Analyzing Big Data using Hadoop MapReduce

Angelo Altamirano

Marshall Plan Scholarship Paper

Utah State University

University of Innsbruck (LFU)

In cooperation with

Medical University of Innsbruck

Advisors:

Dr. Sebastian Schönherr
Lukas Forer, Ph.D.
Univ-Prof. Dr. Hans Dieplinger

Table of Contents

1. Introduction

Today, scientific research faces many challenges that prevent researchers from working with big data efficiently and effectively. Computer science is highly recognized for relieving such challenges in genetic research. As a result of its developed importance in the research world, there is an increased demand for suitable software solutions to attend scientist's needs. However, the cost of storing, processing and moving data, generated by next generation sequencing (NGS), limits scientists analyzing it on a current computer in the lab. Due to the dramatic downfall in costs of NGS data generation, the growth of such 'big data' results in a much longer run-time for available programs, which often can't even run due to the lack of computer resources. This increased demand of resources pushes computer scientists to further develop new methods to circumvent these limitations in today's programs.

The Apache Software Foundation's framework Hadoop is one approach based on Google's MapReduce paradigm to provide the needed computing power, allowing the development of reliable, expandable and parallel computing. The framework allows the processing of large scale data among a cluster of computers. Hadoop is becoming constantly more popular in Bioinformatics for processing and analyzing an extensive amount of data. For example, Hadoop is used in mapping NGS data to the human genome as well as matching strings in a large genotype file which would normally take a significant amount of time on a single computer. For example, the software system Cloudgene [7], which has been developed at my visiting institute (Innsbruck Medical University, IMU) in cooperation with the Institute of Computer Science, Research group Databases and Information Systems at the University of Innsbruck, uses the Hadoop framework to execute MapReduce workflows graphically and simplify the data management with the distributed file system HDFS. One important feature of this system is that new use cases can be integrated easily.

One of the goals of my project was to recognize why Hadoop can be important in Genetics and get familiar with its overall architecture. As a first use case, the so called "Inverted Index" was adapted to the MapReduce programming model and was executed on command line. In a next step, this use case was integrated into Cloudgene. The second task of this project was then to implement a genetic use case to create statistics using NGS data based on Hadoop MapReduce. This has been first developed together with the researchers from the University of Innsbruck and has been then integrated into Cloudgene. The genetic application generates

statistics on NGS data in a massive parallel way. This work starts with an introduction into Big Data and MapReduce. Moreover, it describes the implementation details of the inverted index and shows the produced NGS statistics.

2. Big Data and MapReduce

Big data centers like Google demonstrated successfully that for data-intensive applications based on MapReduce a large number of commodity hardware is sufficient in order to achieve equivalent results. Such a low budget cluster consists of hundreds and even thousands of machines, which are used to solve the problem in parallel. This section gives an overview on MapReduce and describes the idea behind this parallel programming model.

2.1. What is Big Data?

It is generally understood by most people that 'Big Data' is composed of large datasets that could be no longer managed by standard processing methods. As a matter of fact, big data is defined as data sets too large and complex to be operated by any standard method or apparatus. Big data is not only limited to numerical values. It also consists of informational research such as records obtained in human genetics. Vivien Marx, a technology editor, states that the European Molecular Biology Laboratory in Hinxton, United Kingdom holds one of the largest biological-data repositories currently storing 20 petabytes of data and backups about genes, proteins, and small molecules **[**5**]**. Genomic data, which includes recombinant DNA and DNA sequence methods, already utilizes two of the twenty petabytes and doubles every year as shown in Figure 2.1 **[**5**]**:

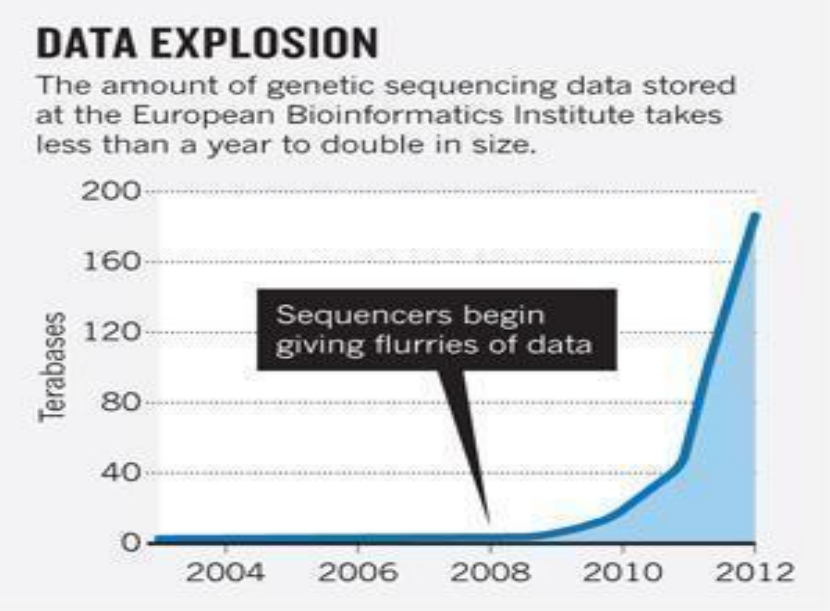

Figure 2.1: Data Explosion [5]: This figure shows the exponential growth in data use and the increased demand for various alternatives of storing big data in most institutions.

Currently, Marx emphasizes that novel research gives particular importance and dependence on data manipulation in order to push it forward. Many researchers are starting to explore and handle big datasets, however, they encounter many difficulties in processing and moving such information. Ewan Birney, an associate director at the European Bioinformatics Institute, (EBI) stated that "biology data mining has challenges of its own" and most biologists would agree that this is a valid statement to make. Biological data has the ability to encompass a wide range of experiments that divide out into many types of information including but not limited to genetic sequences, the interactions of proteins, or simply finding medical information **[**5**]**.

For analyzing data, two aspects are crucial in the area of bioinformatics: (1) Parallelization of data and (2) the usability of developed applications for domain experts. Especially due to the huge amount of data, a scalable approach on analyzing this data is necessary. For that, the focus has been set on Hadoop MapReduce. To improve the usability of implemented use cases, the novel workflow system Cloudgene (http://cloudgene.uibk.ac.at) has been developed at IMU.

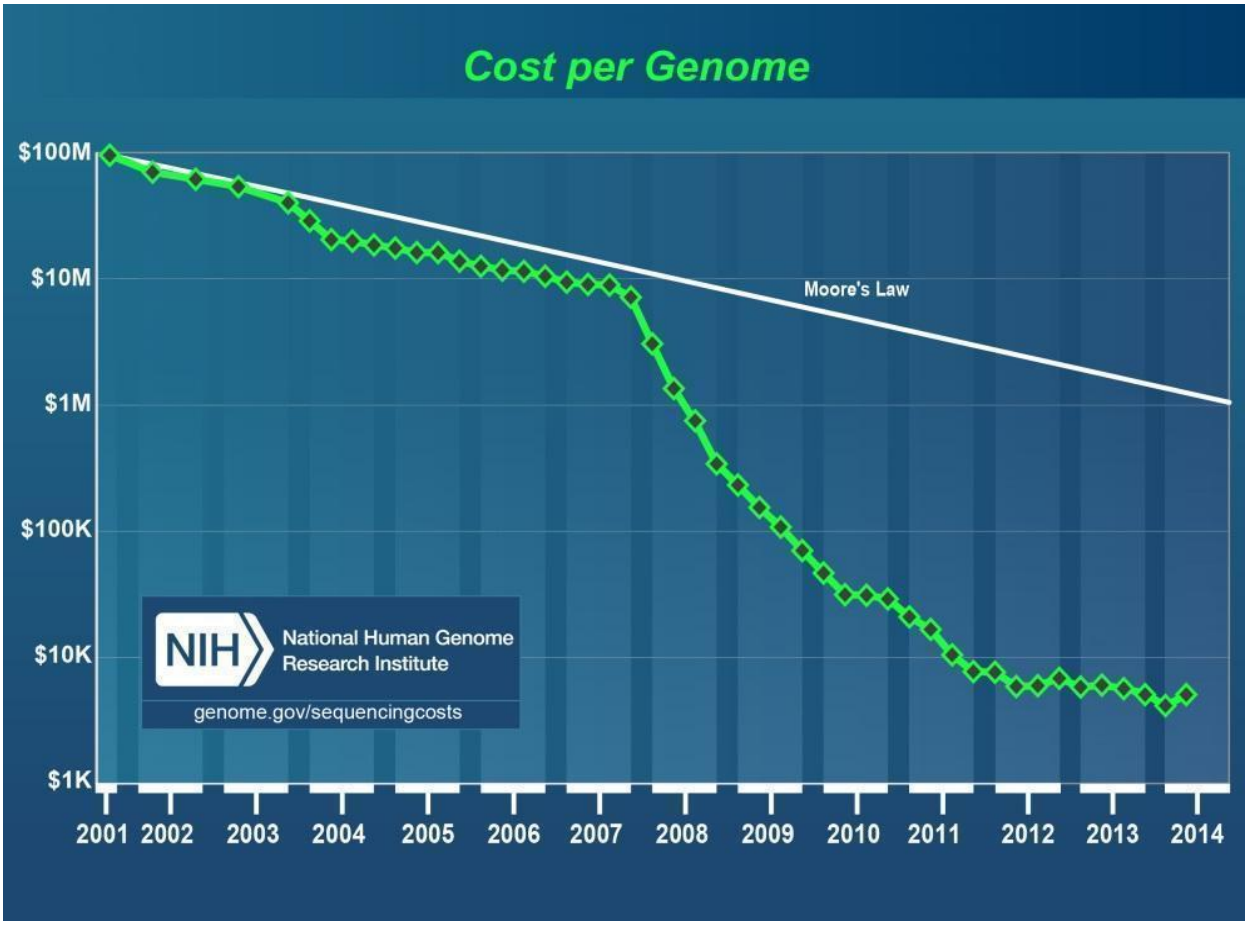

Figure .2.2: Cost per Genome [8]

Figure 2.2 [8]: shows a simple relationship between the costs of generating genome sequences from 2001 until now and the development of computing hardware (Moore's law). As one could tell by the graph, costs for sequencing genomes are falling exponentially, compared to Moore's law, showing here that transistor costs halves approximately every two years. If the cost diminishes, this essentially means that more Institutes will be able to afford the sequencing of human genomes. This essentially means the amount of data is growing exponentially and the need of managing and analyzing this data is crucial.

2.2. What is MapReduce?

MapReduce is generally defined as a programming model for processing and generating large sets of data **[**2**]**. The user specifies a map function that takes a so called key/value pair as an input to generate a set of intermediate key/value-pairs (see Figure 2.3). Then, the reducer function merges all of the intermediate values associated with the same intermediate key **[**2**]**.

In detail, input data can be given by a list of records or words that are represented by (key, value) pairs, as shown in Figure 2.3. There are two phases in MapReduce, the "Map" phase and the "Reduce" phase. First, the data is distributed and processed independently from other data items between different nodes called, "mappers." These mappers are represented by the rectangular figures under the "Map" column. Next, the mapper outputs intermediate (key, value) pairs which then enter the "Reduce" phase, represent by the "Reduce" column. The data having the same key (word) are bundled together and processed into the same node which is done by the "reducer." The reducer then combines different values with the same key into nodes and sums up the intermediate results from the mapper. The results are then put into the distributed file system.

The programs written using this model are automatically parallelized, since the mapper- and reducer-functions can be executed in parallel among a cluster of machines **[**2**]**. It functions primarily on the principle that large problems can be divided into smaller ones. To explain the functionality of MapReduce in richer detail, a tutorial how to create an inverted index will be introduced (see section 3.1).

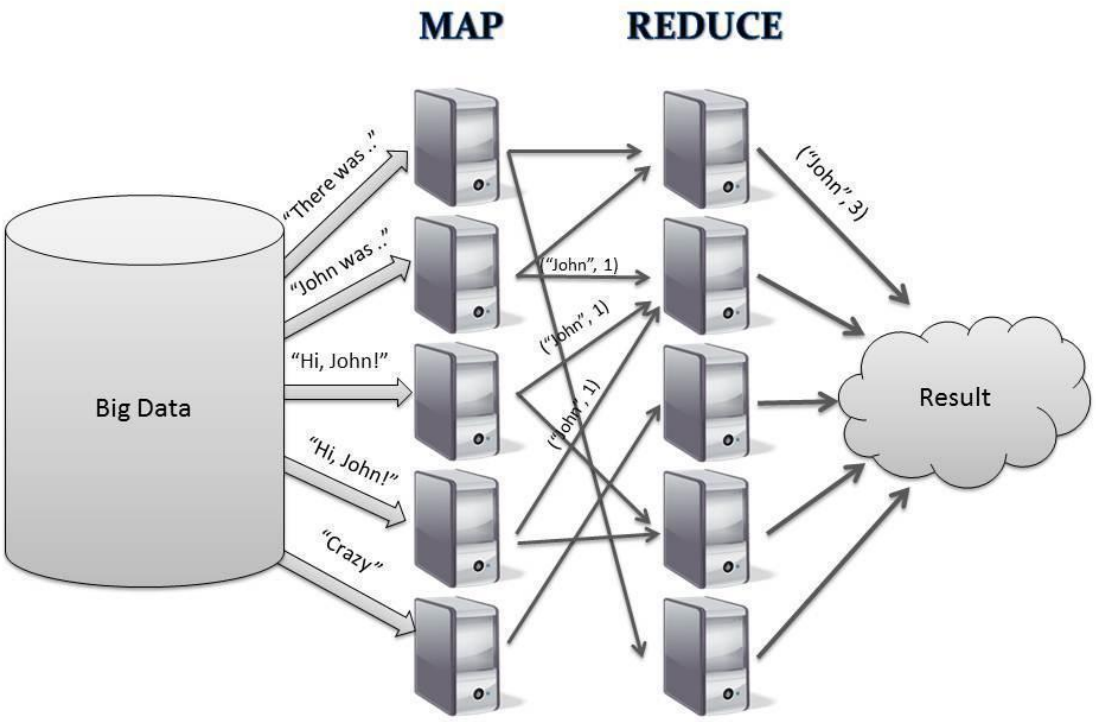

Figure 2.3, MapReduce Overview Diagram [3]

2.3. Apache Hadoop

Apache Hadoop is a popular open-source framework used to process large sets of data that are sent across a cluster of computers using the MapReduce programming model [1**]**. The framework of Hadoop is designed to work on thousands of machines and provides high availability. Depending on the number of machines, the more machines there are in a cluster, it is more likely a machine node in the cluster would fail. Thus, the library itself is designed to monitor and handle failures at the application layer. Delivering a highly available service on top of a cluster of computers may be more prone to failures **[**1**]**. The architecture of Apache Hadoop consists of two core components, which are needed to implement a MapReduce use case:

2.3.1. Hadoop Distributed File System (HDFS)

The '*Hadoop HDFS*' is a distributed file system, which is used to store datasets among all nodes in a cluster. This is absolutely necessary when trying to run a MapReduce program. The design of HDFS was meant to perform on commodity hardware, having much higher fault rate

compared to expensive server systems [1]. Something amazing about HDFS is being very reliable by providing the needed fault-tolerance, which means being able to operate during a failure of a machine. Moreover, the design of HDFS therefore enables itself to operate on such low-cost hardware and provides high throughput access for applications that process large data sets.

2.3.2. Hadoop MapReduce (MRv1) or YARN (MRv2)

Since the introduction of YARN, in MapReduce 2.0, further processing libraries (e.g. for streaming data) are included as well in the Hadoop framework. In this project, the focus lies on the MapReduce library (MRv1), which builds an ideal candidate to process data from the inverted index and NGS data.

2.4 Limitations of MapReduce

The MapReduce library is an ideal candidate for batch processing large amount of data. For interactive analysis or working with graph data, MapReduce is not a natural fit. Therefore, Hadoop provides with YARN a way to separate the processing engine and resource manager and integrates different processing engines. For example it includes the graph processing engine for graph data or Spark for in-memory processing.

3. Inverted Index

A first use case that utilizes and incorporates the functionality of MapReduce is the Inverted Index. An Inverted Index is a structure most commonly used by search engines and databases to research terms from files or documents [4]. A fully inverted index will let you know not only what documents the term is included in, but also the location of the file itself [4]. The next section displays the basic steps on how to implement the inverted index with Java using the Hadoop MapReduce framework.

3.1. Implementation

The certain use case I am using for this tutorial on how to use MapReduce is an ‘Inverted Index’. The inverted index is a structure that mapped out contents such as words or numbers to its location in the database file or in a document or set of documents. The inverted index was modified to simply keep count of repeating words and assign its appropriate quantitative number

to the word. To understand how the implementation of the Inverted Index works, I will explain in the detail the mapper and reducer class.

Mapper

The Mapper class of the Inverted Index (TokenizerMapper) receives as an input a key value pair (Object key, Text value). Here, the value includes one input line of the overall HDFS input. This can be for example a line of a log file of a search engine. For each line, new key/value pairs are generated. This is achieved by splitting the line after each token. The token describes a word and defines the new value. As a key (the split level), the document name is used. The interface to the MapReduce library is established with a *context.write* call.

```
public static class TokenizerMapper extends
        Mapper<Object, Text, Text, Text> {

    private Text word = new Text();

    public void map(Object key, Text value, Context context)
            throws IOException, InterruptedException {
         String fileName = ((FileSplit) context.getInputSplit()).getPath().getName();
        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()) {
            word.set(itr.nextToken());
            context.write(word, new Text(fileName));
        }
    }
}
```

Reducer

As an input to the reducer ("IntSumReducer"), all values (all words) for a specific document are provided. The goal of the Inverted Index is to provide a list of terms with all documents, which they are appearing in. For that, we define a new list, iterate over all values and add all values to this list. Again, as a final output, the keys (terms) and the lists are written to the HDFS (context.write).

```
    public static class IntSumReducer extends
            Reducer<Text, Text, Text, Text> {
        //private IntWritable result = new IntWritable();

        public void reduce(Text key, Iterable<Text> values,
                Context context) throws IOException, InterruptedException {
            ArrayList<String> list = new ArrayList<String>();
            int sum = 0;
            for (Text val : values) {
                if(!list.contains(val.toString()))
            list.add(val.toString());
            }
            //result.set(sum);
            context.write(key, new Text(list.toString()));
        }
    }
```

3.2. Executing the Inverted Index via the command line

First, I made two text files to use as data in the program.

Next, I put the two local text files into the HDFS file system like shown:

```
[adminuser@localhost target]$ hadoop fs -put utahstate.txt engineer
[adminuser@localhost target]$ hadoop fs -put aggies.txt engineer
```

Throughout all of the commands having to be executed, next the MapReduce Program is executed:

```
[adminuser@localhost target]$ hadoop jar Examples-0.0.1-SNAPSHOT.jar index.Index engineer calc
```

Now the processing with MapReduce starts. For that, the input is automatically distributed on all available cluster nodes and the map and reduce function is executed.

Finally, once the Inverted Index processed the data, the last step is to export the data to the local file system:

```
[adminuser@localhost target]$ hadoop fs -getmerge calc calculations
```

(*calc* = HDFS folder / *calculations* = New local folder)

The output of the inverted index includes the term and the documents in which this term is included:

For example the output line "innsbruck - innsbruck.ac.at, austria.gov, cloudgene.uibk.ac.at" means that the term Innsbruck is included in three different websites.

3.3. Evaluation

Executing a MapReduce program for researchers in Genetics can be quite complicated. For them, tasks are very redundant, tedious, and complicated. For example, the user has to (1) connect to a Hadoop MapReduce cluster, (2) import data into the HDFS (in our case log data), (3) executing the use case on the command line and (4) download data to the local workstation. The knowledge the user needs in order to execute such a program would have to be fairly advanced in computer science, thus preventing scientists from using currently available and useful software solutions based on MapReduce **[**2**]**. However, computer scientists have realized a more user friendly method of executing programs graphically.

4. Integration of the inverted index into Cloudgene

As mentioned before, executing MapReduce programs on the command-line could be very hard for biologist without deeper knowledge in computer science. One possible solution to overcome this issue is Cloudgene. Cloudgene has been implemented at Innsbruck Medical University. The upcoming idea was to integrate the Inverted Index into Cloudgene to benefit from the graphical execution possibilities. Thus, the next sections give an overview about Cloudgene and describe the integration of the inverted index example into this system.

4.1. Cloudgene

Cloudgene is a free open-source platform to improve the adaptively of MapReduce programs in bioinformatics by providing graphical interfaces for the execution, import and export of data, and the reproducibility of workflows on private and public clouds **[**7**]**. The aim of Cloudgene is to construct automatically a graphical environment for the available and future MapReduce programs. One of the functionalities enables execution on private clusters, which means that

delicate datasets can be kept privately while data transfers are limited [7]. For a basic overview of the Cloudgene MapReduce interface, the window structure has a top toolbar containing buttons to submit a job (run the program) and importing big datasets automatically to the HDFS **[**7**]**. Other very useful features include account details and cluster information that tells you about how many nodes are being used **[**7**]**. Users could also notice on the upper panel the running and finalized jobs, which are presented with the name, progress and execution time **[**7**]**. Finally the lower panel shows the job-specific information that includes input/output parameters, and execution time and results **[**7**]**. Before you can run a program you must have data in the HDFS therefore the data source has to be selected **[**7**]**. Cloudgene can handle big data imports from HTTP, Amazon S3 buckets or even direct file imports from your local computer **[**7**]**. Right after running the program the process can be monitored and reviewed for viewing and/or downloaded. Figure 4.1 displays a comparison of the amount of steps it takes to run a MapReduce program the traditional way via the command-line versus Cloudgene.

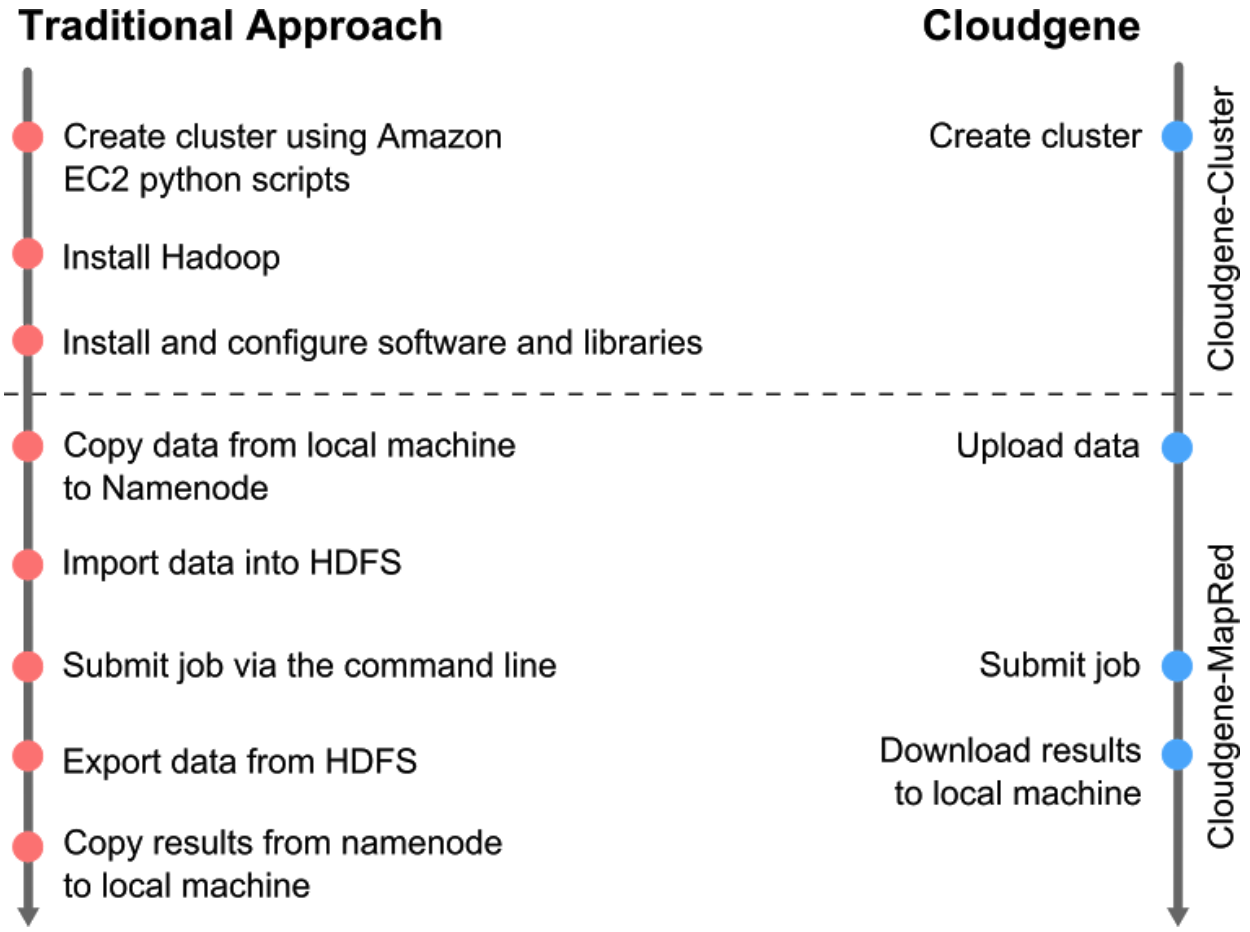

Figure 4.1: Comparison of Approaches

4.2. Integration into Cloudgene

To integrate a running command line program into Cloudgene, a YAML configuration file has to be specified. This configuration file, defines all necessary information to execute the program on a public and private cloud (cluster part) additionally, metadata and input/output parameters are included as well (mapred part). After starting Cloudgene, the new use case can be selected and a simplified way to execute MapReduce jobs has been generated from the YAML file automatically. Listing 1 shows the developed YAML configuration file to run the Inverted Index within Cloudgene.

```
name: Inverted Index
category: Hadoop Examples
version: 1.0
author: Angelo


cluster:
image: us-east-1/ami-da0cf8b3
type: m1.large,m1.xlarge
ports: 80,50030,50070
service: hadoop
installMapred: true


mapred:
jar: Examples-0.0.1-SNAPSHOT.jarinverted-index.jar
params: index.Index invertedIndex $input_folder $output

inputs:
- id: input_folder
description: Input Folder
type: hdfs-folder


outputs:
- id: output
description: Output File
type: hdfs-folder
mergeOutput: true
download: true
```

Listing 1: cloudgene.yaml file for the inverted index

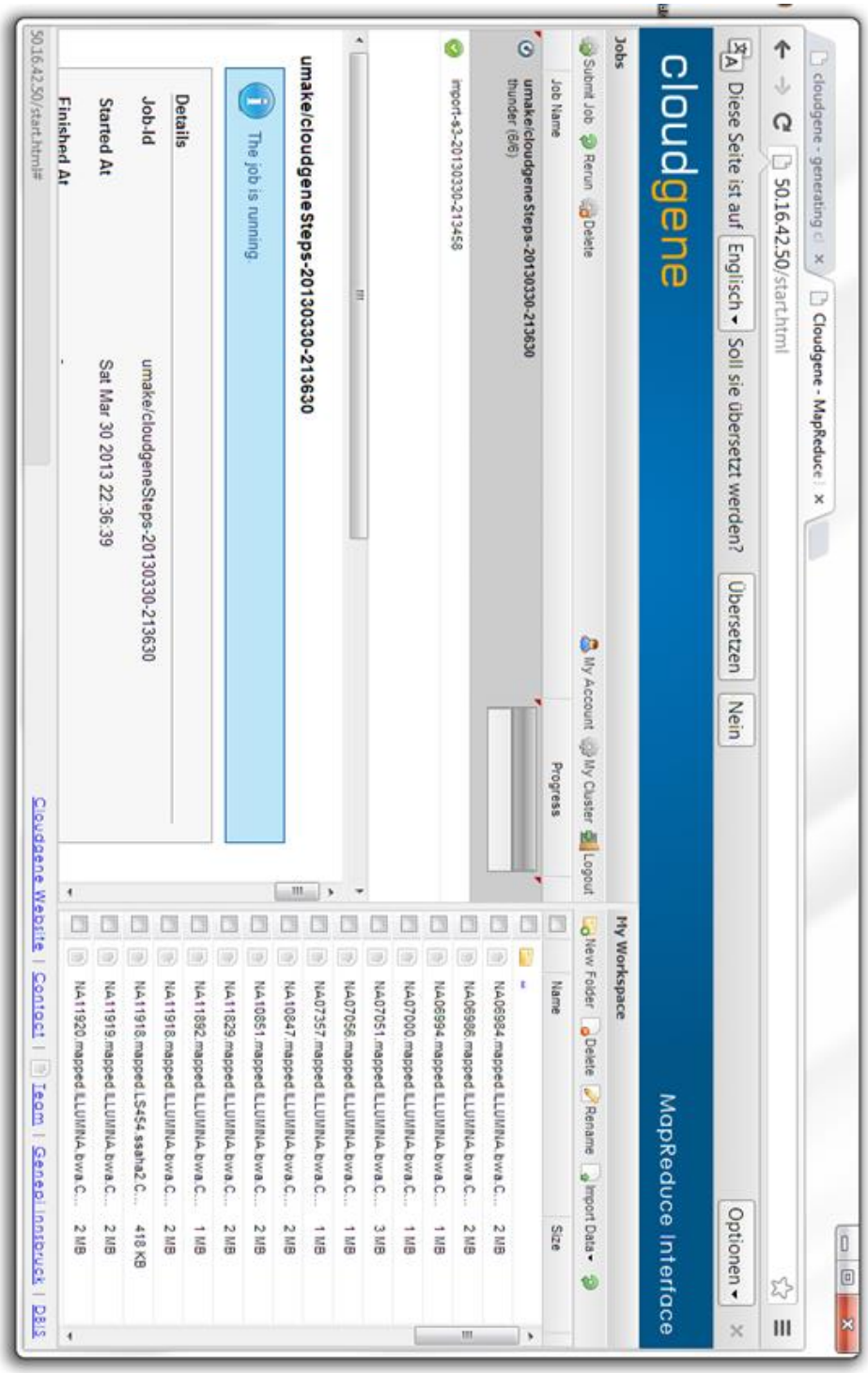

Figure 4.2.: MapReduce Interface: This is a screenshot of the MapReduce interface in Cloudgene. You can view the specific job being processed, along with a progress bar keeping track of the amount of time it takes, distinguishing between the Map and Reduce phase. Below that is another window that displays the details of the job along with its results.

5. NGS Data Quality Control

The second use case is an analysis tool to check and validate FASTQ files using MapReduce. This application was published in [7] and the aim of this task was to integrate it into Cloudgene.

5.1. Input Data

FASTQ data is simply a sequence of multi-line records. The records go in the order of the name, the DNA sequence (usually nucleotides), a separator line for optional comments, and a set of per base quality scores, see Listing 2. Each of the base quality scores actually measure the margin of error at each of the ordered bases. The errors are measured within a single character which could be interpreted on an ASCII table, and from that the base quality code can be interpreted. By finding the base code and subtracting 33 (Sanger format) from the base value you can evaluate the base quality.

```
@READNAME
GATCAGTGGGTTCAAAGCAGTATCGATCAAATAGTAAATCCATTTGTTCAACTCACAGCC
+
!''*((((***+))%+%++)(%+%%).1***-+*''))**56CCF>>>>>>CCCCCCC63
```

Listing 2: Example of 1 read as an entry in a Fastq file

5.2. Implementation

My mentors already implemented a MapReduce pipeline for FASTQ pre-processing to check the quality of large NGS datasets, in which statistics like the sequence quality, base quality, length distribution, and sequence duplication levels are calculated. After processing all sequences, a consecutive program transforms the results into meaningful plots and can finally be downloaded as pdf-files [6].

5.3. Integration into Cloudgene

To integrate this program into Cloudgene, a YAML configuration file had to be specified. This configuration file, defines all necessary information to execute the program on a public (cluster part) and private cloud (mapred part). Additionally, metadata and input/output parameters are included as well. Listing 3 shows the developed YAML configuration file to integrate this program (called exome.jar) into Cloudgene.

```
name: Fastq Preprocessing
description: Quality Control for high throughput sequence data in fastq
format.
version: 1.0.0
category: NGS
author: Sebastian Schoenherr and Lukas Forer
mapred:


 steps:


 - name: Calculating Base Quality
   jar: exome.jar
   params:  -step  base  -input  $input  -encoding  $encoding  -baseJob
$baseQual -outputLocal $output_local -outputHDFS $hdfs_tmp


  - name: Generate graphs
   jar: exome.jar
   params: -step export -outputLocal $output_local -baseJob $baseQual -
duplJob $seqDupl -lengthJob $lengthDist -seqJob $seqQual


 inputs:
   - id: input
     description: Fastq Folder
     type: hdfs-folder


   - id: encoding
     description: Encoding
     type: list
     values:
       0: Sanger/Illumina1.8+
       1: Illumina1.8-
       2: Solexa

 outputs:
   - id: output_local
     description: Output Folder Local
     type: local-folder
     download: true
```

Listing 3: cloudgene.yaml file for the NGS use case

6. Discussion and Conclusion

The idea of this project was to analyze big data by developing several use cases based on MapReduce and integrating them into Cloudgene. For that, several prerequisites needed to be fulfilled. First I had to learn several new technologies such as Hadoop MapReduce and the basics behind the Unix file system. After that, it was essential for me to learn Cloudgene's features and functionalities, developed at IMU.

Afterwards, having to comprehend the importance of Hadoop in genetics, I implemented my first use case, the "Inverted Index". The limitations of a command line execution have been analyzed and the advantages of a graphical execution with Cloudgene discovered. I was able to integrate the MapReduce program into Cloudgene. The benefit of Cloudgene is that it guides the user whether he or she has a computer science background or not and improves the usability of MapReduce programs in Bioinformatics. The last part of my project was to design a new genetic use case with the help of my mentors to process FASTQ data from next generation sequencing (NGS).

Prior to being given the opportunity to come to Austria, I had no experience regarding how I could use my computer science knowledge in a real life application. I was fortunate enough to have worked under Dr. Sebastian Schönherr who showed me how Hadoop could be used in genetics. The knowledge and experience that I gained from my stay in Innsbruck was truly enriching. First, I specifically learned the overall architecture and design of Hadoop in order to familiarize myself with the system, develop new skills and eventually make a connection between how Hadoop benefits genetics. Second, I learned how to adapt such a program like the Inverted Index to the MapReduce programming model and ultimately execute it on the terminal line as well as integrating it into Cloudgene. Lastly, I found that the opportunity fulfilling yet challenging to learn handling massive amounts of data and producing statistical output in under a few minutes.

References

[1] "Apache Hadoop." Hadoop. 30 June 2014. Web. 8 June 2014. <http://hadoop.apache.org/>.

[2] Dean, Jeffery, and Sanjay Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters." Google Research Publication (2004): 1-13. Google. Web. 18 May 2014. http://research.google.com/archive/mapreduce.html.

[3] Fries, Sergie. "MapReduce: Fast Access to Complex Data." Data Management and Data Exploration Group. 1 Jan. 2014. Web. 15 July 2014. <http://dme.rwth-aachen.de/en/research/projects/mapreduce>.

[4] "Inverted Indexes – Inside How Search Engines Work." NullWords Blog. 18 Apr. 2013. Web. 14 Sept. 2014. <http://nullwords.wordpress.com/2013/04/18/inverted-indexes-inside-how-search-engines-work/>.

[5] Marx, Vivien. "Nature." The Big Challenges of Big Data 498 (2013): 255-58. Print.

[6] "Tutorial: Low Pass Sequence Analysis." Center of Statistical Genetics. 12 May 2014. Web. 10 Oct. 2014. http://genome.sph.umich.edu/wiki/Tutorial:_Low_Pass_Sequence_Analysis#A_quick_look_to_the_fastq_files>.

[7] Schönherr, Sebastian, Lukas Forer, Hansi Weißensteiner, Florian Kronenberg, Günther Specht, and Anita Kloss-Brandstätter. "Cloudgene: A Graphical Execution Platform for MapReduce Programs on Private and Public Clouds." *BMC Bioinformatics*. 13 Aug. 2012. Web. 24 July 2014. http://www.biomedcentral.com/1471-2105/13/200

[8] Wetterstrand, Kris. "DNA Sequencing Costs." National Human Genome Research Institute. 31 Oct. 2014. Web. 15 June 2014. <http://www.genome.gov/sequencingcosts>.