Early DNA Analysis Using Incomplete DNA Data

Minfeng Li

<CE-MS-2018-22>

Abstract

In the past few years, considerable attention has been paid to reduce the computational time for the analysis of genome data, which eliminated critical computational bottlenecks in the time needed for the analysis of DNA information. However, the analysis of genome data is still facing time consuming challenges due to the slow speed of DNA sequencing machines. DNA sequencing is a time-consuming process that could take days to sequence even a single sample. This limits the speed of existing DNA analysis methods since they all need to wait for getting the full sequenced DNA data before they start the analysis. As a result, DNA analysis pipelines are not able to benefit from the reduced computational analysis time. Recently, a new method called early DNA analysis was introduced where the genome analysis pipeline is started with incomplete DNA data before all DNA sequencing finishes, which opens the door to decrease the total time consumption of DNA analysis including the sequencing time. In this thesis, a parallel implementation of the early DNA analysis approach based on the Apache Spark big data framework is proposed to improve its performance. Besides, using incomplete DNA data sets brings also a slight drop of the accuracy in genome analysis. The original method proposed a few simple methods to complete the unknown DNA data, but these can be improved to increase the accuracy. Therefore, a few new algorithms are also proposed and tested to increase accuracy in this thesis. Results show that the proposed scalability solution towards early DNA analysis could achieve a 7.6$\times$ speed-up with 97.48% correctness when deployed on a 4-node Power7+ cluster, while one of the advanced completion algorithms could increase the classification accuracy for unknown DNA data by 0.006%.

Early DNA Analysis Using Incomplete DNA Data

by

Minfeng Li

in partial fulfillment of the requirements for the degree of

Master of Science
in Embedded Systems

at the Delft University of Technology,
to be defended publicly on Thursday August 23, 2018 at 2:00 PM.

Supervisor:	Dr. ir. Z. Al-Ars	
Thesis committee:	Dr. ir. A. J. van Genderen,	TU Delft
	Dr. J. S. Rellermeyer,	TU Delft

TU Delft Delft University of Technology

Dedicated to my family and friends

CONTENTS

LIST OF FIGURES

List of Tables

ACKNOWLEDGEMENTS

During my thesis research, I received a lot of help from many people. First of all, I want to extend my heartfelt gratitude to my supervisor Dr. Z. Al-Ars, for granting me the opportunity to work on this thesis topic and his guidance during this research. Thank Dr. ir. A. J. van Genderen and Dr. J. S. Rellermeyer for attending my thesis defense as the core members of the committee. I also want to express my thanks to my daily supervisor Nauman Ahmed. He gave me the background introduction and many helpful suggestions during the thesis.

In addition, I want to thank my friends in Delft : Xiaoming Wen, Mengyu Zhang, Lichen Yao, Xiang Teng, Saiyi Wang, Xin Liu, Lu Liu, Chengqiu Zhang, Chengxin Ma. Thanks for the happiness they brought to me during this two years.

Finally, I want to thank my family, my parents. Thanks for the support and motivation they gave me all along my lifetime.

Minfeng Li
Delft, August 2018

1 INTRODUCTION

In this chapter, we start by discussing the context of this thesis. Subsequently, the current challenges in this research area are presented, which is then followed by a brief problem definition. Finally, we present the outline for this thesis.

1.1. CONTEXT

Over the past years, the cost of DNA sequencing has been dramatically decreased, which allows scientists to easily obtain genome data and analyze it in various approaches. In the field of diagnostics, a DNA sample from a patient is analyzed to detect the mutations which are considered as a potential indicator of genomic diseases. These mutations are called DNA variants, and the analysis pipeline used to identify them is called variant calling pipeline. As the cost and time for genomic diagnostics decrease, this approach can be used to serve a larger portion of patients and promises to save the lives of many people.

To obtain the DNA mutation information for further investigation, both DNA sequencing as well as DNA analysis take a huge amount of time. Although more recent high-throughput sequencing machines (such as the Illumina Hiseq 2500) have significantly reduced the sequencing time, it still consumes a week at most to sequence the whole DNA. Meanwhile, the large amounts of data generated from sequencing machines also results in a huge computational analysis time to process as well.

A lot of efforts have been dedicated to decreasing the computational time consumption of the genome analysis pipeline, from the acceleration of individual stages in the genome analysis pipeline to the acceleration of the whole pipeline based on big data techniques. DNA mapping is the first step in the analysis pipeline. An acceleration of the DNA mapping algorithm is conducted in [4]. A hardware acceleration of the variant calling algorithms is given in [5]. A cluster-based parallelized implementation of the DNA analysis pipeline is proposed in [6].

Although this effort to significantly reduce the computational analysis time of DNA has been successful, the time-consuming sequencing process still limits the capacity to obtain the mutation information within a relatively short time. The sequencing of a DNA sample takes days to complete [7], while the cluster based parallel implementation of genome analysis pipelines could finish the analysis and output the mutations information in less than two hours [3]. Considering the fact that the current sequencing technology will still dominate the market in the foreseeable future, the sequencing speed will remain a bottleneck at least for many years to come.

Reducing the sequencing time of DNA from a computer scientific point of view has not yet been fully exploited. Recently an approach called early DNA sequence analysis has been proposed by Ahmed et al. [8], that starts the variant calling pipeline using incomplete DNA data. This approach starts the analysis pipeline before DNA sequencing finished, thus partially hiding the delay from DNA sequencing. Since this approach allows for the sequence analysis to be started before DNA sequencing is completely finished, the approach has to perform the analysis without last DNA bases and their respective quality scores which should be generated from sequencing machines. A few simple algorithms (such as a consensus-based approach and a probabilistic approach) have been proposed to complete the partially sequenced DNA data by predicting unknown bases and their respective base quality scores.

1.2. CHALLENGE

Despite the fact that the input genome data of early DNA analysis is incomplete, the data generated from the DNA sequencing machine is typically still rather large, in the range of hundreds of GBs in size. To fully utilize the saved sequencing time, a scalability solution for early DNA analysis of incomplete DNA data is required. This scalability solution should be able to parallelize the completion tasks to complete the unknown part of the DNA data that are not fully sequenced. The data dependency of the completion task might limit the degree of parallelism thus affect the overall performance.

Additionally, using incomplete DNA data in genome analysis sacrifices a little in terms of variant detection accuracy. To eliminate the slight drop of accuracy in variants detection, the original work on early DNA analysis proposed a simple completion approach that can be improved and replaced by other smarter methods. There is a need to design a number of more sophisticated completion algorithms to improve the accuracy. Not only saving a huge amount of sequencing time but also assuring a high accuracy of variant detection could allow the early DNA analysis approach to be widely applied in the clinic to help medical professionals in genomic disease detection.

1.3. PROBLEM DEFINITION AND RESEARCH METHODOLOGY

The early DNA analysis approach of incomplete DNA data given by Ahmed et al. [8] can be still improved in terms of speed and accuracy. In this thesis, two major problems are addressed as key research directions.

1.3.1. SCALABILITY PROBLEM

In the early DNA analysis pipeline for incomplete DNA data, incomplete DNA data are firstly processed by a completion phase and then fed into the normal genome analysis pipeline. The large scale of genome data and its complexity make it difficult to finish the completion work in an efficient way. Therefore, the first problem of this thesis is as follows:

How could we improve the scalability and efficiency of the early DNA analysis pipeline?

Three steps are adopted to solve this problem:

1. Considering the fact that the early DNA analysis approach shares quite a few steps with the normal genome analysis pipeline except the completion phase, a precise comparison between the existing cluster-based parallel solutions for mainstream genome analysis pipeline could provide insights into a reasonable solution. The criterion of evaluating the scalable solutions of mainstream genome analysis pipelines are based on the factors below:

    - The capacity of scaling up the computational analysis of DNA data.
    - The correctness of detecting the right variants in the final output.
    - The difficulty of integrating this solution with the completion phase of the early DNA analysis approach.

2. After a thorough comparison, a scalability solution is selected and will be considered as the infrastructure of the early DNA analysis approach. This will be implemented to enable a scalable early DNA analysis pipeline while considering the two following aspects:

    - Parallelize the completion phase of early DNA analysis.
    - Integrate the scalable completion phase with the existing scalability solution of genome analysis.

3. Finally, the performance of the scalable early DNA analysis approach is evaluated and analyzed. Two factors are taken into consideration in the evaluation:

    - Speed of the scalable early DNA analysis implementation.
    - Results consistency compared to the original early DNA analysis.

1.3.2. ACCURACY PROBLEM

The early DNA analysis approach paves a new way to save massive DNA sequencing time by predicting the unknown bases and their quality scores for incomplete DNA data. The accurate prediction guarantees the effectiveness of this early DNA analysis approach. This represents the second problem statement of this thesis, which can be described as follows:

How could we improve the accuracy of the early DNA analysis approach using more advanced algorithms?

As is widely acknowledged, machine learning methods nowadays are commonly used in many areas and illustrated as a strikingly effective solution in many research fields. Instead of using the simple consensus model in the original early DNA analysis pipeline, machine learning methods may give us better correlation between the incomplete DNA sequence and the fully sequenced genome data. A few machine learning algorithms will be tried to solve this problem.

1.4. THESIS OUTLINE

This thesis is organized as follows. In Chapter 2, the background information of this thesis is described. Chapter 3 discusses a number of alternative solutions from the literature to address the thesis problem definition. This chapter also identifies the most relevant solutions to the problem. Next, Chapter 4 illustrates the implementation details of the selected scalability solution. Then the measurement results and discussion of this scalability solution are shown in Chapter 5. Afterwards, the experiment results towards the accuracy problem of the thesis are presented in Chapter 6. At last, Chapter 7 summarizes the conclusion of the thesis and suggests future work.

2
Background

In this chapter, a few background topics related to this thesis are introduced for better understanding the problem and goal of this thesis. Firstly, the history and working mechanism of DNA sequencing machines are introduced in Section 2.1. After that, current DNA analysis pipelines are discussed in Section 2.2. Different stages of the pipeline are described in this section. In Section 2.3, the early DNA analysis approach that could partially hide DNA sequencing delay is introduced. At last, different big data technologies and their features are discussed.

2.1. Genome Sequencing & DNA Data

2.1.1. DNA Sequencing

DNA has a long-chain structure composed of different kinds of nucleotides. The nucleotide of DNA consists of a deoxyribose sugar molecule, a phosphate group, and one of four nitrogenous bases (called a *base* for short in biology): adenine (A), cytosine (C), guanine (G), thymine (T). The type of nitrogenous base decides the type of nucleotide. [9]

Figure 2.1 presents the actual structure of a partial DNA molecule. The order of nitrogenous bases is encoded to determine gene expression of creatures, thus unblocking the bottleneck of a wide variety of research fields.

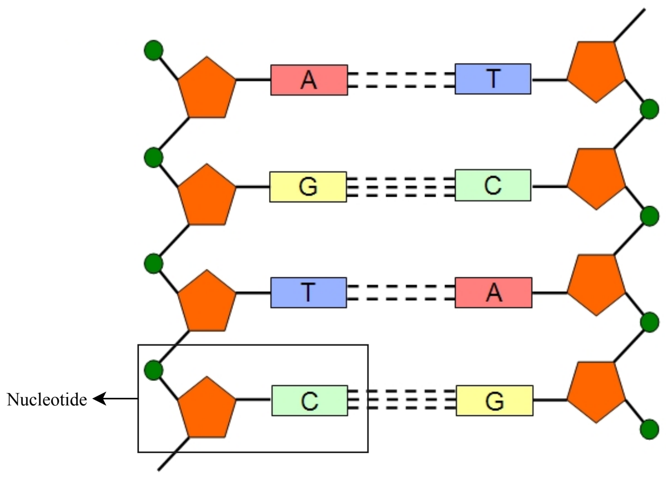


Figure 2.1: DNA structure from [1]

Over the last fifty years, sequencing the order of DNA bases has been widely explored in so many countries. A number of researchers over the world dedicated themselves to the innovative productions on this topic. In general, there are two types of sequencing: whole genome sequencing and whole exome sequencing. The whole genome sequencing sequences all the DNA bases in the sample while the whole exome sequencing only sequences the bases of DNA exome. DNA exome is the subset of DNA that encodes proteins, thereby determines gene expression [10].

The tremendous achievements have been made during the long time-scale, from sequencing a very short piece to millions of bases, from limited capacity of sequencing a single gene to widely available whole genome sequencing [11]. The first generation sequencing technology starts the history of sequencing DNA sample. The length of the sequenced DNA could reach 1000 bases with more than 99.99% accuracy [12]. However, the lack of high throughput capacity and its significant cost impede its wide usage, especially in the commercial market. Along with more advanced sequencing methods, the second generation sequencing machines, also known as NGS (Next Generation Sequencing) machines come out. These machines significantly reduce the time needed for the whole genome sequencing from 3 years to days and dramatically decrease the price from billions of dollars to a thousand dollars [13].

The second generation sequencing machines (e.g., Illumina) divide DNA into fragments. These fragments usually range from 200 bases to 500 bases in length. After DNA fragmentation, machines sequence these DNA fragments from both sides to generate paired-end DNA reads. This paired-end sequencing technique brings more accuracy to DNA analysis since it provides a better chance to correctly identify the location and composition of DNA fragments as compared to single-end sequencing [14]. Sequencing by Synthesis (SBS) is the sequencing technology proposed by Illumina and widely adopted in NGS machines. In SBS, sequencing machines generate the first read and the second read in paired-end data by sequencing DNA fragments from both forward and reverse direction. It is a sequential process such that sequencing the first read is followed by sequencing the second read. Moreover, thousands of DNA fragments are sequenced simultaneously to enable the high-throughput sequencing. Sequencing one base in the DNA fragments will generate two output:

1. The actual type of nitrogenous base.

   There are typically four kinds of bases on the DNA sequence: adenine (A), cytosine (C), guanine (G), thymine (T).

2. The corresponding base quality score.

   This score indicates the probability of error in sequencing the base.

After sequencing, the sequenced output is stored in a software file and then analyzed by genome analysis pipelines.

2.1.2. DNA DATA

Due to the methods of sequencing, DNA fragments are usually over-sampled to reduce sequencing error. Multiple short reads are generated for the same genome region in the DNA sample. Normally this over-sample coverage could reach from 30x to 80x depending on the experimental situation. This coverage number represents the average measurement times for each base of the genome. Besides the coverage, the length of sequenced reads is also commonly used to describe DNA data. Illumina provides different kinds of Reagent Kits [15] that support different output read length. The length of DNA reads sequenced by Illumina sequencing machines varies from 50 to 300 base pairs (bp) [16], either single end or paired end.

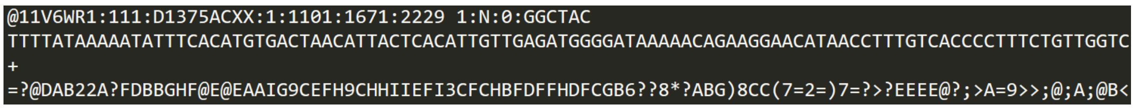
Nowadays a standard file format called FASTQ is used to store the sequenced DNA reads. In paired-end sequencing, the first read and the second read will be placed into two separate FASTQ files. In FASTQ files, both the sequence, namely the order of DNA bases, and their quality scores are given. Each sequenced DNA read is presented as a header line, a string of bases and a string of base quality scores. In paired-end sequencing, the paired two reads are named with an identical sequence identifier in the header line.

Besides the actual base type, a quality score is also given when sequencing a base. This score illustrates the confidence towards the sequenced results from sequencing machines and is calculated as the following method:

$$Q(A) = -10\log_{10} P(A) \tag{2.1}$$

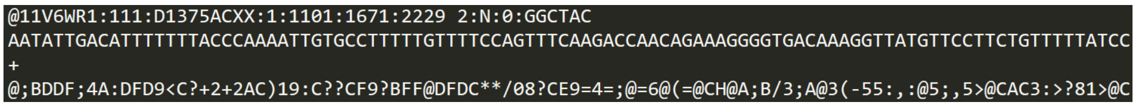
where P(A) describes the estimated probability of error that sequencing machine made in a particular measurement. This value varies within the range from 0 to 40, expressed as a character symbol from '!' to 'I' according to the ASCII table. Therefore, the quality score of a read is described as a string of ASCII characters in the FASTQ file.

A typical example of 100 base pairs DNA read is shown below in Figure 2.2. The header line is given first and followed by the actual sequence of this read. A plus symbol then separates the string of base quality scores of this read. A FASTQ file contains millions of reads like this. The second read generated from the same DNA fragment in this paired-end data is stored in another FASTQ file with same sequence identifier "@11V6WR1:111:D1375ACXX:1:1101:1671:2229" as presented in Figure 2.3.



```
@11V6WR1:111:D1375ACXX:1:1101:1671:2229 1:N:0:GGCTAC
TTTTATAAAAATATTTCACATGTGACTAACATTACTCACATTGTTGAGATGGGGATAAAAACAGAAGGAACATAACCTTTGTCACCCCTTTCTGTTGGTC
+
=?@DAB22A?FDBBGHF@E@EAAIG9CEFH9CHHIIEFI3CFCHBFDFFHDFCGB6??8*?ABG)8CC(7=2=)7=?>?EEEE@?;>A=9>>;@;A;@B<
```

Figure 2.2: A example of 100 bp read stored in FASTQ file



```
@11V6WR1:111:D1375ACXX:1:1101:1671:2229 2:N:0:GGCTAC
AATATTGACATTTTTTTACCCAAAATTGTGCCTTTTTGTTTTCCAGTTTCAAGACCAACAGAAAGGGGTGACAAAGGTTATGTTCCTTCTGTTTTTATCC
+
@;BDDF;4A:DFD9<C?+2+2AC)19:C??CF9?BFF@DFDC**/08?CE9=4=;@=6@(=@CH@A;B/3;A@3(-55:,:@5;,5>@CAC3:>?81>@C
```

Figure 2.3: The second read in this paired-end data

2.2. GENOME ANALYSIS PIPELINE

With the rapid development of NGS sequencing machines, research towards the analysis of DNA data also attracts extensive interests. By comparing the sample DNA with the reference human genome database, different kinds of variants in the sample DNA can be found, i.e., SNPs (single nucleotide polymorphisms) and INDELs (insertions/deletions). These variants are then analyzed by doctors to decide whether this sample person gets diseases. The pipeline to detect these variants are called variant calling pipeline.

Over the years, a wide variety of analyzing tools have been built in different stages of the pipeline. A comparison of the mainstream variant calling pipelines using gold standard personal exome variants are presented in [17]. The performance results show their advantages and disadvantages in different situations. Most of the DNA analysis pipelines contain two major parts, data pre-processing and variant calling. Data pre-processing part is always initiated with an alignment phase, or called DNA assembly, then followed by several data clean up steps to prepare the data for variant calling. The variant discovery phase identifies the genomic variations and determines the output of the whole pipeline. Nowadays, GATK (Genome Analysis Toolkit) best practices pipeline has been widely used in many genomic diagnostics. Figure 2.4 presents the workflow of the latest version of GATK best practices pipeline. Details of each step are presented in this section.

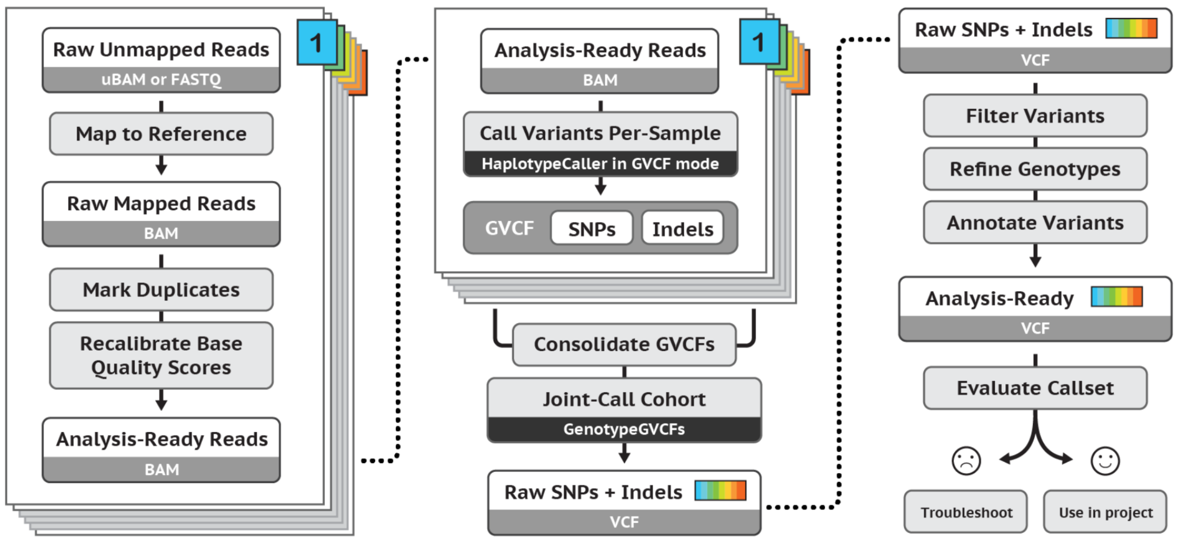


Figure 2.4: GATK best practices pipeline [2]

2.2.1. DNA MAPPING

DNA mapping, also called DNA assembly or DNA sequence alignment, reconstructs the whole DNA sequence from short DNA reads. To compare the sequenced DNA reads with the reference genome for variants discovery, the first step of DNA analysis pipeline is to align these short DNA reads against the reference genome. A few tools are available in this stage, bwa-mem, bowtie2, novoalign and so on [18]. After DNA assembly, all the reads are associated with their position information in the human reference genome. A text-based file format SAM (Sequence Alignment Map) is widely used to store the aligned sequence data, which is then compressed as BAM (Binary Alignment Map) file for later analysis. In the SAM file, apart from the actual sequence and base quality scores, the mapping quality and position information are also given for each read. After alignment, a sorting phase is performed to sort all the reads according to their coordinate, which prepares the data for clean-up stages.

2.2.2. DATA CLEAN UP

After DNA assembly, the genome analysis pipelines attempt to eliminate the influences of errors that sequencing machines made in the sequencing process.

During sequencing, the sequencing machines over-sample the DNA fragments and generate multiple copies of a single DNA read. The influence of these identical copies on different types of variant calling is evaluated in [19]. These copies bring adverse effect during variation detection. This is because the reliability score of each variant detected in variant calling stage is based on the number of times this variant has been observed. Therefore, these identical copies are marked as duplicates and later reduced to a single one in data clean-up part using tool like PICARD.

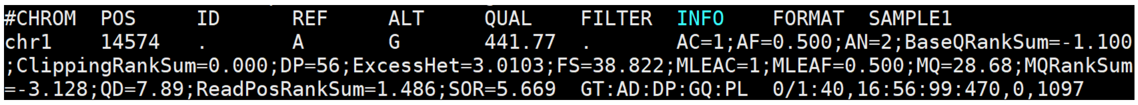
Afterwards, a step called indel-realignment is followed to realign the reads around the INDELs, because the previous sequence alignment stage might have misaligned these reads around this area due to the long insertions or deletions.

Finally, the base quality recalibration stage is performed to correct the quality scores for those bases that were measured with incorrect quality scores by the sequencing machines. Base quality recalibration tries to correct the erroneous quality scores by looking at the human genome databases. The most common variants detected so far have been put in these databases. The mapping differences of the sample DNA absent in these databases will be classified as sequencing errors.

In fact, the data clean-up stage differs in different situations depending on the research proposal. Researchers could decide their own strategy when preparing the data for final variant calling.

2.2.3. VARIANT DISCOVERY

The final step of the pipeline is to detect all variants in the DNA samples. GATK provides two variant calling module: UnifiedGenotyper and HaplotypeCaller. Although HaplotypeCaller is far more computationally expensive, its good accuracy has been widely acknowledged in detecting both SNPs and INDELs. Thus it is commonly used in most of the variant calling pipelines in recent years. A VCF (Variant Call Format) file is saved to store all detected variants from HaplotypeCaller as the final output. An entry in the VCF file looks like below:



```
#CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO    FORMAT  SAMPLE1
chr1    14574   .       A       G       441.77  .       AC=1;AF=0.500;AN=2;BaseQRankSum=-1.100
;ClippingRankSum=0.000;DP=56;ExcessHet=3.0103;FS=38.822;MLEAC=1;MLEAF=0.500;MQ=28.68;MQRankSum
=-3.128;QD=7.89;ReadPosRankSum=1.486;SOR=5.669  GT:AD:DP:GQ:PL  0/1:40,16:56:99:470,0,1097
```

Figure 2.5: An example of variant in the VCF file

This entry describes a SNP variant where the actual base type adenine (A) found in the reference genome is detected as a thymine (T) in the sample DNA. This variant is detected at position 14574 on chromosome 1 with reliability score of 441.77. Doctors will diagnose diseases by analyzing these variants in the VCF file.

2.3. EARLY DNA ANALYSIS

The mainstream genome analysis approaches wait for the full sequencing results and then start running the analysis pipeline. Although the time needed to run the pipeline has been dramatically reduced in the last few years, the relatively slow process of DNA sequencing limits the real-time capacity of genomic diagnostics.

Sequencing a base consumes around half an hour [7]. As a consequence, saving the sequencing time for even just a few bases could achieve a huge time deduction. In paired-end DNA sequencing, a DNA fragment is sequenced from forward direction as the first read, then sequenced from the other end as the second read. In the early DNA analysis approach proposed by Ahmed et al. [8], the sequencing latency is reduced by starting DNA analysis before the second read in paired-end data is fully sequenced. As sequencing the first read and the second read is a sequential process, the analysis hides part of the time needed for sequencing the second read. The genome analysis is started much earlier than the normal situation where the genome analysis pipelines are waiting for the complete results from DNA sequencing machines. Figure 2.6 shows the designed scheme of the early DNA analysis approach. The sequenced bases are streamed out of the machine and written into two FASTQ files, one for the first read, the other one for the second read. After the proposed stopping length of the second read is reached, the existing FASTQ files and the quality profiling of the first read are used as input to start the early DNA analysis.

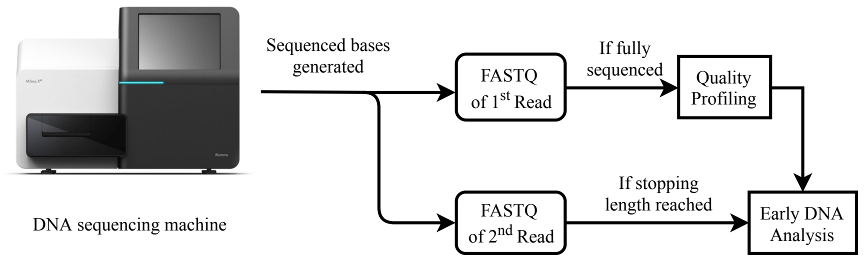


Figure 2.6: Scheme of the early DNA analysis approach

Since the early DNA analysis approach does not wait for the second read completely sequenced, the values of the last bases of the second read and their corresponding base quality scores are unknown for the genome analysis. Therefore, a completion stage was designed in the early DNA analysis approach to complete the values of the unknown bases of the second read and their corresponding base quality scores. Two workflows as shown in Figure 2.7 were also proposed to verify the effectiveness of this completion stage in their research. Details of the completion methods are presented in Section 2.3.1 and 2.3.2.

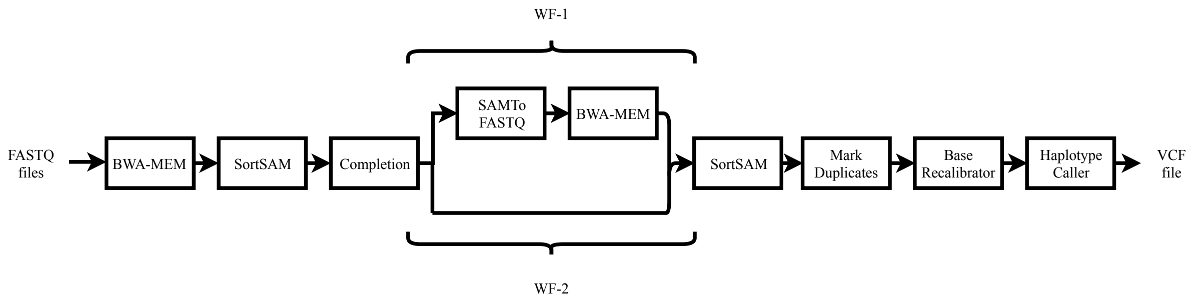


Figure 2.7: Two workflows of the early DNA analysis approach

Workflow-1 (WF-1) and Workflow-2 (WF-2) both share the same first two steps as compared to standard genome analysis pipelines. The fully sequenced first read and the incomplete second read are firstly aligned against the reference genome. After obtaining the position information of each read, a SortSAM step is performed using PICARD tool to sort all the reads according to their positions. Subsequently, the completion stage is performed to complete the unknown bases of the second read and their base quality scores. In WF-1, after the completion stage, both the fully sequenced first read and the complete second read are ready to be used to detect variants. The aligned and complete DNA reads are converted back to raw reads by a SAMToFASTQ stage. Two FASTQ files containing full DNA reads are output from this stage. Afterwards, all the remaining steps are entirely the same as a standard genome analysis pipeline. In WF-2, the completion stage also completes and corrects additional fields of the input SAM file besides the unknown bases and their quality scores. This additional completion eliminates the necessity to redo the sequence alignment job and only requires another sorting step before data clean-up and variant calling.

2.3.1. COMPLETION OF UNKNOWN BASES

The method to complete the unknown bases of the second read is to look at the overlapping between the incomplete second read and the reads which are mapped closely after DNA mapping. Figure 2.8 describes the situation where several reads are mapped around a near location against the reference genome. The black read is an example of an incomplete second read while all the gray reads are the reads having overlap with the black read. The reads containing no dotted lines are the fully sequenced first reads. The dotted line in a read presents the unknown bases that are not sequenced in a second read. The arrow describes the mapping direction.

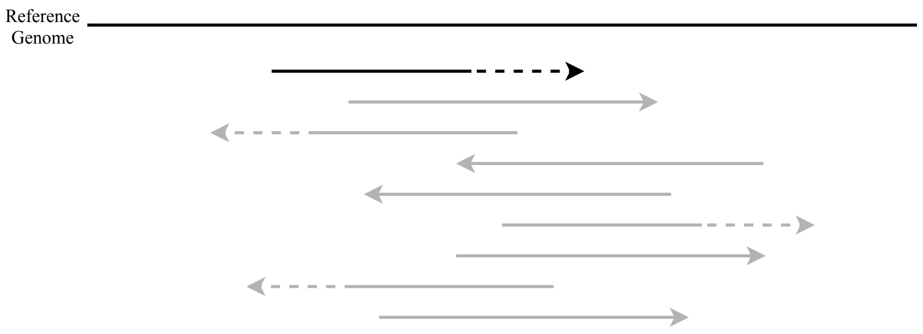


Figure 2.8: Closely mapped DNA reads

Three completion schemes were presented in the early DNA analysis approach:

1. The consensus-based approach.

   For each unknown base in the second read, the majority vote of the bases from the overlapping reads decides the outcome. For an unknown base having no overlap, the actual base located at the corresponding position of the reference genome will be adopted as the outcome. These outcome bases are then appended back to the incomplete second read to form a complete sequence.

2. The approach based on matching rate.

   Instead of taking the majority vote, the overlapping read that has the best matching rate with the known part of the second read will decide the outcome. The corresponding sequence of the unknown positions on this best-match read will be used to complete the unknown bases. If the unknown bases are not entirely covered by this best-match read, the consensus-based outcome will be adopted to complete all those positions.

3. The probabilistic approach.

   For each unknown base, the ratio distribution of the four base types in the overlapping reads decides the probability of a base being predicted at that unknown position.

2.3.2. COMPLETION OF UNKNOWN BASE QUALITY SCORES

In the early DNA analysis, the fitted curve for the average quality scores of the first read and the second read across the read length was found almost the same. To complete the unknown base quality scores, the profiling of the first read is performed after obtaining the fully sequenced first read. This profiling information is presented as a piecewise linear function that describes the linearity between the quality scores and the positions on the full read. Since the second read shares the same linearity, this piecewise linear function is then used to complete the unknown base quality scores of the second read.

2.4. BIG DATA SCALABILITY

Around 90% of the data in the world today are created in the last two years. The huge-scale data exploration and utilization requires ultra-efficient processing capacity. Over the past fifteen years, Big data technologies have been progressed with tremendous achievements such that the exploration of large-scale data-intensive applications become possible.

2.4.1. Hadoop MapReduce

MapReduce is one of the widely used programming models that manage to process large data sets on multiple computing nodes of a cluster. This MapReduce programming model is composed of two sub-steps: a map step and a reduce step. The map tasks firstly convert the input data into a set of key-value pairs and subsequently produce a set of intermediate key-value pairs depending on the application requirements. These map tasks are executed separately on each node of the cluster. Afterwards, the pairs with the same key are merged together and processed by the reduce tasks. A single output will be produced by the reduce tasks in the end. Apache Hadoop is an open source implementation of this MapReduce programming model. It contains three core components:

1. Hadoop Distributed File System (HDFS).

   HDFS guarantees the scalable, fault-tolerant, and cost-efficient storage of large data sets among the nodes in the cluster.

2. Hadoop YARN.

   Hadoop YARN is a resource manager that can allocate the resources and distribute the tasks in the cluster. It is produced in Hadoop 2.0 framework and replaces the JobTracker and TaskTracker in Hadoop 1.0.

3. Hadoop MapReduce.

   Hadoop MapReduce is the same programming model mentioned above that processes the large-scale data by map and reduce tasks.

Besides the core components, Hadoop is also equipped with other components for specific applications. For example, Hive creates a data query scheme similar to standard SQL, which allows SQL developers to build their applications on HDFS. HBase is a database management system that runs on top of HDFS. The whole Hadoop ecosystem provides the solutions to a wide variety of big data problems.

The disadvantages of the MapReduce framework can be addressed in two aspects. On the one hand, only map and reduce tasks are supported in this programming model. Other transformations are only available if the map or reduce functions are modified, which limits the flexibility of the applications. On the other hand, the MapReduce framework is a disk-oriented framework. All the intermediate files generated from the map tasks will be written into the disk and then handled by the reduce tasks. This scheme brings extensive disk access operations and decreases the overall performance.

2.4.2. Spark

To address the disadvantages of Hadoop MapReduce, Apache Spark is produced. Besides the map and reduce functions, more transformations are provided, such as join, filter, intersection, etc. This enables more convenient development for developers as compared to MapReduce. Meanwhile, Spark performs the transformations and saves the output in memory to avoid the intensive disk access overhead. Therefore, Spark could achieve a 100x speed-up as compared to MapReduce for a specific application in [20]. If the size of the data does not fit in the memory, it can still be written into the disk for next step calculations. Additionally, Spark provides the flexibility to implement the applications in various programming languages, i.e., Scala, Java, and Python. An interface is proposed to integrate different programming languages. Therefore, developers could adopt the most suitable language for different applications.

3
ALTERNATIVE SOLUTIONS

In this chapter, a few scalability solutions of the normal genome analysis pipeline are discussed first. The comparison of these solutions provides the insight into the parallel implementation of the early DNA analysis pipeline. Afterwards, the criteria to evaluate the scalability solution of the early DNA analysis approach are presented in Section 3.2. At last, a few machine learning algorithms considered as the potential solution to the accuracy problem of the early DNA analysis approach are introduced in Section 3.3.

3.1. SCALABILITY SOLUTIONS

Over the years, the acceleration for the genome analysis pipeline is conducted in two ways. On the one hand, a lot of researchers focus on the hardware acceleration of the different algorithms used in the pipeline based on FPGA or GPU, i.e., [21]. The hardware acceleration enables the fast genome analysis for each independent module of the pipeline. On the other hand, the big data technologies are used to process the DNA data in a distributed manner. The scalability solutions manage the computing resources and parallelize the tasks over the cluster to achieve speed-up. Since the goal of this project is to build a parallelized early DNA analysis pipeline, a few scalability solutions of the normal genome analysis pipeline are analyzed first as a reference.

3.1.1. CHURCHILL

The first example is Churchill from Kelly et al. [22]. They come up with an approach that utilizes the computational resources in the cluster to parallelize the analyzing tasks. The following four issues are addressed to decide their parallelization strategy.

1. Parallelization.

   The human genome is not balanced since each chromosome has a different length. The longest chromosome in the human genome is chromosome 1, which is five times longer than the shortest chromosome 21. As a result, dividing the raw data by their chromosome locations is not a suitable method for parallelization. A balanced parallelization strategy could improve the overall performance.

2. Dependencies.

   Parallelization brings not only faster-processing capacity but also the problem of data dependency. In some stages of the original genome analysis pipelines, for example, duplicates marking and base recalibration, the whole data sets are suggested as the input. Parallelization of the work splits the entire data sets such that an innovative method is required to remain the data dependency.

3. Determinism.

   Ideally, the parallelization solution will produce the same results with different parallelization set-up. The absolute determinism is essential for the situation where the clinical diagnostics requires 100% confidence.

4. Interchromosomal reads.

   In the original genome analysis pipelines, one large BAM file is fed to the stages after DNA mapping. The interchromosomal reads where the paired two reads are mapped to different chromosomes are still located in the same file. However, the parallelization solution divides the genome into sub-regions thus separates these interchromosomal reads, which could result in information lost as they are processed in a distributed way.

Considering all the factors above, a novel strategy is proposed in Churchill. There are five main steps in the Churchill workflow:

1. Parallelized alignment to a reference sequence.

   Usually, the input data for the mainstream genome analysis pipelines is very large paired FASTQ files. Commonly used DNA sequence alignment tools, such as BWA-MEM, enable the multi-thread running to parallelize the tasks. Running this alignment task on a single machine with multi-threading capacity can still be improved in terms of parallelization. In Churchill, the large input FASTQ files are further split into multiple smaller FASTQ files. This can be directly achieved by controlling the sequencing machines without extra split work. The sequencing machines allow defining the maximal number of reads can be stored in a FASTQ file. Afterwards, these smaller chunks are processed separately in the computing cluster. Utilizing the computing resources among the cluster definitely brings faster processing speed as compared to running the alignment task on a single machine.

2. Parallelized generation and deduplication of subregional BAMs.

   The generation of subregional BAM files is the core step of the Churchill pipeline. The raw BAM files generated from the previous alignment step need to be further divided according to the subregional locations. The whole genome is divided into M sub-regions, where the value of M is pre-defined as the degree of parallelization. The long chromosomes, such as chromosome 1 and chromosome 2, are divided into multiple regions, but the short chromosomes are combined as a sub-region. Each BAM file generated from the alignment stage will be divided into M split parts, creating $N \times M$ split files where N is the number of input BAM files. In each input BAM file, the interchromosomal reads where the pairs are mapped to different sub-regions will be placed into an extra BAM file called *chrI*. Afterwards, the split files corresponding to the same sub-region will be merged into M subregional BAM files. Each subregional BAM file contains only the reads within the boundaries.

   As for deduplication, the parallelized deduplication tasks are performed in the cluster for all the subregional BAM files and BAM file *chrI* containing interchromosomal reads. The extra deduplication for interchromosomal reads overcomes the limit of parallelization in the original genome analysis pipelines. Since the structural variants are usually detected in the interchromosomal reads, this novel strategy also remains the accuracy to identify the structural variants.

   After deduplication, the interchromosomal reads are individually merged back to their corresponding subregional BAM files for later processing.

3. Parallelized local realignment around indels.

   In this stage, an indel-realignment stage is performed for all BAM files. The reads containing INDELs are possibly misaligned in the previous alignment stage. Churchill parallelizes the local realignment tasks across the sub-regions, resulting in significant improvements in speed.

4. Parallelization of base quality score recalibration.

   The base quality scores measured by sequencing machines are known to be inaccurate in some circumstances. These wrongly estimated scores must be recalibrated before calling variants. The covariation of a base such as the quality score, the actual position within the read, and its preceding base, are analyzed by GATK base quality score recalibration (BQSR) algorithm. These covariates are collected across the genome for accurate calculation in original genome analysis pipeline. In Churchill, this dependency is handled by merging all the covariates from each sub-region. Subsequently, the recalibration work is parallelized for each sub-region.

5. Parallelization of variant calling.

   The variant calling tasks are conducted in each sub-region using GATK UnifiedGenotyper or HaplotypeCaller. The output VCF files containing both SNPs and INDELs from each sub-region are merged as the final output in the end.

3.1.2. HALVADE

The second scalability solution is Halvade from Decap et al. [23]. In Halvade, a DNA variant calling pipeline is implemented according to the GATK best practices recommendations. Table 3.1 lists the steps and the used tools in that pipeline. Halvade framework executes all these tasks in parallel on a multi-node cluster to achieve acceleration. This solution is based on a pure intuition that the alignment task for a certain read is independent of the alignment for another read. Meanwhile, the variant calling task for a certain chromosomal region is independent of the variant calling job in another region. Therefore, the raw input data sets can be divided into multiple subsets such that the analyzing tasks will be performed in a distributed manner for each subset.

Table 3.1: Different steps and tools involved in the GATK best practices pipeline

Step	Tool
DNA mapping	BWA-MEM
Reads sorting	PICARD-SortSam
Duplicates marking	PICARD-MarkDuplicates
INDELs realignment	GATK
Base quality recalibration	GATK
Variant calling	GATK

Halvade is a scalability solution based on Hadoop MapReduce framework. As discussed in Section 2.4.1, MapReduce programming model is composed of two part: a map phase and a reduce phase. In Halvade, the map phase corresponds to the sequence alignment step while the variant calling jobs are finished in the reduce phase. The tasks to sort the mapped reads according to their chromosomal positions are completed between this two phases. Figure 3.1 presents the framework of Halvade. Details are described as follows.

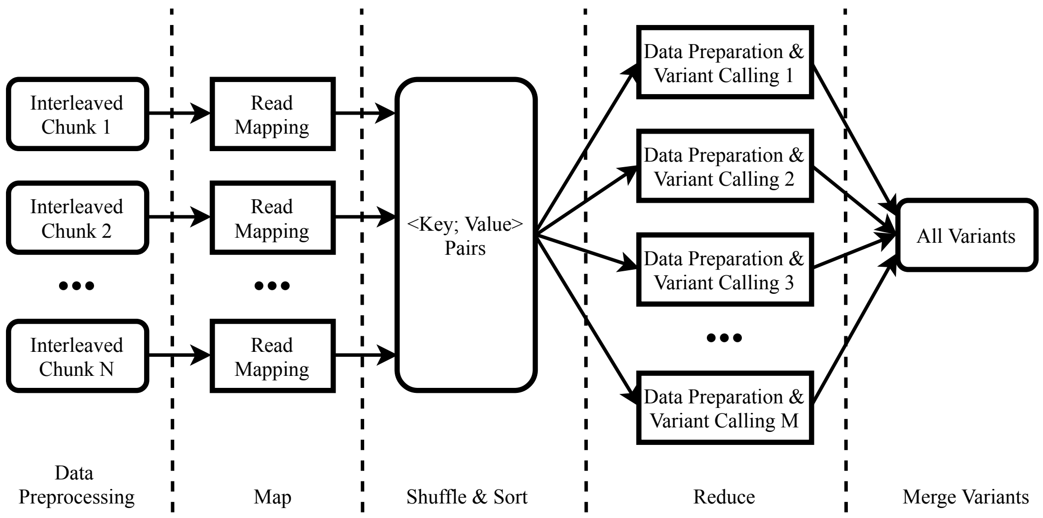


Figure 3.1: The framework of Halvade

1. Input data preparation.

   The parallelization of the map tasks requires to split the entire input datasets into multiple subsets. A separate tool called 'Halvade Uploader' is built in Halvade to interleave the paired reads and create smaller chunks. In each chunk file, the paired reads are placed next to each other. If two paired reads are stored in separate chunk files, the sequence alignment job would fail when executed. The number of chunk files produced in this step is equal to the total number of map tasks to be executed in the map phase. The created chunks are stored in the file system that all the nodes in the cluster could access. Depending on the running environment, it is either HDFS or Amazon EMR.

2. Map phase - read alignment.

In the map phase, each input chunks created in the last step is processed by a map task. The map tasks are executed in parallel on the work nodes of the cluster. Hadoop framework preferably wants to execute a map task on the node that contains the input data locally (as a part of HDFS) to minimize the communication overhead. In each map task, each read in that input chunks is stored as the value of a <key, value> pair using Hadoop-BAM [24] API. These pairs are then assigned to alignment task for getting position information. After alignment, the chromosome number, the position number within that chromosome, and the corresponding sub-chromosomal region number are used as the key for each pair; the value is replaced by the SAM record that contains the read itself and its metadata. The chromosomal region number depends on the actual position of this read within the whole chromosome. The whole chromosome is segmented into multiple sub-chromosomal regions according to the length. Thus, the relatively long chromosomes are divided into more sub-regions. For the read of which the mate read is mapped to a different chromosomal region, two <key, value> pairs will be produced such that this SAM record appears twice, once in each sub-region. This redundancy guarantees the performance of the following variant calling phase in which a degree of data dependency is required.

After map tasks completed, the intermediate pairs that belong to the same chromosomal region will be grouped as a single input of a reduce task. In each chromosomal region, a sorting step is also performed to sort all the SAM records in this region by their position number before the reduce phase, which corresponds to the reads-sorting step in the GATK best practices pipeline.

3. Reduce phase - variant calling.

   After grouping and sorting all the SAM records for each chromosomal region, reduce tasks are executed in parallel in the cluster. Each reduce tasks will produce a VCF file containing all the variants detected in that chromosomal region. These VCF files are merged as a single output in the end by another MapReduce job by Halvade. For the variants that are detected twice in two sub-regions due to the redundancy strategy in map phase, Halvade provides two options: either only keep the variant with higher reliability score, or keep both variants in the final output.

3.1.3. SPARKGA

Spark framework has also been explored to accelerate the genome analysis pipeline. A "Cluster-Based Apache Spark Implementation of the GATK DNA Analysis Pipeline" is proposed in 2015 by Mushtaq et al. [6]. This approach divides the chromosomal regions by not only the length of the chromosome but also the actual number of mapped reads in a sub-region. This strategy brings a more balanced load distribution for variant calling jobs. Nevertheless, fully committing the load balancing job to the memory in this approach brings the Out of Memory (OOM) error for large input genome data.

To solve the problem above, SparkGA comes out also from Mushtaq et al. [3]. By doing the load balancing job in a memory efficient way, SparkGA could further divide the sub-regions that contain too many reads into more segments without OOM error. SparkGA classifies the whole pipeline into three phases: DNA mapping and static load balancing; sorting and dynamic load balancing; duplicates marking and variant calling. To fully optimize system utilization, SparkGA enables the dynamic environment configuration for each phase in the pipeline as they have different computational requirements. SparkGA will read an XML file and alter the settings, such as the memory size, the number of threads, the number of instances, etc., for each specified phase.

SparkGA is also an implementation based on GATK best practices pipeline. Table 3.2 lists the tools used in the original GATK pipeline and SparkGA. As can be seen, only the sorting phase in SparkGA is not inherited, which leaves other developers a free space to modify the pipeline if more advanced tools are invented. Figure 3.2 presents the workflow of SparkGA. The whole pipeline is composed of three core phases. Details of each phase are discussed below.

1. DNA mapping and static load balancing.

   The same as Halvade, the entire input data sets are divided into multiple interleaved chunks where the paired reads are stored in the same chunk file. In this phase, a sequence alignment task is performed for each input chunk. The alignment results are stored in the SAM file. The mapping tasks are executed in parallel in the cluster. Static load balancing is achieved by equally dividing the chromosome into multiple sub-regions according to their length. In each chunk, the reads that are mapped to the same sub-region will be stored in the same SAM file. This step will create $N \times M$ SAM files at most, where the value of N is the number of input chunks, the value of M is the number of sub-regions that can

Table 3.2: Different tools used in the original GATK best practices pipeline and SparkGA

Step	GATK best practices	SparkGA
DNA mapping	BWA-MEM	BWA-MEM
Reads sorting	PICARD-SortSam	Sorting in Scala
Duplicates marking	PICARD-MarkDuplicates	PICARD-MarkDuplicates
INDELs realignment	GATK	GATK
Base quality recalibration	GATK	GATK
Variant calling	GATK	GATK

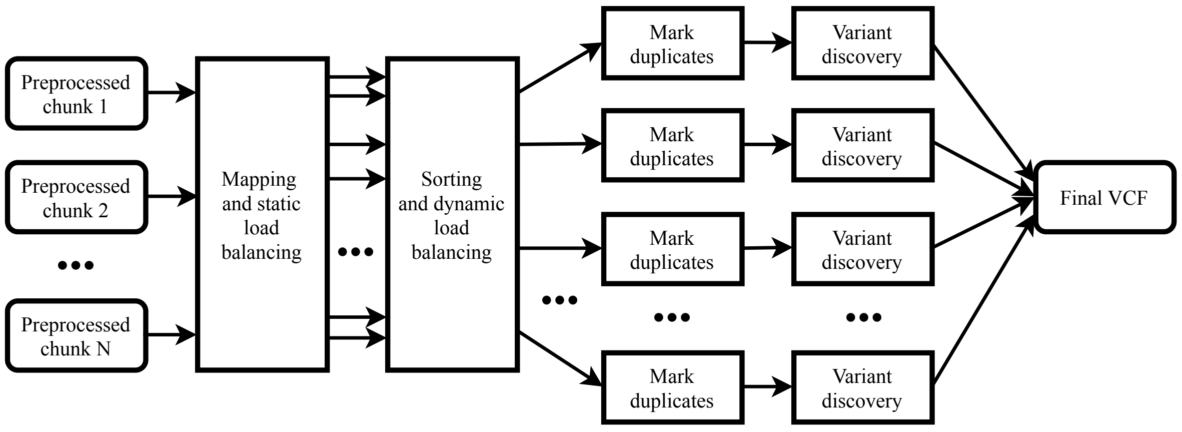


Figure 3.2: The workflow of SparkGA

be predefined. All these SAM files are stored distributedly in the HDFS, thereby making them easily accessible for next stage.

2. Sorting and dynamic load balancing.

   Dynamic load balancing is the novel strategy proposed in SparkGA. In the previous phase, the long chromosomes have been divided into multiple sub-regions. However, equally dividing the chromosomes does not guarantee the balanced workload for variant calling. The actual number of mapped reads located in a certain sub-region is possibly much bigger than another. The SAM files corresponding to the same sub-region will be grouped as one input file for the last phase. Therefore, if too many reads are mapped to a certain sub-region, the corresponding merged file will be much larger than others, thereby slowing down the overall performance of the variant calling phase. SparkGA overcomes this drawback by further dividing this relatively larger sub-region into multiple parts. This is achieved by counting the number of reads in a sub-region also as a factor. The average number of reads in a sub-region will be used as the threshold to further split those large sub-regions. The sub-region that contains n times as big as this threshold reads will be further divided into n smaller regions. After dynamic load balancing, all the reads in the same region are sorted according to their mapping positions and then merged into one BAM file.

3. Duplicates marking and variant calling.

   In this phase, each BAM file will be processed separately on the node of the cluster. Separate VCF files will be produced for each input BAM file. After obtaining all the VCF files for each input, the final output VCF file is generated by merging all these separate results. Input BAM files after dynamic load balancing will have relatively similar size, which ensures that the merging process will not wait for some separate results for too long.

3.1.4. SUMMARY

Different scalability solutions for the normal DNA analysis pipeline are introduced above. Their advantages and disadvantages are addressed in this section such that the potential solution to parallelize the early DNA analysis pipeline could be explored from the comparison of these scalability solutions.

In Churchill [22], all stages are tightly integrated. A customized phase to overcome the data dependencies is included as well. As a result, it is very hard for other researchers to modify any stage in their pipeline for the customized applications. Besides, Churchill currently could only be used on the cluster with a shared file system. This infrastructure requirement significantly limits the freedom to apply their pipeline in other applications.

Halvade [23] is a scalability solution based on Hadoop MapReduce framework, which eliminates the requirement for a shared file system. Any cluster with Hadoop set-up could be used as the environment to run Halvade. However, Halvade can be still improved in several aspects. Firstly, Hadoop MapReduce framework is a disk-based computational model. All the intermediate results generated between the map and reduce phase are stored in the disk. The heavy write and read I/O operations slow down the whole process. Secondly, Halvade only considers the length of the chromosome as the metrics to partition the sub-regions. Although the length of each sub-region has a relatively equal length, the number of reads that actually mapped to a sub-region might be very large. The reduce task to call the variants in the sub-region that contains the most of the reads takes longer than others. As a consequence, it would slow down the overall speed of Halvade as merging the separate VCF files requires the output from each sub-region.

Rather than Hadoop, SparkGA [3] is based on Apache Spark framework. It addresses the disadvantages discussed in Halvade. A phase called dynamic load balancing is included in SparkGA, where the sub-regions containing too many reads are further divided. The experimental result from [3] illustrates that SparkGA outperforms Halvade regarding speed. Figure 3.3 shows the difference of the speed performance given an identical input data between Halvade and SparkGA in that experiment. Although the time consumption of the first two steps (the steps before variant calling) is longer than Halvade due to dynamic load balancing, SparkGA dramatically decreases the time needed for the third step (variant calling), which occupies the majority of the total time consumption. Sacrificing a little in the time to more equally distribute the variants calling workload, SparkGA finally saves a huge amount of time in the last step. Besides, processing the genome data in memory also erases the heavy disk access overhead as compared to Halvade, thereby bringing better speed performance.

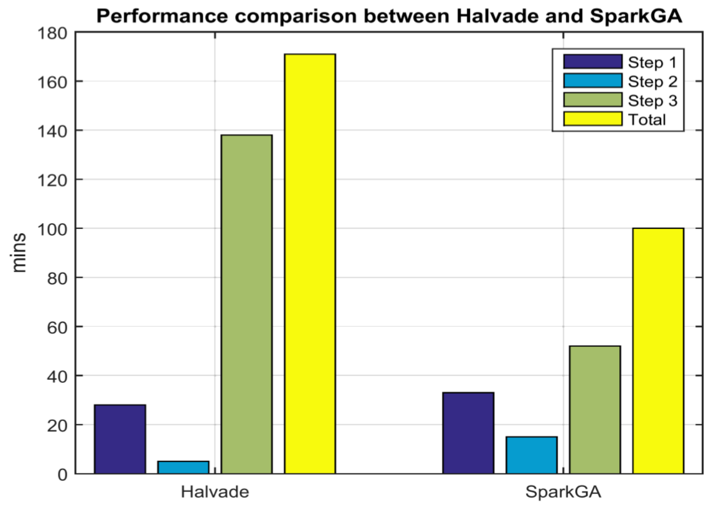


Figure 3.3: Comparison of Halvade and SparkGA from [3]

By comparing different scalability solutions of the normal DNA analysis pipeline, the characteristics of different strategies to parallelize the analyzing tasks in the pipeline are presented to us. The primary method to parallelize the tasks in these solutions is to partition the raw genome data into separate subsets and execute the variant calling tasks on each subset. Two big data frameworks are also compared in the practical applications. From that, Spark-based SparkGA outperforms Hadoop-based Halvade for this genome analysis problem in practice. Therefore, the parallel implementation of the early DNA analysis pipeline will adopt Apache Spark as the big data infrastructure. Details of the implementation will be introduced in the next chapter.

3.2. EVALUATION CRITERIA

To evaluate our parallel implementation of the early DNA analysis pipeline, criteria that can reflect the performance in a straightforward expression should be selected. Generally, to fully utilize the decreased sequencing time consumption, the scalability solution of the early DNA analysis approach should be able to accelerate the whole process and output the result within a relatively short time. Meanwhile, this result generated by our solution should be correct. Therefore, the evaluation of the final solution will be conducted in two aspects: speed and result correctness.

3.2.1. SPEED OF THE SCALABILITY SOLUTION

It is obvious that the motivation to build this scalability solution comes from improving the speed. In the original early DNA analysis approach, running the pipeline on the multi-core platform could be done by executing a simple script. While the parallel implementation of the early DNA analysis pipeline should be performed on the multi-node cluster. The speed performance can be easily compared by doing the same job and recording the total time consumption in both circumstances.

3.2.2. RESULT CORRECTNESS OF THE SCALABILITY SOLUTION

The output file of the early DNA analysis pipeline is the VCF file that contains the variant information. Each line in the VCF file describes a variant detected in the sample DNA. Doctors make the diagnoses based on the variants information they see in the VCF file. In another word, the quality of the VCF file determines the effectiveness of the whole pipeline.

To evaluate the correctness of the result generated by the parallel implementation of the early DNA analysis pipeline, a precise comparison with the result from the original pipeline is required. The area under the precision-recall curve (APR), which reflects the intrinsic trade-off between precision and recall, provide an informative performance score between two results [25]. In our case, the precision and recall are evaluated as follows:

$$precision = \frac{TP}{TP+FP} \times 100\% \tag{3.1}$$

$$recall = \frac{TP}{TP+FN} \times 100\% \tag{3.2}$$

where TP, FP, and FN are defined as true positives, false positives, and false negatives respectively [26]. Precisely, as shown in Table 3.3, true positives describe the same variants detected by both the parallelized early DNA analysis pipeline and the original pipeline; false positives describe the variants appearing in the VCF file of the scalability solution but not shown in the original output; at last, false negatives represent the variants shown in the original VCF file but absent in the VCF file of the scalability solution.

Table 3.3: Variants detected in two approaches

Variants	Scalability solution	Original early DNA analysis approach
TP	Detected	Detected
FP	Detected	Undetected
FN	Undetected	Detected

APR scores could illustrate the correctness of the parallelized early DNA analysis pipeline to detect as much as possible right mutations and as little as possible wrong mutations as compared to the original early DNA analysis pipeline.

3.3. COMPLETION APPROACHES

In the original early DNA analysis approach, the consensus-based scheme to complete the unknown bases is verified as the best one so far. It is clear that this consensus-based completion scheme has not fully explored the correlation between these unknown bases and their overlapping bases. Therefore, more advanced completion methods have stronger possibilities to improve the accuracy. In this section, a few machine learning algorithms are analyzed as the candidates for the advanced completion method. Both classification methods and regression methods in machine learning are introduced.

3.3.1. BASICS

Machine learning methods enable the computer to learn from the data rather than being programmed in a specific manner. Computers do not need to understand the exact mechanism of a problem but the correlation between the input variables and the outcome. To understand the correlation, numerous input data is required to be analyzed. For a particular problem, the data set is normally separated into a training set, a test set, and optionally a validation set. The training set is used to train the model. The validation set could be used to reconfigure the parameters of the model by applying the trained model on it. The test set is used to evaluate the performance of the fully-trained model. Various machine learning algorithms are typically classified as the following three categories:

1. Supervised learning.

   In supervised learning, the correct output for any given input vector is predefined. Thus, supervised learning is an approach based on labeled data sets. Computers use a function to derive the output for an input vector. In the training phase, computers alter this function to minimize the difference between the correct output and the output derived from this function for the input data in the training sets. Afterwards, the input-output pairs in the test sets are used to evaluate the correctness. If this model performs good, this well-trained function will be used to predict the output for unseen input data.

2. Unsupervised learning.

   Rather than using the labeled data sets, unsupervised learning does not have the predefined input-output pairs. The algorithms itself need to understand the underlying information from the input data. Since the training data sets are unlabeled, there is no straightforward method to evaluate the correctness of the model unsupervised learning builds.

3. Reinforcement learning.

   In reinforcement learning, computers learn its behavior based on the feedback from the environment. Instead of directly comparing the labeled output, computers will take actions and adjust the model based on the feedback. Rewards will be given for positive outcomes and punishments will be given for negative outcomes. Feedback leads the model to perform like the actions with rewards for next time.

3.3.2. LINEAR REGRESSION

Linear regression is a regression analysis method that is widely used in many practical applications. It describes the relationship between a scalar dependent variable and one or more independent variables. For more than one independent variable, it is also called multiple linear regression. If the model is further extended with multiple dependent variables, it becomes a multivariate linear regression problem [27]. A linear regression model is denoted as follows [28]:

$$y = \beta_0 + \beta_1 x + \epsilon \tag{3.3}$$

where y is the dependent variable, X is the independent variable vector, β_0 and β_1 are the parameters, ϵ is the error. The parameters of the model are altered in the training to minimize the difference between the observed value y_i and the linear approximation $\beta_0 + \beta_1 x_i + \epsilon$ given an input data x_i. The most common parameter estimation methods are ordinary least squares, ridge regression, and lasso regression.

Ordinary least squares (OLS) method minimizes the residual sum of squares between observed values and predictions by linear estimation. The mathematical notation is written as [28]:

$$min\{\sum_{i=1}^{n}(y_i - x_i^T\beta)^2\} \tag{3.4}$$

However, simple OLS method suffers from overfitting as the input features might not be linearly independent. To overcome this disadvantage, regularization is commonly used by adding additional parameters to the lost function. Ridge regression penalizes the magnitude of coefficients by adding the sum of squares of coefficients to the lost function [28]:

$$min\{\sum_{i=1}^{n}(y_i - x_i^T\beta)^2 + \lambda\sum_{j=1}^{p}\beta_j^2\} \tag{3.5}$$

where λ is the shrinkage parameter that controls the amount of regularization. By doing regularization, large coefficients that cause overfitting is minimized. Another regularization method called lasso regression is also widely used when solving linear regression problems. It penalizes large coefficients by adding the absolute value to the lost function [28]:

$$min\{\sum_{i=1}^{n}(y_i - x_i^T\beta)^2 + \lambda\sum_{j=1}^{p}|\beta_j|\} \tag{3.6}$$

where λ is also the shrinkage parameter that controls the size of the coefficients. These linear regression methods provide easy and effective solutions to various regression problems in practice.

3.3.3. NAIVE BAYES

Naive Bayes is a commonly-used machine learning classifier. The main goal in the classification problem is to decide the class for a given feature set $\{(x_1, x_2, ..., x_n)\}$. Therefore, we want to calculate $P(c_i|x_1, ..., x_n)$ for each class c_i, where $c_i \in \{c_1, c_2, ..., c_k\}$. The class has highest probability becomes the classification outcome.

In order to calculate this probability, Bayes Rule is used. It is described as follows:

$$P(A|B) = \frac{P(B|A)\cdot P(A)}{P(B)} \tag{3.7}$$

where $P(A|B)$ is a conditional probability: the likelihood of event A occurring given that B is true, $P(B|A)$ likewise, $P(A)$ and $P(B)$ are the probabilities of observing A and B. [29]

Therefore, our classification problem becomes:

$$P(c_i|x_1, ..., x_n) = \frac{P(x_1, ..., x_n|c_i)\cdot P(c_i)}{P(x_1, ..., x_n)} \tag{3.8}$$

For easier calculation, $P(x_1, ..., x_n)$ is always ignored. Thus, only $P(x_1, ..., x_n|c_i)$ and $P(c_i)$ are calculated. An assumption that $\{x_1, ..., x_n\}$ are conditionally independent given class c_i such that we can describe the problem as follows:

$$\begin{aligned} P(c_i|x_1, ..., x_n) &\propto P(x_1, ..., x_n|c_i)\cdot P(c_i) \\ &\propto P(c_i)\cdot\prod_{j=1}^{n}P(x_j|c_i) \end{aligned} \tag{3.9}$$

That assumption is most likely not true, hence called Naive Bayes. However, this classifier still performs well in many circumstances. Calculation of $P(x_j|c_i)$ is relatively easy as long as we have the training data. We need to calculate the distribution of feature x_j where $j \in \{1, 2, ..., n\}$ in each class c_i where $i \in \{1, 2, ..., k\}$. In different classification problems, the distribution of the features could be multinomial distribution, Gaussian distribution, and so on. [30]

3.3.4. DECISION TREES

Decision trees approach is a typical solution for regression and classification problems in machine learning. It presents as a tree where each internal node of the tree describes a feature of the input objects, the leaf nodes represent the categories of the objects. Based on a given data set $D = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$, where $x_i = (x_i^1, x_i^2, ..., x_i^n)$ is the corresponding n features for sample i, y_i is the category sample i belongs to, N stands for the total number of samples in this data set, how to build a tree that can correctly classify the data from their features is the question decision trees model addresses. Whether a feature could provide enough information to do this is an intermediate question, since, in some circumstance, two objects might have the same features but are classified into different categories. [31]

An example of decision trees application is introduced as follows. The weather of a morning is described from four aspects: $\{outlook, temperature, humidity, windy\}$. Table 3.4 presents the values of each attribute. According to the table, a specific morning can be described as $\{sunny, hot, high, false\}$. A simple training set is shown in Table 3.5. These mornings are classified as good or bad depending on their attributes.

In this example, decision trees method tries to establish a classification rule that can correctly classify a morning as good or bad according to the attributes that morning has. This rule is expressed as a decision tree. An example of a decision tree to classify the samples in the training set is shown in Figure 3.4. The internal

Table 3.4: Four attributes of a morning

Attributes	Values
Outlook	sunny, overcast, rain
Temperature	cool, mild, hot
Humidity	high, normal
Windy	true, false

Table 3.5: A simple training set.

No.	Attributes				Class
	Outlook	Temperature	Humidity	Windy	
1	sunny	hot	high	false	bad
2	sunny	hot	high	true	bad
3	overcast	hot	high	false	good
4	rain	mild	high	false	good
5	rain	cool	normal	true	bad
6	overcast	cool	normal	true	good
7	sunny	cool	normal	false	good
8	overcast	hot	normal	false	good
9	rain	mild	high	true	bad

nodes of the tree, including the root, are the attributes, while the leaves of the tree are the classes. To classify a sample, we start from the root, take the branch with the same value in the specified attribute. This action is performed in iterations until the leaves reached. The type of the class in that leaf node determines the class. For the given morning $\{sunny, hot, high, false\}$ mentioned above, it is classified as bad after going through this decision tree from the root. [32]

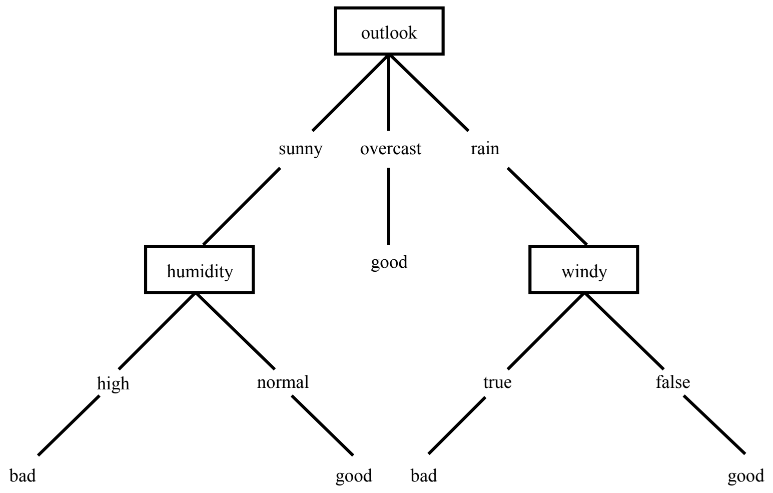


Figure 3.4: An example of a decision tree

To be noted, it is obvious that there are different trees that can correctly classify the samples in the training set. If the humidity attribute is selected as the root, the corresponding decision tree will look different. Starting with determining the root, constructions of the decision tree are progressed in an iterative manner such that a certain number of attributes are selected as the internal nodes while the leaves represent the classes. The selected attributes should have a good capacity to distinguish different samples. Two criteria are commonly used to evaluate the attribute when building the trees: gini impurity and information gain. These two indexes are measured as follows.

1. Gini impurity.

For a given set Y, the gini impurity of this set expressed as $IG(Y)$ is calculated by:

$$IG(Y) = 1 - \sum_i P_i^2 \tag{3.10}$$

where P_i describes the probability of class i. The gini impurity after separating this set Y by a given attribute X is calculated as the weighted average of the gini impurities of the subsets:

$$IG(Y|X) = \sum_s P_s \cdot IG(s) \tag{3.11}$$

where P_s describes the proportion of the subset s, $IG(s)$ stands for the gini impurity of the subset s.

2. Information gain.

   Entropy is used to describe the uncertainty of a class. The entropy of a set Y is calculated as:

$$H(Y) = -\sum_i P_i \cdot \log_2 P_i \tag{3.12}$$

   where P_i describes the probability of class i. The information gain from separating this set Y by a given attribute X is presented as:

$$IG(Y|X) = H(Y) - \sum_s P_s \cdot H(s) \tag{3.13}$$

   where P_s describes the proportion of the separated subset s, $H(s)$ stands for the entropy of the subset s. The difference between the entropy of set Y and the weighted average for the entropy of the subsets describes the eliminated uncertainty by doing this separation.

Attribute with lower gini impurity or higher information gain has better chance to separate the data set such that each divided subset belongs to a unique category. When building the decision trees, the best attribute, that is, the attribute with the lowest gini impurity or the highest information gain will be selected as the root. Subsequently, the second best attribute will be selected as the internal node to further split the subset. The extension of the tree stops until separation by any attribute does not provide more information gains.

Decision trees are easy to be built. A good decision tree should have a simple structure that can capture the correlation between the attributes and the classes.

3.3.5. RANDOM FORESTS

Random Forests is a method built based on simple decision trees. As indicated by the name, it creates a forest composed of a number of trees. These trees in the forest are all different decision trees. Building and using the forest follows simple bagging (bootstrap sample and aggregate) rule :

1. For the training set with size N, a bootstrap data set is obtained by doing a random selection for N times while the probability of selecting every sample in the training set is equal in each independent selection. As a consequence, a particular sample in the training set might be selected for multiple times, while a few samples are not selected when creating a bootstrap data set. A tree in the final forest is built from a bootstrap data set. When building the random forest, a number of bootstrap data sets are created. Each of them is N in size and corresponds to a tree in the forest.

2. When building a decision tree in the forest, one bootstrap data set generated from the last step is used now. Instead of counting all the features, only random selected m ($m < M$) features are used to split this bootstrap data set when building the corresponding decision tree, where M stands for the total number of the features the training set has.

3. After building a number of decision trees based on different randomly selected features given different bootstrap data sets. To classify a sample, the majority vote of the classes output by the trees in the forest represents the aggregate result.

For the training data shown in Table 3.5, an example bootstrap data set is presented below:

As can be seen in Table 3.6, sample $2,5,6$ are all repeated in this bootstrap data set, while sample $1,4,8$ are not selected. Only $outlook, windy$ are selected as the split attribute. The corresponding decision tree created by this bootstrap data set is shown in Figure 3.5.

Table 3.6: A bootstrap data set.

No.	Attributes		Class
	Outlook	Windy	
2	sunny	true	bad
2	sunny	true	bad
3	overcast	false	good
5	rain	true	bad
5	rain	true	bad
6	overcast	true	good
7	sunny	false	good
6	overcast	true	good
9	rain	true	bad

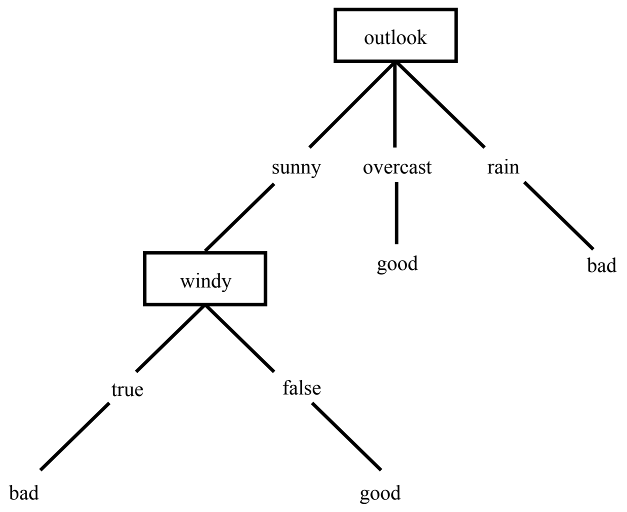


Figure 3.5: The decision tree created from the bootstrap data set shown in table 3.6

Random forests method has a unique advantage when compared to other machine learning algorithms. As discussed above, the bootstrap data sets are created by randomly selecting the samples in the training set. Normally, after a number of bootstrap data sets created, there are still samples never being selected in any bootstrap data set. These never selected samples are called OOB (Out of Bag) samples. As these OOB samples are not used in the training process to create the forest, they can be used as the test data to evaluate the trained model. Therefore, in random forests learning, there is no need to divide the raw data set into a training set and a test set. The error rate estimated by testing these OOB samples is called OOB error. [33]

3.3.6. SUMMARY

A brief summary of the analyzed machine learning algorithms is shown in Table 3.7. Linear regression is a commonly used regression method in practical applications. It estimates the real value of a dependent variable. While Naive Bayes is a typical classification method that classifies a given feature set into a class. In addition, decision trees and random forests are also introduced as two effective methods applied both in regression problems and classification problems.

Table 3.7: Analyzed machine learning algorithms.

Method	Regression	Classification
Linear Regression	$\surd$	$\times$
Naive Bayes	$\times$	$\surd$
Decision Trees	$\surd$	$\surd$
Random Forests	$\surd$	$\surd$

4

Implementation of the Scalability Solution

In the previous chapter, different methods proposed to parallelize the normal genome analysis pipeline are discussed. After absorbing the advantages and disadvantages of these solutions, a parallel workflow of the early DNA analysis pipeline is designed in this chapter. The overview of this parallelized early DNA analysis pipeline is illustrated first in Section 4.1. The implementation details of each step in the workflow are presented in Section 4.2.

4.1. Overview

In the original early DNA analysis approach, two workflows are proposed as shown in Figure 2.7. WF-1 completes the unknown bases and their base quality scores in the input SAM file while WF-2 also completes and modifies the fields that will be used in the later stages. Therefore, WF-1 requires two more steps after completion to prepare the data for calling variants. According to the experiment result in [8], WF-1 outperforms WF-2 regarding variants detection accuracy. As a result, the parallel implementation of the early DNA analysis pipeline will be based on WF-1.

Figure 4.1 shows the whole pipeline of WF-1. It is composed of two major phases: completion and standard genome analysis. As the early DNA analysis approach does not wait for the fully sequenced second read, the input data is incomplete. Precisely, the last few bases and their quality scores of the second read in the paired-end data are unknown. In the completion phase, the incomplete input data set is sequentially processed by four steps: DNA mapping, Reads sorting, Completion, and SAMToFASTQ. The tools used in this completion phase are shown in Table 4.1. After completion, the unknown bases and their quality scores of the second read are completed. The standard genome analysis is followed then performed on the complete data set from the completion phase. The final VCF file is output by the standard genome analysis phase in the end.

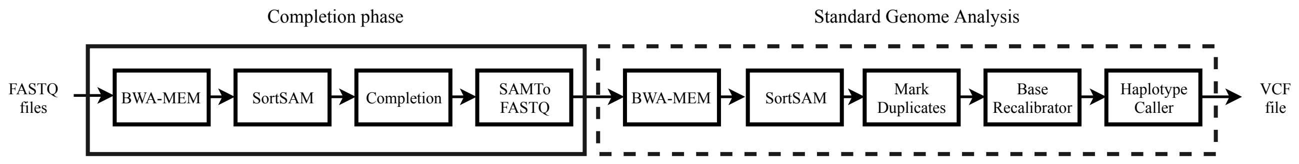


Figure 4.1: WF-1 in the original early DNA analysis approach

As discussed in Section 3.1, a few sophisticated solutions to scale up the normal genome analysis pipeline has been proposed. The primary method to parallelize the analyzing tasks is to divide the raw input data set into a number of subsets, while the variant calling tasks are performed for each subset. Meanwhile, the data dependency of the stages in the pipeline should remain after parallelization. For instance, the redundancy strategy in Halvade guarantees the data dependency requirement for variant calling, in which the paired reads mapped to different sub-regions are stored in both sides.

Table 4.1: Different tools used in completion phase

Step	Tool
DNA mapping	BWA-MEM
Reads sorting	PICARD-SortSam
Completion	Completion program
SAMToFASTQ	PICARD-SamToFastq

As for parallelizing the early DNA analysis pipeline, the standard genome analysis phase in the pipeline can be easily replaced by those scalability solutions introduced in Section 3.1. SparkGA outperforms other solutions, thereby being selected as the replacement for the standard genome analysis phase. Spark framework adopted in SparkGA is selected as the big data infrastructure. In this thesis, the major work is to implement the scalability solution for the completion phase and integrate it with SparkGA. When parallelizing the completion phase, not only the method to partition the data set, but also the data dependency requirement of the completion task should be considered.

In the original early DNA analysis pipeline, the completion step takes a single SAM file containing all the sorted reads as the input, consuming even as long time as standard genome analysis pipeline. Thus, parallelizing the completion tasks could strikingly improve the speed performance of the whole pipeline. Since we could obtain the length information of different chromosomes from the reference genome, we can divide the long chromosomes into a few sub-regions according to their length. The reads mapped to the same sub-region will be merged into one individual SAM file as an input for a completion task. In SparkGA, not only the length of the chromosome but also the actual number of reads in the sub-regions is selected as the metrics to partition the input data set. However, for the completion step, the overlapping reads of the incomplete second read determine the completion result. Since only very few reads have no overlapping with other reads, the dynamic subregional division based on the reads numbers will decrease the completion quality as the further division always separates the overlapping reads. Therefore, only the chromosomal length is counted when partitioning the sub-regions for completion tasks.

The workflow of the parallelized early DNA analysis pipeline is presented in Figure 4.2. It also includes two major phases. The parallelized completion phase is presented in solid while the standard genome analysis phase in the original WF-1 is replaced by SparkGA in dotted.

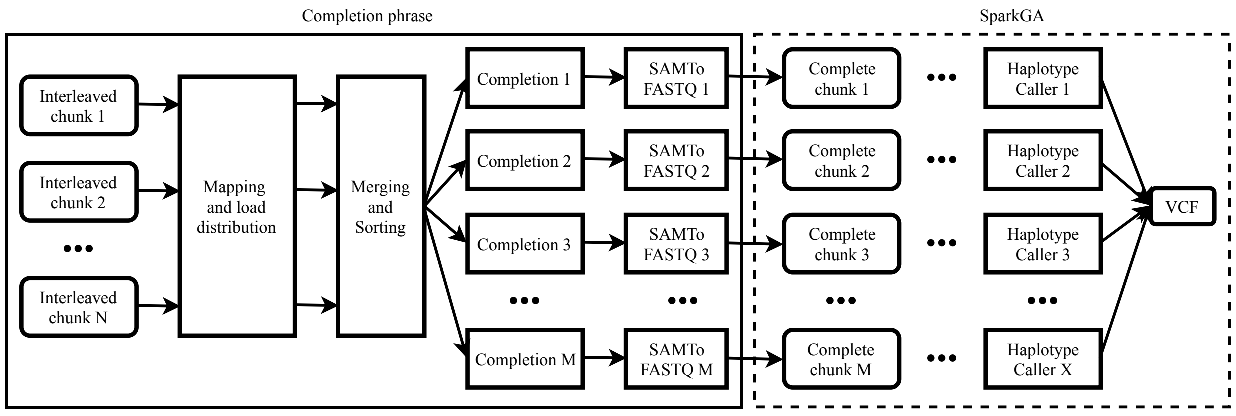


Figure 4.2: Workflow of parallelized early DNA analysis pipeline

At the beginning of the completion phase, the input two FASTQ files are divided into multiple interleaved chunks. These chunks are uploaded to HDFS, where all the nodes in the cluster could access and process them. After these chunks created, the DNA mapping jobs are assigned separately among the nodes in the cluster to align the reads against a reference genome for each chunk. As discussed above, to parallelize the completion tasks, a load distribution step is performed after alignment. Chromosomes are divided into multiple sub-regions according to their length. The aligned reads located in the same sub-region are sorted first by their alignment position and then placed into an individual SAM file. Subsequently, the completion jobs are started distributedly over the nodes for each sub-region. Obtaining the SAM files containing completed reads from the completion tasks for each sub-region, a SAMToFASTQ step using PICARD tool is followed to convert a SAM file back to two FASTQ files for each sub-region. Afterwards, an integration step that interleaves these FASTQ files and uploads them into HDFS performs as the first step of SparkGA. Finally, a full process goes through SparkGA and outputs the final VCF result in the end.

4.2. IMPLEMENTATION DETAILS

As shown in Figure 4.2, the parallelized early DNA analysis pipeline is composed of two major parts: completion phase and SparkGA. The parallelized completion phase is the primary work of this thesis. Different steps in this pipeline are introduced below.

4.2.1. CREATION OF INTERLEAVED CHUNKS

In the early DNA analysis approach, two FASTQ files are streamed out of the sequencing machines. Each one of them contains one of the paired-end reads. Thus, the FASTQ file containing the incomplete second read is relatively smaller in size. In SparkGA, a chunk-segmentation tool is built to divide this two FASTQ files into interleaved chunks. To maximize the speed, SparkGA reads this two FASTQ files by a certain byte of size and put them in a chunk file. As the two input FASTQ files are not same in size in the early DNA analysis approach, this segmentation method brings trouble as the paired-end reads are separated in two chunks, which is not acceptable for later DNA mapping stage. Therefore, a line-by-line chunk creation tool is built in this implementation. As shown in Figure 4.3, by reading both two FASTQ files line by line, an interleaved chunk file containing an equal number of the first and second reads will be generated. Both the first and second read of a pair will be placed next to each other in an identical chunk file. By doing this, the entire input data set is divided into a number of chunks.

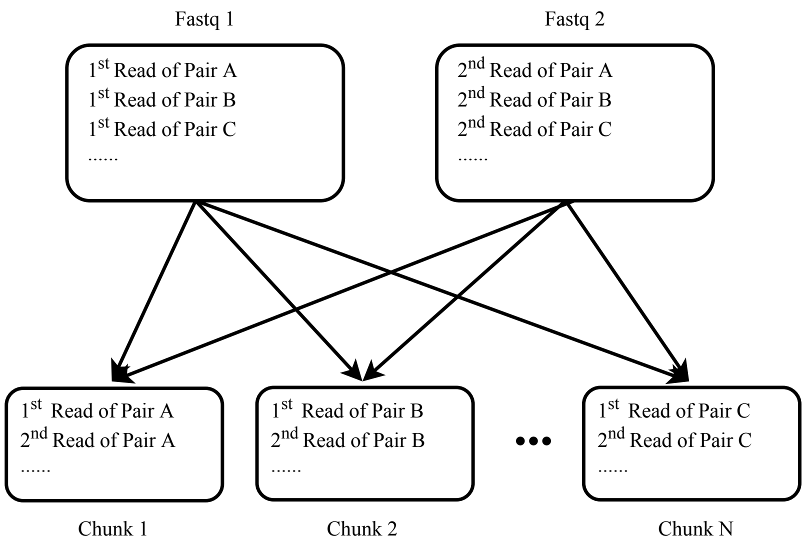


Figure 4.3: Segmentation method to create interleaved chunks

4.2.2. MAPPING AND LOAD DISTRIBUTION

Figure 4.4 presents the scheme of mapping and load distribution. During this stage, every interleaved chunk created by the last step is assigned with a DNA mapping and load distribution job. By doing the DNA mapping job using i.e., BWA-MEM, each read in a chunk will be aligned against a reference genome. In general, the mapping result is a combination of the chromosome name and a position number that indicates the location within that chromosome. The load distribution is achieved by dividing chromosomes into multiple sub-regions according to their length. As a result, longer chromosomes will be divided into more sub-regions. In the human genome, chromosome one is the longest chromosome, thereby having the most sub-regions. The aligned DNA reads are placed into the corresponding regions according to their alignment result. The redundancy strategy proposed in Halvade is also adopted here. The paired reads mapped to different sub-regions will be placed in both regions. The SAM files created in this part are written into HDFS directly such that later steps could easily retrieve these data. The mapping and load distribution task for different input chunks could generate the SAM files that correspond to an identical sub-region in a chromosome. For instance, chromosome one is divided into two parts: sub-region $1a$ and $1b$. Both $chunk1$ and $chunk2$ have many reads that are mapped to the sub-region $1a$ of chromosome one, thus generating two SAM files $Sam1\_1a$ and $Sam2\_1a$, where $SamN\_Mn$ indicates the SAM file generated from chunk N and mapped to sub-region n of chromosome M.

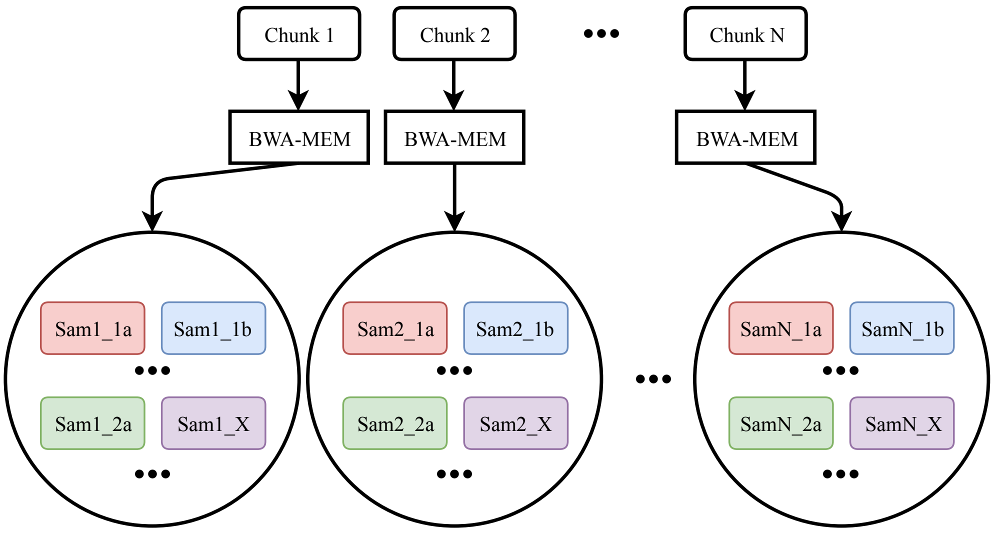


Figure 4.4: Mapping and load distribution

4.2.3. Merging and Sorting

In this step, the SAM files containing the reads that are mapped to the same sub-region will be merged. The merging process is achieved with the help of a global information file created in the previous mapping stage. This file contains the information about all the SAM files generated in the mapping step. Each line in this file describes the information of a SAM file by the name of its belonging sub-region and the name of this SAM file itself. In this step, this file is read first and placed as a Spark's RDD (Resilient Distributed Dataset). Each record in this RDD is presented as a pair. The key of the pair is the sub-region ID, for example, $1a$ (sub-region a of chromosome one). The value of the pair is the SAM file name. Subsequently, a groupBykey operation is performed for this RDD. As a result, for each sub-region, the name of all the SAM files belonging to this sub-region are collected. Afterwards, the tasks to read the SAM records from the collected SAM files and then sort these records by their position number are executed in parallel for each sub-region. Finally, the sorted SAM records for each sub-region are written into a single file in HDFS as an input for the next step. As shown in Figure 4.5, $Sam1\_2a$, $Sam2\_2a$, and $SamN\_2a$, corresponding to region $2a$ of chromosome two, are merged first, then sorted, lastly stored as a single file Sam_2a in HDFS.

4.2.4. Completion and Integration with SparkGA

In this step, the completion tasks are performed in parallel among the nodes for each sub-region. Separate completion tasks retrieve the sorted SAM files from HDFS. All the second reads in these SAM files are not complete. For each sub-region, the completion task generates the output SAM file that contains the completed second read. Afterwards, a SAMTOFASTQ stage is performed to convert the output SAM file into two FASTQ files, which now have equal length in the first and second read. As shown in Figure 4.6, for each sub-region, a pair of FASTQ files is generated. Since the standard genome analysis phase is replaced by SparkGA in our solution, a good integration interface should be designed. In SparkGA, two very large FASTQ files are divided into multiple interleaved chunks at the beginning. As we have obtained FASTQ files for each sub-region, we could directly interleave this two small FASTQ files and upload the interleaved chunks into HDFS, which performs like the first stage of SparkGA, thus achieving an effective integration.

Afterwards, obtaining the completed DNA data from the entire parallelized completion phase, the genome analysis to detect the variants for these data is performed as the scheme of SparkGA. For each interleaved chunk, the DNA mapping job is performed. The static load balancing and dynamic load balancing guarantees the flat distribution of the workload for later variant calling. In each sub-region, the variant calling tasks are executed independently in parallel. Finally, the result VCF files output from each sub-region are merged

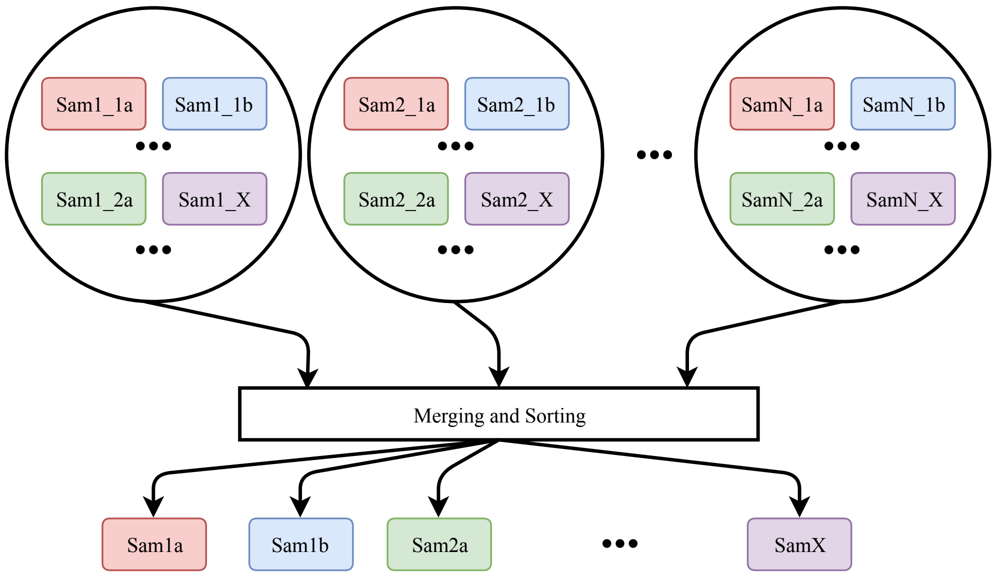


Figure 4.5: Merging and sorting

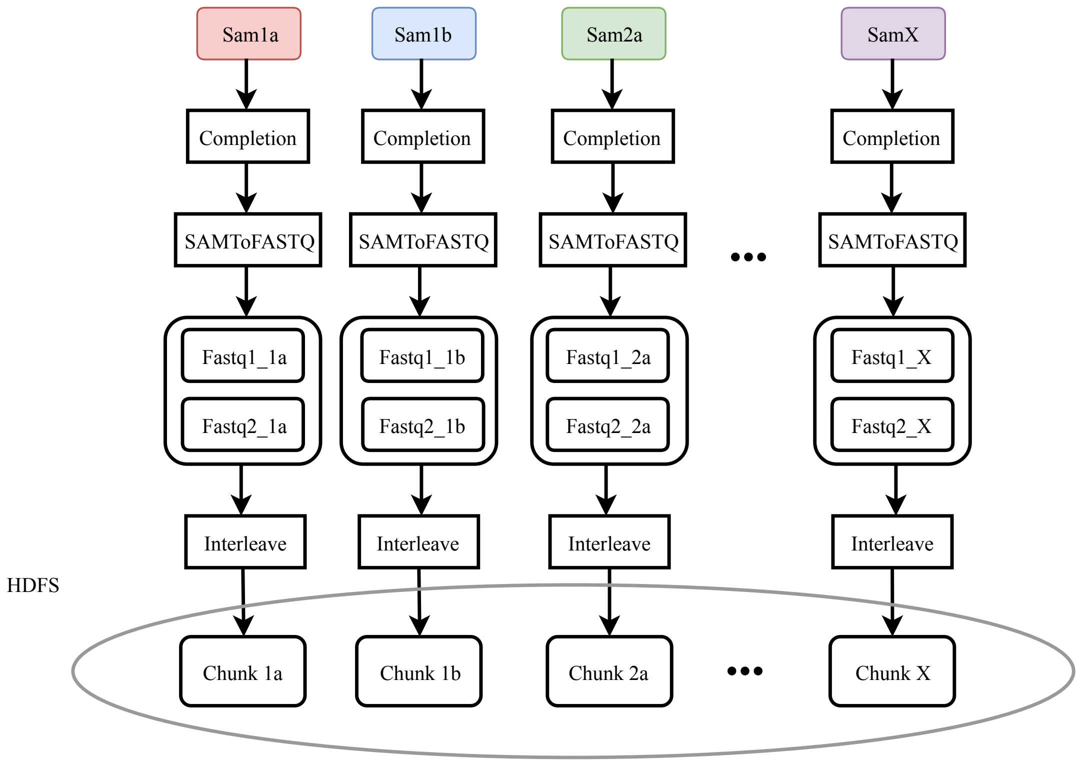


Figure 4.6: Completion and integration with SparkGA

as the final output.

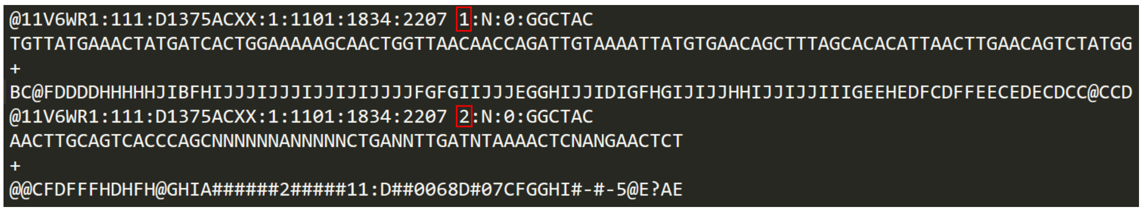
5 Evaluation of the Scalability Solution

In this chapter, the performance of our parallel implementation of the early DNA analysis pipeline is evaluated. Section 5.1 describes the experimental environment. The performance of our scalability solution is presented in Section 5.2. A few additional analyses are also described in this section. Lastly, a summary based on the evaluation results are given in Section 5.3.

5.1. Experimental Environment

The experiment was conducted on an IBM Power7+ cluster with one master node and four data node. Each node in the cluster has 16 physical cores and 128 GB of memory. Each Power7+ core is capable of 4-way simultaneous multi-threading [34], thereby acting as 4 virtual cores. Spark could be run in local mode or YARN mode on this platform. Our scalability solution provides the option to run the parallelized early DNA analysis pipeline on Spark local mode or Yarn mode. Running on the Yarn mode provides the access to utilize all the computing resources in the cluster. On the contrary, running on the local mode means that only one node in the cluster is accessible, which is the same as the environment when running the original pipeline using a script.

In the experiment, we used the whole exome sequencing of sample NA12878 which is widely used in benchmarking. It contains paired-end reads with 150x coverage and 100 base pairs for each read [16]. As in the early DNA analysis pipeline, the input data is incomplete because the second read is not fully sequenced as described in Chapter 2. To simulate the situation where we do not know the last few bases and their quality scores of the second read, we manually cut all the second reads in the experimental dataset. In our experiment, the last 40 bases of all the second reads and their base quality scores were removed. Therefore, the incomplete paired-end dataset we used is composed of the first read with 100 bases and the second read with 60 bases. An example of the paired-end read containing incomplete data is shown in Figure 5.1.



```
@11V6WR1:111:D1375ACXX:1:1101:1834:2207 1:N:0:GGCTAC
TGTTATGAAACTATGATCACTGGAAAAAGCAACTGGTTAACAACCAGATTGTAAAATTATGTGAACAGCTTTAGCACACATTAACTTGAACAGTCTATGG
+
BC@FDDDDHHHHHJIBFHIJJJIJJJIJJIJIJJJJFGFGIIJJJEGGHIJJIDIGFHGIJIJJHHIJJIJJIIIGEEHEDFCDFFEECEDECDCC@CCD
@11V6WR1:111:D1375ACXX:1:1101:1834:2207 2:N:0:GGCTAC
AACTTGCAGTCACCCAGCNNNNNNANNNNNCTGANNTTGATNTAAAACTCNANGAACTCT
+
@@CFDFFFHDHFH@GHIA######2#####11:D##0068D#07CFGGHI#-#-5@E?AE
```

Figure 5.1: Example of the incomplete paired-end data

5.2. Performance Measurement

In this section, the best performance of the original early DNA analysis is explored first. The multi-threading performance of the original pipeline is illustrated in the first section. Then the speed performance of our scalability solution is measured to compare with the original approach. A few additional measurements are also included. In the end, the result correctness of our solution is presented as a validation.

5.2.1. Multi-threading of the Original Pipeline

In the original early DNA analysis pipeline, each stage in the pipeline is executed sequentially with the specific tool. Table 5.1 shows the scalabilities of each stage in the pipeline. The scalability of a stage depends on the tool used in that stage and its data dependency itself. DNA mapping stage using BWA-MEM alignment tool has very good scalability as mapping for a read is independent from mapping for another read. While the stage using PICARD tool is lack of parallelization since PICARD does not provide multi-threading option. In GATK packages, a multi-threading option is provided to accelerate the pipeline. Besides, in HaplotypeCaller, we could also use the $-L$ option with a bed file to parallelize the analyzing tasks for different chromosomal regions.

Table 5.1: The scalability of each stage in the early DNA analysis pipeline (More '+' means better scalability.)

Phase	Stage	Tool	Multi-threading	Scalability
Completion Phase	DNA mapping	BWA-MEM	√	+++
	Reads sorting	Picard-SortSam	×	+
	Completion	Completion program	√	++
	SamToFastq	Picard-SamToFastq	×	+
Standard genome analysis	DNA mapping	BWA-MEM	√	+++
	Reads sorting	Picard-SortSam	×	+
	Duplicates marking	Picard-MarkDuplicates	×	+
	Base recalibration	GATK-BaseRecalibrator	√	++
	Reads printing	GATK-PrintReads	√	++
	Variant calling	GATK-HaplotypeCaller	√	+++

Figure 5.2 shows the speed performance of DNA mapping stage with different multi-threading numbers. The input data has 100x coverage, size of 15 GB in total. Time consumption dramatically decreases from 216 minutes to 27 minutes when we used eight threads as compared to the single thread running. However, increasing the threading numbers does not bring significant time deduction after 8. Running BWA-MEM with 64 threads gives the best performance since our experimental platform has 64 (virtual) cores. Increasing threads from 64 to 100 even slows down the whole process. There is a trade-off as the threading number increases. Running with more threads does not always improve the speed as in the ideal cases.

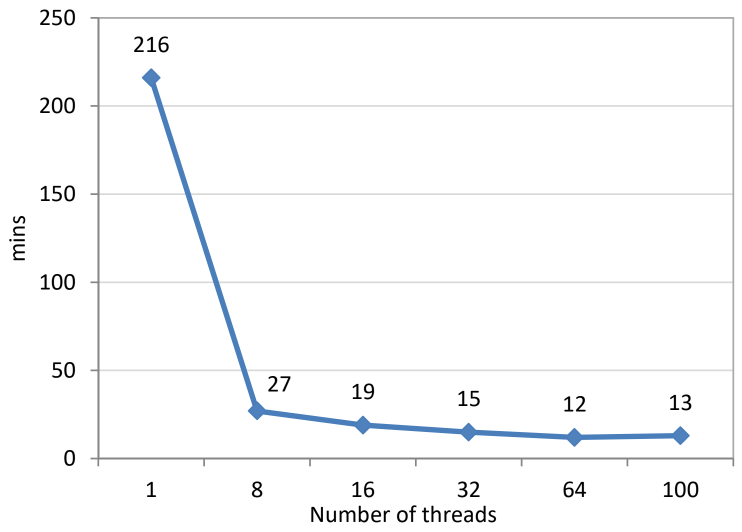


Figure 5.2: Time consumption of DNA mapping with different threads

We optimized the parallelization settings for all scalable stages in the original early DNA analysis pipeline by doing the same experiment. Running the pipeline with these best parameters gives us the minimal time consumption of the original approach. For a fair comparison, we will compare the performance of our scalability solution with the best performance from the original pipeline in the next section.

5.2.2. SPEED PERFORMANCE

As described in Chapter 4, the parallelized implementation of the completion phase in the pipeline consists of three major parts: mapping and load distribution; merging and sorting; completion and integration with SparkGA. The standard genome analysis phase is replaced by SparkGA which is composed of another three steps: mapping and static load balancing; sorting and dynamic load balancing; duplicates marking and variant calling. To have a clear comparison for each step, we define this six steps from step 1 to 6. Their corresponding stages in the original pipeline are shown in Table 5.2.

Table 5.2: All steps of the scalability solution and their corresponding stages in the original early DNA analysis pipeline

Step	Original pipeline	Scalability solution
1	DNA mapping	Mapping and load distribution
2	Reads sorting	Merging and sorting
3	Completion & SamToFastq	Completion and integration with SparkGA
4	DNA mapping	Mapping and static load balancing
5	Reads sorting	Sorting and dynamic load balancing
6	Duplicates marking & Base recalibration & Reads printing & Variant calling	Duplicates marking and variant calling

To compare the speed performance of the original early DNA analysis pipeline and our scalability solution on different modes, we conducted the experiment on the Power7+ cluster with the input data set in 150x coverage and size of 18.5GB. Table 5.3 shows the time consumption of each step in different solutions. In the original early DNA analysis pipeline, the most time-consuming steps are the completion step and the variant calling step, taking 237 minutes and 364 minutes respectively. However, running the scalability solution on the local mode already decreases the total time consumption of the pipeline from 685 minutes to 195 minutes. Running on YARN provides even more speed-up. The time consumption of the completion step drops to 40 minutes, which is around a 6 times speed-up as compared to the original pipeline. A time deduction of 595 minutes is achieved by our scalability solution, from 685 minutes to 90 minutes, around 7.6× acceleration for the whole process.

Table 5.3: Time consumption of each step in different solutions

	Original pipeline		Scalability solution (local)		Scalability solution (Yarn)	
Step	Minutes	Percentage	Minutes	Percentage	Minutes	Percentage
1	12	1.8 %	25	12.8 %	8	8.9 %
2	31	4.5 %	8	4.1 %	5	5.6 %
3	237	34.6 %	90	46.2 %	40	44.4 %
4	11	1.6 %	19	9.7 %	6	6.7 %
5	30	4.4 %	6	3.1 %	10	11.1%
6	364	53.1 %	47	24.1 %	21	23.3 %
1 & 2 & 3	280	40.9 %	123	63.1 %	53	58.9 %
Total	685	100 %	195	100 %	90	100 %

As discussed in Chapter 4, the primary work of this thesis is to implement the parallelized completion phase of the original early DNA analysis pipeline since the standard genome analysis phase could be easily replaced by SparkGA to achieve scalability. The speed performance of the completion phase in our scalability solution is illustrated in Figure 5.3 in particular.

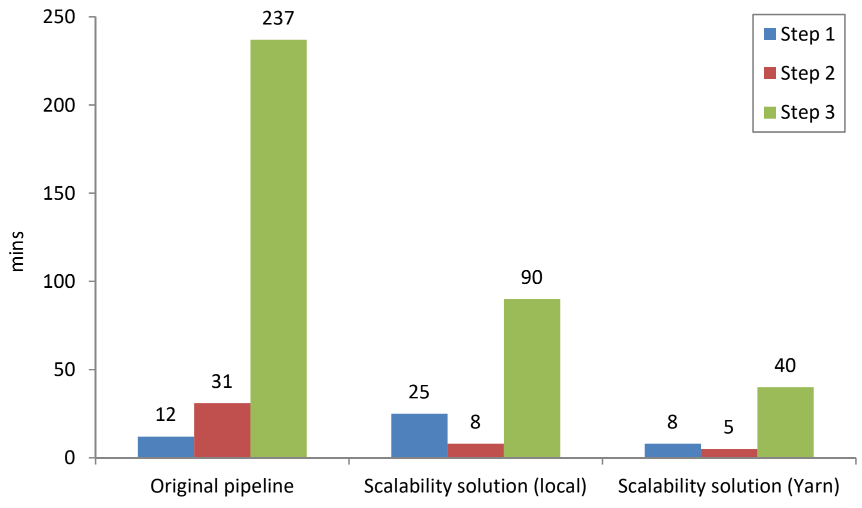


Figure 5.3: Speed performance of the completion phase in different solutions

In the original pipeline, the first step using BWA-MEM tool has very good scalability. By optimizing the multi-threading numbers, it can be finished within 12 minutes. The second step using PICARD tool to sort the aligned reads takes 31 minutes. However, the third step to complete the unknown bases and their corresponding quality scores for the second read takes 237 minutes, drastically slowing down the whole process. On the contrary, our scalability solution could finish the completion step in 90 minutes on the local mode and 40 minutes on the YARN mode. Although the first step does not benefit from the parallelization, even sacrificing a little in the local mode since it also performs the load distribution work, the significant time deduction of the completion step saves a huge amount of time.

5.2.3. ADDITIONAL ANALYSIS

In order to address the parallelization bottleneck of each step in our scalability solution, we recorded the CPU usage of the node when running the program on the local mode. Figure 5.4 illustrates the results. In the first step, a very high CPU usage rate can be found as the mapping tasks are executed in parallel. However, a drop of the CPU usage is observed in a regular interval. This comes from the uncompress job of the input data since we stored the interleaved input chunks as compressed in the input folder. In the second step, the SAM records from the same sub-region are merged first and then sorted. The heavy I/O operations to read the SAM file and write the reads into the new file results in some *idel* time. Step 3 shows good CPU usage as a certain number of completion tasks for different sub-regions are launched in parallel. Mapping and static load balancing step of SparkGA is followed after completion. In this step, the input chunks directly come from the completion step, thereby unequal in size. As a result, step 4 has more *idel* time as compared to step 1 which also performs the mapping job. Step 5 has the same behavior as step 2. In the last step, a relatively high CPU usage can be observed. The parallelized variant calling jobs show good scalability.

In the completion phase of the parallelized pipeline, we divide the chromosomes into multiple sub-regions. The completion tasks are executed in parallel for each sub-region. To evaluate the influence of subregional load distribution on the speed performance of the completion step, we altered the number of sub-regions to run the pipeline. This number determines the total number of sub-regions when dividing the chromosomes. We tested this number with 50, 150 and 300. Table 5.4 presents the features of the input SAM files for the completion step in different circumstances.

With more sub-regions divided, we can observe that the standard deviation of the input SAM file sizes drops. The standard deviation with 50 sub-regions and 300 sub-regions are 250 MB and 57 MB respectively. Smaller deviation means more balanced load distribution for the completion tasks, thereby improving the speed performance. Experiment results are illustrated in Figure 5.5. As can be seen from the figure, dividing the chromosomes into 300 sub-regions gives the best performance. The time consumption of the completion

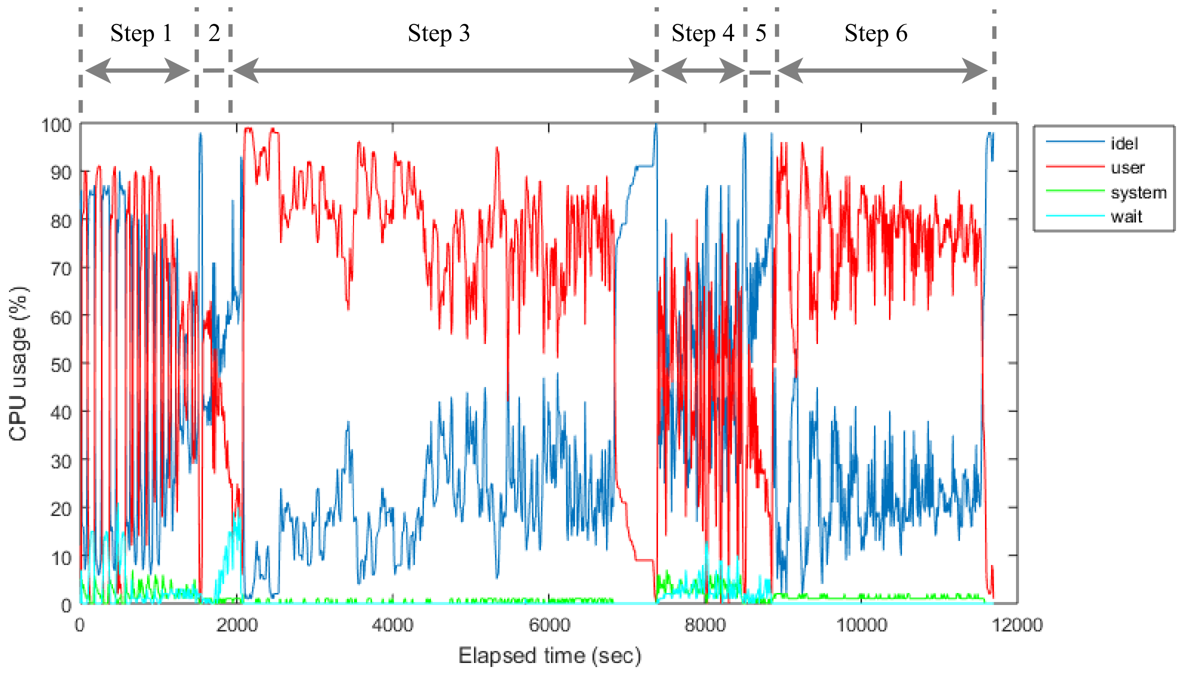


Figure 5.4: CPU usage of each step in the pipeline

Table 5.4: Size features with different subregional numbers

Number of sub-regions	Size of the input SAM files (MB)			
	Minimum	Maximum	Mean	Standard deviation
50	25	1218	585	250
150	7	599	182	97
300	3	337	89	57

step decreases from 76 minutes with 50 sub-regions to 54 minutes with 150 sub-regions, finally 40 minutes with 300 sub-regions.

5.2.4. Result Correctness

Apart from speed, we also evaluated result correctness of our scalability solution. Our scalability solution should detect as many as possible right variants and as little as possible wrong variants in the final VCF file. The VCF file generated by the original early DNA analysis pipeline is the golden standard our scalability solution should output. The correctness of the results is verified by measuring the APR value between the VCF files from the scalability solution and the original pipeline as described in Chapter 3.

The comparison result is presented in Table 5.5 and illustrated in Figure 5.6. There are 87238 variants detected by our scalability solution overlapping with the detection result of the original pipeline, namely, 87238 true positives. There are 2082 false positives wrongly detected by our parallelized early DNA analysis pipeline, 1695 false negatives absent in the output VCF file of the scalability solution but shown in the result of the original pipeline.

Table 5.5: Comparison of the variants detected by two pipelines

Variants	Original pipeline	Scalability solution	Number
True positives	$\surd$	$\surd$	87238
False positives	$\times$	$\surd$	2082
False negatives	$\surd$	$\times$	1695

Table 5.6 presents the precision, recall, and APR of the comparison result. The precision-recall graph is calculated by RTG tools [35]. As shown in the table, our scalability solution achieves 97.48% effectiveness to

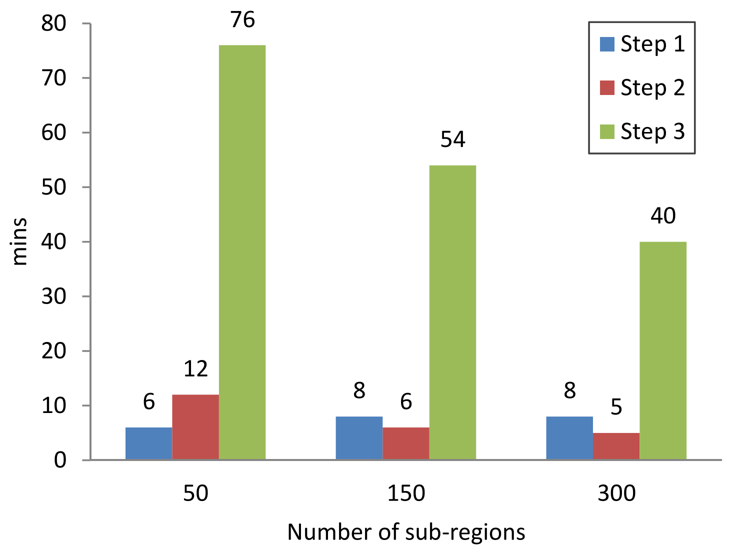


Figure 5.5: Time consumption with different subregional numbers

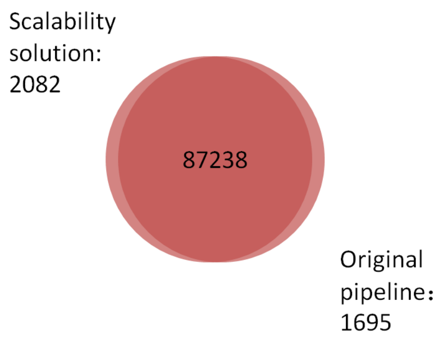


Figure 5.6: Variants detected by two pipelines

detect correct variants as compared to the original pipeline.

Table 5.6: Result correctness of our scalability solution

Parameter	Method	Reuslt
Precision	TP/(TP+FP)	97.67%
Recall	TP/ (TP+FN)	98.09%
APR	Area under the precision-recall curve	97.48%

5.3. Summary

Given the experimental results we obtained, the performance of our scalability solution was evaluated by the selected criterion to compare with the original early DNA analysis pipeline. Our scalability solution could strikingly improve the parallelization of the original pipeline and accelerate the entire process by more than 7 times speed-up. Our solution also achieves 97.48% effectiveness to detect the right variants in the result VCF file. Our parallelized early DNA analysis pipeline addresses the scalability problem of the thesis and shows good performance.

6
Experiment of Advanced Completion Approaches

In this chapter, the accuracies of different advanced completion approaches are evaluated. To train machine learning models, we need to have a training data set. The data extraction method is introduced first in Section 6.1. The evaluation of different machine learning methods is presented in Section 6.2 and Section 6.3, illustrating the accuracy of the classification methods and the regression methods respectively. Finally, a summary of the evaluation is presented in Section 6.4.

6.1. Data Extraction

As described in Section 5.1, we used NA12878 data set with 150x coverage and 100 base pairs for each read. In machine learning analysis, we divided this input data set into two part: a training set with 100x coverage and a test set with 50x coverage. Thus two-thirds of the raw data is used for training, while one-third is used when doing the test.

As shown in Figure 6.1, the overlapping reads are used to complete the unknown bases of the incomplete reads in the original early DNA analysis approach. For instance, there are two incomplete reads containing unknown bases at position X. The overlapping bases at this position consist of 6 A and 1 G. For the consensus-based approach, the major vote of the overlapping reads is A, thereby base A will be used to complete both incomplete read 1 and incomplete read 2 at this position.

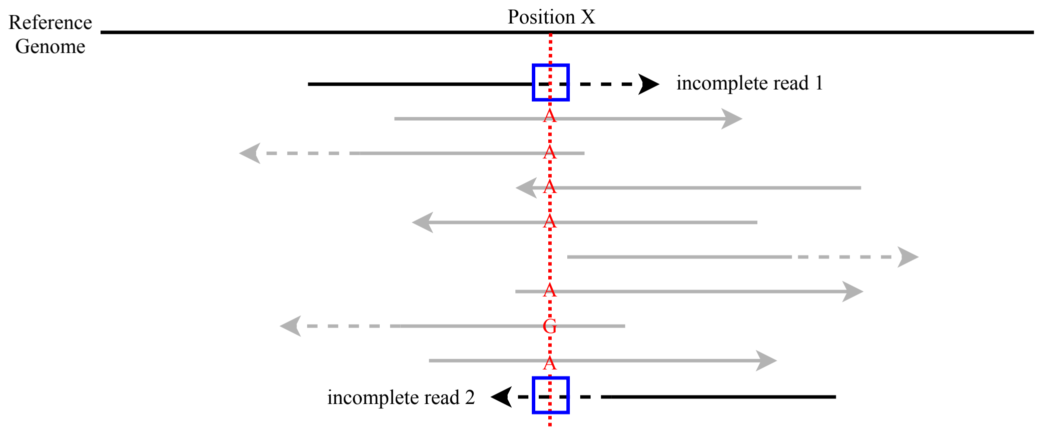
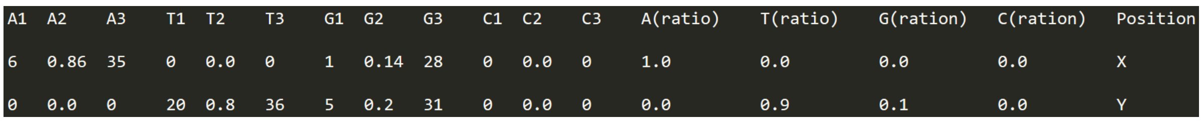


Figure 6.1: The unknown bases and their overlapping bases at Position X

In the machine learning model, we also use the overlapping bases as the input. The original value at the unknown position sequenced by the machines is the ground truth our model should predict. At position X, input data consists of all overlapping bases in the gray reads while the ground truth of this position is the actual sequenced values from the machines. Besides the base types in the overlapping reads, their average base quality scores are also included for the input data. The bases in the input data are described from two aspects: the number of each base and their corresponding ratios. The ground truth is also presented as the ratio. As we manually cut our raw data set, the actual values in that two blue rectangles in Figure 6.1 still can be found in the cut pieces of the incomplete reads. For position X, if the actual sequenced results in that two blue rectangles are 2 A, this position will be described by as Figure 6.2 in our data set. Another example record at position Y in the data set is also presented in the figure.



A1	A2	A3	T1	T2	T3	G1	G2	G3	C1	C2	C3	A(ratio)	T(ratio)	G(ration)	C(ration)	Position
6	0.86	35	0	0.0	0	1	0.14	28	0	0.0	0	1.0	0.0	0.0	0.0	X
0	0.0	0	20	0.8	36	5	0.2	31	0	0.0	0	0.0	0.9	0.1	0.0	Y

Figure 6.2: Two samples in the data set. The features of the input data are presented in the first twelve columns. A1, A2, and A3 stand for the number of A, the ratio of A, and the average quality score of A respectively. The same expression also applies for base T, G, C. The last 4 columns represent the ratio of A, T, G, C in the ground truth before the position column.

For all positions containing unknown bases in chromosome 1, we collected the overlapping bases and their ground truth, which forms our learning data set. For both training set and test set, the same data extraction method was performed. There are 29298538 positions recorded in the training set, and 15403715 positions recorded in the test set. The training data was used to train the models while the test data was used as the evaluation.

6.2. CLASSIFICATION METHODS

In the training set, 27413500 out of 29298538 samples contain only one type of base in the ground truth. To simplify the model, we first regard this completion task as a classification problem. As a result, the dominant base in the ground truth becomes the actual value our machine learning models should predict instead of the ratio distribution. As long as the output base from the model is the same as the dominant base in the ground truth, it is regarded as a correct prediction. In this section, several classification methods were tested as compared to the consensus-based approach.

6.2.1. CONSENSUS-BASED APPROACH

The consensus-based model in the original early DNA analysis approach acts as the baseline in the comparison. If the consensus base in the input data is the same as the dominant base, namely the majority vote of the bases in the ground truth, it is regarded as a correct classification by the consensus model. There are 15195003 out of 15403715 samples in the test set correctly classified by the consensus approach, which achieves 98.645% accuracy.

6.2.2. DECISION TREES

An algorithm called decision trees was explored first. We used Scikit-learn machine learning library to test different algorithms. A typical application of the machine learning algorithms in Scikit-learn is shown in Listing 6.1.

Listing 6.1: Application of decision trees in Scikit-learn

```
import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# read the training data from the txt file
df = pd.read_csv('training_data.txt', delim_whitespace=True, header=None)

# training features: column 0 to 11.
# A1  A2  A3  T1  T2  T3  G1  G2  G3  C1  C2  C3
X = df.loc[:,0:11].as_matrix()

# ground truth ratios: column 12 to 15
# A(ratio) T(ratio) G(ratio) C(ratio)
```

```
Y = df.loc[:,12:15].as_matrix()

# dominant base in the ground truth
Y_n = Y.argmax(axis=1)

# create the test set
test = pd.read_csv('test_data.txt', delim_whitespace=True, header = None)
X_test = test.loc[:,0:11].as_matrix()
Y_test = test.loc[:,12:15].as_matrix()
Y_test_n = Y_test.argmax(axis=1)

# create a DecisionTreeClassifier with maximal tree depth of 10
clf = tree.DecisionTreeClassifier(max_depth=10)

# train the model with the training data
clf.fit(X, Y_n)

# use the trained model to predict the value for input data in test set
predictions = clf.predict(X_test)

# compare the prediction value with the ground truth in test set
print(accuracy_score(Y_test_n, predictions))
print(accuracy_score(Y_test_n, predictions, normalize=False))
```

We ran this program to evaluate the accuracy of decision trees. Results show that 15195147 output of 15403715 samples in the test set are correctly classified by `DecisionTreeClassifier`, which is only 144 more than the consensus model. Other parameters were also tested, such as the node separation criterion. Listing 6.2 shows the `DecisionTreeClassifier` with entropy-based construction method. This application gave us a slightly better result with 15195509 correctly classified samples and 98.648% accuracy.

Listing 6.2: Application of decision trees using entropy

```
# create a DecisionTreeClassifier with maximal tree depth of 10 and node separation
# criterion of entropy, other parameters as default
clf = tree.DecisionTreeClassifier(max_depth=10, criterion='entropy')
```

6.2.3. RANDOM FORESTS

A random forest model is implemented in Scikit-learn as the following Listing 6.3. After optimization, the presented running parameter below gave the best accuracy. Random forests gave us better classification results than simple decision trees with 15195858 correctly classified samples and 98.651 % accuracy. However, it still increases very little as compared to the baseline, only 855 more correctly classified samples.

Listing 6.3: Application of random forests in Scikit-learn. Data loading job is the same as Listing 6.1.

```
from sklearn.ensemble import RandomForestClassifier

# load training data and test data as X, Y_n, X_test, Y_test_n
......

# create a RandomForestClassifier with maximal tree depth of 10, node separation
# criterion of entropy, parallel running jobs as the cores number
clf = RandomForestClassifier(criterion='entropy', max_depth=10, n_jobs=-1)

# train the model with the training data
clf.fit(X, Y_n)

# use the trained model to predict the value for input data in test set
predictions = clf.predict(X_test)
```

```
# compare the prediction value with the ground truth in test set
print(accuracy_score(Y_test_n, predictions))
print(accuracy_score(Y_test_n, predictions, normalize=False))
```

6.2.4. NAIVE BAYES

Other methods such as Naive Bayes were also tested. An application of Multinomial Naive Bayes is shown in Listing 6.4. However, they are all less accurate than random forests. Multinomial Naive Bayes gave us 98.631% accuracy, which is even worse than the consensus-based model.

Listing 6.4: Application of Multinomial Naive Bayes in Scikit-learn. Data loading and evaluation method are the same as Listing 6.1.

```
from sklearn.naive_bayes import MultinomialNB

# create a MultinomialNB classifier
clf = MultinomialNB(alpha=.01)

# train the model with the training data
clf.fit(X, Y_n)
```

6.2.5. ERROR ANALYSIS

Among these machine learning methods discussed above, random forests gave us the best accuracy. 15195858 out of 15403715 samples in the test set are correctly classified by random forests with accuracy 98.651%. We recorded those wrongly classified samples which are 207857 in total and summarized them in three major types as shown in Table 6.1.

Table 6.1: The number of different errors

Error type	Number
1	156209
2	51648
3	10265

Details of each type are introduced as follows:

1. Type 1 errors account most in all three types. They are samples that only contain one base type in the input data. Most likely the trained model will pick this base as the prediction while the dominant base in the ground truth is different. These errors either come from lack of overlapping or sequencing machine errors. On the one hand, random forests model could not perform a correct prediction without enough overlapping bases at some positions. On the other hand, sequencing machines could also make mistakes during the sequencing process. Ideally, the bases sequenced at a position should be the same. If the dominant base in the ground truth is just the errors machines made, our random forests model would also make a wrong prediction.

2. Type 2 errors are the samples that only contain bases with equal counts in the input data. In this situation, our random forests model does not perform very well and picks the wrong value out of the tie bases. Other features such as the average quality scores do not lead the model to make correct predictions.

3. Type 3 errors are mainly from the samples that contain more than one dominant bases with equal counts in the ground truth. In this situation, our $argmax()$ function used when loading the data will fail. It always picks the base with smaller index as the dominant base by default.

6.3. REGRESSION METHODS

Instead of taking the dominant base in the ground truth, now we need to predict the actual ratio of each base without any modification. Thus it becomes a regression problem. We evaluated the prediction accuracy by a simple criterion: if any predicted ratio of four bases differs from its ground truth ratio by more than 0.1, it is regarded as an incorrect prediction. Experiment results of different regression methods are described in this section.

6.3.1. BASELINE

The consensus model outputs a base with 100% ratio as the prediction. A simple intuition that this approach will have a bad accuracy turned out to be incorrect. In fact, the consensus-based approach achieves 96.668% accuracy. The probabilistic approach proposed in the original early DNA analysis approach was also used as a baseline comparison. The prediction ratio of each base will be the same as the ratio in the input data. This probabilistic approach achieves 96.174% accuracy.

6.3.2. LINEAR REGRESSION

To evaluate the regression methods, we also used Scikit-learn library to build different regressors. We first explored a ridge regression model as presented in Listing 6.5. A evaluation function that verifies the absolute value of the difference between Y_test and $predictions$ is defined. If none of the differences are beyond threshold, it will return 1 as a correct prediction.

Listing 6.5: Application of ridge regression in Scikit-learn

```python
import pandas as pd
import numpy as np
from sklearn import linear_model

# read the training data from the txt file
df = pd.read_csv('training_data.txt', delim_whitespace=True, header=None)
# training features: column 0 to 11.
X = df.loc[:,0:11].as_matrix()
# ground truth ratios: column 12 to 15
Y = df.loc[:,12:15].as_matrix()

# create the test set
test = pd.read_csv('test_data.txt', delim_whitespace=True, header = None)
X_test = test.loc[:,0:11].as_matrix()
Y_test = test.loc[:,12:15].as_matrix()

# create a ridge regressor
reg = linear_model.Ridge()

# train the model with the training data
reg.fit(X, Y)

# use the trained model to predict the value for input data in test set
predictions = reg.predict(X_test)

# evaluation criterion
threshold = 0.1
def evaluation( x ):
    if x[0]>threshold or x[1]>threshold or x[2]>threshold or x[3]>threshold :
        return 0
    else:
        return 1

# compare the prediction value with the ground truth in test set
diff = np.absolute(np.subtract(Y_test, predictions))
correct = np.apply_along_axis(evaluation, axis=1, arr=diff)
correct_num = np.sum(correct)
total_num = len(Y_test)
accuracy = float(correct_num)/total_num
```

As compared to the baseline accuracy, ridge regression model gave a much worse result with only 72.101% accuracy. There is no such linear function found yet that could correctly describe the correlations between input features and the ratios in the ground truth.

6.3.3. DECISION TREES

In Scikit-learn library, regressor based on decision trees is also provided. Listing 6.6 describes a decision tree regressor with maximal tree depth of 10. Parameters of the tree were optimized. Unlike decision trees classifiers, regressors output the actual value rather than the base type itself. The accuracy evaluation method is the same as the experiment in ridge regression. Decision trees regressor achieves 95.525% accuracy, which is worse than both the probabilistic model and consensus-based model.

Listing 6.6: Regression application of decision trees in Scikit-learn. Data loading and evaluation method are the same as 6.5.

```python
from sklearn.tree import DecisionTreeRegressor

# create a decison tree regressor
reg = DecisionTreeRegressor(max_depth=10)

# train the model with the training data
reg.fit(X, Y)
```

6.3.4. RANDOM FORESTS

We also tested random forests in the regression problem. An application of random forest regressor is described in Listing 6.7. The experiment result of this regressor turned out to be better than ridge regression and decision trees, with accuracy 95.739%. However, it still underperforms the baseline approaches.

Listing 6.7: Regression application of random forests in Scikit-learn. Data loading and evaluation method are the same as 6.5.

```python
from sklearn.ensemble import RandomForestRegressor

# create a random forest regressor
reg = RandomForestRegressor(max_depth=10, n_jobs=-1)

# train the model with the training data
reg.fit(X, Y)
```

6.3.5. FEATURE IMPORTANCE ANALYSIS

As described in section 6.1, the input data is described by 12 features, the number of each base, the ratio of each base, and the average quality score of each base. To understand how much each feature influences the final prediction, we conducted a few experiments to explore the importance of each feature in random forests. The importance of the features can be obtained as follows:

Listing 6.8: Feature importance exploration

```python
from sklearn.ensemble import RandomForestRegressor

# create a random forest regressor
reg = RandomForestRegressor(max_depth=10, n_jobs=-1)
# train the model with the training data
reg.fit(X, Y)
# obtain the importances of the features
importance = reg.feature_importances_
```

According to the results, if all twelve features are used in the input data, the ratio of T, A, G, C are the most important features that determine the prediction. Since the ratio of each base is calculated from the counts, we removed the counts in the feature set and reran the training process with only ratio and average quality score in the input data. As a result, the ratio of T, A, G, C still dominate the output as compared to the average quality score. Nevertheless, if we ran the library with only count and average quality score of each base in the input data, the most important feature changed. The average quality score of T, A, G became the dominant features in the input data.

6.4. SUMMARY

In this chapter, different machine learning methods were tested to predict the unknown bases in the incomplete DNA data. We started the analysis by regarding this problem as a multi-label multi-class classification problem since we have more than one features in the input data and four types of bases. When it comes to deciding the dominant base type for unknown bases, random forests gave the best accuracy as compared to other machine learning approaches and the consensus-based approach proposed in the original work. As shown in Table 6.2, the consensus-based model achieves a high prediction accuracy of 98.645 % while random forests method is able to increase the baseline accuracy by 0.006%. Afterwards, we conducted the experiments for regression. It is a multi-output regression problem as we need to predict the actual ratio of each base. As presented in Table 6.3, probabilistic model decreases the prediction accuracy a little bit as compared to the consensus-based model, which reflect the research in [8]. None of the machine learning methods tested yet could beat the consensus-based approach.

Table 6.2: The accuracy of different classification methods

Completion methods	Accuracy
Consensus	98.645 %
Decision Trees	98.648 %
Random Forests	98.651 %
Multinomial Naive Bayes	98.631 %

Table 6.3: The accuracy of different regression methods

Completion methods	Accuracy
Consensus	96.668 %
Probabilistic model	96.174 %
Ridge regression	72.101 %
Decision Trees	95.525 %
Random Forests	95.739 %

7
CONCLUSIONS

In this chapter, a summary of all previous chapters is given first in Section 7.1. Conclusions of the thesis are listed in Section 7.2. Finally, the future work of the research is suggested in Section 7.3.

7.1. SUMMARY

The first problem statement of the thesis was:

How could we improve the scalability and efficiency of the early DNA analysis pipeline?

To answer this question, the following three steps were established:

1. Compare the scalability solutions of the mainstream genome analysis pipeline and select one as the infrastructure to be integrated with.

   In Section 3.1, we first investigated three scalability solutions of the genome analysis pipeline. The advantages and disadvantages of the solutions were addressed. SparkGA provides the best speed performance and good convenience to be integrated with, thus being selected as the infrastructure of our parallel implementation of the early DNA analysis pipeline.

2. Design and implement the scalable early DNA analysis solution.

   Since the standard genome analysis phase in the original pipeline was replaced by SparkGA, the primary work of the thesis was to implement the parallelized completion phase of the early DNA analysis pipeline. An integration step was also included in the implementation to integrate with SparkGA. The parallelized completion phase consists of three major steps: mapping and load distribution; merging and sorting; completion and integration with SparkGA. The implementation details were introduced in Chapter 4.

3. Evaluate and validate the performance of the implementation.

   In Chapter 5, we conducted several experiments to evaluate the performance of the scalable early DNA analysis approach. Based on the measurement results, the speed performance of the scalability solution was illustrated as compared to the original approach. The speed of the parallelized completion phase was also described in particular. Besides, the result correctness of the implementation was evaluated to validate our scalability solution.

The second problem statement of the thesis was:

How could we improve the accuracy of the early DNA analysis approach using more advanced algorithms?

In the original early DNA analysis approach, a consensus-based approach and a probabilistic approach are adopted to complete the unknown bases and their respective base quality scores in the incomplete DNA reads. To answer the second question, a few machine learning algorithms were introduced as the candidate solutions in Section 3.3. In Chapter 6, we extracted the input data set from the raw data and trained these data with the advanced machine learning methods. The completion accuracies of these methods were evaluated as compared to the simple completion methods in the original approach.

7.2. Conclusions

In this thesis, a parallel implementation of the early DNA analysis pipeline is proposed to solve the scalability problem. This scalability solution could distribute the tasks in the computing cluster and accelerate the whole pipeline. The accuracy problem of the original early DNA analysis approach is also explored with more advanced completion methods. From the analysis and the measured results, the following conclusions of the thesis are obtained:

1. The original early DNA analysis approach has a sequential pipeline where the completion step and the variant calling step consume a huge amount of computational time. By spreading the time-consuming tasks uniformly to the nodes in the cluster, our scalability solution towards early DNA analysis could bring a 7.6× speed-up when deployed on a 4-node Power7+ cluster as compared to the original pipeline. Meanwhile, the output VCF file from the scalable early DNA analysis approach achieves 97.48% effectiveness to detect correct variants. Saving massive computational time of the early DNA analysis approach and ensuring correctness of the result allow the novel idea of saving sequencing time for genome analysis to be applied in practice.

2. The simple consensus-based completion model for the incomplete DNA data proposed in the original early DNA analysis approach has a good prediction accuracy. When it comes to classifying the unknown bases into 4 base types (A, T, G, C), our advanced completion method using random forests could beat the consensus-based model by 0.006%. Nevertheless, none of the machine learning algorithms tested yet could outperform the consensus-based model if the actual ratios of the unknown bases are required to be predicted. The consensus-based model provides 96.668% accuracy to predict ratios, which reflects its correctness in the original early DNA analysis research.

7.3. Future Work

This thesis provides a scalability solution to accelerate the pipeline for early DNA analysis. A few machine learning methods are also explored to improve the accuracy of the completion step in the original pipeline. To fulfill wider application of our solution, the current research can be continued from several aspects as follows:

1. The completion phase in the scalability solution can be extended with dynamic load balancing to accelerate the pipeline further. Sub-regions containing too many reads can be further split. Creation of a method that could equally divide these sub-regions and remain the data dependencies is an option for the future work.

2. The simple consensus-based completion method proposed in the original early DNA analysis approach turns out to be more accurate than the tested machine learning algorithms in predicting the ratios. Better tuned machine learning methods or deep neural network could be implemented to improve the accuracy in the future.

3. In this thesis, we used a simple criterion to evaluate the accuracy of different completion approaches. To better understand the influence of the completion results on the original pipeline, the predicted bases from different advanced completion methods should be placed back to the incomplete DNA data and processed by the pipeline itself. An interface that could connect the pipeline and the predicted bases should be created in order to efficiently evaluate the accuracy of different completion approaches on variant detection.

BIBLIOGRAPHY

[1] Colten, *Dna structure,* http://tiger21-coltensblog.blogspot.com/2012/04/dna-structure.html (2012), accessed July, 2, 2018.

[2] Broad Institute, *Germline short variant discovery,* https://software.broadinstitute.org/gatk/best-practices/workflow?id=11145 (2018), accessed July, 10, 2018.

[3] H. Mushtaq, F. Liu, C. Costa, G. Liu, P. Hofstee, and Z. Al-Ars, *Sparkga: A spark framework for cost effective, fast and accurate dna analysis at scale,* in *Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology,and Health Informatics,* ACM-BCB '17 (ACM, New York, NY, USA, 2017) pp. 148–157.

[4] N. Ahmed, V. Sima, E. Houtgast, K. Bertels, and Z. Al-Ars, *Heterogeneous hardware/software acceleration of the bwa-mem dna alignment algorithm,* in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)* (2015) pp. 240–246.

[5] S. Ren, K. Bertels, and Z. Al-Ars, *Gpu-accelerated gatk haplotypecaller with load-balanced multi-process optimization,* in *Proc. 17th annual IEEE International Conference on BioInformatics and BioEngineering* (Washington DC, USA, 2017).

[6] H. Mushtaq and Z. Al-Ars, *Cluster-based apache spark implementation of the gatk dna analysis pipeline,* in *Proc. IEEE International Conference on Bioinformatics and Biomedicine* (Washington DC, USA, 2015).

[7] Illumina, *Illumina hiseq-2500 system specifications,* https://www.illumina.com/documents/products/datasheets/datasheethiseq2500.pdf (2015), accessed July, 2, 2018.

[8] N. Ahmed, K. Bertels, and Z. Al-Ars, *Predictive genome analysis using partial dna sequencing data,* in *Proc. 17th annual IEEE International Conference on BioInformatics and BioEngineering* (Washington DC, USA, 2017) pp. 119–124.

[9] Wikipedia, *Nucleotide,* https://www.wikiwand.com/en/Nucleotide (2018), accessed July, 2, 2018.

[10] A. Warr, C. Robert, D. Hume, A. Archibald, N. Deeb, and M. Watson, *Exome sequencing: Current and future perspectives,* G3: Genes|Genomes|Genetics **5**, 1543 (2015), exported from https://app.dimensions.ai on 2018/08/13.

[11] J. M. Heather and B. Chain, *The sequence of sequencers: The history of sequencing dna,* in *Genomics* (2016).

[12] Biog, *Brief introduction on three generations of genome sequencing technology,* https://www.creative-biogene.com/blog/index.php/2016/11/01/brief-introduction-on-three-generations-of-genome-sequencing-technology/ (2016), accessed July, 8, 2018.

[13] K. Schwarze, J. Buchanan, J. Taylor, and S. Wordsworth, *Are whole exome and whole genome sequencing approaches cost-effective? a systematic review of the literature, Value in Health,* **21**, S100 (2018).

[14] I. Albert, *The biostar handbook,* (2016) Chap. Sequencing Instruments.

[15] Illumina, *Sequencing kits for every lab,* https://www.illumina.com/products/by-type/sequencing-kits.html (2018), accessed July, 8, 2018.

[16] G. Highnam, J. J. Wang, D. Kusler, J. M. Zook, V. Vijayan, N. Leibovich, and D. Mittelman, *An analytical framework for optimizing variant discovery from personal genomes,* in *Nature communications* (2015).

[17] S. Hwang, E. Kim, I. Lee, and E. Marcotte, *Systematic comparison of variant calling pipelines using gold standard personal exome variants, Scientific Reports,* **5**, 17875 (2015).

[18] H. Li, *Aligning sequence reads, clone sequences and assembly contigs with bwa-mem,* (2013).

[19] M. T. W. Ebbert, M. E. Wadsworth, L. A. Staley, K. L. Hoyt, B. Pickett, J. Miller, J. Duce, for the Alzheimer's Disease Neuroimaging Initiative, J. S. K. Kauwe, and P. G. Ridge, *Evaluating the necessity of pcr duplicate removal from next-generation sequencing data and a comparison of approaches,* BMC Bioinformatics **17**, 239 (2016).

[20] The Apache Software Foundation, *Apache spark,* `https://spark.apache.org/` (2018), accessed July, 10, 2018.

[21] S. Ren, K. Bertels, and Z. Al-Ars, *Exploration of alternative gpu implementations of the pair-hmms forward algorithm,* in *Proc. 3rd International Workshop on High Performance Computing on Bioinformatics* (Shenzhen, China, 2016).

[22] B. J. Kelly, J. R. Fitch, Y. Hu, D. J. Corsmeier, H. Zhong, A. N. Wetzel, R. D. Nordquist, D. L. Newsom, and P. White, *Churchill: an ultra-fast, deterministic, highly scalable and balanced parallelization strategy for the discovery of human genetic variation in clinical and population-scale genomics,* Genome Biology **16**, 6 (2015).

[23] D. Decap, J. Reumers, C. Herzeel, P. Costanza, and J. Fostier, *Halvade: scalable sequence analysis with mapreduce,* Bioinformatics **31**, 2482 (2015).

[24] M. Niemenmaa, A. Kallio, A. Schumacher, P. Klemelä, E. Korpelainen, and K. Heljanko, *Hadoop-bam: directly manipulating next generation sequencing data in the cloud,* in *Bioinformatics* (2012).

[25] Scikit-learn, *Precision-recall,* `http://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html` (2017), accessed July, 12, 2018.

[26] Wikipedia, *Precision and recall,* `https://www.wikiwand.com/en/Precision_and_recall` (2018), accessed July, 12, 2018.

[27] Wikipedia, *Linear regression,* `https://www.wikiwand.com/en/Linear_regression` (2018), accessed July, 12, 2018.

[28] *Regularization: Ridge regression and the lasso,* `http://statweb.stanford.edu/~tibs/sta305files/Rudyregularization.pdf` (2006), accessed July, 12, 2018.

[29] Wikipedia, *Bayes' theorem,* `https://www.wikiwand.com/en/Bayes%27_theorem` (2018), accessed July, 14, 2018.

[30] D. Soni, *Introduction to naive bayes classification,* `https://towardsdatascience.com/introduction-to-naive-bayes-classification-4cffabb1ae54` (2016), accessed July, 14, 2018.

[31] C. Apté and S. Weiss, *Data mining with decision trees and decision rules,* Future Generation Computer Systems **13**, 197 (1997), data Mining.

[32] J. R. Quinlan, *Induction of decision trees,* Machine Learning **1**, 81 (1986).

[33] A. Liaw and M. Wiener, *Classification and Regression by randomForest,* R News **2**, 18 (2002).

[34] IBM, *Power7 and power7+ optimization and tuning guide,* (2013) Chap. The POWER7 processor.

[35] J. G. Cleary, R. Braithwaite, K. Gaastra, B. S. Hilbush, S. Inglis, S. A. Irvine, A. Jackson, R. Littin, M. Rathod, D. Ware, J. M. Zook, L. Trigg, and F. M. M. De La Vega, *Comparing variant call files for performance benchmarking of next-generation sequencing variant calling pipelines,* bioRxiv (2015), 10.1101/023754, https://www.biorxiv.org/content/early/2015/08/02/023754.full.pdf .