Università degli Studi di Cagliari
Dottorato di Ricerca in Ingegneria Biomedica
Ciclo XXVII

Unlocking Large-Scale Genomics

Presentata da:	Luca Pireddu
Coordinatore dottorato:	Prof. Giacomo Cao
Tutor:	Prof. Riccardo Scateni e
	Prof. Giacomo Cao

Settore scientifico-disciplinare di afferenza: ING-INF/06
Esame finale anno accademico 2014–2015

Abstract

The dramatic progress in DNA sequencing technology over the last decade, with the revolutionary introduction of next-generation sequencing, has brought with it opportunities and difficulties. Indeed, the opportunity to study the genomes of any species at an unprecedented level of detail has come accompanied by the difficulty in scaling analysis to handle the tremendous data generation rates of the sequencing machinery and scaling operational procedures to handle the increasing sample sizes in ever larger sequencing studies. This dissertation presents work that strives to address both these problems. The first contribution, inspired by the success of data-driven industry, is the Seal suite of tools which harnesses the scalability of the Hadoop framework to accelerate the analysis of sequencing data and keep up with the sustained throughput of the sequencing machines. The second contribution, addressing the second problem, is a system is developed to automate the standard analysis procedures at a typical sequencing center. Additional work is presented to make the first two contributions compatible with each other, as to provide a complete solution for a sequencing operation and to simplify their use. Finally, the work presented here has been integrated into the production operations at the CRS4 Sequencing Lab, helping it scale its operation while reducing personnel requirements.

Acknowledgements

I would like to thank my supervisor and mentor, Dr. Gianluigi Zanetti, for his guidance and encouragement over the past several years. I would also like to thank my colleagues at CRS4 for providing a stimulating environment where I can solve problems and find new problems to solve. Another thank you to my employer, CRS4, who gave me the opportunity to test my hypotheses within the organization and kindly granted me the necessarily flexibility that allowed me to work on this doctoral program. In addition, I would like to thank the program coordinator, Prof. Giacomo Cao, and my tutor, Prof. Riccardo Scateni, for giving me the opportunity to work on this Ph. D.

Finally, and most of all, a special thank you to Valentina for all her help and support, and to my family who, though far, are always close to me.

Contents

Chapter 1

Introduction

The past decade has yielded dramatic progress in the technologies for DNA sequencing, beginning with the introduction of next-generation sequencing (NGS) [87]. Previous to this technological breakthrough, DNA sequences were generally acquired through Sanger sequencing [78] which, though quite accurate, is not a high-throughput technique. In fact, compared to the previous state-of-the-art, the development of next-generation sequencing – also known as high-throughput sequencing – caused an exponential decrease in the data acquisition costs, along with an inversely proportional increase in data generation rates. The progress has been so rapid that from 2008 to 2016 the throughput of NGS machines and the cost of sequencing a human-sized genome have improved by four orders of magnitude! As it became more economically accessible, DNA sequencing opened up to a myriad of new applications that were previously technologically or economically unfeasible [86]. High-throughput sequencing can now be used for research into understanding human genetic diseases [66] and population-wide studies to better understand human phylogeny [25]. There are also obvious applications in oncology to study and identify tumours [9], and even at the level of personalized diagnostic applications [13]. Moreover, the same technology can be used to study plant and animal genomes [28, 61].

Nevertheless, the data produced by the NGS process is not directly interpretable from a biological point of view. Instead, the product of a high-throughput next-generation sequencing run is billions of short DNA fragments, called *reads*, which need to be analyzed computationally to understand the structure of the original whole DNA sequence from which they came. While the specific analyses to be performed will vary depending on the application, they are all irremediably data intensive due to the sheer data production rate of the NGS process. Unfortunately, the dramatic progress in the data acquisition technology has not been directly followed by compa-

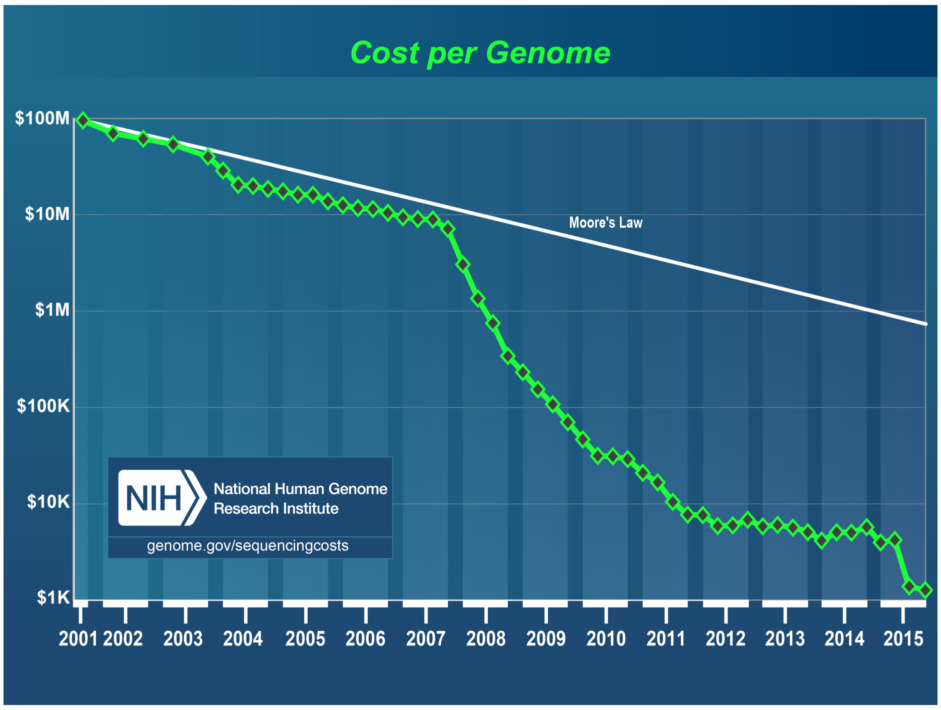


Figure 1.1 – The progression of the cost of sequencing a human-sized genome. The impact of NGS technology can be clearly seen starting on January 2008. Note that the graph uses a logarithmic scale. For comparison, the dotted white line depicts Moore's Law [59] which – loosely speaking – describes the trend of computing power to double every year. The sequencing costs considered include direct and indirect costs of producing the data, but not the cost to analyze it. This graph is graciously provided by the American National Human Genome Research Institute (NHGRI) [36].

rable progress in the computational techniques and infrastructure that support the analysis. The result has been a shift of the bottleneck in sequencing studies from data acquisition to data processing and analysis [79] (see Fig. 1.2). Moreover, many of the applications of sequencing become interesting as the number of samples grow – consider, for instance, population-wide genetic studies with thousands of samples. However, increasing the number of samples, in addition to multiplying data size, also multiplies the number of datasets, thus introducing new logistical and data management problems that must be handled appropriately to ensure scalability, reproducibility and to curtail the probability of errors.

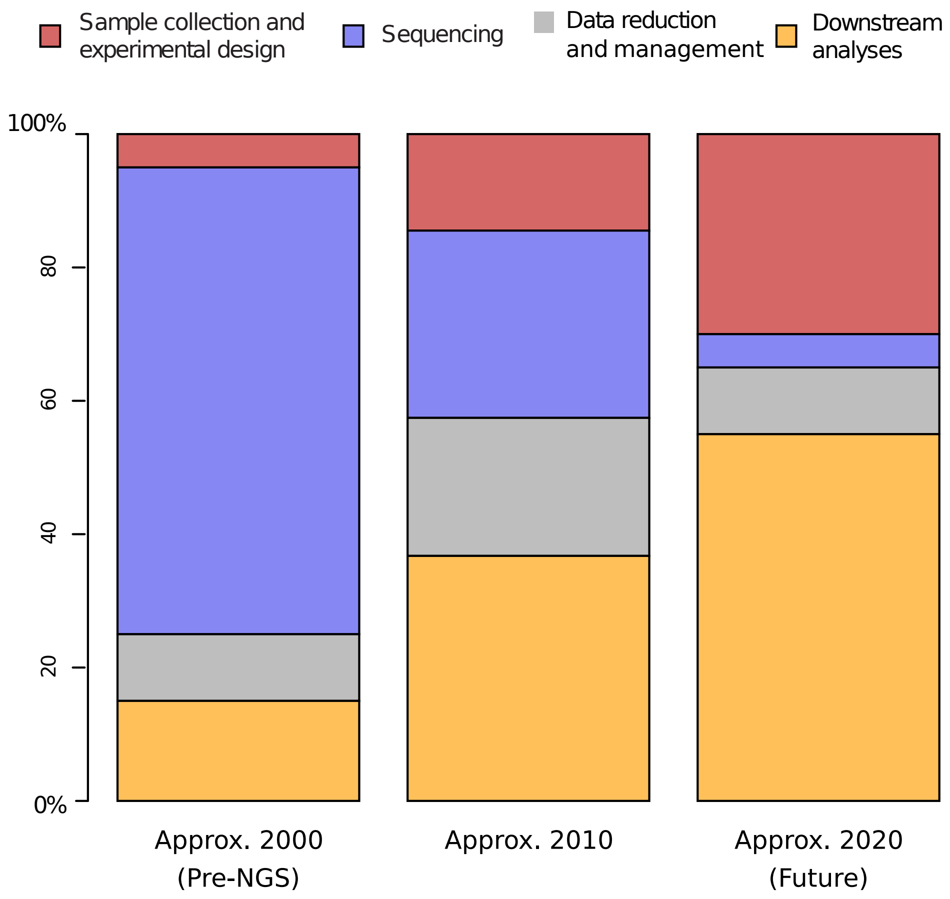


Figure 1.2 – Illustration (from Sboner et al. [79]) of the relative costs in three different time periods of a typical sequencing-based study, which consists of four main components: (i) sample collection, (ii) data acquisition (sequencing), (iii) data reduction management and (iv) data analysis. While in the pre-NGS era the data acquisition phase was the most expensive part of the study, NGS is shifting those costs to data analysis.

1.1 Motivation

1.1.1 Next-Generation Sequencing

Next-generation sequencing actually refers to a number of different techniques that emerged around the same time (around 2005). These are perhaps better described as "second-generation DNA sequencing" (after 10 years one might be justified in arguing that these are no longer the "next" generation!). In this work we will be mostly concerned with the technology used in Illumina's sequencers [10]. For a brief but complete review of all the main NGS methods the reader may refer to the work by Shendure and Ji [87].

The NGS method is a *shotgun* technique, where the DNA to be acquired is broken up into random fragments that are all read simultaneously in the sequencing process. This "flaring" of small simultaneous observations over the entire genomic sequence is what gives the technique its name. On Illumina sequencers, each individual fragment results in one or two *reads*: one from one end of the fragment and, optionally, one from the opposite end (see Fig. 1.3). Two reads resulting from the same fragment are said to be *paired*. The reads usually do not cover the entire fragment, leaving an unsequenced portion of DNA between the pair.

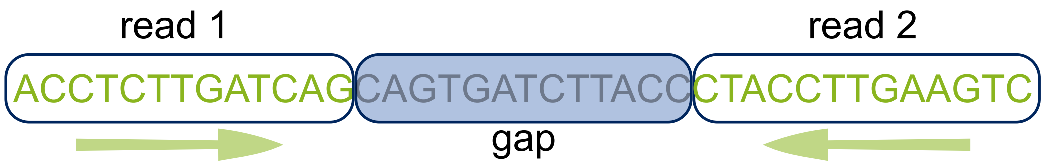


Figure 1.3 – A DNA fragment is sequenced from its two opposite ends, resulting in two paired reads.

The most common sequencing scenario – and the one that concerns us in this work – is the one where there is a reference genome available for the sample being sequenced. This type of operation is known as *resequencing*. When resequencing, we use the reference sequence as a guide to reconstruct the genome of the sample. Further analysis then highlights the variations between the sample and the reference genome – information which can form the basis for further studies, such as identifying associations between variations and phenotypes.

The resequencing process is ideal to leverage the type of data produced by shotgun sequencing. However, it entails a multi-step analysis phase that requires significant amounts of computational resources – both in terms of processing power and input/output (I/O) throughput.

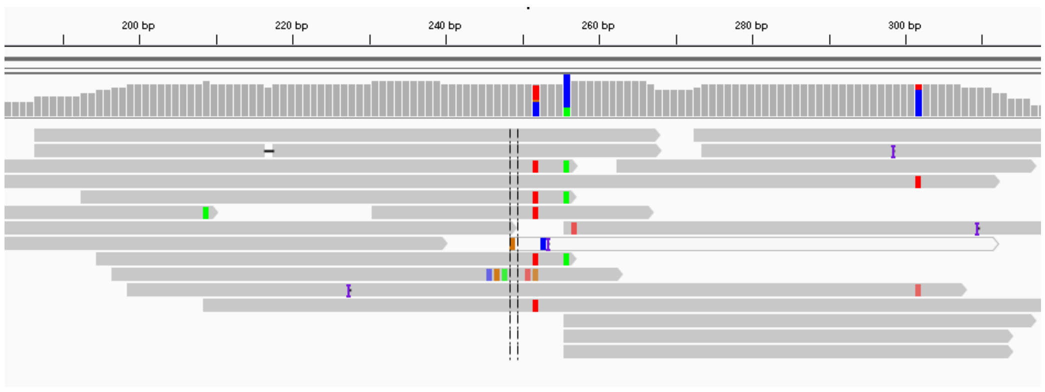


Figure 1.4 – A detail of a screenshot of the Integrative Genome Viewer (IGV) [74, 92] showing an area of the reference with a number of overlapping reads (horizontal gray bars). The vertical gray bars indicate coverage – i.e., the number of times that specific position of the reference was sequenced.

1.1.2 Large sets of large datasets

Next-generation sequencing activity quickly results in a lot of data. Consider that a modern high-throughput sequencer like the Illumina HiSeq 4000 can produce 1500 Gigabases, which equate to about 4 Terabytes of uncompressed sequencing data, in just 3.5 days [31]. That much data is sufficient to reconstruct only up to 12 human-sized genomes (3 Gigabases in length) due to the oversampling required to reconstruct the genomes from the short reads generated by the sequencing machine. With some quick arithmetic it is easy to conclude that a single sample of this type equates to a bit over 300 GB of uncompressed data. Of course, the data from each sample needs to be computationally processed and digitally stored. Such datasets quickly become too big to process on a single computer, giving rise to *Big Data*-type problems where sophisticated computational techniques need to be applied to achieve results in a reasonable time.

Moreover, in the context of a study or a sequencing center, this hefty per-sample data size is typically multiplied by a significant number of samples. Consider that for sequencing-based studies large numbers of samples need to be treated to achieve high analysis sensitivity or to get population-wide statistics. The population-wide study by Orrù et al. [66] provides an example of this situation. For that study, 2870 whole genomes were sequenced, resulting in approximately one *petabyte* of sequencing data to be analyzed and stored. Processing such a large quantity of data in less time than the sequencing machines require to generate them requires a substantial computing infrastructure and the scalable computational software to leverage it.

Sequencer	Read length	Time (days)	Gigabases	Gigabytes (GB)	Rate (GB/day)
HiSeq 2000	2 x 100	8	400	1000	125
HiSeq 2500	2 x 125	6	1000	2500	417
HiSeq 4000	2 x 150	3.5	1500	3750	1071

Table 1.1 – Maximum sequencing capacities for a number of Illumina high-throughput sequencers. Note that the machines can be configured to run in several modes trading off the amount of bases sequenced with the duration of the sequencing run or the read length. Here we report the configuration that maximizes the former and uses two flow cells. The size of the output in gigabytes is intended for uncompressed reads with base qualities and id strings, considering a constant total size of 2.5 bytes/base. Although the actual size of the sequencer's output directory is larger, this size serves well to quantify the amount of data to be processed. Source: Illumina specification sheets [31–33].

In addition to the sheer data size, executing the large studies that are so well suited to NGS technology presents difficulties related to making the processing and analysis operation scale with respect to the number of samples. For each sample, at least one computation analysis procedure needs to be executed, and it will produce a number of datasets spread over an even larger number of files. Moreover, every step presents a possibility of error which must be kept in check, since the presence of errors in the project will scale superlinearly[1] with respect to the number of samples.

Finally, as with all science, it is of paramount importance to ensure the reproducibility of such large-scale sequencing studies by documenting the *provenance* of all results. This can be a rather difficult task because of the sheer number of parameters involved in the generation and analysis of sequencing data. While the reproducibility of the sequencing run is assured by the fact that the sequencing apparatus documents all its parameters, the onus falls on the user to document the details of the analysis procedure. It is necessary to document the analysis steps performed, the software used and the version of said software – lest its behaviour change in newer or older versions – the software parameters, as well as any model data used (e.g., a reference genome) and its version as well. Providing such detailed tracing of operations at scale would be a daunting task without the implementation of proper automation and support tools.

[1]This stems from the fact that in addition to the errors strictly related to processing the sample at hand, one must also consider the errors originating from desired or accidental interactions with the analysis procedures of other samples

1.2 Contributions

This dissertation makes three main research contributions that work towards addressing the issues described in the previous section. They are summarized below.

Scalable sequence processing software. A suite of scalable distributed software tools for processing sequencing data, called *Seal*, has been implemented and released as an open source project. The software is currently used at the CRS4 NGS Lab (Chapter 3).

Automation of primary processing. A system to automate the standard primary processing that needs to be performed on raw sequencing data. The system fully traces all the operations performed on the data and it is currently in use at the CRS4 NGS Lab (Chapters 4 and 5).

Scalable scripting for sequence analysis. A specialization of a scripting language to work on sequencing data. With this system, a user that is not an expert in distributed computing can write programs that manipulate and analyze sequencing data using the full power of a computing cluster (Chapter 6).

The remaining chapters of this dissertation provide background and discuss the details of these contributions.

1.3 Achievements

The research work presented in this dissertation resulted in the publication of the following articles over the course the Ph. D. program.

- G. Cuccuru,. . . , L. Pireddu, et al. "An automated infrastructure to support high-throughput bioinformatics". In: *High Performance Computing & Simulation (HPCS), 2014 International Conference on.* IEEE. 2014, pp. 600–607.

- L. Pireddu, S. Leo, et al. "A Hadoop-Galaxy adapter for user-friendly and scalable data-intensive bioinformatics in Galaxy". In: *Proceedings of the 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics.* ACM. 2014, pp. 184–191.

- A. Schumacher, L. Pireddu, et al. "SeqPig: simple and scalable scripting for large sequencing data sets in Hadoop". *Bioinformatics* 30 (1), 2014, pp. 119–120.

In addition, the methods presented in this work were used to analyze most of the sequencing data within the context of this population-wide genome-wide association study:

> V. Orrù,. . . , L. Pireddu, et al. "Genetic variants regulating immune cell levels in health and disease." *Cell* 155 (1), Sept. 2013, pp. 242–56.

Finally, the experiences in computationally processing large quantities of sequencing data gathered over the course of this doctoral program, along with with those of other colleagues working in other research institutions, were published in the article:

> O. Spjuth,. . . , L. Pireddu, et al. "Experiences with workflows for automating data-intensive bioinformatics". *Biology Direct* 10 (1), 2015, pp. 1–12.

Chapter 2

Background

2.1 The CRS4 NGS Lab

CRS4[1] is a non-profit public multidisciplinary research organization in Italy. It hosts a high-throughput sequencing facility that is directly interconnected to its computational resources (3000 cores, 4.5 PB of storage). With three Illumina HiSeq 2500 sequencers, it is the largest NGS platform in Italy, with a production capacity of up to about 1.25 TB of raw sequencing data per day. Among its most important projects the lab counts large-scale studies on the genetics of autoimmune diseases [66] and longevity [25].

The CRS4 NGS Lab is relevant to this dissertation because over the course of this work it served as a testbed and early adopter for the proposed solutions, not to mention steady source of new challenges and ideas.

2.2 Best-practice sequence processing

As already mentioned in Chapter 1, the raw data produced by the sequencers needs to be processed to infer information about the genome from which the fragments were generated. Depending on the sequencing application at hand, there are standard processing workflows that are accepted by the community as being "best practice." For the purpose of this work, we are mainly concerned with experiments performing resequencing (see Section 1.1.1) for the purpose of *variant calling* – the procedure to identify the specific variations of the sample's genome with respect to the reference; in particular, the tools presented in Chapter 3 are used for the *primary processing* phase. Note that this does not imply that these contributions of this dissertation are not

[1] `http://www.crs4.it`

applicable in other circumstances.

Within the context of resequencing for variant calling, van der Auwera et al. from the Broad Institute[2] and Wellcome Trust[3] published the gold-standard procedure in 2013 [8]. However, the tools and procedures used in sequence processing evolve very quickly, so the most up-to-date recommendations are published directly on the Broad Institute's web site [34]. Although the Broad's recommendations are well accepted, they are centered around their own tool suite – the Genome Analysis Toolkit (GATK). Nevertheless, their useful advice can be easily interpreted and applied with alternative tools.

Once the sample's reads are available, the primary analysis part of the best practice workflow for DNA consists of the following steps.

1. Map to reference. As mentioned in Sec. 1.1.1, next-generation shotgun sequencing produces a shattered view of the genome in the form of many small unordered DNA reads. To reconstruct the genome of the sample being resequenced one takes the reads generated by the sequencer and uses specialized *mapping* or *alignment* software, which uses approximate string matching algorithms to find the position in the reference sequence from which the reads were most likely acquired. In this manner, we use the reference sequence as a guide and as a coordinate system to reconstruct the sample's genome. The mapping algorithms are designed to be robust to sequencing errors and the small genetic variations that characterize the sample. Moreover, when sequenced the genome is typically oversampled many times; for instance, for whole-genome sequencing 30 times oversampling – also known as *30X coverage* is common. The oversampling results in reads that overlap, which will be essential in later phases to detect heterozygosity and to distinguish genomic variants specific to the sample from mere sequencing errors. In addition to the reference position, the mapping process also calculates the differences between the sample's read and reference sequence at the mapped position. This difference is expressed as a sequence of string editing operations: match, insert, delete.

2. Mark duplicates. The sequencing process generates a number of duplicate reads – meaning that from one original genome fragment the process produces two or more reads. The duplicates can originate from the amplification process used while preparing the material for sequencing or from optical effects in the sequencing apparatus itself. Either way, these duplicates are a form of error that can skew statistics and need to be identified and properly handled.

[2]`https://www.broadinstitute.org`

[3]`http://www.wellcome.ac.uk/`

3. Realign indels. When mapping reads to a reference, the alignment algorithm works on one read at a time. While mapping the ends of the read the aligner is at a higher risk of committing the error of accepting a substitution error instead of inserting a sequence insertion or deletion (which, in general, are less likely). However, this situation can be more easily detected when analysing together all the reads mapped at the location in question. In short, the end of the read in question may coincide with the middle of a number reads – where the alignment is more accurate – or with a known variant location. This step looks at these high-error situations and "fixes" the original read mapping when appropriate.

4. Recalibrate base qualities. The sequencer provides a probability of error for each base it produces based on its own internal error model. However, it has been observed that the probability of error on the bases is more accurate if calculated empirically by assuming that the read bases that match the reference sequence are correct and all the others are wrong – except at locations that are known to vary. Based on this rule, base qualities are recalculated by a likelyhood estimation conditioned on the following factors: the cycle in the sequencing process, the base preceding the current one, and the original probability of error estimated by the sequencer.

2.3 MapReduce and Hadoop

Hadoop [30, 77] is a platform for large-scale data processing. Although with its rising popularity the Hadoop ecosystem has been growing with numerous tools and variations, the crux of the system – and the only parts relevant to this work – are its open source implementation of the MapReduce paradigm [22] and the Hadoop Distributed File System (HDFS). In the past few years Hadoop has established itself as the *de facto* standard for large scale data processing, allowing both commercial and academic institutions to deal with projects of unprecedented size [29].

The Hadoop MapReduce implementation provides a framework into which an application can install appropriate *map* and *reduce* functions that, in combination, compute a desired result. These functions operate exclusively on data in the form of key-value pairs. The *map* function performs a transformation on the data, mapping each input tuple to one or more output tuples. On the other hand, at each invocation the *reduce* function simultaneously receives all the values associated to the same key, and therefore has the opportunity to perform aggregation-type operations. Both functions output key-value pairs exclusively through the framework. By following this programming model, procedure side-effects and unstructured interactions between processes are eliminated, which makes it possible to create a framework that automatically execute multiple instances of *map* and *reduce* functions in parallel and even on different computers. It can even re-execute the functions on the same input data – for instance, in the case of stragglers or to recuperate dead tasks – by adopting appropriate measures to ensure idempotency and safe concurrency. Among these measures we find that each work task executed by the framework writes to its own output file. As such, when processing data with Hadoop MapReduce datasets are typically kept in directories, which may contain any number of files, each of which contains a part of the entire dataset. Perhaps the most important consequence of following this rigid model is that the application-specific logic in the *map* and *reduce* functions is completely separated from the framework's, so that the latter can be packaged into a well-tuned and reusable distributed computing framework – exactly like Hadoop.

As an example, Algorithm 1 illustrates how to count word frequency with an algorithm that follows the MapReduce model. The *map* function splits the text into words and emits tuples *(word, 1)*. Then, for each different word, the reducer is invoked with all the '1's that were emitted by the maps – one for each occurrence of the word. By summing all the values, the observed frequency of the word is computed.

Algorithm 1 A MapReduce word count algorithm

```
function MAP($key, text$)
    for $word \in text$ do
        emit($word$, 1)
    end for
end function
function REDUCE($word, values$)
    $count \leftarrow 0$
    for $v \in values$ do
        $count \leftarrow count + v$
    end for
    emit($word, count$)
end function
```

2.3.1 HDFS

The Hadoop Distributed File System (HDFS) is a file system designed to store large quantities of large files by spreading them over the local disks of all the nodes in the computing cluster. HDFS splits files into large blocks

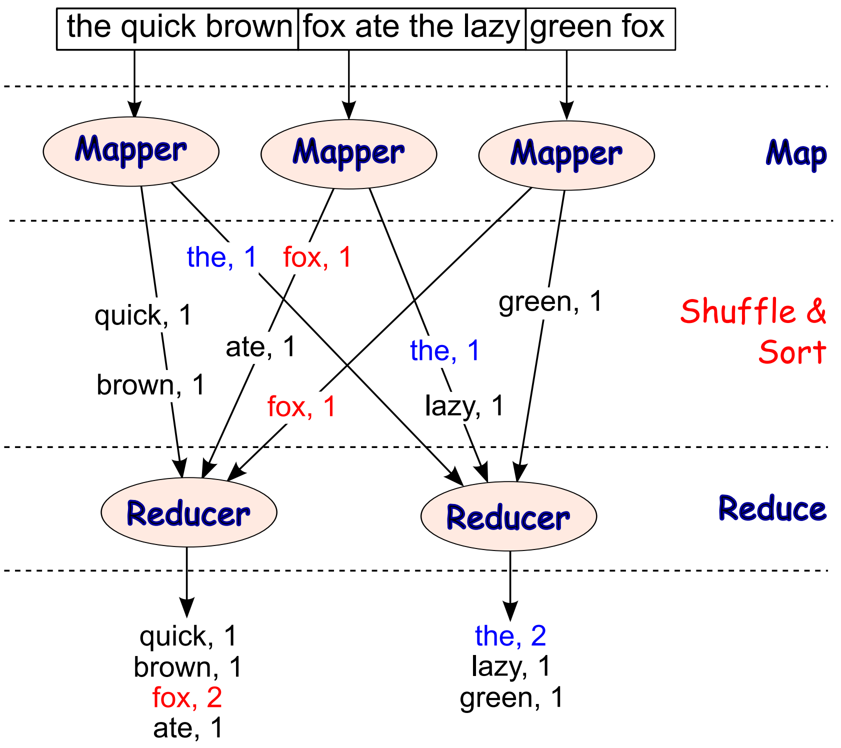


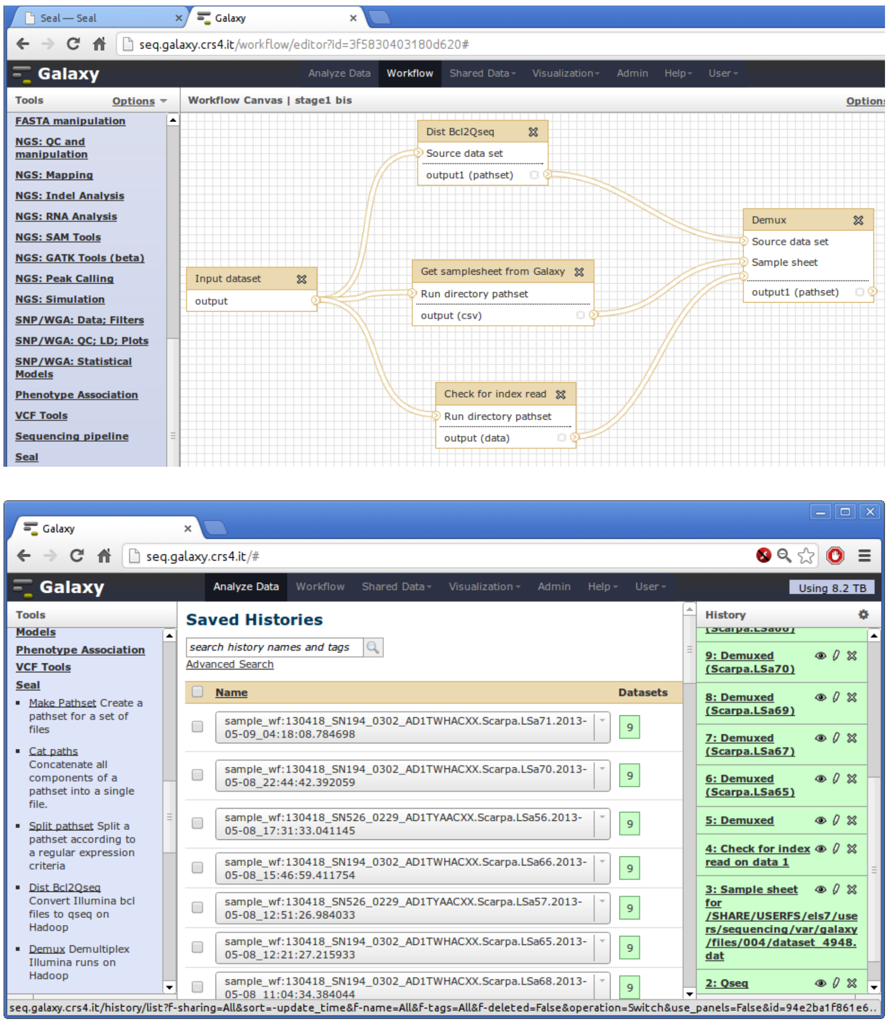
Figure 2.1 – Schematic diagram of the execution of the word count algorithm shown in Alg. 1 (*Image courtesy of Simone Leo*).

(i.e., usually 128 MB) and sends the various blocks to different nodes. To ensure robustness, each block is replicated a number of times. By storing data directly on the computing nodes, HDFS creates the opportunity to bring the computation to the data – i.e., executing the computation directly on the node containing the data block to be processed, thus avoiding data transfer and reducing network I/O. Another corollary to its architecture is that HDFS makes its total bandwidth scale with the number of nodes in the cluster. In addition to these performance advantages, by using regular disks HDFS relies on cheap hardware, and so is able to provide cost-effective storage solution.

2.4 Galaxy

Galaxy [27] is a user-friendly bioinformatics workflow management system. Through its web interface it allows users to graphically describe workflows by drawing pipes to connect tools. Once a workflow is launched, the system can then automatically manage its execution, running steps in an appropriate order and, when dependencies allow, executing them concurrently. The platform is extensible, allowing users to integrate new tools which can then be included in new workflows. In fact, there exists a large community-managed library of tools that are already integrated and thus are ready to use or easily added after installation.

To support managing workflows in this automated and extensible manner, Galaxy imposes a model on the tools that are integrated and the datasets that they process and generate – this is relevant for the work in Chapter 5. Specifically, tools are invoked by command line, where Galaxy provides them with input and output dataset paths and user-selected options. In the Galaxy model, tools that are invoked in succession in a workflow "communicate" – as in passing datasets from one to the next – by sharing file system space. So, Galaxy is configured to use a directory as a workspace, where it creates paths for the various datasets that are generated during its operation. It is important to note that Galaxy needs to be able to access and read these dataset files and thus the workspace needs to be on a file system mounted on the system where the platform is running.



Saved Histories

Name	Datasets
sample_wf:130418_SN194_0302_AD1TWHACXX.Scarpa.LSa71.2013-05-09_04:18:08.784698	9
sample_wf:130418_SN194_0302_AD1TWHACXX.Scarpa.LSa70.2013-05-08_22:44:42.392059	9
sample_wf:130418_SN526_0229_AD1TYAACXX.Scarpa.LSa56.2013-05-08_17:31:33.041145	9
sample_wf:130418_SN194_0302_AD1TWHACXX.Scarpa.LSa66.2013-05-08_15:46:59.411754	9
sample_wf:130418_SN526_0229_AD1TYAACXX.Scarpa.LSa57.2013-05-08_12:51:26.984033	9
sample_wf:130418_SN194_0302_AD1TWHACXX.Scarpa.LSa65.2013-05-08_12:21:27.215933	9
sample_wf:130418_SN194_0302_AD1TWHACXX.Scarpa.LSa68.2013-05-08_11:04:34.384044	9

Figure 2.2 – Screenshots of the Galaxy web application. The first one shows a simple workflow open in Galaxy's workflow editor. The second one shows a list of "histories", each one recording the steps executed in a data analysis procedure – be it defined by a workflow or interactive.

Chapter 3

Scalable, distributed processing of sequencing data

The data-intensive revolution [54, 93] in the life sciences has next-generation sequencing (NGS) machines among its most prominent officers. One of the main challenges brought forth by this phenomenon is to develop scalable computing tools that can keep up with such a massive data generation throughput. To date, it appears that most sequencing centers have opted to implement processing systems based on conventional software running on High-Performance Computing (HPC) infrastructure [89] – a set of computing nodes accessed through a batch queuing system and equipped with a parallel shared storage system. While with enough effort and equipment this solution can certainly be made to work, it presents some issues that need to be addressed. Two important ones are that developers need to implement a general way to divide the work of a single job among all computing nodes and, since the probability of node failures increases with the number of nodes, they also need to make the system robust to transient or permanent hardware failures, recovering automatically and bringing the job to successful completion. Nevertheless, even with these measures, the architecture of the HPC cluster limits the maximum throughput of the system because it is centered around a single shared storage volume, which tends to become the bottleneck as the number of computing nodes increases – and this is especially true for sequence processing which tends to perform a lot of I/O with respect to processing activity.

This chapter presents specialized software suite, called *Seal*, that adopts a completely different strategy by processing sequencing data using the Hadoop framework (see Sec. 2.3). It has been updated and extended since the initial versions that were published in previous literature [69, 70]. The following sections motivate the decision to use this framework and present the tools that have been developed.

3.1 Hadoop for processing sequencing data

The Hadoop framework has been designed from the ground up to scale computing throughput up to very high levels while containing costs. Indeed, its origins lie in data companies such as Yahoo! and Google that process much more data than any modern sequencing center. Because of this, its developers have already addressed many of the difficulties that arise when trying to scale computational throughput. Therefore, by adopting this framework, the software that uses it automatically benefits from their experience, rather than repeating the work by developing new ad hoc solutions.

Hadoop addresses the aforementioned issues that can limit the scalability of processing systems on a HPC cluster. It automatically splits input data among all the computing nodes available, thus distributing the work over the cluster. It is robust to hardware failures, automatically taking care to restart tasks that do not complete because of a broken node – without any user intervention. Perhaps most importantly, it implements its own storage system that distributes data over the disks on the computing nodes themselves. As a result, the data can be usually processed directly on the node where it is stored, drastically reducing the amount of data that has to travel over the network. Further, since with this architecture the computing and storage resources increase hand-in-hand, the capacities of both systems scale together with the number of nodes in the cluster, thus raising the level of I/O activity at which storage becomes the process' bottleneck. What is more, it does so while containing the cost per unit of storage volume with respect to the typical costs of a shared parallel file system. Another point worth mentioning is that by adopting Hadoop, one also acquires access to Hadoop-based Platform-as-a-Service (PaaS) offerings – for example, Amazon Elastic MapReduce (EMR) [4] – which entails having access to scalable cloud computing infrastructure without incurring any initial investment cost.

On the other hand, the Hadoop framework also has a number of disadvantages. For one, it is not well suited for the typical HPC cluster, which is implemented around a batch-queuing system. In the HPC scenario, the computing nodes are temporarily assigned to a user upon request and, once the job is finished, then they are supposed to be immediately cleaned of all temporary data and returned to the system, which will consider them available to other users. On the contrary, Hadoop assumes that it has complete and permanent control of its computing nodes. In addition, nodes in an HPC cluster are usually equipped with limited amounts of local storage space, which limits the usefulness of the HDFS. However, the most significant disadvantage is probably the fact that Hadoop is not compatible with conventional software. Therefore, to adopt this framework in a sequencing operation and reap the benefits it offers, one has to find or implement new Hadoop-compatible software to perform the processing steps required by the sequence processing workflow: that is the problem that the work presented here is addressing.

Of course, there exist other approaches to scale the throughput of sequence data processing. The Message Passing Interface (MPI) [24] is a widely used set of interfaces and supporting libraries that provide primitives to implement distributed applications. However, this tool works at a much lower level than the Hadoop framework so its functionality does not include features such as robustness to hardware failure, automatic splitting of input data, or a distributed file system; to create an application with an equivalent feature set a developer would have to address those issues from scratch. Another viable way to speed up computation is to take advantage of hardware "accelerators" such as Graphical Processor Units (GPUs) [42, 53] or Field-Programmable Gate Arrays (FPGAs) [14, 85, 94]. These types of devices are able to achieve extremely high speeds by simultaneously performing many operations in parallel. However, while these accelerators address the problem of computing speed, but do not help to scale I/O throughput. Therefore, they should be seen as complementary to a distributed computing solution such as Hadoop. Indeed, it would be totally reasonable to implement a Hadoop-based software that used hardware accelerators to speed up computing tasks within its map and/or reduce tasks.

3.2 The Seal toolbox

This chapter presents the *Seal* suite of Hadoop-based tools for the primary analysis of sequencing data. These tools are scalable alternatives to conventional software tools. The tools implement the functionality necessary to interpret the raw data from Illumina sequencing runs, execute most of the primary analysis steps in the "best practice" variant calling workflow (see Sec. 2.2), and provide the functionality necessary to convert the data from Seal's storage format to the conventional SAM format so that the output can be fed to conventional tools downstream. Therefore, by using Seal one can run a significant part of the variant calling workflow on a Hadoop-based infrastructure. The following Sections describe in more detail the various components of the tool suite. While the tools described in Sections 3.3 and 3.4 are specifically designed for Illumina-based sequencing, the other tools are generic and can be used with data from any sequencing machine.

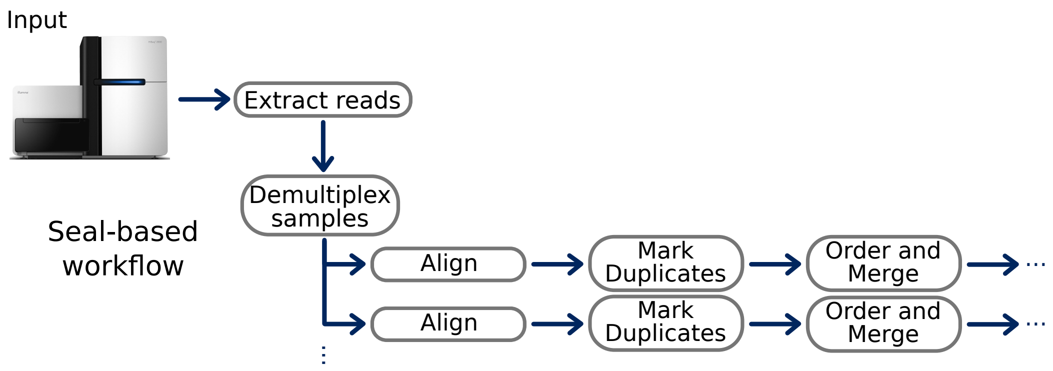


Figure 3.1 – The part of the primary analysis that can be implemented with Seal. The suite includes a distributed tool to directly process an Illumina sequencer's binary output (*BCL* base call files), separate the sequences by sample id (demultiplex), align and mark duplicates. The last tool orders the sequences by position and produces a BAM file that can be used to continue processing with conventional tools.

3.3 Seal: Decoding Data in BCL Format

The first tool of the Seal suite is used to decode the data written by the sequence to generate standard reads: nucleotide sequences, with associated probability of error values for each base and a sequence id. As the sequencer is running, at each cycle it acquires a single base from all the reads by snapping a picture of the flow cell where a chemiluminescent reaction is taking place. Each color detected corresponds to a specific base – **A**denine, **C**ytosine, **G**uanine, or **T**hymine. At each step, the software on the controlling workstation perform base calling from the image data. Performing the base calls on-line, while the sequencer is running, allows the procedure to discard the images almost immediately, saving significant amounts of temporary storage space. Since each image, and corresponding base calls file (BCL), only contains the bases that were acquired from all the reads in that specific sequencing cycle, the individual DNA fragments are actually split over n files, where n is the number cycles in the sequencing run – and therefore n is also equal to the length of the reads.

Thus, the first step to be performed after sequencing is to reconstruct the actual base pair sequence of the DNA fragments. The procedure is illustrated in Fig. 3.2. Seal includes the `bcl2qseq` tool which performs this operation by distributing the work to the entire Hadoop cluster on which it runs. The tool is implemented as a map-only Hadoop job that parallelizes the work on the tiles of the flow cell (one tile per map task). Internally, the program includes a "launcher" component that examines the details of the sequencing run and creates a plan for the work to be done – i.e., number of cycles to decode, location of the files, etc. Finally, the launcher starts the Hadoop job and thus the worker tasks; these drive the `bclToQseq` program provided by Illumina, directing it at the appropriate location and capturing the output so that it can be redirected into the Hadoop framework. The data is written in *qseq* format [84], which is a simple text-based tabular format that is easy to read and process. The code is implemented in Python by taking advantage of the Pydoop library [49] which provides a Python interface for Hadoop.

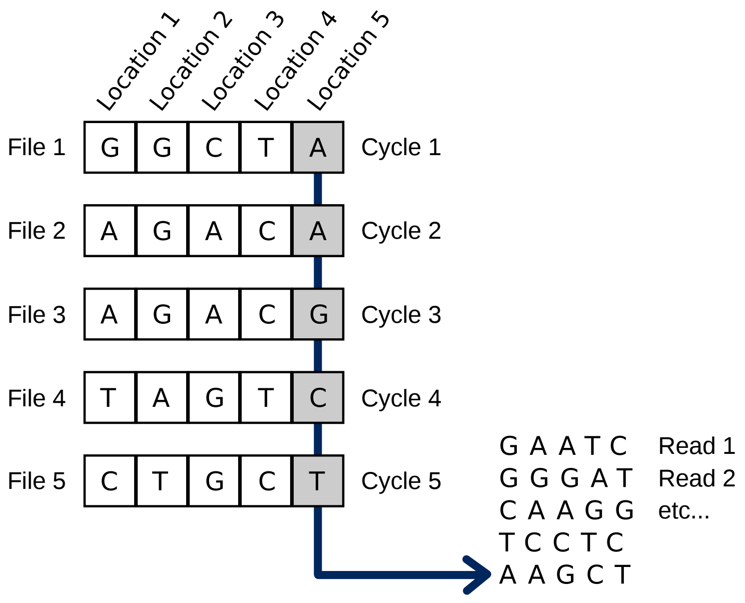


Figure 3.2 – Recomposing nucleotide sequences from the raw data in Illumina BCL files. The operation requires reading data simultaneously from a number of files equal to the number of cycles in the sequencing run – each file contains one base from each read. Once all the bases of a read are acquired, the read can be composed and emitted; then processing can advance to the next one.

3.4 Seal: Demultiplexing

Often, for reasons of efficiency, the DNA fragments from individual samples are tagged with a short identifying nucleotide sequence and then mixed and sequenced with other samples in a single batch. This procedure is known as *multiplexed sequencing*. In this way, the sequencing capacity of the run is divided among more biological samples than would be possible if it was necessary to keep them physically separated. This technique is especially useful for applications that do not require the acquisition of a very high number of bases – exome sequencing is an example. Being that the genetic material of multiple samples is in the same biological solution, the sequencer will acquire their fragments of DNA indiscriminately and they will be output in the same dataset – though the barcode sequence will be decoded as an additional read, so that paired multiplexed sequencing will generate 3 reads per DNA fragment. Separating them requires processing the reads to identify the original sample based on the read's flow cell lane and the nucleotide barcode that was attached in preparation – a procedure known as *demultiplexing*.

Seal includes `demux`, a distributed tool to perform read demultiplexing in a scalable fashion. Demux is implemented as a MapReduce application running on the Hadoop platform. The `demux` algorithm works by grouping all the reads deriving from the same DNA fragment, including the barcode read, into the same reducer instance. Consequently, the reducer is in a position to perform a look-up on the barcode to find the sample to which the reads belong and then write them all to the appropriate dataset.

In more detail, the map phase of the algorithm – which emits key-value pairs – computes a key for each read that is itself a pair formed by: 1) the location on the flow cell from where the read was sequenced, identified by a tuple `(lane, tile, x position, y position)`; 2) the read number. On the other hand, the value of the emitted pair is the read itself along with all its complementary data. While it is true that such a key is unique for any given flow cell, `demux` overrides the default framework behaviour to group only on the location – i.e., the same biological fragment – while ignoring the read number in the key. Further, the key ordering behaviour is also modified as to give precedence to read two – the barcode read. In this way, the reducer receives all a given fragment's reads in this order: 1) barcode read; 2) read 1; 3) if present, read 2. The ordering makes the implementation of the reducer straightforward, since it can decipher where all the reads need to go as soon as it receives the first read in the list. Alternatively, had the ordering not been customized, it would have to cache reads in memory until it found the barcode read.

3.5 Seal: Align Reads and Mark Duplicates

The reference mapping step – also known as *alignment* – is critical to the quality of the results of a resequencing analysis, since it effectively reconstructs, as best as it can, the original genome from the fragments produced by shotgun sequencing. In the "best practice" variant calling workflow (see Sec. 2.2) mapping is immediately followed by duplicate identification. Seal includes a single MapReduce-based tool, called `seqal`, that perform both these operations in a single iteration over the data. The two steps are implemented as a single tool because they fit well in a single MapReduce iteration. The bulk of the tool is implemented in Python, taking advantage of the Pydoop library.

The alignment step is implemented in seqal's Map phase. Rather than implementing a new aligner from scratch, seqal currently indirectly integrates the BWA-MEM aligner through the RAPI read aligner API (developed as part of this project; see Subs. 3.5.3). To integrate BWA-MEM, its C source code was significantly modified to repackage it as a software library, so that its functions and data structures could be called by seqal. Indeed, currently seqal is, to the best of the author's knowledge, the only Hadoop-based read mapping program that directly integrates the aligner core into its code; other options choose to execute conventional read alignment programs in their unmodified form [39, 44, 45] as a separate child process. The technique adopted by seqal, while being more difficult to implement, provides improved efficiency and flexibility.

In addition to transforming BWA-MEM into a software library, the original code was also modified to use memory-mapped I/O to read the reference sequence and its index, instead of simply writing the index components into `malloc`'d memory blocks. This improvement provides two significant advantages over BWA-MEM's original technique. The first is that by using `mmap` all instances of the software using the same reference on the same computer will share the same data in memory, thereby saving significant amounts of memory space. This feature is especially important to seqal's use case since it can, through the Hadoop framework, end up running multiple map tasks on the same computing node and thus risks exhausting the available memory. The second advantage is that after its use the reference is automatically cached in memory for a period of time, so that subsequent invocations of the aligner with the same reference will save the time required to load it from storage (a significant amount of time for large references such as for *Homo sapiens*). While unused, the reference is cached until the memory it occupies is needed for other purposes. Since the BWA-MEM project is open source, these improvements were sent to the author of the original program to be considered for integration into the main line of development.

After alignment, seqal also performs duplicate identification (this is an optional step; seqal can also be used in align-only mode). To identify duplicate reads, seqal uses the same criterion that is used by the popular Picard tool [35]. Duplicate reads are likely to map to the same reference coordinates. This observation can be exploited to identify them. The specific criterion used defines two pairs as duplicates if their alignment coordinates are identical, both for their first and second reads. Likewise, lone reads are considered duplicates if they are aligned to the same position. When a set of duplicate pairs is found, all but the one with the highest average base quality are marked as duplicates. The behaviour is analogous when running in unpaired read mode.

3.5.1 MapReduce algorithm for duplicate identification

To implement duplicate identification criteria used by seqal within the MapReduce paradigm, the following algorithm was designed.

1) First, the mapper aligns the read pair (or single read) that it received.

2) Keys are crafted for each aligned read as the triple

$$(chromosome, position, orientation)$$

where the orientation refers to the forward or reverse genomic strand; this is the information required to define the exact position of alignment. In the key, the chromosome is always represented by its numerical id, through in the output it can be named for the sake of making the alignment data easier to read. The key is then formed by fixed-size fields that are delimited by a colon (':'); the values are left-padded with zeros to reach the required size. See Fig. 3.3 some examples. This design results in keys whose alphabetical ordering is equivalent to their numerical ordering – which is important since Hadoop by default uses alphabetical ordering.

3) The keys generated in step 2 are ordered. Only the "smaller" key is emitted, along with the entire read pair and accompanying alignment information (the value). If working with unpaired reads, then a single key will be emitted with the single read.

4) The reducer, for a given key (alignment position), collects all the read pairs and computes their left- and right-most position – i.e., the start and end positions at which the *entire fragment* from which the reads were sequenced would align to the reference. Pairs that have the same start and end coordinates are considered equivalent. Therefore, all except the one with the highest average base quality are marked as duplicates. The reads are finally output to the final result. Again, for unpaired reads the behaviour is analogous.

Chr	**Pos**	**F/R**		**Key**
2	113423446	F	$\rightarrow$	0002:000113423446:F
13	69112518	R	$\rightarrow$	0013:000069112518:R

Figure 3.3 – Examples of keys formed for MapReduce-based duplicate read identification. The headings stand for *Chromosome* (or, more generally, contig), *Position*, *Forward/Reverse* (as in which strand of the genome). Although the chromosome (or contig) can be named in the alignments file, when building the key it is always represented by a numerical id. The keys are formed by fixed-size fields; the fields are left-padded with zeros to reach the required size. This way, the alphabetical and numerical ordering of the keys is equivalent.

Algorithm 2 Seqal's Map algorithm

```
procedure MAP(mode, read_pair)
    aligned_pair ← align(read_pair)
    if mode = aln-only then
        emit('', aligned_pair)
    else                                          ▷ mode is aln+markdups
        keys ← [ makeKey(a) | a ∈ aligned_pair ]
        ordered_keys ← sort(keys)
        emit(ordered_keys[0], aligned_pair)
    end if
end procedure
```

Algorithm 3 Reducer algorithm to identify duplicates

```
procedure REDUCE(pos, aligned_pairs) ▷ All pairs are at same position
    if |aligned_pairs| > 1 then
        aln_with_score ← [ (a, mean(a.qualities)) | a ∈ aligned_pairs ]
        ordered ← sort(aln_with_score, by=tuple→tuple[1])
        for tuple ∈ ordered[1:] do                    ▷ All but the first
            emit(mark_dup(tuple[0]))
        end for
        aligned_pairs ← ordered[0:1]
    end if
    emit(aligned_pairs[0])
end procedure
```

CPUs	dual quad-core Intel Xeon CPUs @ 2.83 GHz
RAM	16 GB
Disks	one 250-GB SATA hard disk
Network	single 1-Gbit Ethernet connection

Table 3.1 – Hardware configuration of the nodes used to evaluate Seal seqal's performance.

3.5.2 Speed and Scalability

To evaluate seqal's scalability, the tool was tested with datasets of varying input size (DS data sets from Table 3.2) and various cluster sizes (16, 32, 64, and 96 computing nodes). For each cluster size, a Hadoop cluster was set up, including a Hadoop Distributed File System using the nodes' local disk. Before starting the experiments, the input data and a tarball with the indexed reference sequence were copied onto the configured cluster's HDFS. The seqal application was run on all the DS data sets in both align-only and align+mark duplicates modes. The runs were repeated 3 times, with the exception of DS8 which was only run once. The runtime was measured and the throughput computed. The averages of the results are reported in Fig. 3.4.

Looking at Fig. 3.4 we see that seqal is generally capable of throughput/node levels comparable to single-node operation – that is to say that the application and Hadoop keep the distribution overhead to a minimum. As the cluster size increases we would ideally see a constant throughput per node, giving a linear increase in overall throughput. In practice, when the input is too small with respect to the computational capacity, nodes are often underutilized. Therefore, the throughput per node with DS1 at 96 nodes is much lower than the other configurations. On the other hand, we see that seqal is capable of utilizing available resources efficiently when more data is available, although while scaling up from 64 to 96 nodes, the system achieved better throughput on the small DS3 data set as opposed to the larger DS8. This effect might be due to network congestion, which could be alleviated choosing a better-organized network topology.

3.5.3 The RAPI Read Aligner API

In work related to NGS data, read aligners are one of the most fundamental pieces of analysis software, so read alignment is an operation that has been intensely studied [52, 76], resulting in a number of effective algorithms and implementations. Furthermore, many of the aligners are continuously

Data set	No. lanes	No. pairs	Size (GB)	Read length
5M	0	$5.0 \cdot 10^6$	2.3	91
DS1	1	$1.2 \cdot 10^8$	51	100
DS3	3	$3.3 \cdot 10^8$	147	100
DS8	8	$9.2 \cdot 10^8$	406	100

Table 3.2 – Seal seqal evaluation: input data sets. The 5M data set consists of the first 5M pairs from run id ERR020229 of the 1000 Genomes Project [23]. The three DS data sets are from a production sequencing run on an Illumina HiSeq 2000.

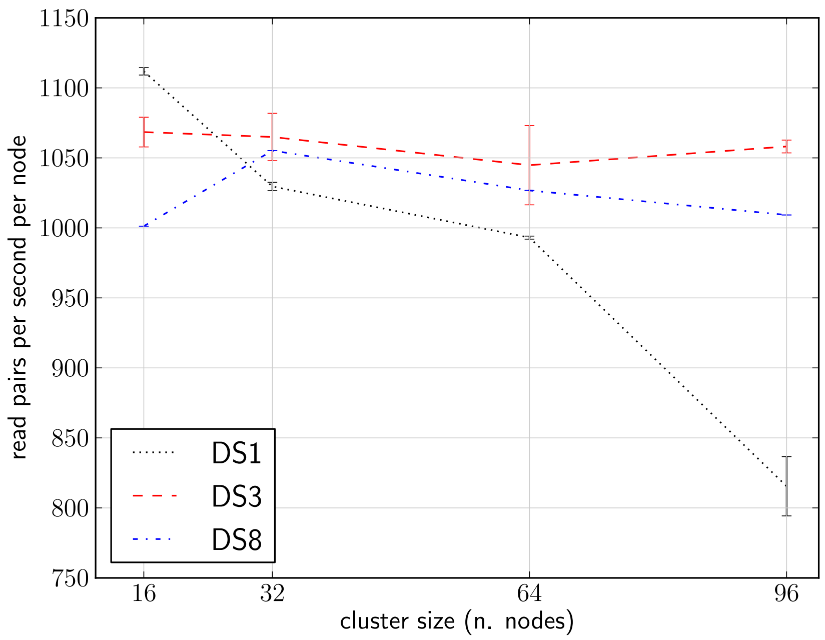


Figure 3.4 – Throughput per node of Seal seqal computing alignments and marking duplicate reads. An ideal system would produce a flat line, scaling perfectly as the cluster size grows. The 3 data sets used are described in Table 3.2. By comparison, a single-node baseline test – performing the same work as Seal seqal but using the standard multi-threaded BWA and Picard programs, on the same hardware – reaches a throughput of approximately 1100 pairs/sec on the 5M data set.

Data set	**BWA time** (h, 1 node)	**Seqal time** (h, 32 nodes)	**Speed-up**
5M	0.49	0.04	12.25x
DS1	11.26*	0.63	17.87x
DS3	32.39*	1.72	18.83x
DS8	89.35*	4.78	18.69x

Table 3.3 – Comparison of running time in hours between BWA on a single node with 8 cores and Seal seqal running on 32 nodes in align-only mode. *) Time is predicted as a linear extrapolation of the throughput observed on the 5M data set.

evolving, as the authors work to improve the quality and the performance of their software. However, while the alignment machinery receives continuous attention, much less is payed to the external interface of the implementations.

Aligners are typically packaged as command-line programs that expect to receive three main arguments: the path to the indexed reference sequence, the path to the input read file in Fastq format, and the path to the output file in SAM or BAM format. This simple interface makes a number of assumptions: that all the required data files (reference, input and output) are accessible on locally mounted file systems; that the data are in the formats that the aligner supports; and that the use case supports executing a new process each time an alignment is required. Although these assumptions may appear to be reasonable, they are actually extremely limiting to work aimed at exploring novel programmatic ways to *use* aligners – in particular, seeking to advance the standard sequencing pipeline's data flow architecture by, for instance, using distributed computing to improve scalability.

A more flexible approach would be to have read aligners define an application programming interface (API) and package their functionality as a library, potentially in addition to the standard command line interface. The experience mentioned in the previous section, patching the BWA-MEM aligner, teaches that this approach is very time-consuming and difficult to execute correctly (the code that needs to be modified may not be suitably structured or documented). More importantly, such work is rapidly obsoleted by the rapidly evolving nature of popular aligners.

As a more general solution, the development of Seal has resulted in the definition of a read aligner API that can be implemented with any underlying read mapping technology: the *Read Aligner Application Programming Interface (RAPI)*. For maximum compatibility the RAPI is defined in C. It includes generic functions and data structures to support typical alignment operations: index a reference sequence, load and unload the reference, map reads to the reference, interpret the results. Further, RAPI includes Python bindings, making it simple to load an aligner as a Python module and use it in unconventional ways – e.g., for scripting or in an interactive session. The project includes a reference aligner interface implementation that wraps the BWA-MEM [50] aligner. This aligner plug-in is used in the Seal project to compute sequence alignments on the Hadoop distributed computing platform. The fact that RAPI does not make any assumptions about the source or destination of the data makes it possible to easily integrate unconventional scalable computing and data storage technology, such as the Hadoop, into an alignment workflow. It makes it equally feasible to transparently implement aligner plug-ins based on GPGPU or FPGA accelerators. Also, since RAPI does not make any assumptions about the data formats, it also facilitates research into alternative data structures for persistent storage. Finally, since RAPI standardizes the aligner interface, one could parametrize the aligner to use within a RAPI-based pipeline, swapping aligner without changing any of the code.

RAPI is being proposed to the community as an option to standardize the read aligner interface. A standard interface would open up new use cases, reducing maintenance for existing applications, and make it simple and safe to harness aligner plug-ins to prototype and create novel functionality. The interface has been released under an open source license[1].

3.6 Data storage formats

Seal can use several file formats to read and store data, depending on the user's preference. It can work with both text or binary formats. The text formats are the standard ones used in conventional bioinformatics: qseq [84] and fastq [18] for unaligned data and SAM [51] for aligned data. For the former two, Seal relies on implementations from the Hadoop-BAM project [60]; for the latter, it Seal includes its own implementation. Further, Seal can transparently compress or decompress these data formats with standard codecs.

On the other hand, the binary formats available in Seal use the Apache Parquet [7] columnar storage format that has emerged from work at data-intensive companies [58]. Unlike more conventional file formats, which store data record-wise, Parquet groups the data by field – i.e., it writes all values for `field1`, then all the values for `field2`, etc. This strategy is especially advantageous for analyses that do not require all the fields (since the unnecessary fields do not have to be read from disk) and in terms of data compression (storing multiple values of the same field results in lower entropy and thus

[1] `https://github.com/ilveroluca/rapi`

better compression rates). Incidentally, the binary data formats supported by Seal are also compatible with ADAM [55], ensuring that the tools from the two projects can be used together.

3.7 Related work

Usage of "BigData" tools for bioinformatics applications started in late 2008 with work on Hadoop [26, 56] and has since continued to increase [91]. Although the model has been successfully applied to a number of "classic" applications, such as BLAST and GSEA [48, 56], the area of most advancement is deep sequencing data analysis, where many Hadoop-based tools have been developed [63]; tools such as Crossbow [44] for SNP discovery, and Myrna [45] for RNA-Seq [95] differential expression analysis were pioneers in this area.

Several Hadoop-based sequencing alignment tools have already been published. Crossbow [44] (based on the Bowtie [43] aligner, which it executes internally) and Cloudburst [81] were two of the first. The alignment algorithms used by these programs has been superseded by others, including the BWA-MEM algorithm [50] which is used by Seal. Another Hadoop-based program integrating BWA-MEM has recently been published [1]; its authors even claim that it is slightly faster than Seal's aligner. Alas, it only aligns sequences and does not provide any other functionality to help scale up a sequence processing workflow – unlike Seal which offers a suite of tools.

There are also other Hadoop-based tools for the analysis of sequencing data. For instance, Quake [41] is a tool to detect and correct sequencing errors; SAMQA [75] is a tool for quality assurance of sequencing data; Contrail [80] runs on Hadoop to perform *de novo* assembly. However, to the best of the author's knowledge there are no Hadoop-based tools for duplicate sequence identification; the state of the art are conventional tools such as Samtools [51] and Picard [35].

Also, there is a MapReduce-based tool that does not run on Hadoop. The Genome Analysis Toolkit (GATK) [57] is a MapReduce-like programming framework that provides a set of data access patterns commonly used by sequencing data analysis programs. GATK defines a programming contract where components are organized into data producers, or traversals, and data consumers, or walkers (i.e., analysis modules). It also includes a set of tools that are built upon its own framework – many of which are useful to implement the "best practice" variant calling workflow (see Sec. 2.2). However, it uses MapReduce only as a programming design pattern, not as a distributed computing model. While it does include it's own custom module to perform distributed computing, it is not compatible with the Hadoop ecosystem.

Finally, it is worth mentioning that there is a promising project called ADAM [55] that is developing tools for the analysis of sequencing data based on "Big Data" technology. ADAM runs on the Spark framework [99] rather than Hadoop MapReduce. However, Spark and MapReduce are compatible, meaning that they can run on the same cluster and access data on the same HDFS volume (Spark is part of the Hadoop "ecosystem"). Since ADAM is focussing on later steps in the variant calling workflow than the tools included in Seal, the two projects currently complement each other and together come close to implementing an entire variant calling workflow on the Hadoop platform.

3.8 Conclusion

This chapter has described the problem encountered by medium- and large-sized sequencing facilities in ensuring their analysis procedures can sustain a throughput sufficient to keep up with the data generation rates of their sequencing machines. The Hadoop platform provides an ideal foundation for software that needs to scale to quickly process large quantities of data – its success in data-based companies such as Twitter, Facebook and Amazon is a testament to this [29]. The Seal suite of tools leverages the scalability of the Hadoop platform to bring its advantages to the analysis of sequencing data. Its tools can be used to process the raw output of Illumina sequencers and perform most of the primary processing in the "best practices" variant calling algorithm. To integrate with conventional bioinformatics tools, the last step in a Seal-based analysis procedure can use the Hadoop cluster itself to reformat the data into a file that is compatible with legacy software.

Chapter 4

Automated and traceble processing of sequencing data

While the scalable computing techniques described in Chapter 3 strive to address the issue of analyzing the data at an equal of faster rate than it is generated, they do not help to address the problem of handling the ever larger study sizes – in terms of number of samples – that are enabled by NGS [54, 93]. In this Chapter, presents a strategy to confront this issue by automating all routine processing and adopting advanced data management techniques.

The work in this Chapter has been recently included another publication [20]; for the reader of this dissertation, the contents of that article are presented in the following sections after being distilled to include only the contributions to which the author contributed directly and in a significant manner.

4.1 Introduction

The data-intensive revolution [54, 93] in the life sciences is being driven by the increasing diffusion of massively parallel data acquisition system, next-generation sequencing (NGS) machines [67] being among the most cited examples. One of the main challenges brought forth by this phenomenon is to develop scalable computing tools that can keep up with such a massive data generation throughput [44, 70, 71]. However, efficient data processing is only part of the problem: additional issues include dealing with highly structured data where individual components are connected by multiple relationships, keeping track of data provenance, managing complex processing workflows, minimizing operational costs and providing simple access interfaces to nontechnical users.

This Chapter describes the author's experience in contributing to the development of a fully automated infrastructure for the analysis of DNA sequencing data produced by the CRS4 NGS facility — currently the largest in Italy by throughput, number of samples processed and amount of data generated (see Sec. 2.1. The system, which was introduced to the production pipeline in July 2013, has reduced the amount of human resources required to process the data from four to one full-time individual. The infrastructure has been built by composing open source tools — many written at CRS4 — with new purpose-built software which will also be contributed to the open source community.

The remainder of this Chapter is structured as follows. In Section 4.2, it gives an overview of the system architecture and its components, following with a discussion on overall system performance in Section 4.3. Section 4.4 delineates the related work in this area, after which we conclude and describe future work in Section 4.5.

4.2 General System Architecture

The data production rate of the NGS laboratory presents a significant challenge with respect to operator effort, data management complexity and processing throughput. The system developed at CRS4 can efficiently and autonomously perform the standard primary processing of the data produced by the NGS lab, thus preparing it for further ad-hoc analysis by bioinformaticians or for shipping to external collaborators. The system achieves scalability via three main features. The first is the automatic execution and monitoring of standard operations, which reduces the human effort required to process the data, thus lowering the occurrence of errors and allowing the analysis to scale to large numbers of datasets. The second is the handling of provenance information on all datasets, which allows to reconstruct their history up to the original raw data: this is crucial to effectively manage large data collections, allowing to quickly query data interdependencies and facilitate integration between multiple studies. Finally, the system is designed for high processing throughput, which is a strict requirement given the growing volumes of data produced by modern data-intensive acquisition technologies.

Fig. 4.1 summarizes the overall system architecture: computational engines are the core analysis tools that process raw data to yield the final results; OMERO.biobank handles metadata storage and provenance tracking; iRODS [37, 73] acts as a single point of access for all datasets; the workflow manager takes care of composing and executing the various steps

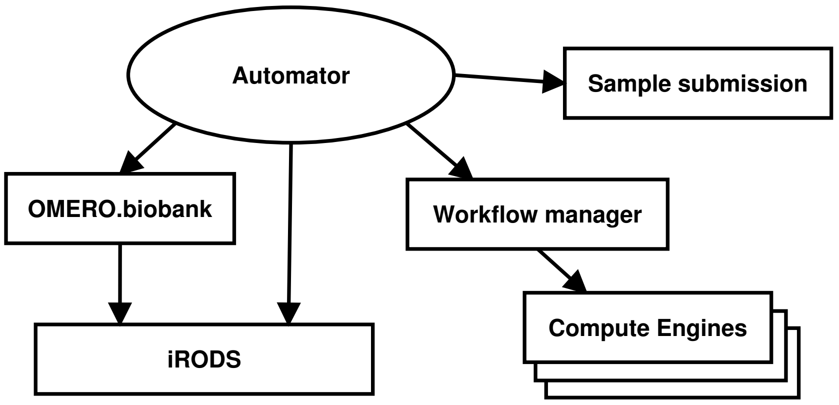


Figure 4.1 – Overall architecture of the automation and processing system used at CRS4. The Automator is programmed to orchestrate operations and control the other components, which have specific duties pertaining to data processing, storage, metadata maintenance and interaction with the users.

that make up each analysis pipeline; the sample submission system allows researchers to provide detailed specifications on input samples and request specific processing; finally, the automator performs global orchestration of other components in order to minimize human intervention and increase reliability. The components relevant to the author's work are described in more detail in dedicated subsections.

4.2.1 Automator

The automator performs standard processing on data produced by the laboratory, including format conversion, demultiplexing, quality control and preparation for archival or shipment. When appropriate, further sample-specific workflows are also run. The system is based on a reactive, event-driven design. For example, the activity diagram in Fig. 4.2 shows what happens when a sequencing run is completed: an event announces that the run is finished; the automator reacts by executing the appropriate handler, which registers the new datasets with the iRODS [73] catalogue and with OMERO.biobank (see Sec. 4.2.2). The system's kernel is implemented by an event queue built with RabbitMQ [72]; clients can add new events to the queue to notify the system that something has occurred: for instance, a periodic check adds an event when new data is ready for processing; one or more daemons monitor the queue and execute appropriate actions for each event. The design allows multiple instances of the automator to run concurrently, thus making the system more robust to node failures and other technical problems.

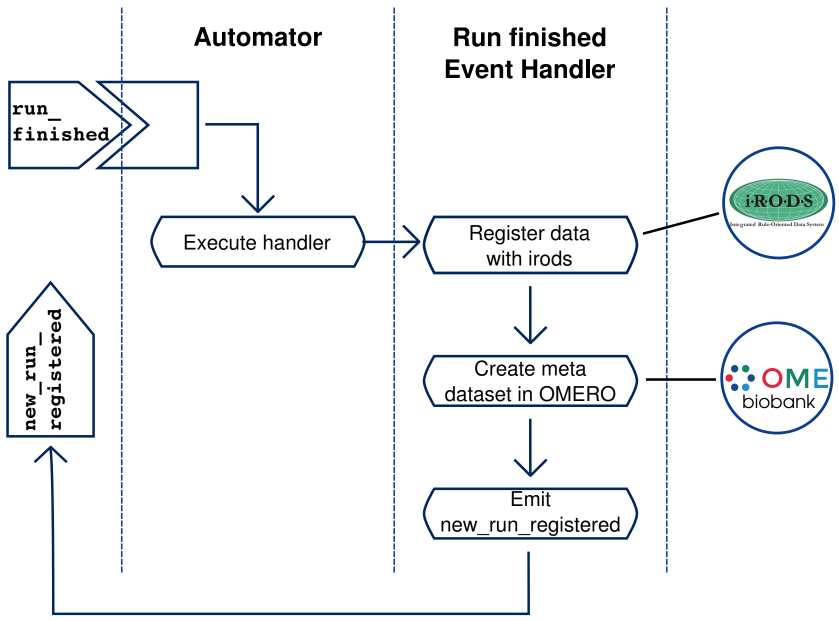


Figure 4.2 – Activity diagram illustrating the registration process for a new sequencing run.

In addition to the event-dispatching kernel, the automator consists of a number of purpose-built event handlers that are specific to the process implemented at CRS4, and a software library to communicate programmatically with the other components. In fact, the automator does not execute operations directly on the data; instead, these are grouped into workflows that are defined and executed through the workflow manager (see Sec. 4.2.3). The automator *monitors* the execution of these workflows and, when they complete, registers new datasets in OMERO.biobank along with a detailed description of the operations that generated them, thus ensuring reproducibility. The automator's role in the overall architecture is therefore that of a middleware layer that serves to drive the automation, integrate the various components and execute specialized site-specific operations.

4.2.2 OMERO.biobank

OMERO.biobank is a robust, extensible and scalable traceability framework developed to support large-scale experiments in data-intensive biology. The data management system is built on top of the core services of OME Remote Objects (OMERO) [2], an open source software platform that includes a number of storage mechanisms, remoting middleware, an API and client applications.

At its core, OMERO.biobank's data model (see Fig. 4.3) consists of entities (e.g., biological samples, analysis results) connected by *actions* that keep track of provenance information. The system is designed to avoid strong bindings with respect to static data flow patterns: each entity is only aware of the action from which it derives, and vice versa. Additional information is provided by *devices*, which are linked to actions and hold all the details required to describe hardware components (e.g., an NGS machine), software programs or whole pipelines involved in the data generation process. An example of pipelines mapped as devices is given by Galaxy workflows, which can be easily manipulated with the BioBlend.objects package [47].

OMERO.biobank's kernel is complemented by an indexing system that maintains a persistent version of the traceability structure by mapping entities to nodes and actions to edges in a graph database. The system is able to manage a large number of items: at CRS4, the latest sampling counted over 130000 entities linked by over 190000 actions.

4.2.3 Workflow Manager

The continuously increasing size of the data produced in the life sciences has led to a progressive intensification of the effort required for their analysis. Large and diverse datasets must be processed by workflows consisting of many steps, each with its own configuration parameters. In addition, the entire analysis process should be transparent and reproducible, and the analysis frameworks usable and cost-effective for biomedical researchers. Since keeping track of all information associated with complex pipelines can be very time consuming and error prone, easy-to-use data processing platforms that can automate at least part of the process are highly sought-after.

Our workflow management system has been specifically designed to address the above challenges. It is based on Galaxy [27], an open platform for biomedical data analysis that provides a standard way to encapsulate computational tools and datasets in a graphical user interface (GUI), together with a mechanism to keep track of execution history. The BioBlend.objects API is used to programmatically interact with the Galaxy service.

4.2.4 iRODS

NGS platforms generate a significant amount of data split over a large number of files and datasets. In addition, frequent collaborations among geographically dispersed entities introduce a requirement for fast and controlled remote data access. To simplify this process, CRS4 adopted iRODS as a front-end to its large-scale heterogeneous storage system (about 4.5 PB distributed over

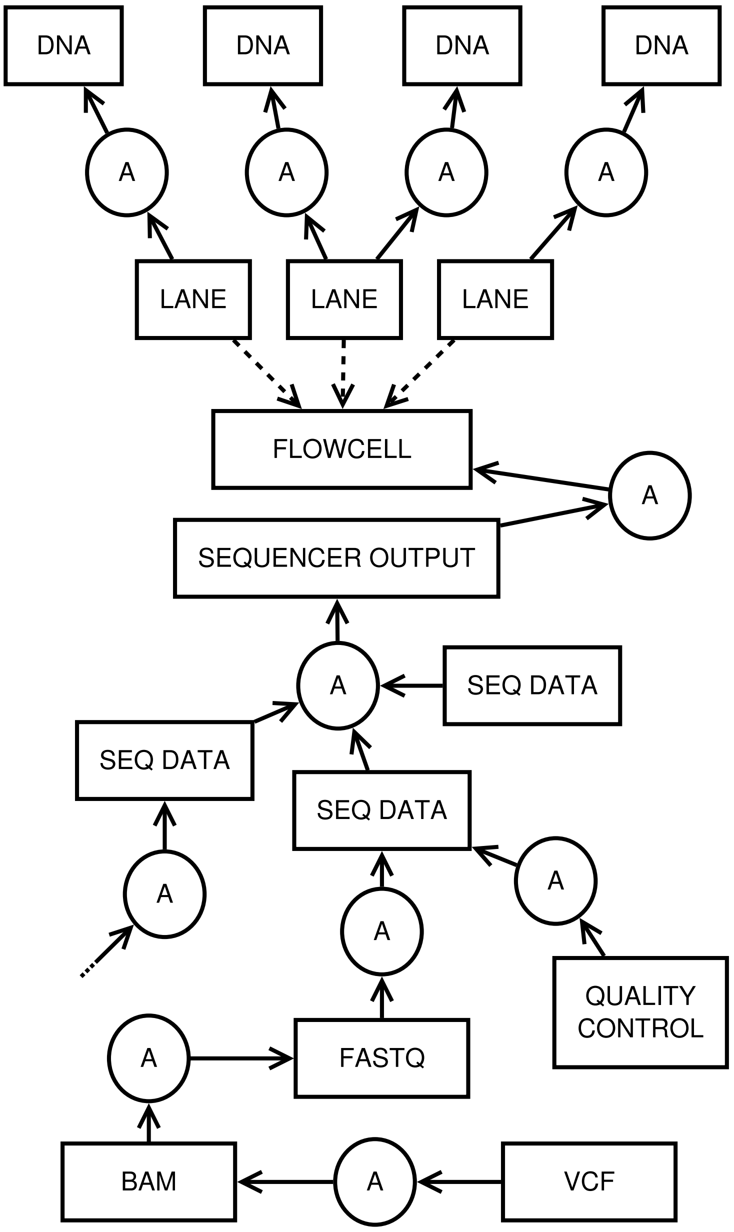


Figure 4.3 – Traceability graph for an exome processing workflow stored within OMERO.biobank. Rectangles represent entities, while circles represent actions.

various volumes). The service stores and maintains sequencing datasets in a way that allows users to safely access and manage them through a variety of clients, such as web browsers and command line interfaces. It relies on general-purpose file systems to store data and on SQL databases for metadata. Designed to scale to millions of files and petabytes of data, iRODS is a key component of our infrastructure, providing a single point of access to datasets that may be distributed across a number of disjoint storage systems.

4.2.5 Compute Engines

For its computational requirements, the operation maily relies on the Seal suite of tools described in Chapter 3. It is the central tool for the analysis of high-throughput sequencing data at CRS4. Seal's main goal is to remove the processing bottlenecks presented by conventional tools by providing a suite of scalable, distributed applications to perform common time-consuming sequence processing operations.

To simplify their use and incorporate them into the process automation mechanisms, the Seal tools have been integrated into Galaxy, thus allowing their usage as workflow components; the integration mechanism is described in Chapter 5. Incidentally, the toolbox has also been independently integrated into other high-level workflow tools such as Cloudgene [82].

Of course, other conventional tools are used as well. Moreover, for ad hoc analysis of large quantities of sequencing data the tool of choice is often SeqPig (see Ch. 6).

4.2.6 Hadoocca

While the Hadoop platform is a strong vector for computational scalability [97], it imposes some requirements on the underlying computational infrastructure that are not compatible with the established resource allocation patterns used on HPC clusters, like the one at CRS4. Namely, Hadoop has its own mechanisms for job submission, queueing, and scheduling that conflict with HPC batch scheduling systems. In addition, Hadoop needs to run daemons and store data locally on the nodes it uses, essentially assuming their exclusive and long-term allocation.

Thus, to support Hadoop at CRS4, a strategy was devised to make both scheduling paradigms co-exist in an efficient and manageable manner. The work entailed the creation of a dynamic Hadoop-node allocation system that seamlessly integrates with the existing HPC infrastructure (based on Open Grid Scheduler). The system occupies resources on-demand (see Fig. 4.4), improving node utilization over static allocation approaches without breaking

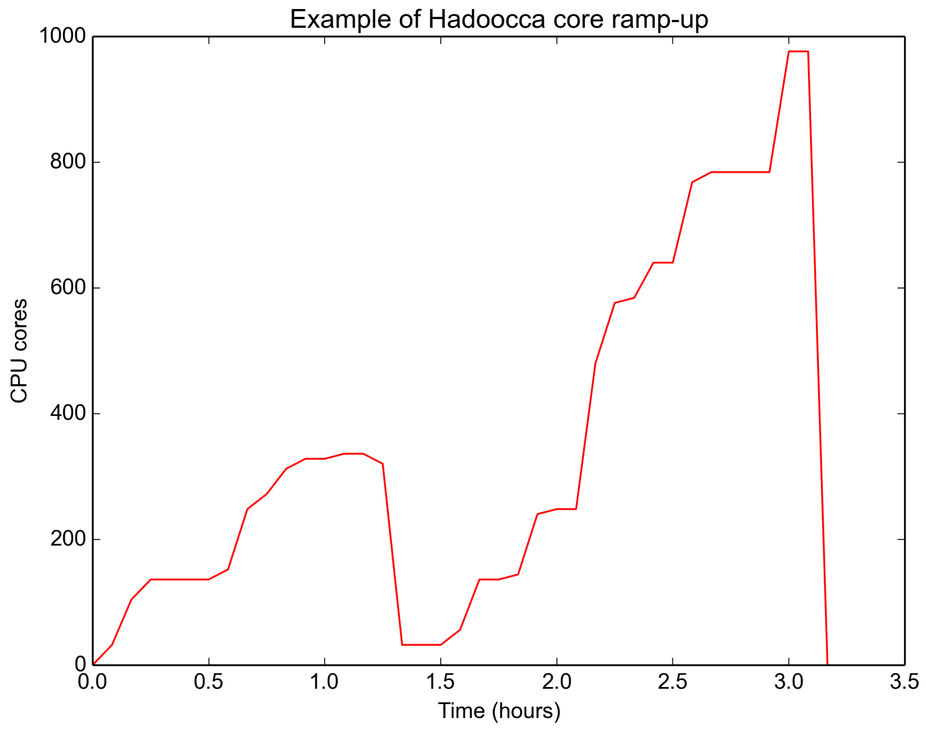


Figure 4.4 – Progression of CPU core assignment by Hadoocca during the execution of a workflow. The assigned core count varies adapting to load and all machines are automatically released once the analysis is concluded.

existing scheduling policies. Therefore, it provides a low-cost and low-risk path to testing and adopting Hadoop which effectively allowed our HPC center to set up a Hadoop cluster with minimal investment, albeit with some trade-offs. Specifically, CRS4's set up – affectionately named *Hadoocca* – foregoes the Hadoop Distributed File System (HDFS) and instead relies on the cluster's shared parallel file system. Thus, though the approach does not benefit from the advantages of HDFS [77], it allows to run HPC and Hadoop jobs at the same time in the same computing environment. The system is being used in production at CRS4 to run computational biology pipelines and other workloads on a 3200-core HPC cluster that is shared with other jobs.

4.3 Production Capabilities

With the introduction of the framework described in this Chapter, CRS4 has been able to scale its operations while containing research costs. Specif-

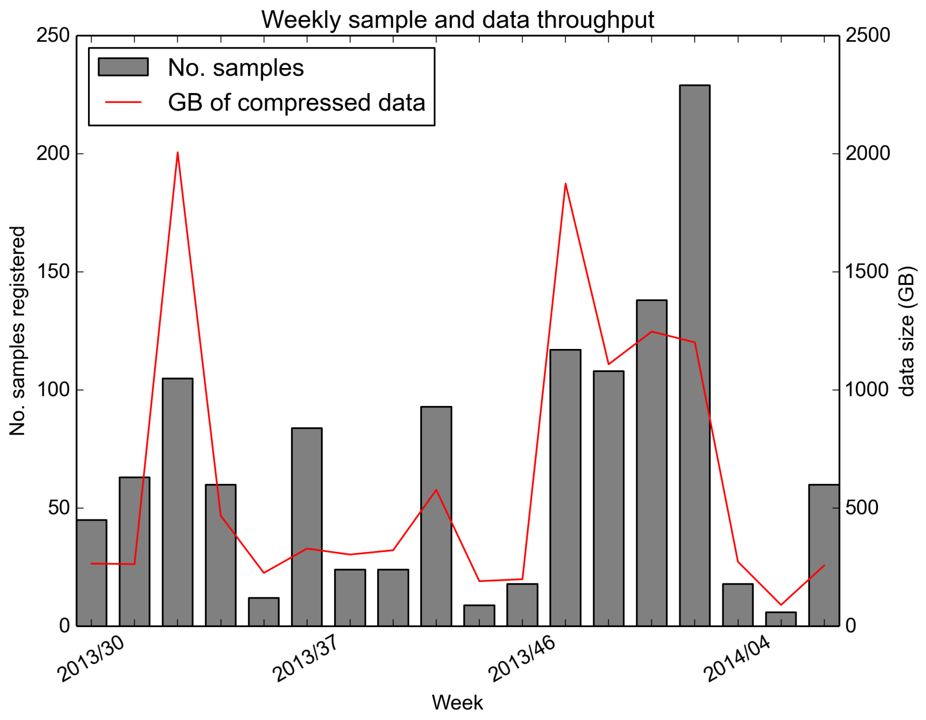


Figure 4.5 – Weekly throughput of operations at the CRS4 sequencing facility. For the period from the last week of July 2013 to the first week of February 2014, the graph shows the number of samples processed each week and the corresponding volume of gzip-compressed sequence data generated.

ically, the number of full-time individuals required to operate the processing went down from approximately three to less than one, freeing resources for downstream, research-specific analysis. Its adoption has also enforced complete digital tracking of all analysis operations and datasets. In addition to ensuring reproducibility, this feature provides an important source of information for the quantitative monitoring, evaluation and management of the facility. In addition, the automation system, together with the high-throughput distributed computing applications, has allowed the center to reach its throughput targets. Fig. 4.5 shows the number of samples processed and the corresponding amount of gzip-compressed sequence data generated week-by-week for the extended period after the framework went into production. The system has coped with peak loads of over 200 samples per week and about 2 TB of compressed data (approx. six flow cells, or 20 TB of raw input data) per week. This rate is already sufficient to handle the capacity of CRS4's sequencing facility, but the system could probably scale to higher numbers. Fig. 4.6 shows the rate of data production since system start-up.

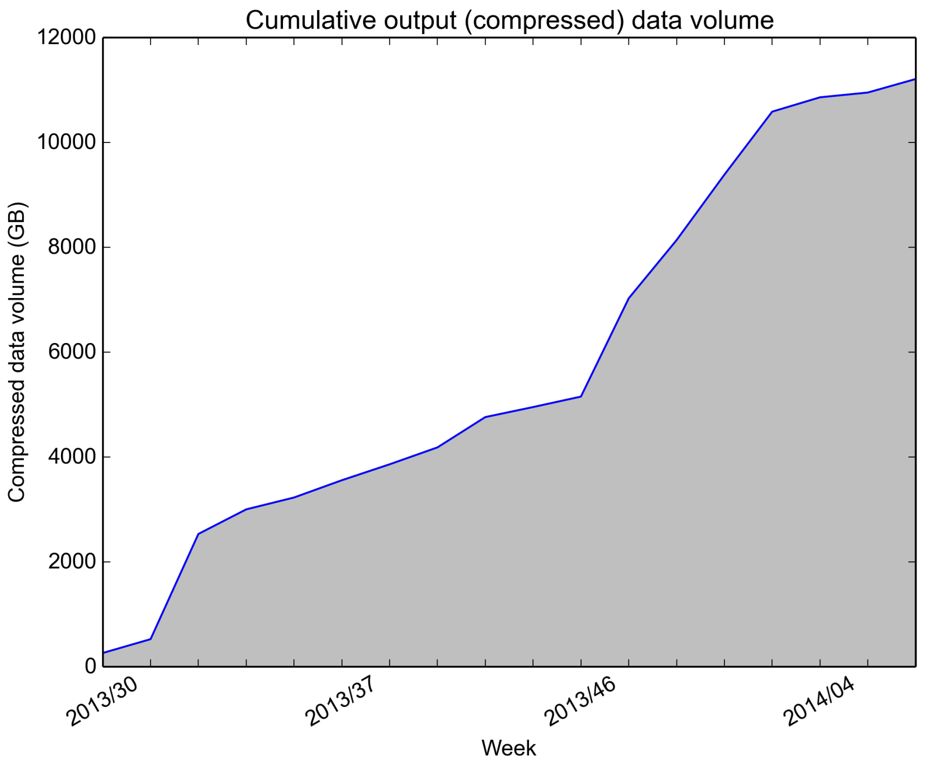


Figure 4.6 – Growth of compressed output data accumulated at the CRS4 sequencing facility since July 2013, when the automated system fully entered production use, to February 2014.

4.4 Related Work

The development of a comprehensive data infrastructure for the management and analysis of NGS data has been pursued extensively in different contexts and with varying goals in mind. However, automatically piloting and monitoring standard operations as well as ensuring reproducibility and traceability of analysis are issues that have been less comprehensively addressed.

Previous work has been carried out at The Genome Analysis Centre (TGAC), an institute in the UK that conducts research in genomics and bioinformatics. Their work [46] is primarily focused on the initial analysis of sequencing data and provides a number of tools, packages and pipelines to ascertain, store, and expose quality metrics. The computed quality metrics and contamination screening analyses are stored using a flexible MySQL database and API – useful for storing any run metric or metadata. Furthermore, an iRODS layer is provided, through which data can be annotated with descriptive metadata enabling consolidated searching and discovery of grouped datasets. In principle, the combination of these tools offers the potential to provide richer contexts for downstream analysis. There are, however, a number of issues that have not been addressed. For example, there is no support for automated selection of the processing pipeline based on the nature of the sequencing project. Furthermore, the lack of integration with an analysis platform such as Galaxy hinders the possibility of automatically and rapidly exposing sequence data for downstream analysis.

Since 2010, iRODS has been running as a production data management system at the Wellcome Trust Sanger Institute (WTSI), one of the world's major sequencing centres. The WTSI uses iRODS as an archive system [17]. Currently, WTSI users are mainly using iRODS for managing and accessing sequencing Binary Alignment/Map (BAM) files for further analysis and research. Moreover, the WTSI uses iRODS to manage user-defined metadata related to BAM files, whereas more advanced uses (e.g., metadata queries, and management of experimental output for further analysis) are currently under investigation on various internal testbeds. In addition to the WTSI, iRODS has also been used in several other large-scale biological and biomedical initiatives and institutes, including the Broad Institute, the Genome Biology Unit at the University of Helsinki, and the National Center for Microscopy and Imaging Research (NCMIR) at UCSD.

iPlant is a collaborative 5-year, NSF-funded effort to develop a cyberinfrastructure to address a series of grand challenges in plant science based on iRODS. Interestingly, the iPlant data infrastructure [38] is designed to support preservation of the experimental provenance of data and of the computational transformations applied to them, providing support for rerunning a workflow using the same data from reference databases or for reproducing experiments and the processing done on resulting experimental data.

The UPPNEX initiative provides high-performance computing resources and large-scale storage together with a software infrastructure for NGS research in Sweden [21]. Currently managing about 300 projects concurrently, UPPNEX is being used by three sequencing platforms, each with their own data delivery workflow; research groups may then analyze their data using the installed software or with custom pipelines. UPPNEX uses iRODS to facilitate moving data between different types of storage resources and to share resources with other domains. Currently, most of the installed software at UPPNEX is only available via command line interface, and limited support is provided for workflow management systems such as Galaxy. In addition to that, there is no evidence on how provenance information is treated and if support for reproducibility and traceability is offered to the users.

4.5 Conclusions

This chapter has described a fully automated infrastructure to support the analysis of the data produced by the NGS facility at CRS4. The system, in production since July 2013, integrates open source tools – either internally developed or publicly available – into a framework that can autonomously handle the primary analysis process and support downstream analysis. The automation middleware, which is built around a distributed event queue, drives a workflow manager and executes custom housekeeping tasks. The system has successfully handled peak weekly data production periods of over 200 samples and 2 TB of compressed sequence data. As reusable components become available, they will be released to the community as open source software.

Chapter 5

Integrating Hadoop-based processing with Galaxy

The work described in Chapters 3 and 4, respectively on scalable software and automation, while both contributing important solutions they were actually born incompatible with each other for technical reasons. This issue is important since on their own neither contribution provides a complete solution for a center processing sequencing data.

The work presented in this Chapter solves this incompatibility through a special Galaxy data type and adapter program. The solution was published in an article [68] which is summarized in this Chapter for the reader of this dissertation.

5.1 Introduction

Rapid progress in biological and biomedical data acquisition technologies is turning modern biology into a data-intensive science [54, 93]. This can be mostly attributed to the increasing diffusion of massively parallel data acquisition systems, next-generation sequencing (NGS) machines [67] being among the most cited examples. The introduction of NGS has allowed researchers to explore important, but previously inaccessible, biological questions and has paved the way to a host of significant new techniques and protocols that have a wide range of important applications in biology and medicine [88]. However, to use NGS data one has to surmount two main, interrelated, difficulties: processing complexity and dataset size. In fact, the extraction of biologically significant information from the raw sequencing data requires complex, multi-step, computationally intensive processing workflows. In its bare form, this processing is not trivial and requires specialized computing skills, such as familiarity with shell programming and high-performance computing (HPC), and knowledge about the layout of the local storage infrastructure. These issues are typically not of high interest to biologists, since they are system-level obstacles that only need to be defeated in order to achieve the final objectives of their work.

The desire to simplify such analyses from the user's point of view has historically motivated the creation of a number of graphical bioinformatics platforms – some examples are Galaxy [27], Taverna [64] and Chipster [40] – which allow the system-level details tied to implementing such processing workflows to be hidden below a high-level graphical interface. This type of simplification is particularly beneficial for Hadoop-based programs since, being "unconventional", may be perceived to be difficult to use by users not accustomed to its model of operation. In this manner, the biologists can express the workflow they would like to run by graphically connecting operations. Galaxy in particular has been garnering a growing level of popularity among biologists in recent years, as can been seen by the growing number of related papers and of public installations[1]. Indeed, CRS4 has also adopted Galaxy for their DNA and RNA sequence processing operations.

This high-level graphical user interface isolates the users from the technical details required to execute the computation. For instance, it allows system administrators to configure the system to transparently use computational resources and methods appropriate for the tasks at hand. More specifically, such installations can be configured to run jobs through standard HPC batch-queue systems, thus accessing available computing cluster nodes transparently, where they may also have access to high-performance or particularly large parallel shared storage systems.

For many types of biology-related computation this arrangement is sufficient. However, NGS poses particularly challenging data throughput requirements. Consider that a modern sequencing machine can generate about 8TB of raw data per week which needs to be processed and stored, so even medium-size sequencing operations can face significant operational and infrastructural challenges. The need for scalable computing solutions in this sector has prompted work in applying "BigData" technologies, especially Hadoop [30, 77], to sequencing-related operations [63, 70, 82, 83]. These new Hadoop-based tools are designed to be scalable both in the amount of data to be processed and in the use of available computing resources, meaning that in general one can simply add more computing nodes to achieve higher throughput. Unfortunately, these new Hadoop-based tools are not di-

[1]For an up-to-date list of public Galaxy installations and related publications see `https://wiki.galaxyproject.org/PublicGalaxyServers`

rectly compatible with Galaxy and thus cannot be natively used through its user-friendly interface for reasons discussed in more detail in Section 5.2.1.

This work addresses the incompatibilities between these two systems to provide a functional and extensible integration layer, allowing users to mix and match Hadoop-based and conventional tools in their Galaxy workflows. The solution is simple to use, but requires the workflow designer to keep in mind when the data is being processed in the "conventional space" and when in the "Hadoop space" – something which is done through explicit operations that are inserted in the workflow. This Hadoop-Galaxy adapter has been released as open source software[2].

The rest of this Chapter is structured as follows. Section 5.2.1 describes the incompatibilities between these two systems and thus explains why this Hadoop-Galaxy adaptor is required; the rest of Section 5.2 describes how the adaptor works and how to use it. Section 5.3 then explains how to use the Hadoop-Galaxy adaptor to integrate new Hadoop-based tools with any Galaxy installation, while Section 5.4 follows with the details of a concrete use case developed at CRS4. The Chapter then describes related work in Section 5.5 and finally concludes. For background information, the reader can refer to Chapter 2.

5.2 Galaxy-Hadoop Integration

5.2.1 Incompatibilities

The execution models used by Galaxy and Hadoop are incompatible, meaning that even Hadoop-based tools that are invoked from the command-line cannot be easily integrated into a Galaxy workflow. There are two main issues that keep this integration from working, both pertaining to the two platforms' view of datasets. The first is that the two systems store datasets differently. Specifically, Galaxy assumes that a dataset is entirely contained in a single file, while Hadoop MapReduce purposefully splits datasets into multiple files – possibly bundled into a single directory – to be more amenable to processing in multiple parallel tasks. Because of this incompatibility, one cannot present the output of a Hadoop job to Galaxy as a single dataset. Though at the time of this writing work is being done by the Galaxy developers to resolve this limitation, in the way of multi-file datasets, it is still in the early stages of development (see Pull Request #386 in the `galaxy-central` reposi-

[2]Hadoop-Galaxy is available at `https://github.com/crs4/hadoop-galaxy/` and in the Galaxy Tool Shed: `http://toolshed.g2.bx.psu.edu/view/crs4/hadoop_galaxy`.

tory for a complete description: `http://goo.gl/UVuRq7`[3]). The second issue relates to the file system where the data is stored and the fact that Galaxy expects to be able to access it directly. This issue is a problem since Galaxy can only access conventional file systems that are mounted on the machine where the server runs. On the other hand, Hadoop clusters are typically configured to use the HDFS for storage, which cannot be readily accessed by Galaxy. In fact, HDFS was not designed to be mounted on a system and is instead accessed through its own client application programming interface (API) or web service. Although some methods exist to mount HDFS like a conventional file system – e.g., NFS gateway to HDFS or mounting HDFS through Linux Filesystem in Userspace (FUSE) – using them would entail putting the entire Galaxy workspace on HDFS and these solutions would not be suitable to handle intense file system activity. A similar argument can be made for Amazon S3 storage, which is also accessible transparently to Hadoop programs but not Galaxy.

5.2.2 Solution by Indirection: Pathsets

Due to the incompatibilities between Hadoop's and Galaxy's tool models, Galaxy cannot use Hadoop-based tools directly. However, as David Wheeler once said, "All problems in computer science can be solved by another level of indirection" [98]. Following his advice, this work introduces the *pathset* data type. In brief, a pathset is a list of one or more complete paths that define a single dataset. Concretely, it is an ordered list of uniform resource identifiers (URIs), each identifying one or more files or directories, augmented by the ability to use the shell-like wildcards. The dataset defined by the pathset can be materialized by concatenating the contents of all the files spanned by the pathset: therefore, the order in which the URIs are listed is important. When a URI references a directory the files in the entire directory hierarchy are included in the pathset as if by recursive traversal. When selecting paths through patterns or recursion, the elements are included in alphabetical order. In addition to the paths themselves, the pathset includes some minimal metadata, such as the format of the data being referenced.

The pathset is implemented as a Python class which, in addition to basic operations such as adding, removing and iterating over URIs, implements a text-based serialization. An example of the serialization format is shown in Fig. 5.1). Such a file can be readily handled as a dataset by Galaxy, for which a `pathset` data type is defined to ensure that only pathset-compatible tools

[3]`https://bitbucket.org/galaxy/galaxy-central/pull-request/386/dataset-collections-initial-models-api/`

can be connected to each other.

By executing Hadoop-based tools in Galaxy through an adapter, we are able to use the pathsets – which are single-file entities – to reference the real data which may be located on file systems that can be referenced through a URI and split between any number of files and directories.

```
# Pathset       Version:0.0     DataType:Unknown
hdfs://hdfs.crs4.int:9000/data/sample-cs3813-fastq-r1
hdfs://hdfs.crs4.int:9000/data/sample-cs3813-fastq-r2
```

Figure 5.1 – Example of a pathset file. The header begins with a sequence to identify the file type, the version of the pathset format, and then contains metadata describing the format of the referenced data.

5.2.3 Hadoop-Galaxy Adapter

Although the pathset datatype provides a way to indirectly reference data that is too large to be easily placed in Galaxy's workspace or that is located on file systems that cannot be mounted, on its own it is insufficient to allow Galaxy to use Hadoop-based tools in its workflows. To complete the solution, Hadoop-Galaxy includes an adapter program – the *Hadoop-Galaxy adapter* – that can "dereference" the input and output pathsets and pass the contents to the Hadoop-based program that needs to run.

The Hadoop-Galaxy adapter can work as an executable or as a Python function called by another program. To use it, one specifies: the input pathset, referencing the input data; the output pathset, which will reference the output data after execution; a location where the Hadoop program can write its output data, typically on HDFS; finally, the name or path of the executable of the Hadoop-based tool and any additional arguments it needs to operate. The adapter will stage the data output directories as needed and write their path to the output pathset. It will then read the input paths from the input pathset and then call the executable, passing the real data input and output paths as arguments.

A configuration file can be provided either by command-line argument or, more conveniently for Galaxy-related applications, by setting an environment variable which could be used to simultaneously affect all Hadoop-based tools integrated through our adapter. Through this configuration file the user can vary several operational aspects, including which Hadoop cluster to use and the environment variables to set when the Hadoop-based tools are run. The job's output data path is decided autonomously by the adapter based on the configuration provided by the user. This setting is quite important when operating at scale because this integration technique, at the moment, does not provide garbage collection for the data referenced by the pathsets. Indeed, a notable shortcoming of this system in its current form is that when a pathset dataset is deleted by Galaxy, the data it referenced remains in existence and occupies disk space. To alleviate this situation, the user can configure a directory where all Hadoop-based tools write, knowing they can schedule occasional clean-up sessions to delete old intermediate data. This strategy is being successfully used at CRS4 for its production sequencing pipeline, which uses Galaxy and Hadoop for many of its processing stages.

5.2.4 Utility Programs

In addition to the adapter itself, Hadoop-Galaxy provides a number of utility tools to facilitate the integration of Hadoop-based and regular tools in the same workflow. They are briefly described in the following paragraphs.

`make_pathset`: a tool to create a new pathset that references files or directories provided as input. The input can be provided by connecting the output of another Galaxy tool, thus providing a bridge to take a "regular" Galaxy dataset and pass it to a Hadoop-based tool. Alternatively, the input can be given as a direct parameter, which can be useful, for instance, for creating workflows where the user specifies the input path as an argument or when performing *ad hoc* analysis with Galaxy.

`cat_paths`: a tool to take the list of part files that make up a Hadoop dataset and concatenate them into a single file. This tool is effectively the inverse operation of `make_pathset`: while `make_pathset` creates a level of indirection by writing a new pathset that references data files, `cat_paths` copies the referenced data into a single file. The new destination file exists within Galaxy's workspace and can therefore be used by other standard Galaxy tools. Thus, with the combination of `make_pathset` and `cat_paths`, a single Galaxy workflow can mix processing with standard tools and Hadoop-based operations.

However, since Hadoop-based processing can generate a lot of data, copying it to the Galaxy workspace one chunk at a time in a serial process can be very time-consuming. To ameliorate this issue, `cat_paths` provides a distributed mode that uses the entire Hadoop cluster to copy data chunks to the same file in parallel. For this feature to work, the Galaxy workspace must be on a parallel shared file system accessible by all the Hadoop nodes. When applicable, its effects can be significant: with datasets on CRS4's computing cluster speed-ups of up to 40x have been observed.

`put_dataset`: a tool to copy data from the Galaxy workspace to Hadoop storage. It is possible that in some configurations the Galaxy workspace may be stored on a volume that is directly accessible by the Hadoop cluster. In this case, before feeding the data as input to a Hadoop-based program it is necessary to copy it from Galaxy's storage to the Hadoop-accessible storage. For this task Hadoop-Galaxy provides the `put_dataset` tool, which works in a way analogous to the command

```
    hadoop dfs -put <local file> <remote file>
```

The `put_dataset` operation receives a pathset as input and creates a new one as output, where for each input path element there is a corresponding output path for the file at its new location. It is important to keep in mind that this copy operation is serial in nature, as it needs to run directly on the server that has access to the Galaxy workspace. As such, it provides limited bandwidth and can be time-consuming for large files. If large files need to be passed between the Galaxy and Hadoop workspaces, it is a much faster solution to have the Galaxy workspace on a parallel shared file system that can be accessed directly by the Hadoop cluster, thus eliminating the need for `put_dataset`.

`split_pathset`: a tool that can be used in a Galaxy workflow to split a pathset into two parts based on path or filename. It allows the user to define a regular expression as a test: the path elements are then accordingly placed in the "match" or "mismatch" output pathset. This type of tool is handy in cases when the Hadoop tool associates a meaning to the output file. For instance, some tools in the Seal suite for processing sequencing data [70] can separate DNA fragments based on whether they were produced in the first or second sequencing phase (i.e., read 1 or read 2) putting them each under a separate directory. This tool allows a workflow to split the Galaxy dataset into two sets according to this criteria.

`dist_text_zipper`: a tool for parallel (Hadoop-based) compression of text files. Although this tool is not required for the use of Hadoop-Galaxy in user workflows, it is a generally useful utility that doubles as an example illustrating how to integrate Hadoop-based tools with Galaxy using our adapter.

5.3 Integrating new tools

The components of the Hadoop-Galaxy project are designed to simplify the integration of Hadoop-based tools with Galaxy. The `dist_text_zipper` utility, distributed with Hadoop-Galaxy, provides a clear example. The Hadoop-based program has a regular command-line interface, receiving input path(s) and an output path – both of which must be accessible by the Hadoop nodes

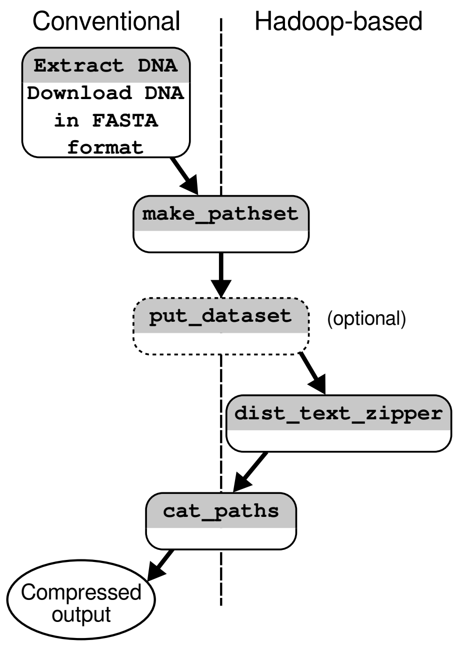


Figure 5.2 – A Galaxy workflow that fetches a dataset (DNA sequences) in text format, using a standard Galaxy tool (*Fetch Sequences → Extract Genomic DNA*), and uses Hadoop-Galaxy to compress them using a Hadoop cluster. The `dist_text_zipper` program (run over the Hadoop-Galaxy adapter) is used to compress the data, while the `make_pathset` and `cat_paths` utilities are used to provide the appropriate data bridge between the Galaxy workspace and the Hadoop file system. The optional `put_dataset` component is only needed if the Hadoop cluster cannot access the Galaxy workspace directly.

and may be on an unconventional file system such as HDFS. To use the program through Galaxy, it is sufficient to write a normal tool definition in XML; but, rather than invoking the program directly, it should be called through the `hadoop_galaxy` adapter (illustrated in Fig. 5.3). No modifications to the program are necessary. Also, passing additional arguments to the tool is simple since the adapter forwards any arguments that it does not recognize. In its standard form, the adapter assumes that the slave program can be called in the form:

```
cmd_name [options] input output
```

To accommodate programs with a different interface (for instance, a common case may be programs that identify the desired output path with a `-o` option) the Hadoop-Galaxy adapter supports user-provided pluggable command runners that can implement the required interface, though they need to be written in Python.

```
<tool id="hg_dtz" name="Dist TextZipper">
<description>Compress lots of text</description>
<command>
        hadoop_galaxy
        --input $input
        --output $output
        --executable dist_text_zipper
</command>
<inputs>
        <param name="input" type="data" format="pathset"/>
</inputs>
<outputs>
        <data name="output" format="pathset" />
</outputs>
<stdio>
        <exit_code range="1:" level="fatal" />
</stdio>
</tool>
```

Figure 5.3 – Example of a Hadoop-based tool (`dist_text_zipper`) integrated as a Galaxy tool. Notice how the tool is called through the `hadoop_galaxy` adapter, rather than being invoked directly.

For additional examples showing how to use Hadoop-Galaxy to integrate Hadoop-based tools with Galaxy, developers can look at the adaptors for the Seal suite of tools for processing DNA sequencing data (`https://github.com/crs4/seal-galaxy`). These tools and wrappers are used to process the data produced by CRS4's three Illumina HiSeq 2000 high-throughput sequencers.

5.4 Sample use case

Recently at CRS4 Hadoop-Galaxy was used to develop a custom bioinformatics workflow for the analysis of viral vector integration sites (ISs) to assess the safety of a novel hematopoietic stem cell gene therapy (HSC-GT) treatment for metachromatic leukodystrophy (MLD) [11]. The patient is infused with HSCs modified by viral vectors, enabling them to express a particular enzyme whose absence leads to the condition. However, despite its efficacy in treating the disease, GT can cause serious side effects: changes in the genomic area close to the vector's IS can trigger the expression of harmful genes, a phenomenon known as *insertional mutagenesis*. Thus, the ability to predict and monitor the genomic distribution of viral vectors is crucial to the safety of the procedure. IS identification starts in the wet lab, where host DNA is amplified via polymerase chain reaction (PCR) and then sequenced, which produces a considerable amount of plain text data (sequencing reads). The rest of the analysis is done *in silico*: bioinformatics tools are needed to remove viral and artificial fragments from the reads, align them to a reference genome, apply a series of filters to ensure unambiguous mapping, merge equivalent ISs and annotate them with nearby genomic features (e.g., genes). The custom pipeline developed at CRS4, called VISPA (`https://github.com/crs4/vispa`) takes care of the computational part of the analysis, from raw sequencing data to annotated ISs.

While most of the analysis steps could be implemented efficiently as ordinary single-core programs, the alignment and filtering step proved to be significantly more challenging. During the experiments carried out in the course of the MLD study, nearly 14 million sequences had to be aligned to the human reference genome with BLAST [3]: on a single processor, such a task can take more than a year to complete, an amount of time which is incompatible with the turnaround requirements of clinical trials. Although BLAST can optionally use multiple CPUs, in practice this leads to minor performance increases, since only a small part of the code is multi-threaded. Thus, to achieve significant speedups, the set of input sequences needs to be split into multiple files, each of which must be fed to a separate BLAST process. Since Hadoop already provides this functionality in a manner suitable for scaling up to a large number of CPU cores distributed across different machines, it provides the ideal framework on which to reimplemented the alignment and filtering steps. Figure 5.5 compares the time required to align and filter 2048 64-base long sequences to the human genome (hg19 assembly) for varying Hadoop cluster sizes. Input sequences were randomly generated as sub-samples of chromosome 22, with a mismatch rate of one base every two sequences. As shown in the figure, the running time can be reduced by adding more nodes to the cluster: in this case, due to the small size of the test dataset, saturation is approached rather quickly as the overhead introduced by the framework becomes comparable with the core computation; in real-world applications, substantial gains can be obtained up to much larger cluster sizes.

Despite the obvious advantages of the distributed implementation, its execution model does not allow its immediate integration into the main Galaxy workflow because of the incompatibilities described in Sec. 5.2.1. Figure 5.4 depicts how the Hadoop-Galaxy adapter was used to integrate the tools without modifying the MapReduce code of the program.

The left flow corresponds to a "standard" Galaxy implementation: trimmed sequences output by the previous step are passed to the BLAST tool [19], while a subsequent filtering step classifies them as unambiguously aligned, not mapped or repeats (i.e., with multiple high-score mappings) and writes them

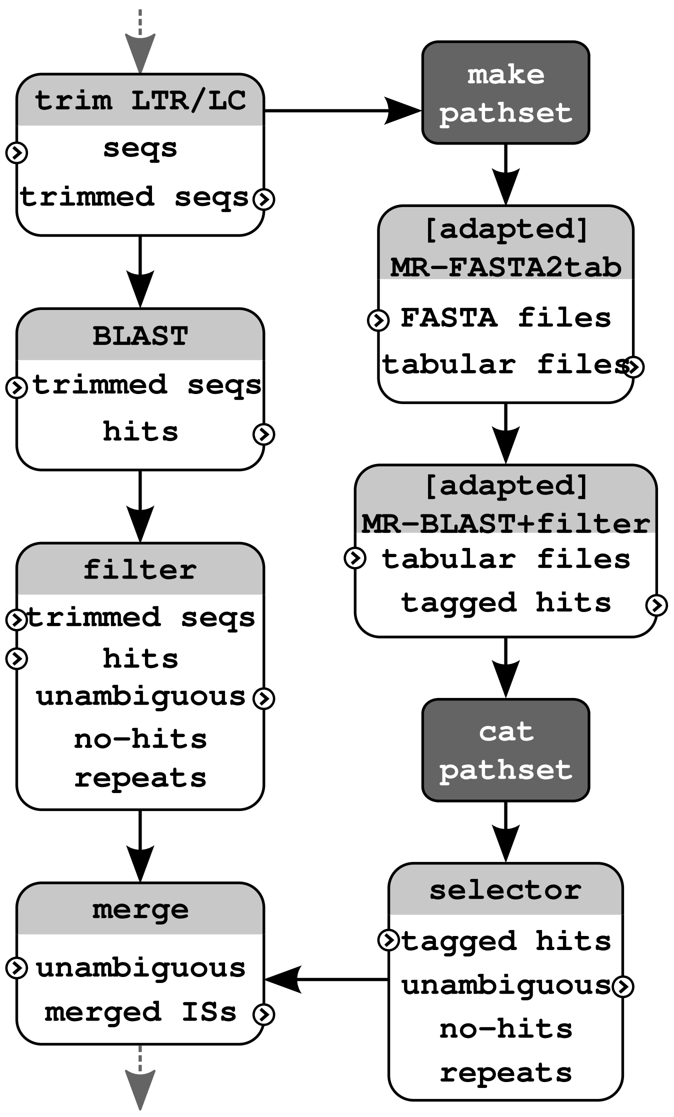


Figure 5.4 – The BLAST and filter section of VISPA's Galaxy workflow, implemented with ordinary single-core tools (left flow) and with MapReduce applications (right flow). Hadoop-Galaxy adapters and tools allow to integrate the latter into the Galaxy workflow unchanged.

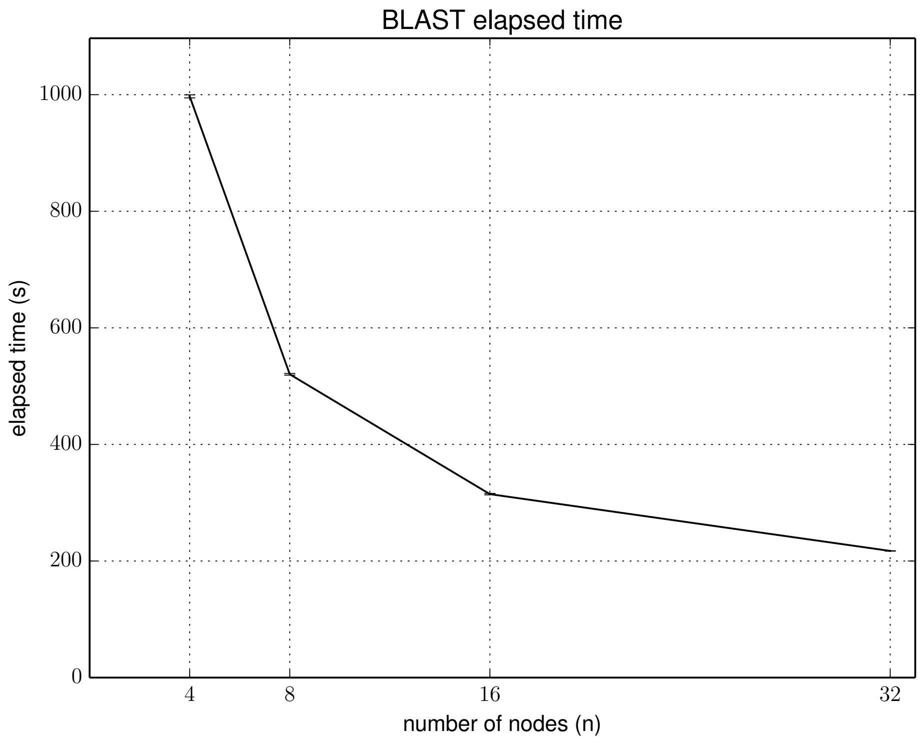


Figure 5.5 – Time required to align and filter 2048 64-base sequences, for different cluster sizes, with our distributed version of BLAST (average over three repeated runs; used BLAST version 2.2.21). Four CPU cores were used on each node. The suboptimal speedup is related to implementation details of the MapReduce application and is not tied to the use of the Hadoop-Galaxy adapter.

to three separate files. Unambiguously aligned sequences are then passed to the IS merging step, while the other two sets are kept for statistical purposes. Note that the filtering step needs the trimmed sequences to compute their length.

The right flow shows the MapReduce implementation of the BLAST and filtering steps, as well as their integration into Galaxy. Since a one-record-per-line format greatly simplifies Hadoop I/O, it is useful to have a separate normalization step do the conversion from FASTA to a tabular format which can subsequently be used by multiple sequence processing application; on the other hand, since the filtering step performs the same operations on the hits corresponding to each sequence, it is more efficient to merge it with the alignment step, so that it can take advantage of the already distributed workload. Since Hadoop writes a separate output file for each concurrent task, classification is done by adding a tag to each hit, so that it can easily be dispatched to the correct output file by a subsequent selector step; due to its relatively light workload, the latter does not need to be parallelized.

The first step to integrate the MapReduce tools into Galaxy consists of adding a `make_pathset` tool after the trimmer, which builds a pathset containing the path of the trimmed sequences file; the fasta-to-tabular converter and the alignment and filtering tools are wrapped via the `hadoop_galaxy` adapter (see Fig. 5.3), so that they can operate on pathsets; finally, `cat_paths` concatenates all files output by the alignment and filtering step, producing an ordinary dataset that can be passed as input to the selector.

5.5 Related work

The integration of Hadoop computational back-ends in Galaxy is a recurring theme in the Galaxy mailing lists. However, at the best of our knowledge, the only published work on the subject is a paper of Chen et al. [15] on a cloud-based image processing toolbox obtained by integrating Galaxy, Hadoop and their proprietary image processing tools. Their motivation was to provide users with a toolbox that would simplify the design and execution of complex image processing tasks using scalable cloud computation capacity. The solution proposed, based on the wrapping of their image processing programs as galaxy programs, is, however, very specialized to their application and does not consider the general problem of integrating Hadoop-based tools in standard Galaxy workflows.

5.6 Conclusions and Future Work

This chapter has presented a strategy and a functional package for easily integrating Hadoop-based applications within the Galaxy bioinformatics platform. The integration is based on the idea of introducing a layer of indirection which is handled through an adapter program, and it has been shown to work through the description of a concrete use case where a significant workflow speed-up was obtained without sacrificing user-friendliness.

Though the Hadoop-Galaxy integration presented in this work is already functional, there are several ways in which it could be improved. The pathset data type needs to be integrated with the Galaxy garbage collector, thus eliminating the problem of having to manually delete data when it is no longer useful. Other improvements will go in the direction of a tighter functional integration with Galaxy, which will require a richer implementation of the pathset datatype within Galaxy and will provide new features such as support for viewing excerpts of the data directly from within Galaxy (the user currently sees excerpts of the serialized pathset file) and tighter type checking, thus only allowing workflow connections between operations and pathsets that reference compatible data formats.

The Hadoop-Galaxy source code is available at `https://github.com/crs4/hadoop-galaxy`. The software is also available through the Galaxy Tool Shed [12], so it can be easily added to a custom Galaxy installation. Once installed, the new "pathset" datatype and Hadoop-Galaxy tools are available allowing the user to build workflows integrating both regular and Hadoop-based tools.

Chapter 6

Simple, scalable, interactive processing

When presenting scalable applications, such as ones based on Hadoop, the focus is usually on improving scalability and speed; rarely is any attention payed to the users of these systems, whose work is often complicated by the need to use more sophisticated software and computing infrastructure. This chapter describes a tool whose goal is to make writing and running Hadoop-based tools for bioinformatics simpler for users by integrating bioinforamtics-specific operations with a Hadoop-based scripting language called *Pig Latin* [65]. This work is a result of a collaboration with Dr. André Schumacher, with whom an article has been already published [83]. This chapter summarizes its text.

6.1 Introduction

Novel computational approaches are required to cope with the increasing data volumes of large-scale sequencing projects, since the growth in processing power and storage access speed is unable to keep pace with them [54, 90]. Several innovative tools and technologies have been proposed to tackle these challenges [60, 62, 70, 75, 82]. Some are based on Hadoop MapReduce [22], which is a distributed computing paradigm that has been designed for processing collections of relatively independent data items and is therefore well suited for processing sequencing data. It is based on the idea of splitting input data into chunks which can be processed largely independently (via a *Map* function). Subresults can later be merged after grouping related subresults (by a *Reduce* function). By projecting the work exclusively into these two phases, MapReduce eliminates unstructured interactions between processes and permits automatic parallelization and scalable data distribution across many computing nodes. In addition, processes have a single, well defined way to emit output. Since the output can be controlled, it becomes simple to automatically restart processes in case of failures. Finally, and perhaps most importantly, the model creates a clear separation between the application's data-processing logic and the code required to distribute the computation. The latter can then be packaged into a well-tuned, general, reusable distributed computation framework. The most popular implementation available as open source software is Apache Hadoop [30], which also comes with its own scalable distributed filesystem. The validity of Hadoop as a general data processing platform is demonstrated by the level of adoption in major data-intensive companies – e.g., Twitter, Facebook and Amazon.

Motivated by the potential scalability and throughput offered by Hadoop, there are an increasing number of Hadoop-based tools for processing sequencing data [63, 91]. These range from quality control [75] and alignment [70, 81] to SNP calling [44] and variant detection [62, 96], including also general purpose workflow management [82].

While Hadoop does simplify writing scalable, distributed software, it does not make it trivial. Such a task still requires a specialized skill set and a significant amount of work, particularly if the solution involves sequences of MapReduce jobs, multiplying the implementation effort required to achieve scalability and introducing a level of complexity in tying these jobs together into a reliable workflow. To alleviate these problems, Apache Pig was introduced [65]. Pig implements an SQL-like scripting language – called *Pig Latin* – which is automatically translated into a sequence of Hadoop MapReduce jobs. Given its flexibility and simplicity, it is not surprising that a large fraction of the computing jobs in contemporary Hadoop cluster deployments originate from Apache Pig or similar high-level tools [16].

SeqPig brings the benefits provided by Apache Pig to sequencing data analysis by extending Pig with a number of specialized features and functionalities. Specifically, it provides: 1) data input and output components, 2) functions to access fields and to transform data and 3) a collection of scripts for frequent tasks (e.g., pile-up, QC statistics).

6.2 Background

Pig Latin defines relational operators much like the ones provided by SQL (e.g., JOIN, GROUP, ORDER BY) and built-in aggregation and transformation functions (e.g., numeric functions such as AVG, COUNT, MIN, MAX, and string functions like INDEXOF, LOWER, REPLACE, REGEX_EXTRACT, etc). Its basic data types are *relations*, *bags*, *tuples* and *fields*. Pig Latin statements operate on relations, which are so-called *outer* bags that are not contained in any other bag. Each bag is essentially a set of tuples. Typically, it is not safe to assume any order of the tuples that are contained in the bag, although an order can be enforced by operations such as ORDER BY. A tuple is an ordered set of fields, which are elementary data types such as integers, strings, etc. Finally, a relation is a bag that is not contained in any other bag. Upon execution, Pig Latin scripts are automatically translated into a sequence of Hadoop MapReduce jobs by the Pig interpreter; the jobs are then executed to compute the results.

6.3 Implementation

Apache Pig provides an extension mechanism through the definition of new library functions, implemented in one of several supported programming languages (Java, Python, Ruby, JavaScript); these functions can then be called from Pig scripts. SeqPig uses this feature to augment the set of operators provided by plain Pig with a number of custom sequencing-specific functions.

SeqPig supports ad hoc (scripted and interactive) distributed manipulation and analysis of large sequencing datasets so that processing speed scales with the number of available computing nodes. It provides import and export functions for file formats commonly used for sequencing data: Fastq, Qseq, FASTA, SAM and BAM. These components, implemented with the help of Hadoop-BAM [60], allow the user to load and export sequencing data in the Pig environment. All available fields, such as BAM/SAM optional read attributes for example, can then be accessed and modified from within Pig. Also, read metadata is appropriately recognized so it can then be directly used in operations – for instance, to employ a Hadoop cluster to calculate the mean insert size of a data set. The Illumina-extended Fastq format is supported, so that reads from both Fastq and Qseq files are loaded with their associated flow cell coordinates, index reads, base quality scores and so on.

SeqPig also includes functions to access SAM flags, split reads by base (for computing base-level statistics), reverse-complement reads, calculate read reference positions in a mapping (for pileups, extracting SNP positions), and more. It comes packaged with scripts that calculate various statistics and manipulations on read data, which also serve as examples. The growing library of functions and scripts is documented in the SeqPig manual.

The following Figures show a number of example scripts.

```
reads = LOAD 'in.qseq' USING QseqLoader();
STORE reads INTO 'out.fq' USING FastqStorer();
```

Figure 6.1 – Converting Qseq into Fastq; the data set is simply read and then written using the appropriate load/store functions. Notably, these two lines of code result in a Hadoop job that takes advantage of all available computing nodes in the cluster.

```
F = load 'in.fastq' using FastqLoader();
top = FILTER F BY tile >= 1000 AND tile < 2000;
sp = FOREACH top GENERATE UnalignedReadSplit(sequence, quality);
bases = FOREACH sp GENERATE FLATTEN($0);
first_10 = FILTER bases BY pos <= 10;
grpd_by_pos = GROUP first_10 BY pos;
result = FOREACH grpd_by_pos GENERATE group AS pos, AVG($1.basequal);
sorted_cycle_avg = ORDER result BY pos;
DUMP sorted_cycle_avg
```

Figure 6.2 – Calculating per-cycle average base quality for the first 10 bases for reads on top surface of an Illumina flowcell. The operation filters by tile to select the desired set of reads. The remaining reads are then split into *(base, quality, position)* tuples. All positions greater than 10 are eliminated; the remaining tuples are grouped by position and their base qualities aggregated into an average. The result is finally ordered by position and stored.

```
reads = LOAD 'in.fq' USING FastqLoader();
read_bases = FOREACH reads GENERATE UnalignedReadSplit(sequence, quality);
read_gc = FOREACH read_bases {
  only_gc = FILTER $0 BY readbase == 'G' OR readbase == 'C';
  GENERATE COUNT(only_gc) AS count;
}
gc_counts = FOREACH (GROUP read_gc BY count) GENERATE group AS gc_cnt,
    COUNT($1) AS cnt;
DUMP gc_counts;
```

Figure 6.3 – Script to calculate a frequency table for the GC content of the input reads. The reads split into bases, which are then filtered to keep only the 'G' and 'C' nucleotides. Then, for each read these left-over bases are counted and, finally, the counts are grouped. Finally, the GC-count and the size of each group (i.e., the frequency) is emitted.

6.4 Evaluation

To evaluate SeqPig, a script was written which calculates most of the read quality statistics that are collected by the popular FastQC tool [5]. We ran a set of experiments which measured the speed-up gained by using SeqPig on Hadoop clusters of different sizes compared to using a single-node FastQC run. We used a set of Illumina reads as input (read length: 101 bases; file size: 61.4 GB; format: Fastq). Software versions were as follows: FastQC 0.10.1; Hadoop 1.0.4; Pig 0.11.1. All tests were run on nodes equipped with dual quad-core Intel Xeon CPUs @ 2.83 GHz, 16 GB of RAM and one 250 GB SATA disk available to Hadoop. Nodes are connected via Gigabit Ethernet. FastQC reads its data from a high-performance shared parallel file system by DDN. SeqPig used the Hadoop Distributed File System (HDFS) which uses each node's local disk drive.

We first ran five different SeqPig read statistics for a different number of computing nodes: the sample distribution of a) the average base quality of the reads; b) the length of reads; c) bases by position inside the reads; d) the GC contents of reads. Finally, we combined them into a single script. Each of the executions results in a single MapReduce job and thus a single scan through the data. All runs were repeated three times and averaged (deviation from average $< 7\%$). From Figure 6.4 one can see that it is possible to achieve a significant speed-up by exploiting the parallelism in read and base statistics computation using Hadoop. Further, the total runtime of the script that computes all statistics is mostly determined by the slowest of the individual ones, since the complete script is compiled into a single Map-only job. A different observation is that for most of the statistics computed we are able to achieve a close to linear speedup compared to FastQC until 48 nodes. We assume that the levelling off is due to the Hadoop job overhead eventually dominating over speedup due to parallelization, depending on input file size.

6.5 Conclusion

SeqPig enables simple and scalable manipulation and analysis of sequencing data on the Hadoop platform. At CRS4 SeqPig is used routinely for ad hoc investigations into data quality issues, comparison of alignment tools, and reformatting and packaging data. SeqPig has also been tested on Amazon's Elastic MapReduce service [4], where users may rent computing time on the cloud to run their SeqPig scripts and even share their S3 storage buckets with other cloud-enabled software. The experiments show that it scales well with many operations, and thus provides an accessible way for bioinformaticians to take advantage of the Hadoop framework's scalability.

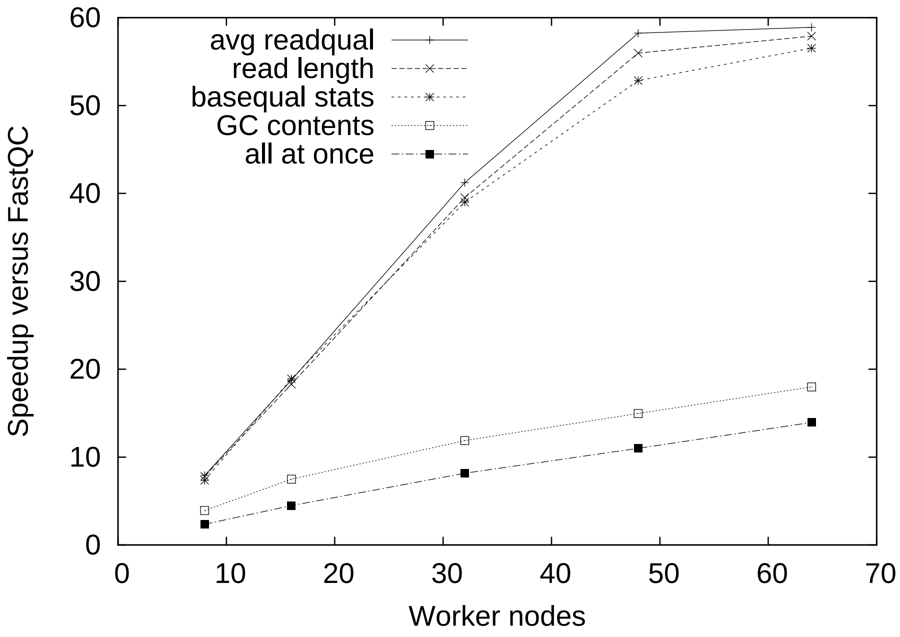


Figure 6.4 – Results of an experiment with an input file of 61.4 GB and varying number of Hadoop worker nodes. The script does not currently implement all FastQC statistics (but the missing ones are expected to scale similarly), whereas the per-cycle quality distribution is not computed by FastQC.

Chapter 7

Conclusion

7.1 Future work

7.1.1 Seal

While the Seal tool suite is certainly useful in its current state, there are many improvements that should be implemented. The work that would have the most impact would be the completion of a full, Hadoop-based, variant calling pipeline. The birth of the ADAM project can help with this target. Thus, an important goal to set for the future of this project is to try explore the potential synergies between Seal and ADAM to arrive at a full Hadoop-based variant calling pipeline.

Also, the Apache Flink [6] and Apache Spark frameworks are have developed into valid alternatives to Hadoop MapReduce. On modern computing hardware, these newer frameworks can better exploit the increased amount of system RAM to improve performance on some types of problems. It would be worthwhile to explore the potential advantages that Flink and Spark could bring to Seal, as well as examine the feasibility of porting the existing software to these new platforms.

7.1.2 Automation

The automation system described in this dissertation could benefit from a number of improvements. The event-driven design of the automator service is simple and robust. In fact, the automator exhibits an "emergent behaviour", where the system's complex behaviour emerges from the combination of a number of simple event handlers. However, these benefits come at the expense of not having an explicit state in the system, which would greatly improve usability and provide the ability to easily track the progress of samples through the system. Future versions of the automator should consider addressing this issue, perhaps sacrificing some the system's robustness in favour of the improved usability derived by an explicit state.

Further, the system would greatly benefit from a unified interface. The system is comprised by a number of independent components that work together. Each of these sub-systems has its own interface, in the form of command-line tools or software modules. As a result, commanding the entire system can be a complex ordeal left only to those that have mastered the use of all the components. To address this issue, the system should be equipped with a single façade implementing the typical high-level functionality that is required by the users – especially the queries required to extract information from the system.

7.1.3 Hadoop-Galaxy

As mentioned in its chapter, the Hadoop-Galaxy adapter has a notable shortcoming that needs to be addressed: the lack of a garbage collection mechanism for the pathset data type. Once this feature is complete, then the adapter can be used as easily as any other Galaxy feature.

7.1.4 SeqPig

The SeqPig language, and especially its function library, could benefit from additional special-purpose functions. The generic functions currently included in the library, when appropriately assembled, can be used to implement a variety of analyses. However, specialized functions can be used to accelerate the same analyses and make them easier to run. This strategy was used to accelerate some of the quality control features used in the evaluation, and should be analogously applied to other functionality.

7.2 Summary

Throughout the chapters of this dissertation we have explored some of the problems faced by medium- and large-scale sequencing centers. The root cause of the issues lies in the revolutionary introduction of next-generation sequencing technology, which greatly increased the amount of sequencing data that can be produced and – consequently – the number of samples that are sequenced. Thus, this technology has introduced two types of scalability problems: the first is scaling analysis with respect to the data size; the second is scaling the entire sequencing operation with respect to the number of samples to be treated.

To address the first problem, this dissertation has presented Seal, a suite of scalable software tools that are designed to fully exploit a suitable computing infrastructure to scale data processing rates on par with the sequencing machines' data production rates. On the other hand, to help operations scale in the numbers of samples they handle, an automation strategy was presented, complete with the means to ensure reproducible science by storing, along with each analysis result, the sequence of computing operations that were performed to generate it. Significant work was also done to integrate these two systems, so that the automation system could use the scalable analysis tools.

Finally, these software tools, while helping sequencing operations scale, they admittedly hinder the simplicity of the system from the user's perspective. Especially in an environment where many users are not computing specialists, to encourage the uptake and the acceptance of new systems it is important to ensure that the systems are not exceedingly difficult to use. This dissertation presented two contributions to improve the usability of the scalable computing solutions proposed. The first is the integration of Seal and Galaxy through the Hadoop-Galaxy adapter. The Galaxy bioinformatics platform provides a simple interface that is easy to use and quite popular popular among biologists and bioinformaticians. By making the Seal tools usable through Galaxy a lot of the complexities connected to the underlying Hadoop framework are hidden from the user, making the tools much simpler to use. The second contribution along these lines is the Hadoop-based scripting language called SeqPig. This SQL-like scripting language allows the user to be expressive in his analysis, while enjoying the benefits of scalable computing through the automatic compilation of SeqPig scripts into distributed Hadoop jobs.

The effectiveness of these tools has been demonstrated by the fact that they are currently in production use at the CRS4 Next-Generation Sequencing Laboratory and by the various scalability experiments presented in this work. Thanks to these contributions, the CRS4 NGS Lab has reduced the number of full-time individuals required to run its analyses from three to just one, freeing resources for downstream, research-specific tasks – a very encouraging result.

Bibliography

[1] José M. Abuín, Juan C. Pichel, Tomás F. Pena, et al. "BigBWA: approaching the Burrows-Wheeler aligner to Big Data technologies". *Bioinformatics* 31 (24), 2015, pp. 4003–4005. DOI: `10.1093/bioinformatics/btv506`.

[2] Chris Allan, Jean-Marie Burel, Josh Moore, et al. "OMERO: flexible, model-driven data management for experimental biology". *Nature Methods* 9 (3), Mar. 2012, pp. 245–253. DOI: `10.1038/nmeth.1896`.

[3] S. F. Altschul, W. Gish, W. Miller, et al. "Basic local alignment search tool". *J. Mol. Biol.* 215 (3), 1990, pp. 403–410.

[4] *Amazon Elastic MapReduce.* URL: `http://aws.amazon.com/elasticmapreduce`.

[5] Simon Andrews. *FASTQC. A quality control tool for high throughput sequence data.* Feb. 2016. URL: `http://www.bioinformatics.babraham.ac.uk/projects/fastqc`.

[6] *Apache Flink.* Feb. 2016. URL: `https://flink.apache.org`.

[7] *Apache Parquet.* Feb. 2016. URL: `https://parquet.apache.org/`.

[8] Geraldine A Auwera, Mauricio O Carneiro, Christopher Hartl, et al. "From FastQ data to high-confidence variant calls: the genome analysis toolkit best practices pipeline". *Current Protocols in Bioinformatics*, 2013, pp. 11–10.

[9] Philippe L. Bedard, Aaron R. Hansen, Mark J. Ratain, et al. "Tumour heterogeneity in the clinic". *Nature* 501 (7467), Sept. 19, 2013. Insight, pp. 355–364. ISSN: 0028-0836. DOI: `10.1038/nature12627`.

[10] David R. Bentley, Shankar Balasubramanian, Harold P. Swerdlow, et al. "Accurate whole human genome sequencing using reversible terminator chemistry". *Nature* 456 (7218), Nov. 6, 2008, pp. 53–59. ISSN: 0028-0836. DOI: `10.1038/nature07517`.

[11] A. Biffi, E. Montini, L. Lorioli, et al. "Lentiviral Hematopoietic Stem Cell Gene Therapy Benefits Metachromatic Leukodystrophy". *Science* 341 (6148), Aug. 2013, p. 1233158. DOI: `10.1126/science.1233158`.

[12] Daniel Blankenberg, Gregory Von Kuster, Emil Bouvier, et al. "Dissemination of scientific software with Galaxy ToolShed". *Genome Biol.* 15, 2014, p. 403. ISSN: 1465-6906. DOI: `10.1186/gb4161`.

[13] Scott D. Boyd. "Diagnostic Applications of High-Throughput DNA Sequencing". *Annual Review of Pathology: Mechanisms of Disease* 8 (1), 2013. PMID: 23121054, pp. 381–410. DOI: `10.1146/annurev-pathol-020712-164026`.

[14] Gabriel Caffarena, Carlos Pedreira, Carlos Carreras, et al. "FPGA acceleration for DNA sequence alignment". *Journal of Circuits, Systems, and Computers* 16 (02), 2007, pp. 245–266.

[15] Shiping Chen, Tomasz Bednarz, Piotr Szul, et al. "Galaxy + Hadoop: Toward a Collaborative and Scalable Image Processing Toolbox in Cloud". In: *Service-Oriented Computing – ICSOC 2013 Workshops*. Ed. by Alessio R. Lomuscio, Surya Nepal, Fabio Patrizi, et al. Vol. 8377. Lecture Notes in Computer Science. Springer International Publishing, 2014, pp. 339–351. ISBN: 978-3-319-06858-9. DOI: `10.1007/978-3-319-06859-6_30`.

[16] Yanpei Chen, Sara Alspaugh, and Randy Katz. "Interactive analytical processing in big data systems: A cross-industry study of MapReduce workloads". In: *Proceedings of the VLDB Endowment*. Vol. 5. Aug. 2012, pp. 1802–1813.

[17] Gen-Tao Chiang, Peter Clapham, Guoying Qi, et al. "Implementing a genomic data management system using iRODS in the Wellcome Trust Sanger Institute." *BMC bioinformatics* 12 (1), Jan. 2011, p. 361. ISSN: 1471-2105. DOI: `10.1186/1471-2105-12-361`.

[18] Peter J. A. Cock, Christopher J. Fields, Naohisa Goto, et al. "The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants". *Nucleic Acids Res* 38 (6), Apr. 16, 2010. gkp1137[PII], pp. 1767–1771. ISSN: 0305-1048. DOI: `10.1093/nar/gkp1137`.

[19] Peter J. A. Cock, Björn A. Grüning, Konrad Paszkiewicz, et al. "Galaxy tools and workflows for sequence analysis with applications in molecular plant pathology". *PeerJ* 1, 2013, e167. ISSN: 2167-8359. DOI: `10.7717/peerj.167`.

[20] Gianmauro Cuccuru, Simone Leo, Luca Lianas, et al. "An automated infrastructure to support high-throughput bioinformatics". In: *High Performance Computing & Simulation (HPCS), 2014 International Conference on.* IEEE. 2014, pp. 600–607.

[21] Martin Dahlö, Samuel Lampa, Pall I Olason, et al. "Lessons learned from implementing a national infrastructure in Sweden for storage and analysis of next-generation sequencing data." *GigaScience* 2 (1), Jan. 2013, p. 9. ISSN: 2047-217X. DOI: `10.1186/2047-217X-2-9`.

[22] J. Dean and S. Ghemawat. "MapReduce: simplified data processing on large clusters". In: *OSDI '04: 6th Symposium on Operating Systems Design and Implementation.* 2004.

[23] Richard M Durbin, David L Altshuler, Gonçalo R Abecasis, et al. "A map of human genome variation from population-scale sequencing". *Nature* 467 (7319), 2010, pp. 1061–1073.

[24] MPI Forum. *MPI: A Message Passing Interface Standard; Version 3.1.* June 2015. URL: `http://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf`.

[25] Paolo Francalacci, Laura Morelli, Andrea Angius, et al. "Low-pass DNA sequencing of 1200 Sardinians reconstructs European Y-chromosome phylogeny". *Science* 341 (6145), 2013, pp. 565–569.

[26] M. Gaggero, S. Leo, S. Manca, et al. "Parallelizing bioinformatics applications with MapReduce". *CCA-08: Cloud Computing and its Applications*, 2008.

[27] Jeremy Goecks, Anton Nekrutenko, James Taylor, et al. "Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences". *Genome Biology* 11 (8), 2010, R86. ISSN: 1465-6906. DOI: `10.1186/gb-2010-11-8-r86`.

[28] Martien A. M. Groenen, Alan L. Archibald, Hirohide Uenishi, et al. "Analyses of pig genomes provide insight into porcine demography and evolution". *Nature* 491 (7424), Nov. 15, 2012, pp. 393–398. ISSN: 0028-0836. DOI: `10.1038/nature11622`.

[29] *List of institutions that use Hadoop in education or production.* URL: `http://wiki.apache.org/hadoop/PoweredBy`.

[30] *Welcome to Apache Hadoop.* 2016. URL: `http://hadoop.apache.org/`.

[31] *HiSeq 3000/HiSeq 4000 System Specifications.* Illumina, Inc. 9885 Towne Centre Drive, San Diego, CA 92121 USA, 2015.

[32] *HiSeq 2000 Sequencing System*. Illumina, Inc. 9885 Towne Centre Drive, San Diego, CA 92121 USA, 2010.

[33] Inc. Illumina. *HiSeq 2500 System Specifications*. Illumina, Inc. 9885 Towne Centre Drive, San Diego, CA 92121 USA, 2015.

[34] Broad Institute. *GATK Best Practices*. Feb. 2016. URL: `https://www.broadinstitute.org/gatk/guide/best-practices.php`.

[35] Broad Institute. *Picard*. Feb. 2016. URL: `http://broadinstitute.github.io/picard`.

[36] National Human Genome Research Institute. Feb. 2016. URL: `https://www.genome.gov/sequencingcosts/`.

[37] *iRODS: integrated Rule Oriented Data System*. Tech. rep. DICE Group: University of North Carolina, University of California, Sept. 2008.

[38] Chris Jordan, Dan Stanzione, Doreen Ware, et al. "Comprehensive data infrastructure for plant bioinformatics". *2010 IEEE International Conference On Cluster Computing Workshops and Posters (Cluster Workshops)*, Sept. 2010, pp. 1–5. DOI: `10.1109/CLUSTERWKSP.2010.5613093`.

[39] Laurent Jourdren, Maria Bernard, Marie-Agnès Dillies, et al. "Eoulsan: a cloud computing-based framework facilitating high throughput sequencing analyses". *Bioinformatics* 28 (11), 2012, pp. 1542–1543. DOI: `10.1093/bioinformatics/bts165`.

[40] M Aleksi Kallio, Jarno Tuimala, Taavi Hupponen, et al. "Chipster: user-friendly analysis software for microarray and other high-throughput data". *BMC Genomics* 12 (1), 2011, p. 507. ISSN: 1471-2164. DOI: `10.1186/1471-2164-12-507`.

[41] D. R. Kelley, M. C. Schatz, and S. L. Salzberg. "Quake: quality-aware detection and correction of sequencing errors". *Genome Biology* 11 (11), 2010, p. 116.

[42] Petr Klus, Simon Lam, Dag Lyberg, et al. "BarraCUDA - a fast short read sequence aligner using graphics processing units". *BMC Research Notes* 5 (1), 2012, pp. 1–7. ISSN: 1756-0500. DOI: `10.1186/1756-0500-5-27`.

[43] Ben Langmead. "Aligning short sequencing reads with Bowtie". *Current protocols in bioinformatics*, 2010, pp. 11–7.

[44] Ben Langmead, Michael C Schatz, Jimmy Lin, et al. "Searching for SNPs with cloud computing". *Genome Biology* 10 (11), 2009, R134. DOI: `10.1186/gb-2009-10-11-r134`.

[45] Ben Langmead, Cole Trapnell, Mihai Pop, et al. "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome". *Genome biol* 10 (3), 2009, R25.

[46] Richard M Leggett, Ricardo H Ramirez-Gonzalez, Bernardo J Clavijo, et al. "Sequencing quality assessment tools to enable data-driven informatics for high throughput genomics." *Frontiers in genetics* 4 (December), Jan. 2013, p. 288. ISSN: 1664-8021. DOI: `10.3389/fgene.2013.00288`.

[47] Simone Leo, Luca Pireddu, Gianmauro Cuccuru, et al. "BioBlend.objects: metacomputing with Galaxy". *Bioinformatics* 30 (19), 2014, pp. 2816–2817.

[48] Simone Leo, Federico Santoni, and Gianluigi Zanetti. "Biodoop: bioinformatics on Hadoop". In: *The 38th International Conference on Parallel Processing Workshops (ICPPW 2009).* 2009, pp. 415–422. DOI: `10.1109/ICPPW.2009.37`.

[49] Simone Leo and Gianluigi Zanetti. "Pydoop: a Python MapReduce and HDFS API for Hadoop". In: *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing.* HPDC '10. Chicago, Illinois: ACM, 2010, pp. 819–825. ISBN: 978-1-60558-942-8. DOI: `10.1145/1851476.1851594`.

[50] Heng Li. *Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM.* 2013. arXiv: `1303.3997v1 [q-bio.GN]`.

[51] Heng Li, Bob Handsaker, Alec Wysoker, et al. "The Sequence Alignment/Map format and SAMtools". *Bioinformatics* 25 (16), 2009, pp. 2078–2079. DOI: `10.1093/bioinformatics/btp352`.

[52] Heng Li and Nils Homer. "A survey of sequence alignment algorithms for next-generation sequencing". *Briefings in Bioinformatics* 11 (5), 2010, pp. 473–483. DOI: `10.1093/bib/bbq015`.

[53] Yongchao Liu, Bertil Schmidt, and Douglas L. Maskell. "CUSHAW: a CUDA compatible short read aligner to large genomes based on the Burrows–Wheeler transform". *Bioinformatics* 28 (14), 2012, pp. 1830–1837. DOI: `10.1093/bioinformatics/bts276`.

[54] Vivien Marx. "Biology: The big challenges of big data". *Nature* 498, 7453 June 2013. DOI: `10.1038/498255a`.

[55] Matt Massie, Frank Nothaft, Christopher Hartl, et al. *ADAM: Genomics Formats and Processing Patterns for Cloud Scale Computing.* Tech. rep. UCB/EECS-2013-207. EECS Department, University of California, Berkeley, Dec. 2013. URL: `http://www.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-207.html`.

[56] A. Matsunaga, M. Tsugawa, and J. Fortes. "Cloudblast: combining MapReduce and virtualization on distributed resources for bioinformatics applications". In: *Fourth IEEE International Conference on eScience.* 2008, pp. 222–229.

[57] A. McKenna, M. Hanna, E. Banks, et al. "The Genome Analysis Toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data". *Genome Research* 20 (9), 2010, pp. 1297–1303. DOI: `10.1101/gr.107524.110`.

[58] Sergey Melnik, Andrey Gubarev, Jing Jing Long, et al. "Dremel: interactive analysis of web-scale datasets". *Proceedings of the VLDB Endowment* 3 (1-2), 2010, pp. 330–339.

[59] Gordon E. Moore. "Cramming More Components onto Integrated Circuits". *Proceedings of the IEEE* 86 (1), Jan. 1998. Reprinted from *Electronics*, pp. 114–117, April 19, 1965, pp. 82–85.

[60] Matti Niemenmaa, Aleksi Kallio, André Schumacher, et al. "Hadoop-BAM: directly manipulating next generation sequencing data in the cloud". *Bioinformatics* 28 (6), 2012, pp. 876–877.

[61] Bjorn Nystedt, Nathaniel R. Street, Anna Wetterbom, et al. "The Norway spruce genome sequence and conifer genome evolution". *Nature* 497 (7451), May 30, 2013. Article, pp. 579–584. ISSN: 0028-0836. DOI: `10.1038/nature12211`.

[62] Brian O'Connor, Barry Merriman, and Stanley Nelson. "SeqWare Query Engine: storing and searching sequence data in the cloud". *BMC Bioinformatics* 11 (Suppl 12), 2010, S2. ISSN: 1471-2105. DOI: `10.1186/1471-2105-11-S12-S2`.

[63] Aisling O'Driscoll, Jurate Daugelaite, and Roy D. Sleator. ""Big data", Hadoop and cloud computing in genomics". *Journal of Biomedical Informatics* 46 (5), 2013, pp. 774–781. ISSN: 1532-0464. DOI: `10.1016/j.jbi.2013.07.001`.

[64] Tom Oinn, Mark Greenwood, Matthew Addis, et al. "Taverna: lessons in creating a workflow environment for the life sciences". *Concurrency and Computation: Practice and Experience* 18 (10), 2006, pp. 1067–1100.

[65] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, et al. "Pig latin: a not-so-foreign language for data processing". In: *SIGMOD Conference*. 2008, pp. 1099–1110.

[66] Valeria Orrù, Maristella Steri, Gabriella Sole, et al. "Genetic variants regulating immune cell levels in health and disease." *Cell* 155 (1), Sept. 2013, pp. 242–56. ISSN: 1097-4172. DOI: `10.1016/j.cell.2013.08.041`.

[67] Chandra Shekhar Pareek, Rafal Smoczynski, and Andrzej Tretyn. "Sequencing technologies and genome sequencing." *Journal of applied genetics* 52 (4), Dec. 2011, pp. 413–35. ISSN: 2190-3883. DOI: `10.1007/s13353-011-0057-x`.

[68] Luca Pireddu, Simone Leo, Nicola Soranzo, et al. "A hadoop-galaxy adapter for user-friendly and scalable data-intensive bioinformatics in galaxy". In: *Proceedings of the 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics*. ACM. 2014, pp. 184–191.

[69] Luca Pireddu, Simone Leo, and Gianluigi Zanetti. "MapReducing a genomic sequencing workflow". In: *Proceedings of the second international workshop on MapReduce and its applications*. MapReduce '11. San Jose, California, USA: ACM, 2011, pp. 67–74. ISBN: 978-1-4503-0700-0. DOI: `10.1145/1996092.1996106`.

[70] Luca Pireddu, Simone Leo, and Gianluigi Zanetti. "SEAL: a distributed short read mapping and duplicate removal tool". *Bioinformatics* 27 (15), 2011, pp. 2159–2160. DOI: `10.1093/bioinformatics/btr325`.

[71] The Ohio State University College of Medicine. *pMap: parallel sequence mapping tool*. 2016. URL: `http://bmi.osu.edu/hpc/software/pmap/pmap.html`.

[72] *RabbitMQ*. URL: `https://www.rabbitmq.com/`.

[73] Arcot Rajasekar, Reagan Moore, Chien-Yi Hou, et al. "iRODS Primer: Integrated Rule-Oriented Data System". *Synthesis Lectures on Information Concepts, Retrieval, and Services* 2 (1), 2010, pp. 1–143. DOI: `10.2200/S00233ED1V01Y200912ICR012`.

[74] James T. Robinson, Helga Thorvaldsdottir, Wendy Winckler, et al. "Integrative genomics viewer". *Nat Biotech* 29 (1), Jan. 2011, pp. 24–26. ISSN: 1087-0156. DOI: `10.1038/nbt.1754`.

[75] Thomas Robinson, Sarah Killcoyne, Ryan Bressler, et al. "SAMQA: error classification and validation of high-throughput sequenced read data". *BMC genomics* 12 (1), 2011, p. 1.

[76] Matthew Ruffalo, Thomas LaFramboise, and Mehmet Koyutürk. "Comparative analysis of algorithms for next-generation sequencing read alignment". *Bioinformatics* 27 (20), 2011, pp. 2790–2796. DOI: `10.1093/bioinformatics/btr477`.

[77] Eric Sammer. *Hadoop Operations*. 1st. O'Reilly Media, Inc., 2012. ISBN: 1449327052, 9781449327057.

[78] F. Sanger, G. M. Air, B. G. Barrell, et al. "Nucleotide sequence of bacteriophage [phi]X174 DNA". *Nature* 265 (5596), Feb. 24, 1977, pp. 687–695. DOI: `10.1038/265687a0`.

[79] Andrea Sboner, Xinmeng Jasmine Mu, Dov Greenbaum, et al. "The real cost of sequencing: higher than you think!" *Genome Biology* 12 (8), 2011, pp. 1–10. ISSN: 1465-6906. DOI: `10.1186/gb-2011-12-8-125`.

[80] M. C. Schatz, D. Sommer, D. R. Kelley, et al. *Contrail: Assembly of Large Genomes using Cloud Computing*. 2016. URL: `http://contrail-bio.sourceforge.net`.

[81] Michael C. Schatz. "CloudBurst: highly sensitive read mapping with MapReduce". *Bioinformatics* 25 (11), 2009, pp. 1363–1369. DOI: `10.1093/bioinformatics/btp236`.

[82] Sebastian Schonherr, Lukas Forer, Hansi WeiSZensteiner, et al. "Cloudgene: A graphical execution platform for MapReduce programs on private and public clouds". *BMC Bioinformatics* 13 (1), 2012, p. 200. ISSN: 1471-2105. DOI: `10.1186/1471-2105-13-200`.

[83] André Schumacher, Luca Pireddu, Matti Niemenmaa, et al. "SeqPig: simple and scalable scripting for large sequencing data sets in Hadoop". *Bioinformatics* 30 (1), 2014, pp. 119–120. DOI: `10.1093/bioinformatics/btt601`.

[84] Seal. *The QSEQ file format*. Feb. 2016. URL: `http://biodoop-seal.sourceforge.net/file_formats.html`.

[85] H. A. Shah, L. Hasan, and N. Ahmad. "An optimized and low-cost FPGA-based DNA sequence alignment – a step towards personal genomics". In: *Engineering in Medicine and Biology Society (EMBC), 2013 35th Annual International Conference of the IEEE*. July 2013, pp. 2696–2699. DOI: `10.1109/EMBC.2013.6610096`.

[86] Jay Shendure and Erez Lieberman Aiden. "The expanding scope of DNA sequencing". *Nature biotechnology* 30 (11), 2012, pp. 1084–1094.

[87] Jay Shendure and Hanlee Ji. "Next-generation DNA sequencing". *Nature biotechnology* 26 (10), 2008, pp. 1135–1145.

[88] Wendy Weijia Soon, Manoj Hariharan, and Michael P Snyder. “High-throughput sequencing for biology and medicine”. *Molecular systems biology* 9 (1), 2013.

[89] Ola Spjuth, Erik Bongcam-Rudloff, Guillermo Carrasco Hernández, et al. “Experiences with workflows for automating data-intensive bioinformatics”. *Biology Direct* 10 (1), 2015, pp. 1–12.

[90] Lincoln Stein. “The case for cloud computing in genome informatics”. *Genome Biology* 11 (5), 2010, p. 207. ISSN: 1465-6906. DOI: `10.1186/gb-2010-11-5-207`.

[91] Ronald Taylor. “An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics”. *BMC Bioinformatics* 11 (Suppl 12), 2010, p. 1. DOI: `10.1186/1471-2105-11-S12-S1`.

[92] Helga Thorvaldsdóttir, James T. Robinson, and Jill P. Mesirov. “Integrative Genomics Viewer (IGV): high-performance genomics data visualization and exploration”. *Briefings in Bioinformatics* 14 (2), 2013, pp. 178–192. DOI: `10.1093/bib/bbs017`.

[93] Kristin M. Tolle, D. Stewart W. Tansley, and Anthony J. G. Hey. “The Fourth Paradigm: Data-Intensive Scientific Discovery”. *Proceedings of the IEEE* 99 (8), 2011, pp. 1334–1337. ISSN: 0018-9219.

[94] Hasitha Muthumala Waidyasooriya, Masanori Hariyama, and Michitaka Kameyama. “FPGA-Accelerator for DNA Sequence Alignment Based on an Efficient Data-Dependent Memory Access Scheme”. In: *Proc. of the 5th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART)*. 2014, pp. 127–130.

[95] Zhong Wang, Mark Gerstein, and Michael Snyder. “RNA-Seq: a revolutionary tool for transcriptomics”. *Nat Rev Genet* 10 (1), Jan. 2009, pp. 57–63. DOI: `10.1038/nrg2484`.

[96] C. W. Whelan, J. Tyner, A. L’Abbate, et al. “Cloudbreak: Accurate and Scalable Genomic Structural Variation Detection in the Cloud with MapReduce”. *arXiv:1307.2331v2*, 2013. arXiv: `1307.2331v2 [q-bio.GN]`.

[97] Tom White. *Hadoop: The Definitive Guide*. first edition. O’Reilly, June 2009.

[98] Wikipedia. *Indirection*. Feb. 2016. URL: `https://en.wikipedia.org/wiki/Indirection`.

[99] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, et al. “Spark: Cluster Computing with Working Sets.” *HotCloud* 10, 2010, pp. 10–10.