alignment	04:00:22	06:54:03	11:41:23
BWA times (hh:mm:ss)				
	Alignment	00:42:03	01:14:59	00:59:26

Table 6.6. Summary of the results for cluster configuration 3 and 50GB dataset

Hadoop, together with HDFS, is designed to perform well on large-scale clusters. As we can see in the results for this cluster, the use of Seal without a fairly large amount of datanodes cuts down drastically its performance. We can see that the alignment process has been extremely slow in comparison with BWA (almost eleven hours of difference for the human genome). Such a big difference was not expected. The use of only two datanotes may introduce an overhead to the whole process, but

even with this overhead, Seal is using 48 cores instead of the 24 that BWA can use simultaneously.

Apart from the overhead introduced for the management of all MapReduce processes, we have already seen that HDFS also introduces an overhead, as the files need to be copied in an out. In this cluster, just 2 datanodes manage all the replication of the data, and are continuously synchronized. If the I/O is too high, this might cause some overload.

A monitoring of the resources was done during the execution of the tests for this cluster. Figures 6.2 and 6.3 show the network traffic in both datanodes during the tests for Escherichia Coli's genome. Figures 6.4 and 6.5 show the disk I/O on the HDFS partition for both datanodes during the same test.

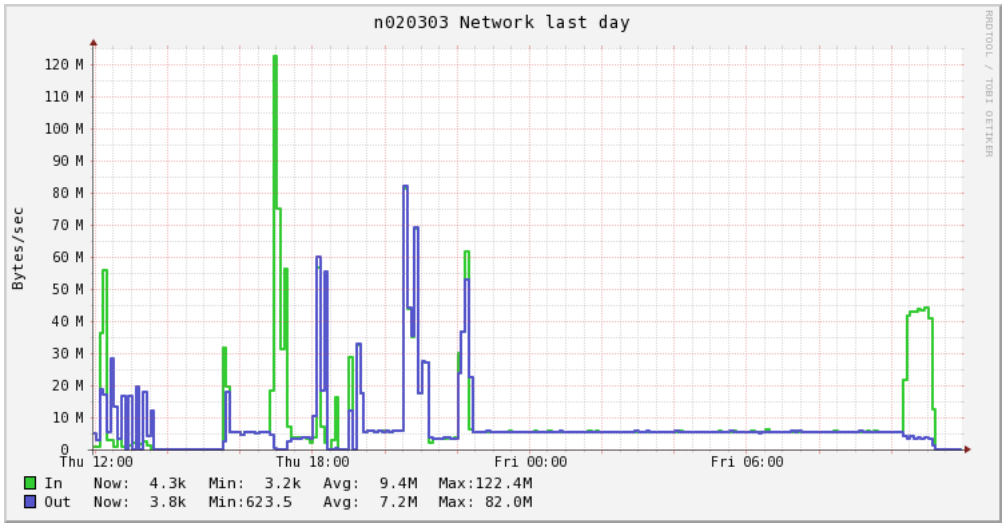

Figure 6.2. Network traffic in datanode 1 during tests for Escherichia Coli's genome in cluster 3.

Several high peaks on the network and on disk confirm our suspicions. As an example, around 18:00 there is a peak of approximately 10.000 reads/sec and 40.000 writes/sec on the disk of datanode 1, and a network input traffic of 120.000 Bytes/sec for the same datanode. This peak occurred when doing the FASTQ - PRQ conversion.

The explanation of these peaks is a bad performance of HDFS. The replication factor for the data is 2 in this cluster. This means that every single chunk of data in HDFS is replicated in both datanodes. In this particular case, moving 50GB of data, converting it to PRQ and aligning the data is saturating the nodes. This replication can be observed specially in the network graphics, where the input and output network traffic on both datanodes is complementary: one is sending data to the other, that stores it in its partition, and viceversa.

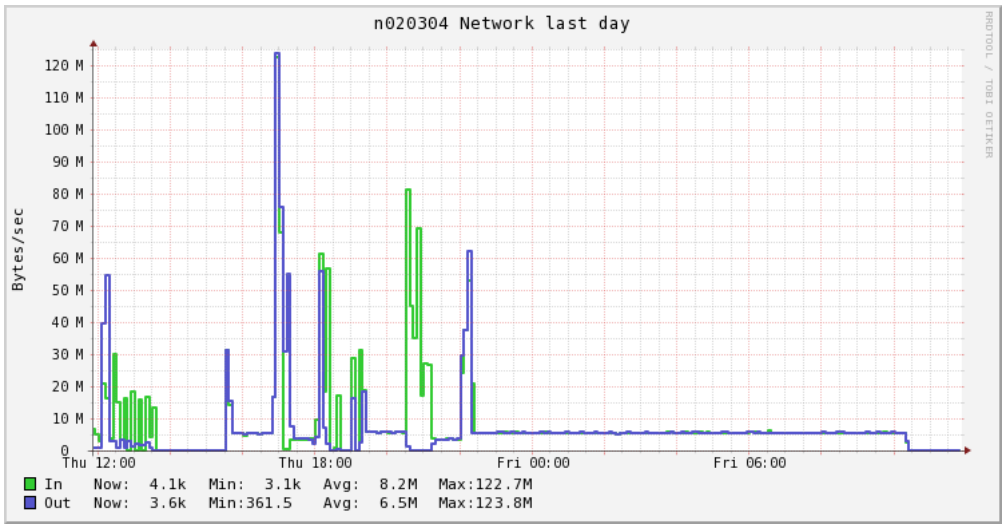

Figure 6.3. Network traffic in datanode 2 during tests for Escherichia Coli's genome in cluster 3.

This is not a big problem with more datanotes, like in the other cluster configurations. The replication factor in the other clusters is 3, and we have 22 and 11 datanodes. Hadoop can in those cases balance the load between datanodes.

A careful dimensioning is needed for a Hadoop cluster.

General conclusions

The big advantage of using a Hadoop based aligner is that one can take profit of large clusters of machines without the need to deal with the common complications of parallel applications. The management of the processes and the restart of failed jobs is done automatically, and therefore the whole process can happen without human intervention. Seal takes profit of all the machines in a cluster to perform the alignment, increasing the performance linearly on the number of machines and cores.

Using HDFS implies data replication and availability. Hadoop makes use of data locality, meaning that the tasks will be scheduled on the datanode that holds the data that they are going to analyze. This means that the computation is moved to the data, and not the other way around. This data locality reduces the payoff of transferring data across datanodes. However, as demonstrated in cluster 3, HDFS can still be the bottleneck if the cluster is not correctly dimensioned. There is a trade-of between the benefits of using HDFS and using a traditional file system that needs to be properly balanced before taking a decision.

Given the amount of data generated and analyzed in a genomics facility, and the results of these tests, we can conclude that a cluster like the second one may

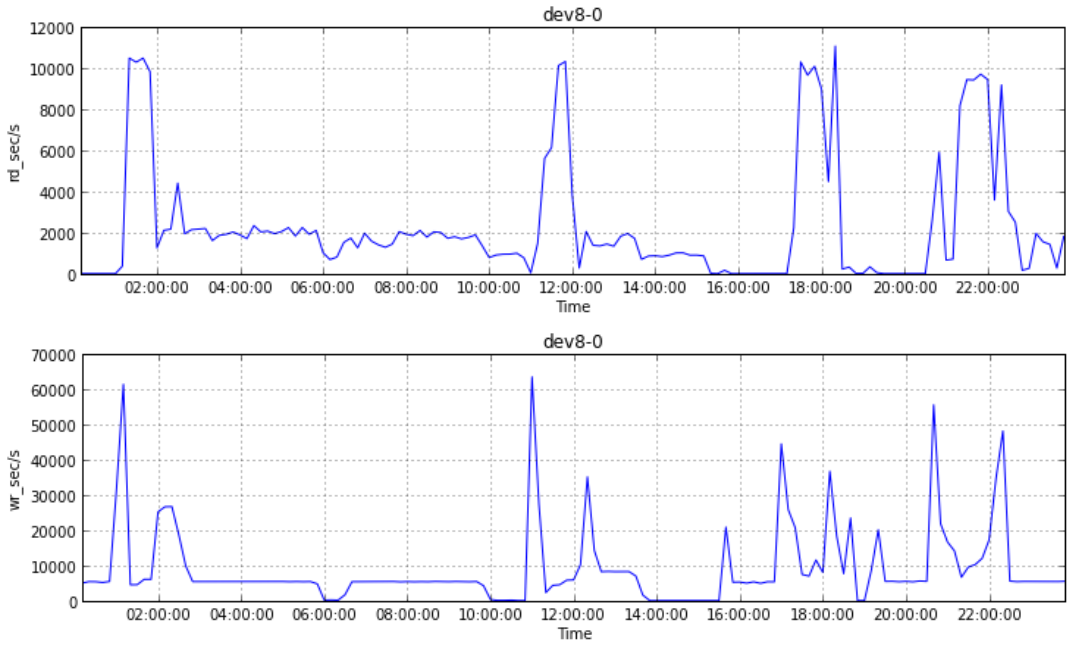

Figure 6.4. Disk I/O in datanode 1 during tests for Escherichia Coli's genome in cluster 3.

be optimal: A couple of nodes with enough computational power and local storage to handle analysis against big genomes, improves the performance of sequential aligners, and helps to the automation of the analysis.

6.2.5 Integrating Seal in bcbio-nextgen

As explained in section 5.3.2, the modularity of *bcbio-nextgen* allows the easy integration of new tools. Code 6.2 contains all what is needed to integrate Seal into *bcbio-nextgen*. It performs all the steps that were performed in the tests automatically, adding some basic checkings on the HDFS:

1. Checks the correctness of the parameters, and the existence of the data to analyze.
2. Moves data into HDFS.
3. Performs the FASTQ - PRQ conversion.
4. Runs the alignment against the given reference genome.
5. Takes the results from HDFS and extracts them in SAM format.

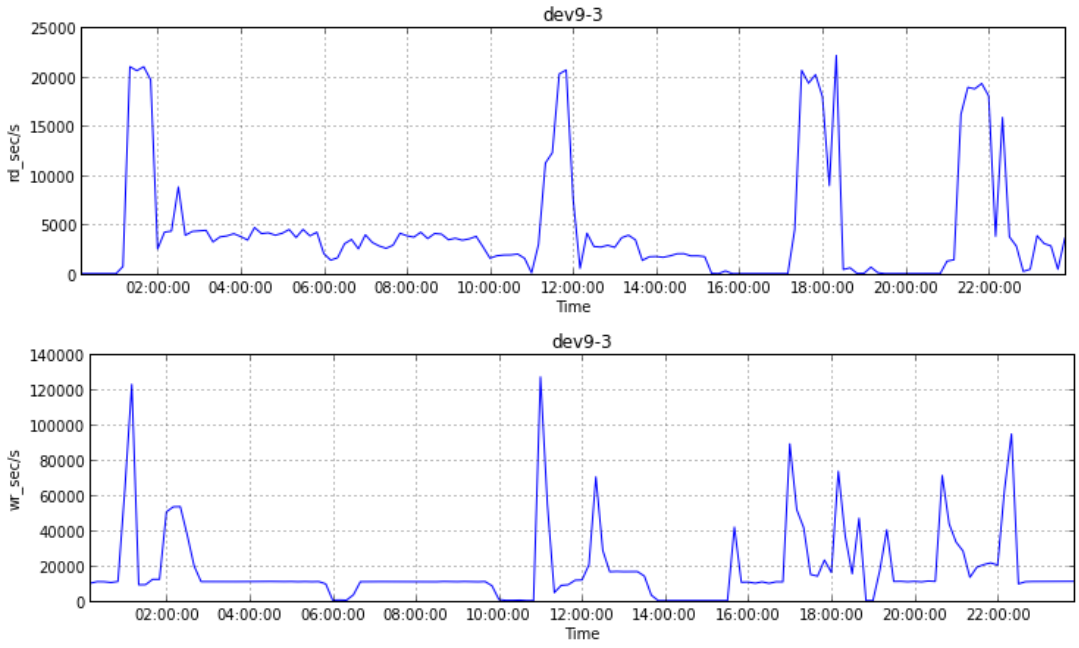

Figure 6.5. Disk I/O in datanode 2 during tests for Escherichia Coli's genome in cluster 3.

```
"""Next-gen alignments with Seal (http://biodoop-seal.sourceforge.net/
    index.html)
"""
import os
import subprocess

from bcbio.log import logger2 as logger
import pydoop.hdfs as hdfs

def align(input_dir_hdfs, output_dir_hdfs, out_base, ref_file_hdfs,
    config):
    """Perform a Seqal alignment, generating a SAM file.
    """
    #Do some basic checking
    if not hdfs.path.exists(input_dir_hdfs) or not hdfs.ls(
        input_dir_hdfs):
        raise IOError("The input dir for the alignment is either empty
            or doesn't exist")
    if not hdfs.path.exists(ref_file_hdfs):
        raise IOError("Could not find the reference genome")
    if hdfs.path.exists(output_dir_hdfs):
        raise IOError("The output directory already exists in the HDFS"
            )
```

```
    logger.info("Performing the alignment")
    _run_seqal_align(input_dir_hdfs, output_dir_hdfs, ref_file_hdfs,
        config)

    #Convert the output to SAM file
    sam_file = out_base + '.sam'
    with open(sam_file, 'w') as sam:
        cl = ['hadoop', 'fs', '-cat', os.path.join(output_dir_hdfs, '
            part-r-*')]
        subprocess.check_call(cl, stdout=sam)



def _run_seqal_align(input_dir_hdfs, output_dir_hdfs, ref_file_hdfs,
    config):
    """Run the alignment with seqal.
    """
    align_only = config['algorithm'].get('align_only', false)
    reducers = config['algorithm'].get('num_reducers', 0)
    cl = [config['program']['seqal']]
    if align_only:
        cl.append('-a')
    if reducers > 0:
        cl.extend(['-r', reducers])
    cl.extend([input_dir_hdfs, output_dir_hdfs, ref_file_hdfs])
    logger.info(' '.join(cl))
    subprocess.check_call(cl)
```

Code 6.2. Seal's wrapper for *bcbio-nextgen*

Thanks to the automated installation procedure developed in chapter 4, and the recipes developed in this chapter, deploying the pipeline in a Hadoop cluster, together with Seal, can be done automatically.

In chapter 3 we also explained how *bcbio-nextgen*'s behavior is driven by configuration files. After the wrapper has been implemented, the user needs to specify that he/she wants to use Seal to align. This is done modifying the configuration file that tells the pipeline which programs to use and how. Code 6.3 shows the relevant part of this configuration file.

```
...
program:
  bowtie: bowtie
  samtools: samtools
  bwa: bwa
  seqal: seal_seqal
  tophat: tophat
  cufflinks: cufflinks
  ucsc_bigwig: wigToBigWig
  picard: $PICARD_HOME
  gatk: $GATK_HOME
  snpEff: $SNPEFF_HOME
  fastqc: fastqc
```

```
    pdflatex: pdflatex
    barcode: barcode_sort_trim.py
  algorithm:
    aligner: bowtie
    max_errors: 2
    num_cores: 8
    platform: illumina
    recalibrate: false
    snpcall: false
    bc_mismatch: 2
    bc_allow_indels: true
    bc_read: 1
    bc_position: 3
    java_memory: 3g
    upload_fastq: false
    save_diskspace: false
    screen_contaminants: false
    write_summary: false
    compress_files: true
    num_reducers: 0
    align_only: false
    ...
```

Code 6.3. Extract of post_process.yaml configuration file, which tells *bcbio-nextgen* pipeline which programs to use and how.

Chapter 7

Planning and resources

In this chapter we detail the costs of this project, as well as the planning for the tasks done.

7.1 Costs

Three roles have been considered for the calculation of these costs:

- **Supervisor**: Supervises the project, specifies the tasks to be done and evaluates the results. Supervisor is also in charge of ensuring the accomplishment in time of the tasks.
- **Analyst**: This person is in charge of the requirements and specification of the project. Analyst also defines the behavior of the system and its architecture.
- **Programmer**: Implements the code and functionalities for the project, following the analyst's specification.

In table 7.1 the details of human resources' costs are shown.

Role	**Cost**	**Dedication**	**Total cost**
Supervisor	60 €/h	80 h	4.800 €
Analyst	40 €/h	464 h	18.560 €
Programmer	35 €/h	360 h	12.600 €
Total			**35.960 €**

Table 7.1. Human Resources' costs for this project (€)

The costs for amortizable goods are shown in table 7.2. The costs are intended for a Hadoop cluster consisting on 50 datanodes with similar specifications that the ones used in cluster configuration 2 in chapter 6.

The totals costs for this project are shown in table 7.3

Description	Price	Amortization	Unit cost	Units	Total cost
Namenode Machine	500€	4 years	10.47 €/month	1	104,7 €
Datanode Machine	900€	4 years	18.75 €/month	50	9.375 €
Total					**9.479,7 €**

Table 7.2. Amortizable goods costs

Concept	Cost
Human resources	35.960 €
Amortizable goods	9.479,7 €
Total	**45.439 €**

Table 7.3. Total costs for this project.

7.2 Planning

Figure 7.1 shows a breakdown of the tasks done to complete this project, as well as an accurate planning.

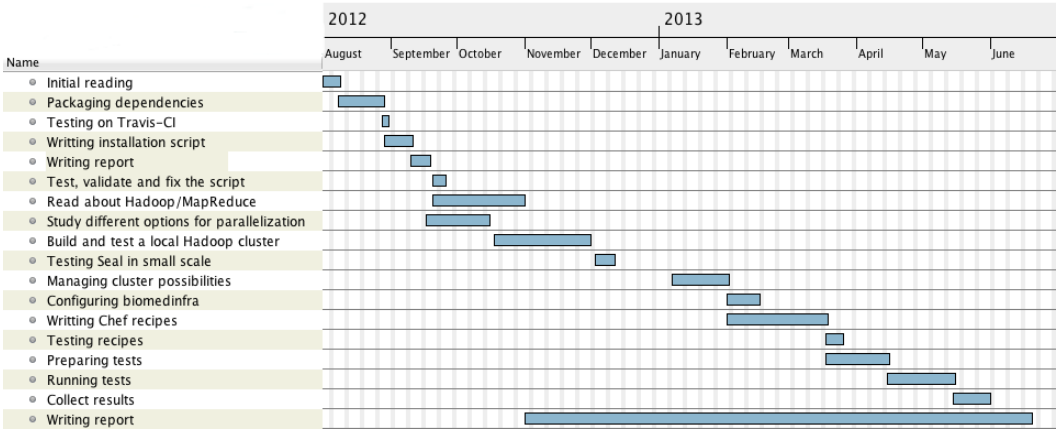

Figure 7.1. Gantt diagram for this project.

Chapter 8

Final remarks

Thorough this project, we have tried to follow a way of working that permits the reusability and reproducibility of our work. All the code implemented during this project are publicly available on GitHub, as well as the software packages for *bcbio-nextgen* dependencies, and the results of the tests.

On one side, we've implemented a method to automatically install and deploy *bcbio-nestgen* pipeline, the NGS-analysis pipeline used in *Science For Life Laboratory*. We've also automated the testing of the aforementioned pipeline, integrating it in a CI system. We have built software packages for *bcbio-nextgen* that are useful not just for us, but also for anyone that may use the same software. All these automations have helped us to improve efficiency and reliability in our development team.

On the other side, in order to increase the performance of the alignment step in the genomics analysis carried out by *bcbio-nextgen*, we have considered and evaluated a Hadoop based sequence aligner, Seal. During this evaluation we've learnt several things about Hadoop and the alignment process:

- Hadoop permits the utilization of commodity hardware to perform heavy computational tasks. This is an advantage in terms of infrastructure: It is easier to build and maintain a cluster of commodity hardware than an HPC. This also has its drawbacks: Firstly, if the hardware is too poor, it may not be able to perform some atomic heavy tasks, like happened to us with the first cluster configuration and the Human genome. Secondly, common hardware may not be as reliable as high performance hardware, meaning that the possibility of a hardware failure increases in this type of clusters. Fortunately, Hadoop ensures reliability and stability against hardware/process failure, handling automatic re-scheduling of failed jobs.

  From my experience in these months in *Science For Life Laboratory* I can say that the situation in which a step on the analysis fails is very common:

memory errors, collisions with other jobs, failures on a machine, etc. Having a completely Hadoop-based pipeline would help to make the analysis smoother.

- BWA is more efficient than Hadoop if the cluster is not properly dimensioned. In our tests, we could see how a bad dimensioning of the cluster can have a really big impact in Seal's performance. HDFS offers data availability and reliability, but one needs to balance the trade-off that these properties imply.

Following the same philosophy of reproducibility, a set of recipes that allow the installation and configuration of a Hadoop cluster were also implemented. This recipes were used to configure the OS image that was the base for all our tests.

During the realization of this project I have been able to apply a lot of the knowledge acquired during my degree. Project based courses like "Projecte de programació" or "Projecte de Xarxes de Computadors" prepared me to work in group, rather than individually, and showed me the importance of working with control version system's in group projects. Operating systems' courses like "Administració de Sistemes Operatius" gave me the bases to understand how to manage OS and concretely Linux conventions, easing me the task of creating standard packages in a correct way for the dependencies. Software engineering courses showed me the importance of concepts like reusability or planning, that I have followed during the whole development of this project. Finally, distributed programming courses like "Sistemes Operatius Distribuits en Xarxa" or "Programació Concurrent i Distribuida" helped me to easily understand the architechture of Hadoop and the code of Seal.

To conclude, during the last period of this project I had the opportunity to participate in a hackathon[1] organized in Sardinia, Italy. In this hackathon, I had the privilege of working with a group of incredible developers, all of them working in Hadoop based bioinformatics applications. Luca Pireddu, author of Seal, was also participating, and he gave me a lot of advices and recommendations on how to work with Seal and Hadoop in general. Together, we implemented two new functionalities for Seal:

1. The process of converting the Illumina Machines' image files (BCL) into QSEQ[2] files can now be carried on by Seal, with the advantages that this implies: more parallelism, data reliability and having more steps in the same pipeline.

2. We integrated Seal into Travis-CI, providing thus a continuous testing. This integration also implicitly tests the chef recipes developed to configure a

[1] A hackathon (also known as a hack day, hackfest or codefest) is an event in which computer programmers and others involved in software development, including graphic designers, interface designers and project managers, collaborate intensively on software projects

[2] QSEQ files are generated by earlier Illumina machines. They're Illumina's specific variant of FASTQ format

cluster: In every build, these recipes are used to install and configure Hadoop in single-machine mode, install Seal and its dependencies and run the test suite.

Unfortunately, this hackathon was just a week before the end of this project, and we could not apply and test all the new features that were developed there. However, this hackathon gave us a lot of new ideas to improve our methods. Hopefully, this project will be the base for positive change in *Science for Life Laboratory*.

Bibliography

[Bla00] David Blackman. Debian package management, part 1: A user's guide. *Linux J.*, 2000(80es), November 2000.

[Bro12] Titus Brown. Our approach to replication in computational science. `http://ivory.idyll.org/blog/replication-i.html`, 2012.

[Cla05] Phil Clapham. Publish or perish. *BioScience*, 55(5):390–391, 2005.

[CW53] Francis Crick and James Watson. Molecular structure of nucleic acids. *Nature*, 171(4356):737–738, 1953.

[DG08] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[Eur13] Elixir Europe. Elixir project web page, 2013.

[Fur91] John L Furlani. Modules: Providing a flexible user environment. In *Proceedings of the Fifth Large Installation Systems Administration Conference (LISA V)*, pages 141–152, 1991.

[gee12] Computer geek. Secondary namenode - what it really do? `http://computegeeken.blogspot.se/2012/06/secondary-namenode-what-it-really-do.html`, 2012.

[Gui12] Roman Guimera. bcbio-nextgen: Automated, distributed next-gen sequencing pipeline. *EMBnet.journal*, 17(B), 2012.

[HE11] Ron Jeffries Hakan Erdogmus, Grigori Melnik. Test-driven development. 2011.

[HMS+09] Jo E. Hannay, Carolyn MacLeod, Janice Singer, Hans P. Langtangen, Dietmar Pfahl, and Greg Wilson. How do scientists develop and use scientific software? In *Proceedings of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*, SECSE '09, pages 1–8, Washington, DC, USA, 2009. IEEE Computer Society.

[Hol12] Alex Holmes. Hadoop in practice, 2012.

[KNQ+09] Iwanka Kozarewa, Zemin Ning, Michael A Quail, Mandy J Sanders, Matthew Berriman, and Daniel J Turner. Amplification-free illumina sequencing-library preparation facilitates improved mapping and assembly of (g+ c)-biased genomes. *Nature Methods*, 6(4):291–295, 2009.

[LD09] Heng Li and Richard Durbin. Fast and accurate short read alignment with burrows–wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009.

[Len09] Joe Lennon. Introduction to couchdb. In *Beginning CouchDB*, pages 3–9. Springer, 2009.

[Les02] A.M. Lesk. *Introduction to Bioinformatics.* Oxford University Press, Incorporated, 2002.

[Les12] A. Lesk. *Introduction to Genomics.* Introduction to Genomics. OUP Oxford, 2012.

[LHW+09] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, Richard Durbin, et al. The sequence alignment/map format and samtools. *Bioinformatics*, 25(16):2078–2079, 2009.

[Lln11] Computing Llnl. Slurm: A highly scalable resource manager, 2011.

[Met09] Michael L Metzker. Sequencing technologies—the next generation. *Nature Reviews Genetics*, 11(1):31–46, 2009.

[MG77] Allan M Maxam and Walter Gilbert. A new method for sequencing dna. *Proceedings of the National Academy of Sciences*, 74(2):560–564, 1977.

[NS11] Stephen Nelson-Smith. *Test-Driven Infrastructure with Chef.* O'Reilly Media, 2011.

[Pir11] SEAL: a distributed short read mapping and duplicate removal tool. *Bioinformatics*, 27(15):2159–2160, August 2011.

[Rom12] Roman Valls Guimera. Enabling Automatic Data Analysis in Bioinformatics Core Facilities, A high-throughput sequencing pipeline, 2012.

[SC75] Fred Sanger and Alan R Coulson. A rapid method for determining sequences in dna by primed synthesis with dna polymerase. *Journal of molecular biology*, 94(3):441–448, 1975.

[SE11] Robert Schmieder and Robert Edwards. Fast identification and removal of sequence contamination from genomic and metagenomic datasets. *PLoS one*, 6(3):e17288, 2011.

[SMLF08] Borja Sotomayor, Rubén Santiago Montero, Ignacio Martın Llorente, and Ian Foster. Capacity leasing in cloud systems using the opennebula engine. In *Workshop on Cloud Computing and its Applications*, volume 2008, 2008.

[Sof12] Apache Software. Apache hadoop main page, 2012.

[upp08] Uppmax progress report, 2008.

[VM12] Erik Vallabh Minikel. Forward and reverse reads in paired-end sequencing. `http://www.cureffi.org/2012/12/19/forward-and-reverse-reads-in-paired-end-sequencing/`, 2012.

[WGS09] Zhong Wang, Mark Gerstein, and Michael Snyder. Rna-seq: a revolutionary tool for transcriptomics. *Nature Reviews Genetics*, 10(1):57–63, 2009.

[Whi12] Tom White. *Hadoop: The definitive guide*. O'Reilly Media, Inc., 2012.