Hadoop-gebaseerde oplossingen voor de identificatie en de analyse van varianten

Hadoop-Based Solutions for Variant Calling and Variant Analysis

Dries Decap

Promotor: prof. dr. ir. J. Fostier
Proefschrift ingediend tot het behalen van de graad van
Doctor in de ingenieurswetenschappen: computerwetenschappen

UNIVERSITEIT
GENT

Vakgroep Informatietechnologie
Voorzitter: prof. dr. ir. B. Dhoedt
Faculteit Ingenieurswetenschappen en Architectuur
Academiejaar 2016 - 2017

ISBN 978-94-6355-020-8
NUR 980, 923
Wettelijk depot: D/2017/10.500/55

Universiteit Gent
Faculteit Ingenieurswetenschappen en Architectuur
Vakgroep Informatietechnologie

Leden van de examencommissie

Promotor
prof. dr. ir. Jan Fostier (Universiteit Gent)

Voorzitter
em. prof. dr. ir. Daniël De Zutter (Universiteit Gent)

Leescommissie
prof. dr. Roel Wuyts (Imec)
dr. ir. Joke Reumers (The Janssen Pharmaceutical Companies of Johnson & Johnson)
prof. dr. Jim Dowling (KTH Royal Institute of Technology, Stockholm)
prof. dr. Tim De Meyer (Universiteit Gent)

Overige leden
prof. dr. ir. Bart Dhoedt (Universiteit Gent)

Universiteit Gent
Faculteit Ingenieurswetenschappen en Architectuur

Vakgroep Informatietechnologie
iGent Toren, Technologiepark 15, B-9052 Gent, België
Tel.: +32 9 33 14900

Dankwoord

In Juni 2011 studeerde ik af aan de Hogeschool Gent. Ik had net mijn masterproef bij Jan en Bart afgewerkt. Maar wat nu? Ik had het idee om verder te studeren om een diploma Master in de Computerwetenschappen te behalen aan de Universiteit Gent. Maar dit zou betekenen dat ik nog twee extra jaar zou moeten studeren. Toen vroeg Jan mij of ik geen interesse had om een doctoraat te doen. Dit was een optie waar ik nog niet over had nagedacht en twijfelde in eerste instantie. Nu weet ik dat ik de juiste beslissing genomen heb.

Bedankt Jan om mij de mogelijkheid te geven om mijn onderzoek te doen. Ik wil je ook graag bedanken voor de enorm leerrijke ervaring die je mogelijk maakte. Je interesse in en kennis van high performance computing verbazen me nog steeds. Telkens wist je mij te helpen als ik een struikelblok tegenkwam. Onze meetings waren telkens interessant en to-the-point, maar daarom niet minder leuk. Dit onderzoek zou nooit geworden zijn wat het nu is zonder uw enorme hulp.

De jury van mijn doctoraat wil ik ook graag bedanken. Thank you **prof. Tim De Meyer, prof. Jim Dowling, dr. Joke Reumers, prof. Roel Wuyts and prof. Bart Dhoedt** for your critical look on my work and the interesting questions that helped improve the quality of this work. The suggestions and comments on my work are certainly interesting and will definitely help me continue my work. Ook wil ik **prof. Daniël De Zutter** bedanken voor het in goede banen leiden van mijn verdediging.

Vervolgens wil ik **Joke, Charlotte, Pascal, Toni en Thomas** bedanken. Jullie interesse in het vak en kritische kijk op mijn onderzoek zijn zeer gewaardeerd. Bedankt ook voor het nalezen van mijn papers en de heel veel en soms moeilijke vragen die je erbij stelde. Daarnaast bedank ik ook graag **Tom en Roel** voor hun interesse in mij en mijn onderzoek alsook voor de interessante gesprekken tijdens de lunchpauzes bij Imec. Het ExaScience project was een enorm leuke ervaring die vooral door jullie zo leerrijk en aangenaam was.

Een grote dankuwel voor het admin team (**Bert, Brecht, Joeri, Simon**), die telkens verbazingwekkend snel een antwoord gaf op mijn vraag. Daarnaast ook een welgemeende dankuwel voor **Martine, Bernadette en Davinia** die telkens raad wisten toen ik tegen een administratief probleem aanliep. Daarnaast bedank ik graag nog het HPC admin team (**Ewan, Ewald, Kenneth en Stijn**), die zorgden voor de wonderbaarlijke hacks en fixes alsook de ongelofelijk snelheid om alle vragen te beantwoorden en op te lossen.

Daarnaast nog een grote dankjewel aan alle collega's die ik tijdens mijn onderzoek leren kennen heb. **Lieven, Carolina, Joeri, Dieter, Kathleen, Pieter, Sergio, Oil, Clemens, Yan, Giles, Leen, Ine, Mahdi, Jimmy, Dries and Camilo** it was amazing to get to know everybody. You really are the best colleagues anyone can wish

for. More often than not, the lunches were filled with interesting and/or entertaining chatter. As well as the very relaxing and fun badminton matches we had!

Daarnaast wil ik zeker ook **Alexander, Kris en Bert** bedanken om telkens met interesse te luisteren naar wat ik aan het onderzoeken was. Ook bedankt voor de zeer toffe, interessante, inspirerende en veelzijdige gesprekken die we over de jaren gevoerd hebben, al dan niet met een drankje. Ook **Yun** wil ik graag bedanken. Zij was er telkens bij toen ik hulp nodig had bij het nalezen en pushte me om telkens door te gaan en telkens harder verder te werken.

Ten slotte zou ik graag mijn **ouders** willen bedanken, jullie wisten altijd raad in de makkelijke, maar ook in de moeilijkere tijden van het onderzoek. Zeker en vast ook bedankt aan mijn zussen **Sara en Fien** en hun partners **Matthias en Simon**. In het bijzonder wens ik Sara te bedanken voor de enorme hulp bij het nalezen van de papers en mijn manuscript. Jullie oprechte interesse heeft me telkens opnieuw moed en wilskracht gegeven om mijn onderzoek verder te zetten.

Aan allemaal een welgemeende dankuwel!

Gent, Juni 2017
Dries Decap

Table of Contents

Dankwoord **i**

Samenvatting **xvii**

Summary **xxi**

1 Introduction **1**

1.1 DNA sequencing 1

1.1.1 Illumina sequencing 3

1.2 The DNA-seq variant calling pipeline 5

1.2.1 Read Alignment 7

1.2.1.1 DNA sequencing read alignment with BWA 8

1.2.1.2 The SAM Format 8

1.2.2 Marking PCR Duplicates 9

1.2.3 Indel Realignment 10

1.2.4 Base Quality Score Recalibration 10

1.2.5 Variant Calling using GATK 12

1.2.5.1 UnifiedGenotyper 13

1.2.5.2 HaplotypeCaller 13

1.2.5.3 The VCF Format 14

1.3 Differences for RNA sequencing data 15

1.3.1 RNA read alignment with STAR 15

1.3.2 Split Reads By CIGAR String 16

1.3.3 Variant calling with RNA-seq data 16

1.4 The MapReduce Programming Model 16

1.4.1 Apache Spark and Spark Notebook 18

1.4.2 VCF analysis and comparison 19

1.5 Overview of the chapters 20

References 21

2 Halvade: scalable sequence analysis with MapReduce **25**

2.1 Introduction 26

2.2 Methods 28

2.2.1 Halvade framework 28

2.2.1.1 Input data preparation 28

2.2.1.2 Map phase – read alignment 29

2.2.1.3 Reduce phase – variant calling 30

2.2.2 Best Practices DNA-seq implementation 30
2.2.3 Optimizations 32
2.2.3.1 The optimal number of parallel tasks per node . . . 33
2.2.3.2 Influence of the map/reduce task size 34
2.3 Results 35
2.3.1 Intel Big Data cluster benchmark 35
2.3.2 Amazon EMR benchmark 37
2.3.3 Exome sequencing analysis benchmark 39
2.3.4 Comparison with Crossbow 40
2.3.5 Halvade accuracy assessment 40
2.4 Discussion and Conclusion 42
References 45

3 Performance analysis of a parallel, multi-node pipeline for DNA sequencing 47
3.1 Introduction 48
3.2 Dataset and tool versions 50
3.3 Single Node Benchmarks 51
3.3.1 Influence of the number of tasks per node 51
3.3.2 Influence of the presence of NUMA domains 52
3.4 Multi-node benchmarks 53
3.4.1 Influence of the use of solid state disks 54
3.4.2 Influence of the interconnection network 54
3.4.3 Influence of the file system 55
3.5 Benchmark of NA12878 dataset on a 15-node cluster 55
3.5.1 Load balance considerations: map phase 56
3.5.2 Load balance considerations: reduce phase 57
3.6 Discussion and Conclusion 62
References 64

4 Halvade-RNA: parallel variant calling from transcriptomic data using MapReduce 67
4.1 Introduction 68
4.2 Materials and Methods 69
4.2.1 RNA-seq variant calling pipeline 69
4.2.2 The RNA Pipeline in MapReduce 70
4.2.3 Benchmark setup 72
4.3 Results & Discussion 72
4.3.1 Parallel Performance 72
4.3.2 Multi-sample Throughput 74
4.3.3 Quality Assessment 76
4.4 Conclusion 77
References 79

5 WGS, WES and RNA variant comparison and analysis **81**
5.1 Introduction 82
5.2 Materials and Methods 84
5.2.1 Preprocessing the CCLE data 84
5.2.2 Hadoop, Spark and Spark Notebook 85
5.3 Analysis and Discussion 87
5.3.1 RNA-WGS-WES concordant variants 87
5.3.2 RNA-WES concordant variants 88
5.3.3 WGS-WES concordant variants 88
5.3.4 RNA-WGS concordant variants 90
5.3.5 WES-only variants 91
5.3.6 WGS-only variants 91
5.3.7 RNA-only variants 91
5.4 Conclusions 94
References 96

6 Conclusions and Future Research **99**
6.1 Discussion 99
6.2 Future Work 104
References 106

A Halvade Documentation **107**
A.1 Introduction 107
A.1.1 Recipes 108
A.2 Installation 108
A.2.1 Build from source 108
A.3 The human genome reference 109
A.3.1 BWA reference for WGS/WES data 109
A.3.2 STAR reference for RNA-seq data 110
A.4 Hadoop setup 110
A.4.1 Single node 110
A.4.2 Multi node 112
A.4.3 Hadoop configuration 112
A.4.4 Intel's Hadoop Adapter for Lustre 114
A.5 Uploading the references 114
A.5.1 HDFS 115
A.5.2 Amazon S3 115
A.5.3 GPFS & Lustre 115
A.6 Halvade Preprocessing 116
A.6.1 Performance 116
A.6.2 Synopsis 116
A.6.3 options 116
A.7 Example datasets 116
A.7.1 Whole genome sequencing sample 118
A.7.2 RNA-seq sample 118
A.8 Run Halvade 118

A.8.1 Configuration 119
A.8.2 Run 119
A.8.3 Amazon AWS 120
A.9 Halvade Options 120
A.10 Troubleshooting 125
A.10.1 Find the logs to solve Halvade errors 125
A.10.2 Halvade with BAM input seems stuck at the MarkDuplicates step 125

List of Figures

1.1 Illumina sequencing starts by adding adapters to the chopped DNA molecules. These molecules are attached to a surface with the adapters. Next a bridge is formed and a reverse strand is created through polymerase. Both strands are denatured and further used to amplify in clusters with identical strands. Every cluster is now sequenced in parallel, base per base the sequence is constructed by determining the light, and corresponding base, that is emitted. During analysis overlapping areas are used to create contigs. 4

1.2 Simple example where the reads are first aligned to the reference. These are then sorted according to the genomic position to which they align. Finally, the differences with the reference and the reads are detected and used to call variants. 6

1.3 Simple SAM format example. The header contains the SAM version and the reference sequence dictionary. The read group information is also added to the header. Three aligned reads are added as an example of the columnar SAM format. 9

1.4 Example of realigning reads to adjust for indels. Reads are grouped in piles with similar variations. If a pile has consistent insertions, then these regions will be realigned taking into account this new insertion. 11

1.5 Trivial example of how BQSR works. The average Phred score is calculated from the given Phred scores. The variants that are not in the dbSNP file are considered to be errors (orange square). With this number a new empirical Phred score is calculated, after which the difference with the average Phred score is added to the Phred scores of each base. 11

1.6 VCF format example. The header contains information about filters, format and info field abbreviations, the genome reference dictionary and information about the programs that processed this file. Includes three variants to show the columnar VCF format. 14

1.7 Word count example of the MapReduce programming model. The input data is split in a number of chunks that are processed in parallel during the map phase. Each mapper emits <key-value> pairs where the key corresponds to an observed word and the value corresponds to 1. The <keys-value> pairs are sorted according to key and sent to a number of parallel reduce tasks where each individual reduce task will process one specific key (in this case: one specific word) and all associated values (in this case: a number of 1s). In the reduce task, values that correspond to specific key are aggregated. In this case the sum of all values is computed. An output <key-value> pair is emitted, with again the key corresponding to a word and the value to the number of times the word has been observed. 17

2.1 Overview of the Halvade framework. The entries of pairs of input FASTQ files (containing paired-end reads) are interleaved and stored as smaller chunks. Map tasks are executed in parallel, each task taking a single chunk as input and aligning the reads to a reference genome using an existing tool. The map tasks emit <key, value> pairs where the key contains positional information of an aligned read and the value corresponds to a SAM record. The aligned reads are grouped and sorted per chromosomal region. Chromosomal regions are processed in parallel in the reduce phase, this includes data preparation and variant detection again using tools of choice. Each reduce task outputs the variants of the region it processed. These variants can optionally be merged into a single VCF file. Note that the name of the tools shown correspond to those of the GATK Best Practices DNA-seq implementation in Halvade. 27

2.2 The parallel speedup (multithreading) of five GATK modules used in the Best Practices pipeline on a 16-core node with 94 GByte of RAM. The limited speedup prevents the efficient use of this node with more than a handful of CPU cores. Option -nt denotes data threads while option -nct denotes CPU threads (cfr. GATK manual). 32

2.3 Runtime of the map phase (BWA-aln and BWA-sampe) using 1 task (24 CPU cores per task), 2 tasks (12 CPU cores per task) and 4 tasks (6 CPU cores per task) in parallel per node. Note that in all three cases, 14 worker nodes were used. 35

2.4 Distribution of the execution time of the map tasks using input chunks with a size of 60 MByte and 240 MByte. 36

2.5 Task slot occupation of Halvade on the Intel Big Data cluster using 15 worker nodes (4 parallel tasks/node) on the complete NA12878 dataset. 37

2.6 The speedup (primary y-axis) and parallel efficiency (secondary y-axis) of Halvade as a function of number of parallel tasks (cluster size) on both an Intel Big Data cluster and Amazon EMR. 39

2.7 Normalized frequency as a function of variant confidence score of variants (a) both found by Halvade and a sequential reference run (b) only found by Halvade (c) only found by the reference run. 42

3.1 Halvade MapReduce implementation of a variant calling pipeline. During the map phase, input reads are aligned, in parallel, to a reference genome using BWA. Next, aligned reads are sorted according to aligned position using the MapReduce sorting functionality. In the reduce phase, variants are identified between the reference genome and aligned reads using GATK. 49

3.2 Comparison of the runtime (summed over all 24 parallel tasks) for each individual tool/module used in Halvade with and without optimized NUMA locality. 53

3.3 Disk I/O (scratch) observed on a worker node. Note that almost no data is actually being read from disk as data are still cached in memory. 53

3.4 Network I/O observed on a single worker node. 54

3.5 Panel A: distribution of runtime for map chunks of size 15, 30 and 60 MByte, respectively. Numbers indicate the median runtime. Panel B: runtime of map tasks as a of function of their position in the input FASTQ file. 56

3.6 Panel A: average per-base quality scores in the first 95% of the FASTQ file. Panel B: per-base quality scores in the final 5% of the FASTQ file. Panel C: runtime of map tasks as a function of their position in the shuffled FASTQ file. 58

3.7 Panel A: distribution of runtime for reduce chunks for different genomic chunk sizes. Numbers indicate the median runtime. Panel B: reduce chunk runtime as a function of the number of aligned reads in the chunk for both the GATK UnifiedGenotyper and HaplotypeCaller modules. 59

3.8 Execution of the NA12878 dataset on a 15-node cluster with improved load balancing. Each node runs 24 tasks in parallel. Note that certain tasks are used by MapReduce for task tracking and scheduling purposes. 61

4.1 Overview of the RNA-seq pipeline in Halvade-RNA. In the first job, reads are aligned in parallel in order to identify splice junctions and the reference genome index is rebuilt using this information. In the second job, final alignments are produced and after sorting and grouping the aligned reads by genomic region, the different Picard and GATK steps are executed in parallel. 70

4.2 Runtime distribution of map and reduce tasks of both jobs. Note that the average runtime of the pass 2 map phase increases slightly due to the sparse index. Also note that the reduce phase of the first job is not displayed as this comprises only a single job. 75

4.3 Comparison of variant Quality between Halvade and the reference pipeline. Shows the quality distribution of all variants taken from all 9 samples. 76

5.1 Overview of the analysis process described before. First Halvade processes RNA-seq, WES and WGS data and stores the output data on a shared file system. Next these variant data (VCF files) are preprocessed: the coverage for all three datasets is computed and also stored on the shared file system. As a final step, the data is analyzed in an interactive Spark Notebook with near real-time performance. All these tools are available to be used in the cloud. 85
5.2 Overview of the distribution of the SNV counts for the different sequencing strategies. The numbers represent the sum of all three samples for each collection. 86
5.3 The fraction of each detected functional annotation for every subset. . 87
5.4 Example of allele-specific expression. Showing that allele-specific expression will not be called in RNA-seq when only the reference allele and not the mutation is expressed. 89
5.5 The homozygous fraction of the WGS-WES concordant variants set with increasing coverage threshold. The first bar represents the variants that have no coverage in the RNA-seq dataset. 90
5.6 The homozygous fraction of subsets of the WGS-WES concordance set. Additionally the fraction of the variants per subset is given to give a better perspective on the numbers. 93

List of Tables

2.1 Overview of the steps and tools involved in the DNA sequencing pipeline according to the GATK Best Practices recommendations described by [19]. 31
2.2 Runtime as a function of the number of parallel tasks (mappers/reducers) on the Intel Big Data cluster and Amazon EMR. 38
2.3 Parallel speedup obtained using Crossbow on Amazon EMR. 41
2.4 Overlap and differences in variants in the VCF files obtained by a Halvade run and a sequential reference run. 42

3.1 Runtime and parallel efficiency as a function of the number of tasks per node. 50

4.1 RNA-seq variant calling pipeline used in this work. 69
4.2 Benchmarks of the RNA-seq variant calling pipeline per sample. . . . 73
4.3 Average runtimes per phase of the RNA-seq pipeline. 74
4.4 Runtime for the batch processing of all 9 RNA-seq samples. 75
4.5 Per sample overlap and average quality score. 77

A.1 Halvade Uploader required options. 117
A.2 Halvade Uploader optional options. 117
A.3 Halvade required options. 121
A.4 Halvade optional options, part 1. 121
A.5 Halvade optional options, part 2. 122
A.6 Halvade optional options, part 3. 123
A.7 Halvade optional options, part 4. 124
A.8 Halvade optional options, part 5. 125

List of Acronyms

AWS	Amazon Web Services
Amazon S3	Amazon Simple Storage Service
API	Application program interface
BAM	Binary sequence Alignment/Map
BWA	Burrows Wheeler Aligner
BWT	Burrows Wheeler Transform
BQSR	Base Quality Score Recalibration
CCLE	Cancer Cell Line Encyclopedia
CIGAR	Compact Idiosyncratic Gapped Alignment Report
CPU	Central Processing Unit
DNA	Deoxyribonucleic Acid
DNA-seq	DNA sequencing
HDFS	Hadoop Distributed File System
EMR	Elastic MapReduce
GATK	Genome Analysis ToolKit
GPFS	General Parallel File System
Halvade	Hadoop ALignment and VAriant DEtection
NGS	Next-Generation Sequencing
NUMA	Non Uniform Memory Access
PBS	Portable Batch System
PCR	Polymerase Chain Reaction
RDD	Resilient Distributed Dataset
RNA	Ribonucleic Acid
RNA-seq	RNA sequencing
SAM	Sequence Alignment/Map
SGE	Sun Grid Engine
SNP	Single Nucleotide Polymorphism
SNV	Single Nucleotide Variation

STAR	Spliced Transcripts Alignment to a Reference
VCF	Variant Call Format
WES	Whole Exome Sequencing
WGS	Whole Genome Sequencing
Yarn	Yet Another Resource Negotiator

Bibliography

A1 publications

[1] **Decap, D.**, Reumers, J., Herzeel, C., Costanza, P., Fostier, J. (2015) Halvade: scalable sequence analysis with MapReduce. *Bioinformatics* 31(15), 2482-2488

[2] De Witte, D., Van de Velde, J., **Decap, D.**, Van Bel, M., Audenaert, P., Demeester, P., Dhoedt, B., Vandepoele, K., Fostier, J. (2015) BLSSpeller: exhaustive comparative discovery of conserved cis-regulatory elements. *Bioinformatics* 31(23), 3758-3766

[3] Herzeel, C., Costanza, P., **Decap, D.**, Fostier, J., Reumers, J. (2015) elPrep: high-performance preparation of sequence alignment/map files for variant calling *PLoS One* 10(7), e0132868

[4] **Decap, D.**, Reumers, J., Herzeel, C., Costanza, P., Fostier, J. (2017) Halvade-RNA: Parallel variant calling from transcriptomic data using MapReduce. *PLoS One* 12(3), e0174575

Conference Papers

[1] **Decap, D.**, Reumers, J., Herzeel, C., Costanza, P., Fostier, J. (2015) Performance Analysis of a Parallel, Multi-node Pipeline for DNA sequencing. *Parallel Processing and Applied Mathematics*

Conference Abstracts

[1] **Decap, D.**, Dhoedt, B., Fostier, J., Saeys, Y. (2012) Flexible motif discovery using feature selection trees: a high performance computing approach (poster). *Symposium on Personal High-Performance Computing*

[2] **Decap, D.**, Dhoedt, B., Fostier, J., Saeys, Y. (2012) Flexible motif discovery using feature selection trees: a high performance computing approach. *2nd Annual MRP symposium on Bioinformatics*

[3] **Decap, D.**, Reumers, J., Herzeel, C., Costanza, P., Fostier, J. (2014) Halvade: whole genome analysis with MapReduce. *Genome Informatics*

[4] **Decap, D.**, Reumers, J., Herzeel, C., Costanza, P., Fostier, J. (2015) Halvade: scalable sequence analysis with MapReduce. *Hadoop for Next-Gen Sequencing workshop*

[5] **Decap, D.**, Reumers, J., Herzeel, C., Costanza, P., Fostier, J. (2015) Halvade: scalable sequence analysis with MapReduce. *Big N2N Annual Symposium*

[6] Moerman, T., **Decap, D.**, Verbeiren, T., Fostier, J., Reumers, J., Aerts, J. (2015) Interactive VCF comparison using Spark Notebook. *Benelux Bioinformatics Conference*

[7] **Decap, D.**, Reumers, J., Herzeel, C., Costanza, P., Fostier, J. (2015) Halvade: scalable sequence analysis with MapReduce. *ISMB/ECCB*

[8] **Decap, D.**, Moerman, T., Reumers, J., Verbeiren, T., Herzeel, C., Costanza, P., Fostier, J. (2016) Scalable, end-to-end variant calling and analysis of genomic and transcriptomic data in the cloud. *Big N2N Annual Symposium*

Samenvatting
– Summary in Dutch –

Een genoom omvat het genetisch materiaal van een organisme. Het bestaat uit DNA (voluit: Desoxyribonucleïnezuur), een molecule die drager is van de genetische code voor alle bouwblokken van levende organismen. DNA kan worden voorgesteld als een sequentie van letters ACGT die corresponderen met de volgorde van de nucleotiden binnen een DNA molecule. Het volledige menselijk genoom bestaat uit ongeveer 3 miljard baseparen, georganiseerd in 23 chromosoomparen. Tussen de genomen van menselijke individuen zijn er ongeveer 0.1% of 3 miljoen verschillen in totaal. Dit is een stuk kleiner dan het aantal verschillen tussen mensen en hun dichtste levende verwanten, de bonobo's en de chimpansees, die ongeveer 4% verschillen qua genoom. De verschillen in DNA tussen individuen worden genetische varianten genoemd en kunnen geïdentificeerd worden als nucleotidevarianten, invoegingen, verwijderingen of structurele variaties. Deze genetische varianten maken ons uniek, maar kunnen tevens de onderliggende oorzaak zijn van ziektes. Het identificeren van varianten is dus belangrijk voor medisch onderzoek of in een klinische omgeving. Bijgevolg is de ontwikkeling van snelle en betrouwbare methodes voor de identificatie van deze varianten een belangrijke onderzoeksvraag.

Sinds het eerste menselijk genoom werd gesequentiëerd, een inspanning die 10 jaar duurde en 3 miljard USD kostte om te vervolledigen, is de techniek van het sequentiëren van DNA sterk geëvolueerd. De zogenaamde volgende-generatie sequentiëringsmachines worden gekenmerkt door een enorme doorvoersnelheid en lage kost. Zo kan bijvoorbeeld een modern Illuminasysteem $\sim$1.5 miljard reads genereren in slechts enkele dagen. Dit leidt tot een enorme hoeveelheid aan ruwe gesequentiëerde data die verwerkt moet worden. Moderne Illuminasystemen ondersteunen tevens het sequentiëren van RNA. RNA is, zoals DNA, een nucleïnezuursequentie. Het is essentiëel voor het coderen, decoderen en reguleren van de expressie van genen (een stuk van het nucleïnezuursequentie dat de moleculaire eenheid is van erfelijkheid). Bij het sequentiëren van RNA wordt eerst een complementaire DNA (cDNA) streng gesynthetiseerd, die dan vervolgens wordt gesequentieerd.

Ruwe data kunnen op verschillende manieren worden verwerkt. In dit werk leggen we de focus op het bepalen van genetische varianten. Dit houdt in dat de verschillen tussen een gekend referentiegenoom en het gesequentieerde staal worden opgespoord. Het Broad Instituut heeft een best practice computationele pijplijn beschreven voor het identificeren van varianten voor zowel DNA en RNA data. Deze computationele pijplijn bestaat uit het uitvoeren van enkele programma's op de data in een bepaalde volgorde met specifieke parameters. Eerst worden de reads gealigneerd ten opzichte van een gekend referentiegenoom. Voor elke read individueel wordt hierbij de meest waarschijnlijke locatie in het genoom gezocht vanwaar de read afkomstig is. Vervolgens worden de gealigneerde reads verder verwerkt ter voorbereiding van het identificeren van de varianten. Dit houdt in dat de kwaliteitsscores van de nucleotiden worden geherkalibreerd en dat reads in moeilijk genomische regio's (gebieden met veel herhalende patronen) worden geheraligneerd. Ten slotte worden de varianten geïdentificeerd en opgeslagen voor verdere verwerking of analyse. Deze computationele stappen zijn zeer rekenintensief: op één 24-core machine duurt het uitvoeren van deze computationele pijplijn ongeveer 5 dagen, dit is langer dan de tijd voor het sequentiëren zelf.

Om de uitvoeringstijd van de computationele pijplijn te minimaliseren hebben we Halvade ontwikkeld. Het is een programma dat de ruwe gesequentiëerde data verwerkt en de identificatie van varianten uitvoert volgens de best practice richtlijnen zoals gespecifieerd door het Broad Instituut. Gebruik makend van het MapReduce programmeermodel kunnen we de computationele pijplijn schalen naar een omgeving met meerdere computernodes en hierdoor de rekentijd reduceren. Dit is gebaseerd op de observatie dat het aligneren van de data parallel is per read: het aligneren van een bepaalde read is onafhankelijk van het aligneren van andere reads. Alle andere stappen zijn dan weer parallel per genomische regio: het identificeren van varianten in een genomische regio is onafhankelijk van het identificeren van varianten in andere genomische regio's. Tussen beide fases in kunnen we gebruik maken van het efficiënte sorteeralgoritme zoals geïmplementeerd in Hadoop. Halvade kan de tijdsduur voor het uitvoeren van de computationele pijplijn op één machine reduceren van 5 dagen naar 2 dagen, terwijl op een cluster met 15 machines de tijd verder gereduceerd wordt naar 2 uur en 39 minuten. Deze versnelling laat ons toe de stap tussen de generatie van ruwe data en het analyseren van de gevonden varianten te minimaliseren. Om de nauwkeurigheid van Halvade te verifiëren hebben we de resultaten van de sequentiële computationele pijplijn vergeleken met de resultaten van Halvade en we halen een overeenkomst van meer dan 99%. Om de RNA-seq computationele pijplijn uit te voeren, waarvoor tevens best practices richtlijnen geformuleerd werden door het Broad Instituut, moest de aligneringsstap opgesplitst worden in Halvade naar een tweestaps aligneringsalgoritme gebruikmakend van het programma STAR. Daarnaast dienden nog enkele bijkomende stappen toegevoegd worden aan de voorbereidingsstap van Halvade om de RNA-seq varianten te identificeren. Het uitvoeren van Halvade op RNA-seq data vertoont een lagere parallelle efficiëntie in vergelijking met de DNA data. Dit wordt veroorzaakt door het ongebalanceerde aantal reads over het genoom, veroorzaakt door de verschillende expressieniveaus van de verschillende genen.

Uit een analyse van de performantie van Halvade is gebleken dat het aantal parallelle taken een grote invloed heeft op de totale tijdsduur. Meer parallelle taken betekent dat meerdere onafhankelijke instanties van één programma opgestart kunnen worden, terwijl ze anders meerdere CPU-cores moeten gebruiken via multithreading. Dit leidde tot de conclusie dat enkele programma's een slechte performantie vertonen als multithreading gebruikt wordt. Daarnaast hebben we vastgesteld dat de verbindingssnelheid tussen de machines in een cluster, het type opslagmedium en het type gedistribueerd bestandsysteem een slechts een kleine invloed heeft op de totale uitvoeringstijd. Ten slotte heeft het optimaliseren van het aantal reducetaken, en daardoor het aantal genomische regio's dat Halvade moet verwerken, een grote invloed. Het gecombineerde effect van deze optimalisaties resulteerde in een totale uitvoeringstijd van 1 uur 21 minuten op een cluster met 15 machines wat overeen komt met een parallelle efficiëntie van 59%.

Tijdens het analyseren van de varianten, worden vaak twee verschillende stalen vergeleken. Zo kunnen bijvoorbeeld stalen uit gezond en kankerweefsel met mekaar worden vergeleken of worden mutaties binnen één staal geobserveerd op twee verschillende tijdstippen. Deze analyse wordt typisch uitgevoerd in een interactieve manier, waarbij telkens de data wordt gefilterd en vergeleken. Om dit type analyse te optimaliseren hebben we een interactieve Spark Notebook ontwikkeld waar de data gemakkelijk gefilterd en aangepast kan worden voor elke use case. De beslissing om dit te ontwikkelen in Spark Notebooks is gebaseerd op de herbruikbaarheid van notebooks waardoor ze gemakkelijk kunnen aangewend worden voor verschillende use cases. We steunen hierbij op het ADAM project dat een API implementeert om VCF en SAM geformatteerde data te verwerken binnen een Spark omgeving. De ontwikkelde notebook ondersteunt het vergelijken van twee VCF bestanden alsook het filteren op verschillende waarden uit het VCF bestandsformaat. Na het filteren worden de data opnieuw vergeleken en wordt de overlap en de verschillen grafisch weergegeven. Dit proces wordt uitgevoerd binnen een interactieve en quasi real-time omgeving.

Daarnaast werd een Spark Notebook ontwikkeld die drie verschillende VCF bestanden met mekaar kan vergelijken. Als voorbeeld worden er drie verschillende sequentiëringstechnieken met mekaar vergeleken: RNA-seq, whole exome sequencing (WES) en whole genome sequencing (WGS) data. Deze data werden verwerkt door Halvade en de geïdentificeerde varianten werden daarna geanalyseerd op concordantie en discordantie. Hierbij focussen we enkel op variatie op nucleotideniveau. In een eerste observatie merken we dat bijna alle varianten die door zowel RNA-seq als WES geïdentificeerd werden ook door WGS bevestigd worden. Slechts 0.5% van deze varianten werden niet door WGS bevestigd. De varianten die enkel door RNA-seq werden geïdentificeerd maar niet door WES, werden voor een groot deel bevestigd door WGS. Ze vertonen een laag aantal reads op de corresponderende plaatsen in WES wat hun afwezigheid in dit datatype verklaart. De varianten die niet door WGS bevestigd werden vertonen een duidelijke aanrijking van RNA editing varianten wat wordt bevestigd door een hoog aantal A-naar-I mutaties. Dit geeft aan dat evenzeer uit RNA-seq data de varianten op een betrouwbare manier kunnen worden geïdentificeerd. WES varianten die niet gevonden werden in RNA-seq data hadden geen of onvoldoende reads in de RNA-seq data. Ook de meeste varianten die enkel in WGS gevonden werden vertonen een zeer laag aantal reads op corresponderende locaties in zowel WES als RNA-seq data.

Met Halvade bieden we een zeer snelle en schaalbare oplossing voor het verwerken van ruwe gesequentiëerde data voor zowel WGS, WES als RNA-seq sequentiëringsstrategiën. Met de toevoeging van Spark Notebooks die de varianten kan vergelijken op interactieve wijze, verkrijgen we een volledige pijplijn van ruwe data tot het analyseren van deze varianten. Door het gebruik van Hadoop MapReduce en Spark is de volledig pijplijn beschikbaar in de cloud of op een lokale cluster.

Summary

A genome is the genetic material of an organism. It consists of Deoxyribonucleic acid (DNA), a molecule that carries the genetic code for all building blocks of living organisms. DNA can be represented as nucleic acid sequences, a succession of letters ACGT that indicate the order of nucleotides within a DNA molecule. The entire human genome is approximately 3 billion base pairs in size. In the cell, the genome is organized in 23 chromosome pairs. Among the genomes of human individuals, there are approximately 0.1% differences or roughly 3 million differences in total. This is considerably smaller than the number of differences between humans and their closest living relatives, the bonobos and chimpanzees, which have approximately 4% differences. The differences among individuals are called genetic variants and can be identified as single nucleotide variants, insertions, deletions or structural variations. This variation is what make us unique, but can also be the underlying cause of certain diseases. The identification of genetic variants is therefore of use in medicinal research or in a clinical setting and the development of reliable and fast methods to identify these mutations is thus important.

Since the first human genome was sequenced, an effort that took 10 years and 3 billion USD to complete, the sequencing of DNA has progressed tremendously. The next-generation sequencing machines have increased the sequencing speed considerably. As an example, modern Illumina systems can generate $\sim$1.5 billion high accuracy reads in a matter of days. This leads to an enormous amount of raw sequencing data that needs to be processed. Modern Illumina systems support sequencing of RNA molecules as well. RNA is, like DNA, a nucleic acid sequence. It is essential in coding, decoding, regulating and expression of genes (a part of the nucleic sequence that is the molecular unit of heredity). The sequencing is done by synthesizing a complementary DNA (cDNA) strand, complementary to the single stranded RNA, and then sequencing the DNA strands.

Raw sequencing data can be processed in several ways. In this work we focus on variant calling applications. In essence, the variant calling procedure involves the identification of differences between a known reference and the sequenced sample. The Broad Institute has formulated Best Practices recommendations for a computational pipeline for variant calling from both DNA and RNA sequencing data respectively. This pipeline involves running a number of specific tools on the data in a specific order with certain parameter settings. It involves aligning the reads to a known reference genome. For every read individually, the most likely genomic location from which the read was derived is sought using a non-exact string matching procedure. Next, aligned reads are prepared for variant calling. This includes recalibrating the base quality scores and performing a realignment in difficult regions (regions with many repeating patterns and/or insertions of deletions). Lastly, the variants are called and stored for further processing and analysis. These computational steps are very compute-intensive: on a single 24-core node the processing time of this pipeline is approximately 5 days, which is more than the time needed for the sequencing step itself.

To minimize the runtime of this computational pipeline, we have developed Halvade. It is a tool that processes raw sequencing data and performs variant calling according to the Best Practices recommendations as formulated by the Broad Institute. The use of the MapReduce programming model allows us to scale the pipeline to a multi-node setup and in turn reduce runtime. This is based on the observation that the read alignment step is parallel by read, i.e., the alignment of one read is independent of the alignment of other reads. The other steps are all parallel by genomic region, i.e., variant calling in one region is independent of variant calling in another region. In between both phases, we can rely on the efficient sorting functionality of the Hadoop framework. Halvade decreases the runtime of the processing pipeline on a single node from 5 days to 2 days, and on a 15-node cluster the runtime is further decreased to 2 hours 39 minutes. This speedup allows for a minimization of time between the production of the raw data and downstream variant analysis steps. To confirm the accuracy of Halvade, we compared the output of Halvade with the output of the sequential pipeline and found a concordance of over 99%. In order to also support the RNA-seq variant calling pipeline, similarly described by the Broad Institute, the mapping step of Halvade had to be adjusted to a 2-step read alignment phase using the STAR alignment tool. Moreover, some additional steps in the RNA-seq variant calling pipeline were added to Halvade. Processing RNA-seq data using Halvade shows a lower parallel efficiency compared to the DNA data. This is because of the unbalanced coverage that in turn is caused by different gene expression levels from the data samples.

When further analyzing the performance of Halvade, we discovered that the number of parallel tasks has a big influence on the runtime. More parallel tasks means that more parallel instances of a certain tool will be started instead of using the available CPU-cores for multithreading. This is associated to the fact that certain individual tools show limited multithreading scaling behavior. Additionally, we note that the interconnection network of the cluster, the type of storage and the type of distributed file system only have a small influence on the total runtime. Lastly, optimizing the number of the reduce tasks and consequently the number of genomic regions Halvade will process in parallel, has a big influence on performance. With all collective optimizations in place we were able to further reduce the total runtime to only 1 hour and 21 minutes on a 15-node cluster, corresponding to a parallel efficiency of 59%.

When analyzing variant data, two samples often need to be compared, for example, cancer versus a matching non-cancer sample or the mutations in a sample observed in two distinct points in time. Typically, this analysis is performed in an iterative way, where the data is filtered and compared in each step. To improve this kind of analysis we have developed an interactive Spark Notebook where the data can easily be filtered and modified for different use cases. The decision to develop this in a Spark Notebook is based on the reproducibility of the notebooks which easily allows processing different samples in a scalable manner. By using Spark we can rely on the ADAM project which provides an API for processing VCF and SAM formatted data. In the notebook we compare two VCF files and support filtering on several fields from the VCF format. After filtering, the VCF files are again compared and overlap is visualized. This process is executed in an interactive and quasi real-time environment.

Additionally, we have developed a Spark Notebook to compare three sequencing strategies on the same sample: RNA-seq, whole exome sequencing (WES) and whole genome sequencing (WGS). Raw data is processed by Halvade and then compared using this notebook. This analysis focused on unfiltered single nucleotide variants. A first observation is that almost all variants that are called from both RNA and WES are also confirmed in WGS. Only 0.5% of these variants are not confirmed by WGS. For the variants only called from RNA-seq data but not from WES, a great deal are confirmed by WGS. They show low only coverage in the WES data which explains why they were not identified. The RNA-seq variants that are also not confirmed by WGS show a clear presence of RNA editing indicated by a very high A-to-I mutation count. This indicates that RNA-seq data can also be a reliable source for variant calling. As for the WES variants not called in RNA-seq, we see that the coverage in the RNA sample is severely lacking and thus does not provide enough information for variant calling. Most of the mutations that are called only in WGS show a very low coverage in both the WES and the RNA-seq data set.

With Halvade we provide a very fast solution to process raw sequenced reads for WES, WGS and RNA sequencing strategies. With the addition of the Spark Notebook that compares the variants achieved from Halvade, we essentially provide a pipeline from raw sequencing reads to analysis ready variants. The use of Hadoop MapReduce and Spark makes the entire pipeline available for use in the cloud as well as on a local cluster.

1 Introduction

In this introduction we will first explain the process of DNA and RNA sequencing with Illumina platforms as these are currently most commonly used for sequencing. Next we explain how raw sequencing data is typically processed to identify genomic variants in a sample with respect to a reference genome. This process involves several computational steps and is described as a sequencing analysis pipeline. Best Practices recommendations for this pipeline have been described by the Broad Institute and will be explained in detail: what are the steps involved and what effect do they have on the variant calling procedure? The parallelization of this pipeline is the main topic of this thesis, and the used programming models and concepts will be introduced. Next, we describe the process of analyzing the variants that have been identified with the pipeline. Finally, we provide an overview of the different papers.

⋆⋆⋆

1.1 DNA sequencing

Deoxyribonucleic acid, or as it is more commonly referred to, DNA, is a molecule that carries the genetic instructions used in the growth, development, functioning and reproduction of all known living organisms. DNA, and RNA, are nucleic acid sequences and are one of the four major types of macromolecules that are essential to all known forms of life. The other three are proteins, lipids and complex carbohydrates. DNA molecules consist of two complementary strands coiled around each other to form a double helix. The two DNA strands are composed of monomer units called nucleotides and are therefore called polynucleotides. Each nucleotide is composed of a sugar called deoxyribose, a phosphate group and one of four nitrogen-containing nucleobases: cytosine (C), guanine (G), adenine (A) or thymine (T). These nucleotides are connected to one another by bonds between the sugar and phosphate of two adjacent nucleotides. The two DNA strands are bound together (A is paired with T while C is paired with G) with hydrogen bonds to form the double-stranded DNA. Biological information is stored in the DNA and both strands contain the same biological information which makes it resistant to cleavage.

The DNA strands form chromosomes, and the human genome contains 23 pairs of chromosomes, 22 pairs of autosomes and one pair of sex chromosomes, two X chromosomes in females, while individuals with both X and Y chromosomes are males. The genotype of an individual or organism consists of its genetic makeup, the genome, and it determines the characteristics of that organism. Only a small portion (2%) of the DNA is actually used for coding protein sequences, these are the genes and they contain the code for making proteins. RNA strands are formed by using the DNA as a template in a process called transcription. These RNA strands are in turn used as a template to specify the amino acids within proteins in a process that is called translation. The genes are made up of exons, which encodes part of the final mature RNA (created by transcription from DNA), and introns which are removed by RNA splicing before translation to proteins. A variant in or around a given gene can result in changes in either the protein makeup or regulation for expression. This variant of a given gene is called an allele, which in turn can produce different phenotypes. Phenotypes are observable characteristics of the organism, e.g. the color of the eyes. If both alleles at a gene on the homologous chromosomes are the same, the organism is homozygous with respect to that gene. Different alleles for that gene make the organism heterozygous with respect to that gene.

The human genome is about 3 billion base pairs in size. Between the human genome and that of its closest relative, the bonobos and chimpanzees, there are about 4% differences. However, also between individual humans there are differences, albeit substantially less: only about 0.1% of the human genome differs between individuals, amounting to about 3 million differences in total. These differences are called genetic variants and can be identified as single nucleotide variants, insertions, deletions or structural variations. An organism's DNA affects how it looks, how it behaves and also its physiology. Consequently, a mutation in the DNA can cause changes in looks, behavior and physiology. Sometimes they can even be the cause for diseases. Finding these variants and linking them to diseases is more and more used in medicinal research and clinical settings. Therefore, it is important to have methods to identify and analyze these mutations in a reliable and timely manner.

The first human genome has been sequenced in the Human Genome Project, a 3 billion USD undertaking that took nearly 10 years to complete. Since then, the sequencing technologies have improved a lot and nowadays human genomes can be sequenced much cheaper and at a faster rate. These new sequencing technologies are typically referred to as next-generation sequencing (NGS) platforms [1]. The output of these sequencing machines are called reads, i.e., short parts of DNA sequences with a typical length of about 50 to 250 nucleotides. For every base pair that is sequenced, a corresponding quality score is provided by the sequencing platform. These per-base qualities are encoded in an ASCII string where the ASCII value represents the Phred-scaled quality score, which is equal to $-10\log_{10} P\{base\ is\ wrong\}$. These sequencing machines are typically capable of sequencing both ends of a genomic fragment resulting in paired-end reads. The use of paired-end reads improves the accuracy of alignment and is especially beneficial in or near genomic rearrangements and repetitive sequences as well as gene fusions and novel transcripts. The Illumina sequencers (e.g. Illumina HiSeq) account for over 90% of all sequenced data today and all Illumina NGS machines are capable of sequencing paired-end reads.

These reads or paired-end reads can be used either for de novo genome assembly projects or for resequencing purposes. The former involves the reconstruction of the genomic sequence without prior information. In the latter case, reads are aligned to a known reference genome after which variation with respect to this reference genome can be identified. The alignment step relies on an inexact string matching procedure where it is assumed that there are only few errors from the sequencing machine and few variants between the reference and the sequenced genome. In this dissertation we focus on the computational aspect underlying the alignment and variant calling steps, both for DNA or RNA sequencing data. Additionally, we investigate scalable solutions for the downstream analysis of variants.

1.1.1 Illumina sequencing

Illumina sequencers use a technique called Illumina dye sequencing, enabling them to determine a series of base pairs, thus forming a read. This technique was developed by Shankar Balasubramanian and David Klenerman of Cambridge University. The method is based on reversible dye-terminators that enable the identification of single bases when they are introduced into DNA strands. It can be used for whole-genome and targeted sequencing, as well as for transcriptome analysis, small RNA discovery and more.

The sequencing technique consist of three steps. First is amplification, followed by sequencing and lastly analysis. An overview of the sequencing process is shown in Fig. 1.1. The first step starts with chopping the isolated DNA molecule into random smaller pieces. Terminal sequences or adapters are added to both ends of all the cut pieces as well as indices and other modifications. The adapter will be used in the following steps of amplification and sequencing. The indices are used during the sequencing to identify the samples. Over a 100 samples can be run in parallel. During the process the sequences with the same indices will be grouped together. Pieces where the modifications were unsuccessful are washed away. The remaining pieces are loaded onto a specialized chip (an acrylamide-coated glass flow cell), the terminal sequences will match with and attach to oligonucleotides that are present on the bottom of the flow cell. The amplification and sequencing occurs on this flow cell.

Once the DNA molecules are attached to the flow cell, identical copies will be

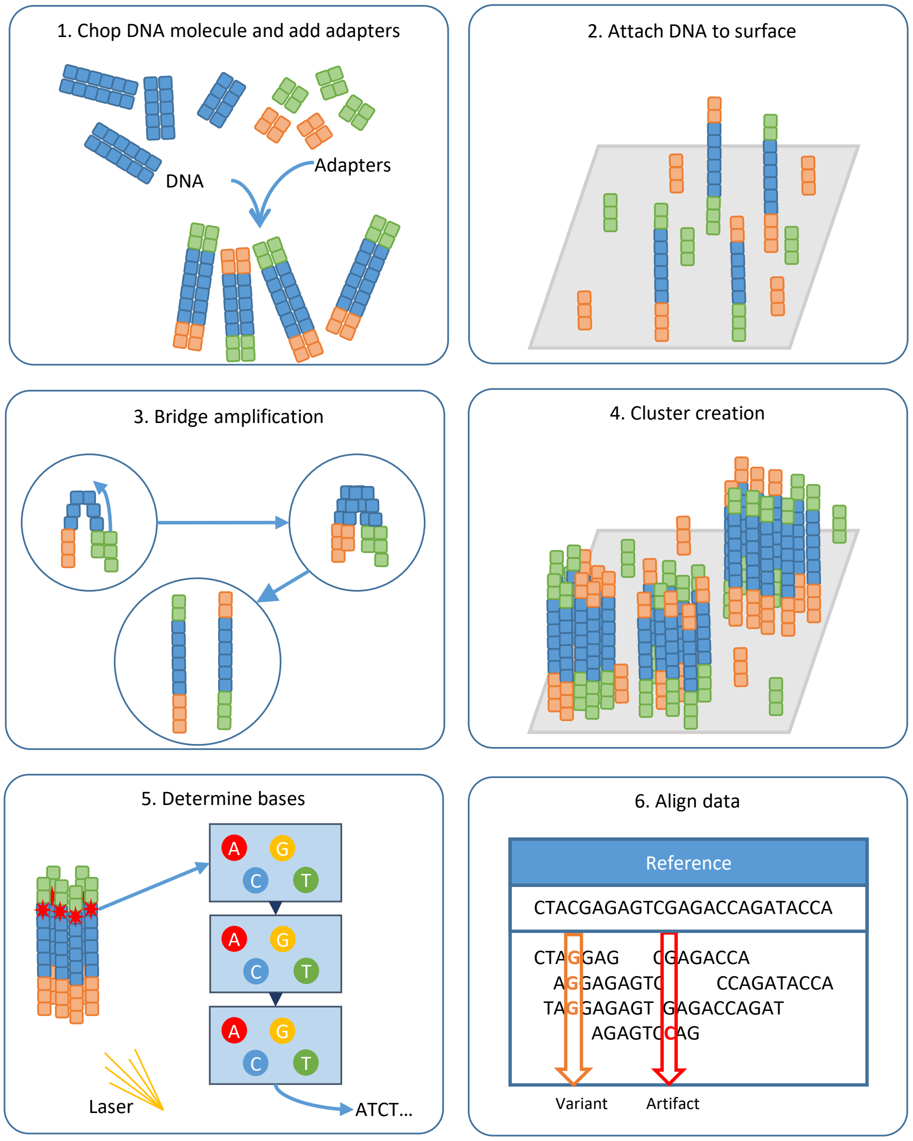


Figure 1.1: Illumina sequencing starts by adding adapters to the chopped DNA molecules. These molecules are attached to a surface with the adapters. Next a bridge is formed and a reverse strand is created through polymerase. Both strands are denatured and further used to amplify in clusters with identical strands. Every cluster is now sequenced in parallel, base per base the sequence is constructed by determining the light, and corresponding base, that is emitted. During analysis overlapping areas are used to create contigs.

created to form clusters. The goal is to amplify a single molecule so that you can get a stronger signal and get better readings of the nucleotides. Several hundred or a few thousand of identical molecules in a cluster are required. The strands are attached to the flow cell with an adapter. Now the strand bends over and the adapter at the other end is also attached to a corresponding oligo on the flow cell, forming a bridge. Polymerases attach to the strand and the complementary strand is made, the reverse of the original strand. Now the strands are straightened by denaturing the strands, while remaining attached to the oligo at the bottom of the flow cell. This is called bridge amplification and occurs in parallel for thousands of clusters on the flow cell. By performing this process over and over again, a cluster of identical (forward and reverse) strands of the original DNA molecule is created. This amplification is important for the quality of the sequencing that will follow. By using identical strands, the same nucleotides are expected at every position in the strand. If an abnormal sequence is found, the other strands in the cluster can be used as verification.

A cluster is sequenced as a whole, but the reverse strands are washed off. This leaves the cluster with all identical strands, this time only forward strands. The sequencing primers are attached to every strand in the cluster and fluorescent nucleotides are added. A single nucleotide is added in every round of base identification. This also means that the length of a sequence can be measured by the number of rounds the sequencing goes through. The four bases have a unique emission when a light source is added, and every round this is recorded to identify one of the four bases. Once the DNA strand is read, this added strand is washed away. Now the other end of the strand will be read in order to create paired-end reads. This is done by again making a bridge to one of an oligo on the flow cell. This time the forward strand is washed away, leaving a cluster of reverse strands. And the sequencing process is repeated on this end of the DNA sequence. The sequencing is performed on millions of clusters in parallel, where each cluster has several hundreds of identical copies of the DNA molecule from the PCR bridge amplification.

In the analysis this sequencing data is used to create longer fragments of DNA, called contigs. These contigs are created by comparing fragments and finding overlapping areas between them and thus extending the contig. Without a reference this is called de novo assembly, where the created contigs will form a new reference. With a known reference, these contigs can be used to find variation between the sample and the known reference. The latter will be the focus of this manuscript.

1.2 The DNA-seq variant calling pipeline

The life science application we focus on in this thesis is variant calling and analysis. Variant calling determines the genetic differences between the sequenced sample and a reference sequence. There are four types of differences, single nucleotide variants, insertions, deletions (typically insertions and deletions are grouped together and called indels) and structural variations. The Broad Institute has proposed pipelines for high-

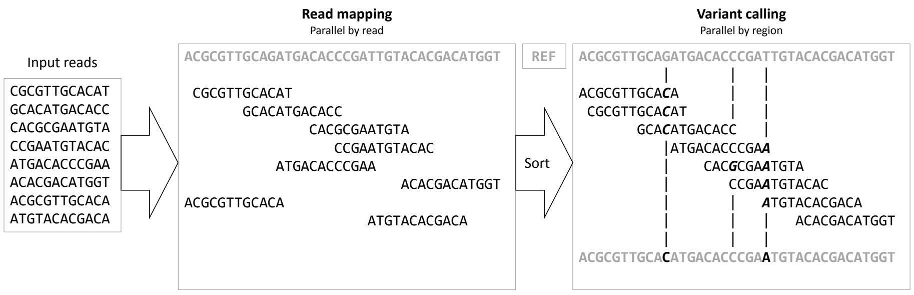


Figure 1.2: Simple example where the reads are first aligned to the reference. These are then sorted according to the genomic position to which they align. Finally, the differences with the reference and the reads are detected and used to call variants.

quality variant calling for both DNA and RNA sequencing data. We will first describe this pipeline for DNA data after which we will clarify the differences and additions that are required to run the same pipeline on transcriptomic data. The raw sequencing data produced by sequencing machines is typically stored as FASTQ files. These files contain the ASCII strings representing the sequenced read and the associated quality string, as well as an identity string for every read to match paired-end reads.

The process involves two big steps. First, the reads are aligned to the reference sequence, this is done using a tool called BWA (Burrows-Wheeler Aligner), which takes the FASTQ file as input and outputs the aligned reads in SAM format. After the alignment, the reads are sorted by genomic location to which they align and then the second big step starts, which includes data preparation and variant calling. The data preparation steps include duplicate marking, i.e., the identification of duplicated reads that thus contain no additional information and the addition of read group information to the SAM records. This preparation step formats the data according to very strict standard imposed by GATK, which will perform the final computational steps. In order to improve the variant calling quality the reads near putative insertions or deletions (indels) are realigned and the quality score of each base is recalibrated based on a list of known variants. GATK is then used to calls the variants using one of two supported modules. A very simple overview of read alignment and variant calling (without the data preprocessing steps in between) is shown in Fig. 1.2. Although sequencing errors are relatively rare, they need to be taken into account to avoid false positive variants from being called.

Calling variants from a typical whole genome sequencing (WGS) sample (50x coverage) using the Best Practices pipeline from the Broad Institute is a very time-consuming endeavor, amounting to 12 days on a single CPU core of a 24-core machine (dual socket Intel Xeon E5-2695 v2 @ 2.40 GHz) or 5 days when using all 24 cores of that machine. This pipeline is even more time consuming than the sequencing itself, which typically takes about two days on a HiSeq Illumina machine. Some clinical applications require the analysis to be performed much faster. As an example, preimplantation genetic diagnosis (PGD) for in vitro fertilization (IVF) allows 48 hours in between taking a cell to sequence and the decision which of the embryos to implant. These 48 hours include the sample preparation, sequencing and the analysis. In this situation, 5 days clearly would not work and faster solutions need to be available. Another important example is the latest Illumina X10 sequencing machine. With this machine up to 18 000 samples can be sequenced in a year with a price under 1000 USD per sample. This price includes the cost of paying for the machine, sample preparation and everything else it requires. This allows relatively cheap analysis of a lot of samples, approximately 50 samples will be ready for analysis every single day. The current pipeline runs 5 days on a single core. This means that 250 machines would be needed to handle the maximum influx of data. This means that a big cluster needs to be purchased, similar in price to that of the full price of the sequencing machine itself. If the runtime of a sample can be reduced on a single core, the cost of the accompanying cluster to analyze the data can be greatly reduced. Given the limited speedup provided by multithreading [2] and the inability of the pipeline to use multiple nodes, the development of a parallel multi-node framework is warranted.

1.2.1 Read Alignment

The first step in the post-sequencing analysis pipeline is the alignment of the reads to a known reference genome. Currently, over 60 short-read mappers are available [3], where most were released after 2008 to cope with advances in high-throughput sequencing technologies. In these mapping tools, the alignment can be stated as finding all substrings m of a set of reference sequences R for a set of query sequences Q that respect certain constraints (e.g. the number of mismatches/gaps allowed) and a distance threshold k. The constraints vary depending on the type of data. Additionally the mapping tool needs to take into account errors from the sequencing platform and structural variation to try and find the true location of the fragment. The distance measure is generally used to allow for a number of variants and/or sequencing errors. Additionally, read mappers take into account paired-end read information to increase the specificity of the mapping process as this provides additional positional information.

Most of these tools are based on a hashing algorithm or on the Burrows-Wheeler Transform (BWT), a space-efficient index that accelerates the string matching procedure [4]. The tools based on the latter are faster, more memory-efficient and can cope better with repetitive reads, but these are typically less sensitive. Hashing algorithm based tools like Novoalign [5] and Stampy [6] are more accurate but have a longer runtime. Typically, the workflow is divided into a preprocessing step and a mapping step. The preprocessing step is done only once and involves indexing the reference which can take several minutes to complete. After this the mapping using the Burrows-Wheeler Alignment (BWA) tool [7], which relies on the BWT, runs respectively 3 and 12 times faster than Novoalign and Stampy. The alignment itself has a big influence on the variant calling in a later step. Misaligned reads can be the cause for detecting false positives during variant calling. Given the large number of data generated from high-throughput sequencing machines, many tools provide a way to align reads in parallel. Typically this is done for a shared-memory system where multithreading is used.

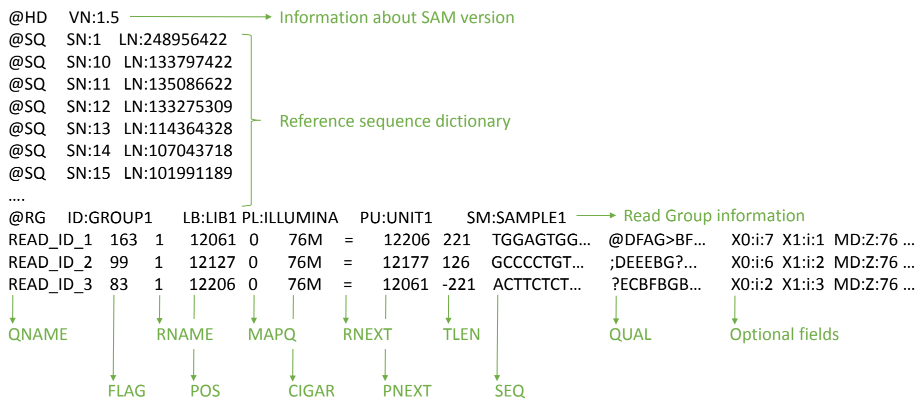
1.2.1.1 DNA sequencing read alignment with BWA

The DNA-seq variant calling pipeline proposed by Broad Institute uses the BWA tool for alignment. BWA or Burrows-Wheeler Alignment tool is an efficient short-read alignment package that is based on backward search with Burrows-Wheeler Transform (BWT). The backward search procedure effectively mimics the top-down traversal on the prefix trie of the genome with a relatively small memory footprint. This allows BWA to count the number of exact hits of a string of length m in $o(m)$ time, independent of the size of the genome. However, for inexact matching BWA samples distinct substrings from the implicit prefix trie with an edit distance less than k with respect to the query read. Additionally, because exact repeats are collapsed on a single path on the prefix trie, BWA does not need to align the query read against every copy of the repeat. With this BWA is able to achieve very high efficiency. To reduce the memory overhead of the prefix trie, they can be made sparse, where only a fraction of the trie is saved in memory and the rest is calculated on the fly.

To align reads with BWA, two options are provided. The first is to independently align both ends of paired-end reads separately (with the *bwa aln* command) and then to later combine these alignments to find the best alignment position (with the *bwa sampe* command). The second method is to use *bwa mem* [8] which chooses between local and end-to-end alignments and is developed with longer paired-end reads in mind. The algorithm is robust to sequencing errors and works on reads from 70 bp to a few megabases. The typical read length of 100 bp per read in our tests is applicable to both options. However, the performance tests in this thesis were conducted with *bwa aln* and *bwa sampe* as those were the standard methods during the time of benchmarking. The Broad Institute has since updated their pipeline to use *bwa mem* instead because reads are typically produced with lengths longer than 70 bp. Both alignment options use the BWA reference, which is a combination of the BWT and a sparse suffix array (SA) of the genome sequence. Together they require approximately 5 GBytes of memory.

1.2.1.2 The SAM Format

The output from both alignment tools is formatted using the SAM format. A simple example of this format is illustrated in Fig. 1.3. This format contains a header, which provides information about the SAM format version, the reference sequence dictionary, read group information and programs that produced or modified the SAM file. After the header each line contains the alignment information for one read. The information is tab separated and has 11 mandatory fields with optional fields at the end. The



```
@HD    VN:1.5
@SQ    SN:1    LN:248956422
@SQ    SN:10   LN:133797422
@SQ    SN:11   LN:135086622
@SQ    SN:12   LN:133275309
@SQ    SN:13   LN:114364328
@SQ    SN:14   LN:107043718
@SQ    SN:15   LN:101991189
....
@RG    ID:GROUP1     LB:LIB1 PL:ILLUMINA    PU:UNIT1     SM:SAMPLE1
READ_ID_1  163   1   12061  0   76M   =   12206  221   TGGAGTGG...  @DFAG>BF...   X0:i:7  X1:i:1  MD:Z:76 ...
READ_ID_2  99    1   12127  0   76M   =   12177  126   GCCCCTGT...  ;DEEEBG?...   X0:i:6  X1:i:2  MD:Z:76 ...
READ_ID_3  83    1   12206  0   76M   =   12061  -221  ACTTCTCT...  ?ECBFBGB...   X0:i:2  X1:i:3  MD:Z:76 ...
```

Figure 1.3: Simple SAM format example. The header contains the SAM version and the reference sequence dictionary. The read group information is also added to the header. Three aligned reads are added as an example of the columnar SAM format.

first column contains the query name of the read, the second is a bit-wise flag containing additional information. This flag contains information about paired-end reads and whether this alignment is a secondary alignment or not, whether it is flagged as a PCR or optical duplicate. The third and fourth column respectively show the reference sequence name and the 1-based leftmost mapping position of the read. The fifth column is the mapping quality which is equal to $-10\log_{10} P\left\{mapping\ position\ is\ wrong\right\}$, the sixth column provides the CIGAR string. This CIGAR string contains the exact positions of indels or mismatches in the read. The next two columns contain the reference sequence name and 1-based leftmost mapping position of the mate read in paired-end alignment. The ninth column shows the signed observed template length, equaling the total length on the reference sequence. The last two columns contain the actual sequence fragment with the corresponding quality string. The quality string contains the Phred-scaled score for every base pair in the sequence fragment as encoded as ASCII characters. The base quality is the Phred-scaled base error probability which equals $-10\log_{10} P\left\{base\ is\ wrong\right\}$ and is encoded as the ASCII character of base quality plus 33. After these 11 mandatory fields, optional columns can be added by the alignment tool. This can include other alignment metrics or information about the read group. Based on the SAM format, which is human readable, a BAM format is available as well. This BAM format is a binary representation of the SAM format and is designed to compress reasonably well.

1.2.2 Marking PCR Duplicates

Some DNA samples only have a few DNA molecules that can be used in the sequencing machine. To get an overall better result of the sequencing we can amplify the sample and get multiple clones. This is called Polymerase Chain Reaction or PCR amplification. In this step multiple copies of each original genomic DNA molecule are created intentionally. This way, enough of them are available in order to sequence more accurately. The duplication occurs when two copies of the same original molecule get onto different beads of different primer lawns in a flow cell. And because of the random hybridization of the DNA molecules to beads, some PCR duplication is inevitable. The ratio of duplicate DNA molecules depends on the amount of starting material available. This can range from as low as 4% or up to 70% in some cases. Another cause of a high ratio is if the variance in fragment size is too big and the smaller fragments end up being over-represented (as they are easier to PCR amplify) [9].

The PCR duplicates are identified and marked as such in order to take this information into account when computing statistics. The duplicates cannot be identified solely by alignment position as two distinct cDNA (a complementary DNA synthesised from a single stranded RNA) fragments can produce reads that align to the same position. The algorithm essentially finds the 5' coordinates and mapping orientations of each read pair. Determining the 5' coordinate takes into account all clipping, indels, gaps etc. in the aligned read. Next all reads with identical 5' coordinates and orientations are marked as duplicates except the "best" pair, where the "best" pair is the read pair with the highest sum of base qualities. This process is typically done with the MarkDuplicates tool from the Picard [10] suite or rmdup from Samtools [11]. In the proposed pipeline Picard is used.

1.2.3 Indel Realignment

Indels can cause incorrectly aligned reads, especially at the end of the reads. In turn, these reads can cause the calling of false positive variants. Indel realignment is a step where alignments are corrected near indels. This process is done in two steps. First a list of regions where indel realignment is required is generated. Next, the reads are realigned around the indels in the list of regions. The targets for realignment are detected from three different origins. First, the known sites from databases, next, where the original alignment shows an indel (in the CIGAR string) and lastly, where evidence can be found that suggests a hidden indel. After the target locations are determined all reads that overlap these regions are realigned.

To realign, the algorithm groups the reads in piles. These piles show reads that have similar mismatches to the references. A score is given to every pile of reads, representing the total sum of quality scores for all mismatching bases. If the score is sufficiently better after realignment, then the proposed realignment is accepted. A simple example is provided in Fig. 1.4 [12]. This step effectively eliminates a number of false positive variants that would otherwise occur and is performed by GATK using the RealignerTargetCreater and IndelRealigner modules for the respective steps.

1.2.4 Base Quality Score Recalibration

Data generated from sequencing machines include a quality score per base pair. This quality score is a Phred score and equals $-10\log_{10} P\left\{base\ is\ wrong\right\}$. This means a

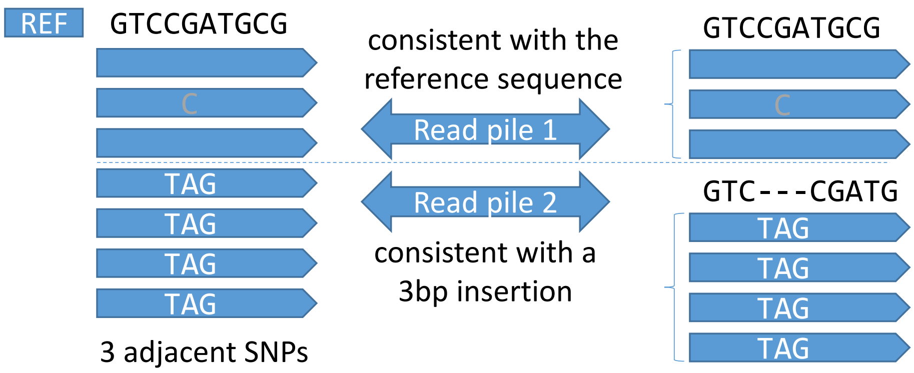
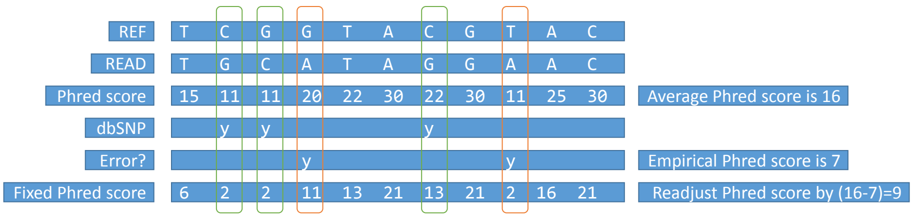


Figure 1.4: Example of realigning reads to adjust for indels. Reads are grouped in piles with similar variations. If a pile has consistent insertions, then these regions will be realigned taking into account this new insertion.



REF	T	C	G	G	T	A	C	G	T	A	C	
READ	T	G	C	A	T	A	G	G	A	A	C	
Phred score	15	11	11	20	22	30	22	30	11	25	30	Average Phred score is 16
dbSNP		y	y				y					
Error?				y					y			Empirical Phred score is 7
Fixed Phred score	6	2	2	11	13	21	13	21	2	16	21	Readjust Phred score by (16-7)=9

Figure 1.5: Trivial example of how BQSR works. The average Phred score is calculated from the given Phred scores. The variants that are not in the dbSNP file are considered to be errors (orange square). With this number a new empirical Phred score is calculated, after which the difference with the average Phred score is added to the Phred scores of each base.

score of 10 represents an accuracy of 90%, 20 represents 99% accuracy, 30 represents 99.9% and so on. Base Quality Score Recalibration (BQSR) is a method to adjust the Phred quality scores to be more accurate by looking at all the bases in the file instead of looking at every base separately. To run BQSR a file is needed with all known Single-Nucleotide Polymorphisms (SNPs). The National Institutes of Health (NIH) provides a database of single SNPs called dbSNP, which is used in this pipeline. This file will be used to identify "real" errors, in other words, SNPs that are not present in this file will be marked as real sequencing errors. This is a statistically sound assumption, given that there are over 12 million SNPs in this database and that a single individual will have, on average, only about a 1000 SNPs not found in this file.

For a single read, the average Phred score is calculated by converting the scores back to probabilities, taking the average of the per-nucleotide probabilities and calculating the Phred score of this average probability. Next, per read the "real" errors are counted and used to calculate an empirical Phred score, equal to $-10\log_{10} P\left\{errors \;/\; readlength\right\}$. Now the difference between the empirical Phred score and the average Phred score is used to correct the individual Phred scores so that the new average will equal the empirical Phred score. This process is illustrated in Fig. 1.5. However, BQSR is performed on a collection of reads. This collection of reads is first binned by read group since different sequencing machines may be calibrated differently. These read group bins are again binned by the quality scores since the goal is to compare the scores reported by the sequencing machine to the empirical scores we derive from the empirical error counts. For every read group quality bin a new empirical score is derived by using the error count in that bin. This empirical score is used as new score for bases with a quality equal to the quality of this bin [13, 14].

BQSR requires two passes over the data, the first pass splits the data up as described above and calculates new empirical scores for every quality present per read group. The second pass uses this data to adjust the quality scores of every base in the data file. The first pass is done by the BaseRecalibrator module from the GATK which produces a table for the recalibration. The second pass is performed by the PrintReads module from the GATK which uses this table to recalibrate all the base scores in the input file.

1.2.5 Variant Calling using GATK

Converting base calls and quality scores from the sequencing machine into a set of genotypes for each individual in a sample is often divided into two steps: calling single nucleotide polymorphisms (or SNPs) and genotype calling [4]. SNP calling or variant calling detects the sites where at least one base differs from the reference sequence whereas genotype calling determines the genotype for every individual, typically for each location where a variant has been detected. Basic variant calling simply counts alleles at each site and uses cutoff rules to determine SNPs and genotypes. Typically, only high-confidence bases are kept, bases with a Phred score below a certain threshold are not used. This works well for regions with high coverage (more than 20 X). However, newer methods tend to include uncertainty in a probabilistic framework, allowing additional information like allele frequency patterns to be used. This yields better results for regions with lower sequencing coverage. This method also provides a measure of uncertainty.

The probability $P(G \mid X)$ for a genotype G given the aligned read information X can be expressed using Bayes' formula as 1.1, where $P(G)$ is the prior, $P(X)$ is the evidence and $P(X \mid G)$ is the probability of observing the data X given the genotype G.

$$\begin{aligned} P(G \mid X) &= \frac{P(X \mid G)P(G)}{P(X)} \\ &= \frac{P(X \mid G)P(G)}{\sum P(X \mid G_i)P(G_i)} \end{aligned} \tag{1.1}$$

This gives rise to the implicit assumption of independence among reads. This is where marking PCR duplicates, indel realignment and BQSR gives additional accuracy as

PCR artifacts and alignment errors can violate this independence assumption. The prior $P(G)$ can be chosen arbitrarily to assign equal probability to all genotypes or based on external reference information like a SNP database (dbSNP). Another possibility is to use the observed genotypes in the actual data (i.e. empirical Bayes). The genotype with the highest probability P is called. GATK provides the UnifiedGenotyper which can call genotypes in DNA sequencing data and the HaplotypeCaller which supports calling genotypes in both DNA and RNA sequencing data.

1.2.5.1 UnifiedGenotyper

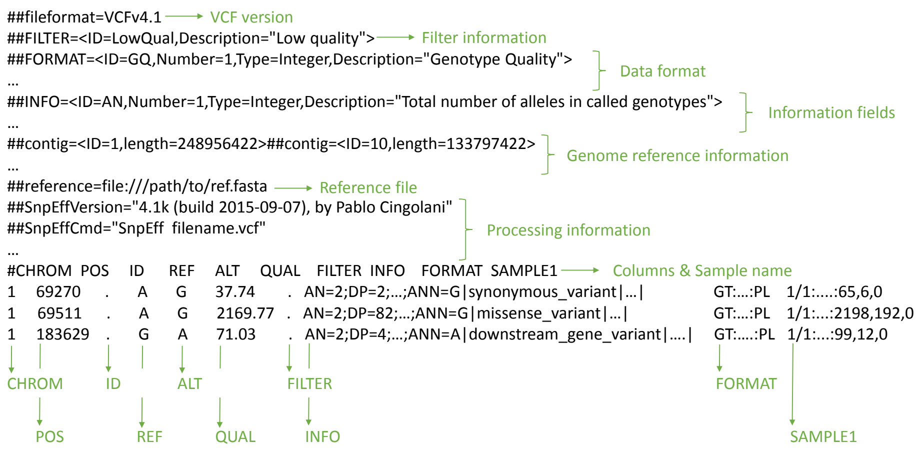
The UnifiedGenotyper is the first variant and genotype calling utility from GATK. It is based on the probabilistic framework to call variants. This algorithm calculates the likelihood $L(G \mid X)$ that a genotype G is present given all the sequencing data for a particular individual at a particular site X [15]. GATK ignores the denominator in the Bayesian model as they are trying to calculate the most likely genotype, and this part is the same for all genotypes.

$$L(G \mid X) = P(G)P(X \mid G) = \prod_{b \in \{good\ bases\}} P(b \mid G) \tag{1.2}$$

Now the likelihood can be calculated as 1.2 where $P(G)$ is the prior and $P(X \mid G)$ is the probability of observing the data X given the genotype G. The prior $P(G)$ is set to 1 for single sample and then applied for multi-sample calculations, so we get a product of the probabilities of the independent bases b given G. The bases are filtered to only have reliable bases left based on base quality, read mapping quality, pair mapping quality, etc. The likelihood $L(b \mid G)$ is calculated using a platform-specific confusion matrix. This error matrix contains information of the performance of a sequencing system. The UnifiedGenotyper tool is very aggressive in calling variants in order to be more sensitive. The side effect of this is that the output can contain many false positives which can be filtered out in a later step. Additionally, it is not recommended to use this tool for RNA-seq data. Tests show that it calls less than 50% of the true positive indels for RNA-seq data that the HaplotypeCaller does call. Some functionality was added to this last tool that intelligently takes into account exon-intron split regions to achieve higher accuracy.

1.2.5.2 HaplotypeCaller

This module uses local de novo assembly of haplotypes, which can process splice junctions in RNA properly, to call SNPs and indels simultaneously [16]. A haplotype is a set of DNA variations that tend to be inherited together, which can refer to a combination of alleles or a set of SNPs. The algorithm used to calculate the variant likelihood is not well-suited to extreme allele frequencies (relative to ploidy) as is the case with somatic (cancer) variant discovery. GATK provides the MuTect2 module



```
##fileformat=VCFv4.1
##FILTER=<ID=LowQual,Description="Low quality">
##FORMAT=<ID=GQ,Number=1,Type=Integer,Description="Genotype Quality">
...
##INFO=<ID=AN,Number=1,Type=Integer,Description="Total number of alleles in called genotypes">
...
##contig=<ID=1,length=248956422>##contig=<ID=10,length=133797422>
...
##reference=file:///path/to/ref.fasta
##SnpEffVersion="4.1k (build 2015-09-07), by Pablo Cingolani"
##SnpEffCmd="SnpEff  filename.vcf"
...
#CHROM  POS    ID   REF  ALT   QUAL    FILTER  INFO  FORMAT  SAMPLE1
1   69270    .    A    G    37.74     .  AN=2;DP=2;...;ANN=G|synonymous_variant|...|           GT:...:PL   1/1:....:65,6,0
1   69511    .    A    G    2169.77   .  AN=2;DP=82;...;ANN=G|missense_variant|...|            GT:...:PL   1/1:...:2198,192,0
1   183629   .    G    A    71.03     .  AN=2;DP=4;...;ANN=A|downstream_gene_variant|....|     GT:....:PL  1/1:...:99,12,0
```

Figure 1.6: VCF format example. The header contains information about filters, format and info field abbreviations, the genome reference dictionary and information about the programs that processed this file. Includes three variants to show the columnar VCF format.

which is more accurate for these types of variant calling but requires both a normal and a tumor sample to run properly. As this thesis focuses on single sample analysis we only consider the HaplotypeCaller. The algorithm used in this module has four steps. The first is to identify active regions, these are regions where significant evidence of variation is found. The second step uses a de Bruijn graph to locally reassemble the active region to identify possible haplotypes. These haplotypes are realigned against the reference haplotype using the Smith-Waterman algorithm to identify potential variant sites. In the third step the paired-end reads are realigned to each haplotype using the pairHMM algorithm. This gives a matrix of likelihoods for each haplotype which in turn is used to obtain likelihood for alleles for each variant site. The last steps uses Bayes' rule described before to calculate the likelihoods of alleles given the read data to calculate the likelihoods of each genotype per sample given the observed data for that sample. Lastly the genotype with the highest likelihood is assigned to the sample.

1.2.5.3 The VCF Format

The VCF format is a text file format that stores variant information, a simple example is shown in Fig. 1.6. It contains meta-information lines, a header and subsequently one data line per variant. The meta-information lines start with '##' followed by a key-value pair. This contains information about the VCF format version, the file date, reference file, contigs etc. The file format must always be present. After this, the meta-information lines contain three optional entries: info, filter and format. The info and format fields contain information about the similarly named columns in the variant entries. The filter entries show the filters that have been applied to the VCF file, e.g. quality equal to or greater than 10. Next comes the header line syntax: one line starting with '#' followed by a tab-separated list that contains the column names of the variant entries. There are 8 fixed, mandatory columns in this order: chrom, pos, id, ref, alt, qual, filter and info. If genome data is present in the files, these are added to the header by first showing the format column and then a column for every sample, where the sample name is the name of the column.

After this, the actual variants are added, one variant per line. The fields are tab-separated and follow the order of the header line. The chrom and pos column identify the exact location and chromosome where the variant is found. The id field shows the unique identifiers when available, e.g. the rs numbers from dbSNP. The ref and alt fields show what nucleotides are found in the reference and the sample respectively. These two fields can also be used to identify if a variant is an indel or a SNP. Qual shows the Phred-scaled quality score for the assertion made in alt, and is equal to $-10log_{10}Pr\left\{call\ in\ alt\ is\ wrong\right\}$. The filter column shows pass if this variant has passed all filters, if it does not this field contains all the failed filters in a semicolon-separated list of codes for the filters. The info field contains additional information that has been introduced in the meta-information fields. This contains a semicolon-separated list of key-value pairs. Next are the genotype fields, with first a format field which specifies the data types and order (colon-separated string). This is followed by one field per sample, which contains a colon-separated string of the values in order for that sample.

1.3 Differences for RNA sequencing data

1.3.1 RNA read alignment with STAR

As described in the Best Practices pipeline for RNA data the read alignment is performed using STAR [17]. This RNA alignment tool achieves high speedups compared with different tools (factor of >50) while keeping successful alignment rate very high. The alignment of RNA data is different compared with the DNA alignment process. This is because most eukaryotic genes consist of multiple exons, which can be spliced together in several possible combinations. Here, two key tasks make the analysis computationally intensive. First, alignments can contain mismatches and indels caused by mutations or sequencing errors, this is identical to DNA sequencing. Second, the alignment algorithm needs to take into account the possibility that reads can span multiple exons and can therefore have potentially large gaps corresponding to the intron regions. The second task is unique to the alignment of RNA sequencing (RNA-seq) data, but is crucial as it provides the connectivity information for the spliced RNA molecules. STAR focuses solely on RNA-seq data whereas other RNA-seq alignment tools are often extensions of existing DNA alignment tools. The algorithm used consists of two major steps: a seed searching step and a clustering/stitching/scoring step. The idea is to first do the sequential search for a Maximal Mappable Prefix (MMP). The search for MMP's gives the opportunity to detect *de novo* splice sites. Next, the seeds found in the first steps are combined and used to find the optimal alignment position. It is possible to use existing splice site information to improve the alignment. When *de novo* splice sites are used from a previous alignment run, the total alignment process is commonly referred to as 2-pass alignment. This algorithm uses an uncompressed suffix array and thus uses a lot of memory for this step (the reference genome for human genome is approximately 27 GBytes).

1.3.2 Split Reads By CIGAR String

This is a step that is only present in the RNA pipeline. This step splits reads into exon segments, this means getting rid of the N's of the alignment but retaining grouping information. This first part includes all bases left of the first N element, the rest is trimmed. The second part starts after this first N element until the next N elements and so on. Additionally, this step also reassigns mapping qualities, because STAR assigns good alignments a quality of 255, which represents an unknown quality according to the SAM format specifications. So this quality is changed to the default value of 60. This step is done by the SplitNCigars module from the GATK tool [18]. After this step, the next two steps are identical to the DNA pipeline. The reads are realigned near indel regions and the scores of the base qualities are recalibrated using a list of known variants.

1.3.3 Variant calling with RNA-seq data

Variant calling for RNA-seq data can only be performed with the HaplotypeCaller module from GATK and not with the UnifiedGenotyper. The latter does not recognize reads that are split in parts caused by alternative splicing of the genes in RNA sequencing data sets. The HaplotypeCaller supports through arguments given to the GATK command to specify that RNA data is being processed. The basic algorithm remains the same as explained in the previous section.

1.4 The MapReduce Programming Model

Prior to the development of a parallelized pipeline, we investigated a number of existing solutions. Crossbow [19], implements a similar pipeline and parallelizes this pipeline using the MapReduce programming model. This similar pipeline is based on alignment with Bowtie [20] and SOAPsnp [21]. MapReduce is similarly used to accelerate the read alignment phase in CloudBurst [22], BigBWA [23] and DistMap [24].

The MapReduce programming model is designed to handle big problems, i.e. a huge amount of data. To handle this big data, the Hadoop implementation of MapReduce provides access to a distributed file system. This means that data can easily be split up and then accessed over multiple nodes. Another important aspect regarding big data, is the fault-tolerant scheme Hadoop has provided. This means that hardware

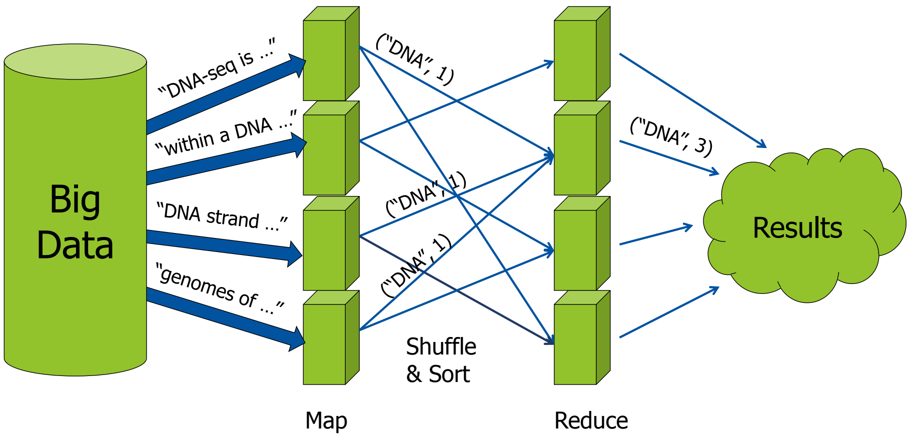


Figure 1.7: Word count example of the MapReduce programming model. The input data is split in a number of chunks that are processed in parallel during the map phase. Each mapper emits <key-value> pairs where the key corresponds to an observed word and the value corresponds to 1. The <keys-value> pairs are sorted according to key and sent to a number of parallel reduce tasks where each individual reduce task will process one specific key (in this case: one specific word) and all associated values (in this case: a number of 1s). In the reduce task, values that correspond to specific key are aggregated. In this case the sum of all values is computed. An output <key-value> pair is emitted, with again the key corresponding to a word and the value to the number of times the word has been observed.

failure does not necessarily mean that the entire job has to be done again, instead only the failed parts can be done again. The relatively easy nature of the problems is handled in two big phases, a map phase and a reduce phase. The map simply organises data by a key, after which the reduce aggregates the data corresponding to a single key. The 'hello world' example that is typically used is counting the frequency of words in a huge text. This problem is simple, counting occurrences of words, but the size of the input data can make it a big problem.

The model consists of three consecutive parallel steps: map, sort and reduce, executed in this specific order. The map phase processes the input data and generates a number of intermediary <key-value> pairs. These records are sorted and collected by key. Each reduce task processes a single key by aggregating (or reducing) all corresponding values. The word count example starts by mapping every word in a given input text to a <string-int> pair where the key is the word and the value is 1. The sort phase sorts all the keys, in this case the words, and collects all values into an iterable list. During the reduce phase every key has a corresponding list of values, these are all summed to produce a total count of that word. This word count example is shown in Fig. 1.7. The map and reduce phase are executed by map and reduce tasks which are started by the framework. These map tasks read (a chunk of) the input data and process this data and emit the <key-value> pairs to the framework. By default Hadoop MapReduce starts only 1 reduce slot which processes all keys in the reduce step. To improve performance this number can be increased to increase the parallelism. To assign every key to a different reduce task Hadoop uses a hash function on the key to assign the key to a reduce task. To have optimal performance every reduce task should have about the same amount of keys to process. Having a good hash function to divide the keys over the reduce tasks results in a balanced load and correlates directly to the achieved parallel efficiency. This simple programming model can be applied to a wide range of problems that can be translated to this three step model.

Hadoop MapReduce version 2.0 and later uses Yarn as a scheduling system. With Yarn the total number of map or reduce containers itself cannot be set, but instead Yarn allows users to assign the required memory and cores for both the map and reduce tasks per job. This gives a more flexible scheduling system as the resources can easily be shared between different applications.

During the course of this thesis MapReduce has been used in several other tools to parallelize this pipeline. MegaSeq [25] has implemented this pipeline to work on Beagle, a Cray XE6 supercomputer at Argonne National Laboratories. However, this tool aims at high throughput with concurrent multiple genome analysis instead of single sample analysis. HugeSeq [26] is similar to MegaSeq as it also parallelizes on a chromosome level instead of splitting on a deeper level like this thesis describes. More similar to this method, with deeper level of splitting of the genomic regions, is Churchill [27]. However, this pipeline is implemented using Bash and Python scripts instead of the MapReduce to distribute the workload over the worker nodes using either the Sun Grid Engine (SGE) or Portable Batch System (PBS).

1.4.1 Apache Spark and Spark Notebook

As the MapReduce programming model has a very linear data-flow structure, the data is first described as key-value pairs which is then sorted, after which it is aggregated by key. With this model, many problems can be solved, but is very limiting if one wants to deviate from this simple data-flow structure. In response, Apache Spark was introduced as a more flexible framework. Apache Spark provides an Application Programming Interface (API) centered on a distributed data structure called resilient distributed dataset (RDD). This is a read-only multiset of data items distributed over a cluster of nodes, and is maintained in a fault-tolerant way. The API contains many functions to be used on data in this RDD structure, including the map, sort and reduce functions from MapReduce. On top of this filter, union, intersection and more functions are provided.

To use Spark, the cluster needs a cluster manager and a distributed file system. For cluster management, Spark can run in stand-alone mode (native Spark Cluster) or use Hadoop YARN or Apache Mesos. As for distributed file systems, Apache Spark supports a wide variety, including Hadoop Distributed File System (HDFS), MapR File System (MapR-FS), Cassandra, Amazon S3 and more. The Spark API has been implemented in several programming languages, including Scala, Python and Java.

The API includes higher-order model of programming, the instruction is passed to a "driver" program which invokes parallel operations such as map, filter or reduce on an RDD by passing a function to Spark, which in turn schedules the function's execution in parallel on the cluster. The RDDs themselves are immutable, so the output of these functions are new RDDs which can be cached in memory if multiple access is required. With this the performance can be considerably improved, if the data needs to be accessed multiple times, e.g. in an iterative algorithm.

Spark Notebook [28] is a fork of the Scala Notebook, which changed its focus to massive data set analysis with Apache Spark. A Notebook is a web-based code editor that combines Scala code, SQL queries, Markup or even JavaScript in a collaborative manner. This allows performing reproducible analysis, here with Scala and Apache Spark as well as making graphs from the data.

1.4.2 VCF analysis and comparison

Often in VCF analysis, different VCF files are compared to one another. As an example tumor and normal samples are often compared, as well as different alignment tools can be compared or different sequencing strategies on the same sample (RNA-seq, WES and WGS). The process of comparing the VCF files is often an iterative process and requires manual intervention to determine the next set of filters. Tools like vcftools [29] can be used to perform functions like compare, filter, intersect and much more on a single or multiple VCF files. The iterative process includes filtering of the VCF files and comparing the variants of both files after the filtering and this is repeated until a certain goal is satisfied. To analyze the variants in a VCF file, these are often annotated with SnpEff [30] and/or SnpSift [31] or tools with similar functions. As an example SnpSift can detect if a variant is present in a database of known variants, e.g. common variants or clinically important variants. SnpEff can be used to annotate genetic variants and predict the effects it has (such as amino changes). Annotation can for example provide information on the location of the variants (exon, intron or intergene) or changes in protein codes like premature stop codons. This information can be used to further filter and select variants during the variant comparison process.

This comparison happens quite often and can be time consuming if big files are compared. This is caused by the tools that are typically command line tools and many lack multithreading. Another reason is the way the process works, first the files are filtered and these output variants are then compared to see if a certain goal is met. With an implementation to compare two or three VCF files in a Spark Notebook we tackle both these problems. The Spark cluster allows for faster processing of files and provides a scalable solution. The Notebook interface allows the iterative filtering to go faster as good visual representations can be used. We implemented such a notebook in the last chapter as a use case for variant analysis.

1.5 Overview of the chapters

In the next chapter we will take a look at the post-sequencing analysis pipeline. We use MapReduce to implement a parallel best-practices pipeline, we dub the program Halvade [2], short for Hadoop ALignment and VAriant DEtection. The parallel efficiency and speedup are determined by running tests on a local Hadoop cluster and Amazon EMR. And lastly, to confirm the accuracy of Halvade, we compare the output to that of the original pipeline. We continue benchmarking Halvade in the third chapter. We look at the influence of the number of parallel tasks, as well as the influence of the interconnection network, type of local disk and distributed file system. These tests help us identify the bottlenecks and we attempt to alleviate them. We conclude the chapter with an overview of the new runtimes and speedups we achieve with the optimizations. The RNA-seq pipeline is somewhat different from the DNA-based pipeline. We implemented these changes to be able to run in Halvade. Halvade-RNA is benchmarked and the accuracy measured. This is discussed in the fourth chapter. The VCF files we get from Halvade can be used in analysis. In the fifth chapter we talk about how we implement this in Spark Notebooks. As a proof of concept we compare RNA-seq, WGS and WES data, we look at the concordance and discordance between the three sequencing strategies. Finally, we end with an overview of all our findings during the implementation and optimization of Halvade. We also give a summary of the VCF analysis we have performed, where we compare RNA-seq, WES and WGS from the same sample. At the end of the chapter we talk about what can still be accomplished and where this work can still be improved upon.

References

[1] Metzker, M. L., 2010. Sequencing technologies - the next generation. *Nature reviews. Genetics*, 11(1):31–46.

[2] Decap, D., J. Reumers, C. Herzeel, P. Costanza, and J. Fostier, 2015. Halvade: scalable sequence analysis with MapReduce. *Bioinformatics*, pages btv179+.

[3] Fonseca, N. A., J. Rung, A. Brazma, and J. C. Marioni, 2012. Tools for mapping high-throughput sequencing data. *Bioinformatics*, 28(24):3169–3177.

[4] Nielsen, R., J. S. Paul, A. Albrechtsen, and Y. S. Song, 2011. Genotype and SNP calling from next-generation sequencing data. *Nat Rev Genet*, 12(6):443–451.

[5] 2012. Novocraft.com: Novoalign short read mapper (http://www.novocraft.com/main/downloadpage.php).

[6] Lunter, G. and M. Goodson, 2011. Stampy: a statistical algorithm for sensitive and fast mapping of Illumina sequence reads. *Genome research*, 21(6):936–939.

[7] Li, H. and R. Durbin, 2009. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics (Oxford, England)*, 25(14):1754–1760.

[8] Li, H., 2013. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM.

[9] Minikel, E., 2012. How pcr duplicates arise in next-generation sequencing.

[10] Wysoker, A., 2009. Picard.

[11] Li, H., B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, et al., 2009. The Sequence Alignment/Map format and SAMtools. *Bioinformatics (Oxford, England)*, 25(16):2078–2079.

[12] Broad Institute, 2017. Indel-based realignment.

[13] Broad Institute, 2016. Base quality score recalibration (bqsr).

[14] Zen Fractal, 2014. Introduction to base quality score recalibration (bqsr).

[15] Broad Institute, 2014. Gatk single sample genotype likelihoods.

[16] Broad Institute, 2017. Haplotypecaller.

[17] Dobin, A., C. A. Davis, F. Schlesinger, J. Drenkow, C. Zaleski, et al., 2012. STAR: ultrafast universal RNA-seq aligner. *Bioinformatics*, 29(1):bts635–21.

[18] McKenna, A., M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis, et al., 2010. The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. *Genome research*, 20(9):1297–1303.

[19] Langmead, B., M. C. Schatz, J. Lin, M. Pop, and S. S. L, 2009. Searching for SNPS with cloud computing. *Genome Biology*, 10:R134.

[20] Langmead, B., C. Trapnell, M. Pop, and S. L. Salzberg, 2009. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology*, 10:R25.

[21] Li, R., Y. Li, K. Kristiansen, and J. Wang, 2008. SOAP: short oligonucleotide alignment program. *Bioinformatics*, 24:713–714.

[22] Schatz, M. C., 2009. CloudBurst: highly sensitive read mapping with MapReduce. *Bioinformatics*, 25:1363–1369.

[23] Abuín, J. M., J. C. Pichel, T. F. Pena, and J. Amigo, 2015. BigBWA: approaching the BurrowsWheeler aligner to Big Data technologies. *Bioinformatics*, pages btv506+.

[24] Pandey, R. V. and C. Schlötterer, 2013. DistMap: A Toolkit for Distributed Short Read Mapping on a Hadoop Cluster. *PLoS ONE*, 8:doi:10.1371/journal.pone.0072614.

[25] Puckelwartz, M. J., L. L. Pesce, V. Nelakuditi, L. Dellefave-Castillo, J. R. Golbus, et al., 2014. Supercomputing for the parallelization of whole genome analysis. *Bioinformatics*, 30(11):1508–1513.

[26] Lam, H. Y. K., C. Pan, M. J. Clark, P. Lacroute, R. Chen, et al., 2012. Detecting and annotating genetic variations using the HugeSeq pipeline. *Nature Biotechnology*, 30(3):226–229.

[27] Kelly, B. J., J. R. Fitch, Y. Hu, D. J. Corsmeier, H. Zhong, et al., 2015. 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(1).

[28] Petrella, A., 2014. Spark Notebook.

[29] Danecek, P., A. Auton, G. Abecasis, C. A. Albers, E. Banks, et al., 2011. The variant call format and VCFtools. *Bioinformatics*, 27(15):2156–2158.

[30] Cingolani, P., A. Platts, M. Coon, T. Nguyen, L. Wang, et al., 2012. A program for annotating and predicting the effects of single nucleotide polymorphisms, snpeff: Snps in the genome of drosophila melanogaster strain w1118; iso-2; iso-3. *Fly*, 6(2):80–92.

[31] Cingolani, P., V. Patel, M. Coon, T. Nguyen, S. Land, et al., 2012. Using drosophila melanogaster as a model for genotoxic chemical mutational studies with a new program, snpsift. *Frontiers in Genetics*, 3.

2
Halvade: scalable sequence analysis with MapReduce

In this chapter, we introduce Halvade, a parallel solution to the Best Practices pipeline for variant calling recommended by the Broad Institute. Halvade uses Hadoop MapReduce as programming model in order to achieve a highly scalable and parallel result. Halvade is tested and evaluated for both correctness and performance.

⋆ ⋆ ⋆

Dries Decap, Joke Reumers, Charlotte Herzeel, Pascal Costanza and Jan Fostier.

Published in Bioinformatics 31(15). 2015. p2482-2488

Abstract Post-sequencing DNA analysis typically consists of read mapping followed by variant calling. Especially for whole genome sequencing, this computational step is very time consuming, even when using multithreading on a multi-core machine. We present Halvade, a framework that enables sequencing pipelines to be executed in parallel on a multi-node and/or multi-core compute infrastructure in a highly efficient manner. As an example, a DNA sequencing analysis pipeline for variant calling has been implemented according to the GATK Best Practices recommendations, supporting both whole genome and whole exome sequencing. Using a 15-node computer cluster with 360 CPU cores in total, Halvade processes the NA12878 dataset (human,

100 bp paired-end reads, 50x coverage) in less than 3 hours with very high parallel efficiency. Even on a single, multi-core machine, Halvade attains a significant speedup compared to running the individual tools with multithreading. Halvade is written in Java and uses the Hadoop MapReduce 2.0 API. It supports a wide range of distributions of Hadoop, including Cloudera and Amazon EMR. Its source is available at http://bioinformatics.intec.ugent.be/halvade under GPL license.

2.1 Introduction

The speed of DNA sequencing has increased considerably with the introduction of next-generation sequencing platforms. For example, modern Illumina systems can generate several hundreds of gigabases per run [1] with a high accuracy. This, in turn, gives rise to several hundreds of GBytes of raw sequence data to be processed.

Post-sequencing DNA analysis typically consists of two major phases: (1) alignment of reads to a reference genome and (2) variant calling, i.e., the identification of differences between the reference genome and the genome from which the reads were sequenced. For both tasks, numerous tools have been described in literature, see e.g. [2] and [3] for an overview. Especially for whole genome sequencing, applying such tools is a computational bottleneck. To illustrate this, we consider the recently proposed Best Practices pipeline for DNA sequencing analysis [4] that consists of the Burrow-Wheeler Aligner (BWA) [5] for the alignment step, Picard [6] for data preparation and the Genome Analysis Toolkit (GATK) [7, 8] for variant calling. On a single node, the execution of this pipeline consumes more time than the sequencing step itself: a dataset consisting of 1.5 billion paired-end reads (Illumina Platinum genomes, NA12878, 100 bp, 50-fold coverage, human genome) requires over 12 days using *a single* CPU core of a 24-core machine (dual socket Intel Xeon E5-2695 v2 @ 2.40GHz): 172 hours for the alignment phase, 35 hours for data preparation (Picard steps) and 80 hours for GATK, including local read realignment, base quality score recalibration and variant calling. When allowing the involved tools to run multithreaded on the same machine, the runtime decreases only by a factor of roughly 2.5 to $\sim$5 days, indicative of a poor scaling behavior in some of the steps of the pipeline.

To overcome this bottleneck, we developed Halvade, a modular framework that enables sequencing pipelines to be executed in parallel on a multi-node and/or multi-core compute infrastructure. It is based on the simple observation that read mapping is parallel by read, i.e., the alignment of a certain read is independent of the alignment of another read. Similarly, variant calling is conceptually parallel by genomic region, e.g., variant calling in a certain chromosomal region is independent of variant calling in a different region. Therefore, multiple instances of a tool can be run in parallel on a subset of the data. Halvade relies on the MapReduce programming model [9] to execute tasks concurrently, both within and across compute nodes. The map phase corresponds to the read mapping step while variant calling is performed during the reduce phase. In between both phases, aligned reads are sorted in parallel according

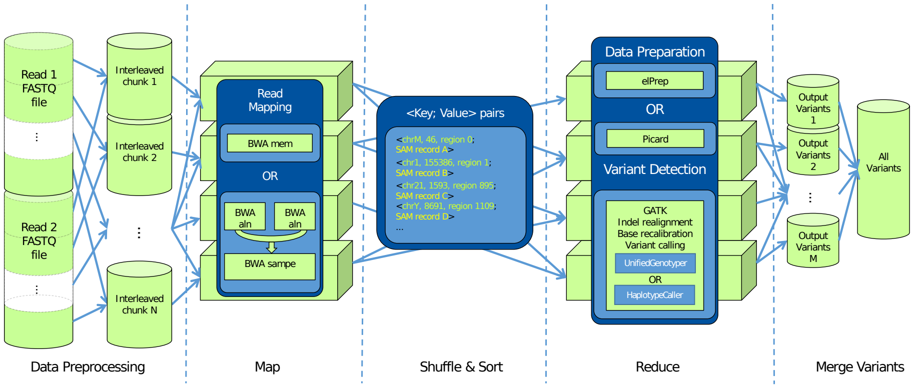


Figure 2.1: Overview of the Halvade framework. The entries of pairs of input FASTQ files (containing paired-end reads) are interleaved and stored as smaller chunks. Map tasks are executed in parallel, each task taking a single chunk as input and aligning the reads to a reference genome using an existing tool. The map tasks emit <key, value> pairs where the key contains positional information of an aligned read and the value corresponds to a SAM record. The aligned reads are grouped and sorted per chromosomal region. Chromosomal regions are processed in parallel in the reduce phase, this includes data preparation and variant detection again using tools of choice. Each reduce task outputs the variants of the region it processed. These variants can optionally be merged into a single VCF file. Note that the name of the tools shown correspond to those of the GATK Best Practices DNA-seq implementation in Halvade.

to genomic position. By making use of the aggregated compute power of multiple machines, Halvade is able to strongly reduce the runtime for post-sequencing analysis. A key feature of Halvade is that it achieves very high parallel efficiency which means that computational resources are efficiently used to reduce runtime. Even on a single, multi-core machine, the runtime can be reduced significantly as it is often more efficient to run multiple instances of a tool, each instance with a limited number of threads, compared to running only a single instance of that tool with many threads. As an example, both whole genome and whole exome variant calling pipelines were implemented in Halvade according to the GATK Best Practices recommendations (i.e., using BWA, Picard and GATK).

The MapReduce programming model has been used before in CloudBurst [10] and DistMap [11] to accelerate the read mapping process and in Crossbow [12] to accelerate a variant calling pipeline based on modified versions of Bowtie [13] and SOAPsnp [14]. The Halvade framework extends these ideas, enabling the implementation of complex pipelines while supporting different tools and versions. The software is designed to achieve a good load balance, maximize data locality and minimize disk I/O by avoiding file format conversions. As a result, Halvade achieves much higher parallel efficiencies compared to similar tools.

More recently, MapReduce-like scripts were used in MegaSeq [15], a workflow for concurrent multiple genome analysis on Beagle, a Cray XE6 supercomputer at Argonne National Laboratories. Like Halvade, MegaSeq implements a whole genome analysis pipeline based on the GATK Best Practices recommendations. However, whereas MegaSeq focuses on a high throughput of many genomes using a specific, extreme-scale compute platform, Halvade aims to maximally reduce the analysis runtime for the processing of a single genome, while supporting a wide variety of computer clusters. This approach is particularly of use in a clinical setting, where the analysis step will typically be performed on a local cluster within a hospital environment, and where the time between obtaining a DNA sample from a patient and diagnosing should be kept as small as possible. The source code of Halvade is publicly available.

2.2 Methods

2.2.1 Halvade framework

Halvade relies on the MapReduce programming model [9] to enable parallel, distributed-memory computations. This model consists of two major phases: the *map* and *reduce* phase. During the map phase, different map tasks are executed in parallel, each task independently processing a chunk of the input data and producing as output a number of intermediate <key, value> pairs. Next, the intermediate <key, value> pairs emitted by *all* map tasks are sorted, in parallel, according to key by the MapReduce framework. During the reduce phase, different reduce tasks are executed in parallel, each reduce task independently processing a single key and its corresponding values.

Conceptually, a read alignment and variant calling pipeline can be cast into the MapReduce framework: read alignment is then performed in the map phase where the different map tasks are processing parts of the input FASTQ files in parallel while the variant calling and, if required, additional data preparation steps, are handled in the reduce phase where the different reduce tasks are processing chromosomal regions in parallel. Using the MapReduce framework, the reads are sorted according to their aligned position and grouped by chromosomal region in between the two phases. The Halvade framework provides access to data streams for individual tools that run in parallel during read mapping and variant calling. An overview of the Halvade framework is depicted in Fig. 2.1. The different computational steps are described in more detail below.

2.2.1.1 Input data preparation

The input data typically consist of paired-end reads stored in two distinct, compressed FASTQ files. We provide a separate tool called 'Halvade Uploader' which interleaves the paired-end reads, storing paired reads next to each other, and splits the data in chunks of about 60 MByte of compressed data. These chunks are transferred on-the-fly to the file system from which they can be accessed by the worker nodes. In case a generic Hadoop system is used, this is the Hadoop Distributed File System (HDFS); in case Amazon EMR is used, chunks are uploaded to the Amazon Simple Storage

Service (Amazon S3) using the Amazon S3 API. The number of input chunks corresponds to the number of map tasks that will be executed during the map phase. The rationale behind the Halvade Uploader is that data have to be copied or uploaded onto the compute infrastructure anyhow, and that decompressing, interleaving, splitting and again compressing can easily overlap with this transfer, thus reducing file I/O to its minimum. The interleaving of paired-end reads ensures that both pairs are accessible in a single task, which is required for the read alignment. The Halvade Uploader is multithreaded and operates on data streams, which means that its execution can overlap with the data generation (i.e., sequencing) step itself.

Prior to the actual execution of the MapReduce job, additional preparatory steps are required. First, the reference genome is partitioned into a pre-determined number of non-overlapping chromosomal regions of roughly equal size. The number of chromosomal regions corresponds to the total number of reduce tasks that will be executed during the reduce phase and can be configured by the user based on the size of the reference genome in question. Next, Halvade ensures that all required binaries and configuration files are available on each worker node. It does so by adding all required files, in a compressed archive, to the distributed cache which is then copied to each worker node and again decompressed. Note that when these files are persistently stored onto the worker nodes, this preparatory step can be omitted.

2.2.1.2 Map phase – read alignment

For the map phase, one map task is created per input FASTQ chunk. These tasks are in turn executed in parallel on the worker nodes by a number of mappers. Typically, the number of map tasks $\gg$ the number of mappers which means that each mapper will process many tasks. Key to the MapReduce model is that a map task will preferably be executed by a worker node that contains the input chunk locally on disk (as a part of the HDFS) in order to minimize remote file access and thus network communication. Each mapper first checks if the indexed reference genome is locally available and retrieves it from HDFS or Amazon S3 when this is not the case. The input chunks are read from HDFS or S3 and parsed into input <key, value> pairs using the Hadoop-BAM [16] API. The values of these pairs contain the FASTQ entries and are streamed to an instance of the alignment tool.

Halvade requires that the alignments are accessible in SAM format and provides a SAM stream parser that will create the required intermediate <key, value> pairs. This intermediate key represents a composite object that contains the chromosome number and alignment position of a read, along with the identifier of the chromosomal region to which it aligns. The value contains the corresponding SAM record, i.e., the read itself and all metadata. Reads that cannot be aligned are optionally discarded. For individual or paired-end reads that span the boundary of adjacent chromosomal regions, two intermediate <key, value> pairs are created, one for each chromosomal region. This redundancy ensures that all required data for the data preparation and variant calling is available for each reduce task.

After all map tasks are completed, the MapReduce framework sorts, in parallel, the intermediate pairs according to chromosomal region (as part of the key). This way, all reads that align to the same chromosomal region are grouped together thus forming the input of a single reduce task. Halvade uses secondary sorting to further sort the SAM records for each chromosomal region by genomic position. Both grouping and sorting effectively replace the sorting of SAM records typically performed by tools such as Picard or SAMtools [17] and are performed in a highly efficient manner by the MapReduce framework.

2.2.1.3 Reduce phase – variant calling

When all data has been grouped and sorted, the different reduce tasks are executed in parallel by different reducers. Again, the number of reduce tasks $\gg$ the number of reducers. Before the reads are processed, Halvade can copy to each worker node additional files or databases that are required for variant calling. A specific task takes as input all (sorted) intermediate <key, value> pairs for a single chromosomal region and converts it to an input stream in SAM format. Halvade iterates over the SAM records and creates a BED file [18] containing position intervals that cover all SAM records in the chromosomal region. This file can optionally be used to specify relevant intervals on which tools need to operate. Finally, instances of these tools are created in order to perform the actual variant calling.

Typically, at the end of each reduce task, a VCF file has been produced which contains all variants identified in the corresponding chromosomal region. Halvade provides the option to merge these VCF files using an additional MapReduce job. In this second job, the map phase uses the individual VCF files as input and the variants are parsed as <key, value> pairs using Hadoop-BAM. The key contains the chromosome identifier and the position of the variant, while the value contains all other meta-information. These values are collected in a single reduce task, which writes the aggregated output to either HDFS or Amazon S3. If variants at the same location are found from SAM records that were sent to adjacent chromosomal regions, either all are emitted or only the variant with the highest Phred-scaled quality score is retained. Note that this second MapReduce job is very light-weight.

2.2.2 Best Practices DNA-seq implementation

In Halvade, a DNA-seq variant calling pipeline has been implemented according to the Best Practices recommendations by [19]. Table 2.1 lists the different steps involved.

During the map phase, read alignment is performed by BWA; both BWA-mem and BWA-aln with BWA-sampe are supported in our implementation. In case BWA-aln is used, paired-end reads are again separated and aligned individually by two instances of BWA-aln after which BWA-sampe is used to join these partial results. The standard output stream of either BWA-sampe or BWA-mem is captured, and its SAM records are parsed into intermediate <key, value> pairs.

Table 2.1: Overview of the steps and tools involved in the DNA sequencing pipeline according to the GATK Best Practices recommendations described by [19].

step	program	input	output
align reads	BWA	FASTQ	SAM
convert SAM to BAM	Picard	SAM	BAM
sort reads	Picard	BAM	BAM
mark duplicates	Picard	BAM	BAM
identify realignment intervals	GATK	BAM	intervals
realign intervals	GATK	BAM & intervals	BAM
build BQSR table	GATK	BAM	table
recalibrate base quality scores	GATK	BAM & table	BAM
call variants	GATK	BAM	VCF

In the reduce phase, the SAM stream is first prepared according to GATK's requirements, i.e., the read group information is added, read duplicates (i.e., reads that are sequenced from the same DNA molecule) are marked and the data is converted to the binary, compressed BAM format. Note that in the Best Practices recommendations, read group information is added during read alignment. In Halvade, this is postponed to the reduce phase in order to avoid sending this extra meta-information (as part of the SAM record) over the network during sorting. For data preprocessing, Halvade can use either Picard or elPrep (http://github.com/exascience/elprep) with SAMtools [17]. ElPrep is a tool that combines all data preparation steps and outputs a SAM file that conforms to the GATK requirements. When using elPrep, the input SAM records are streamed directly to elPrep for marking duplicate reads and adding of read group information. Its resulting SAM file is then converted to BAM format using SAMtools. When using Picard, Halvade first writes the input SAM stream to local disk in a compressed BAM file and then invokes the Picard MarkDuplicates and AddReadGroups modules. Note that both options (elPrep/SAMtools or Picard) produce identical output. However, the combination of elPrep and SAMtools is considerably faster than Picard.

Next, the actual GATK modules are executed. To correct potentially misaligned bases in reads due to the presence of insertions or deletions, the RealignerTargetCreator module is used to identify intervals that require realignment followed by the IndelRealigner module to perform the actual realignment. Next, using the dbSNP database of known variants [20], the BaseRecalibrator module is used to generate co-variation data tables that are then used by the PrintReads module to recalibrate the base quality scores of the aligned reads. Finally, the actual variant calling is done using either the HaplotypeCaller or UnifiedGenotyper module.

The DNA-seq analysis pipeline implementation in Halvade supports both whole genome and exome sequencing analysis. One important difference to note is that an additional BED file is required, containing the coordinates of the exome regions to be

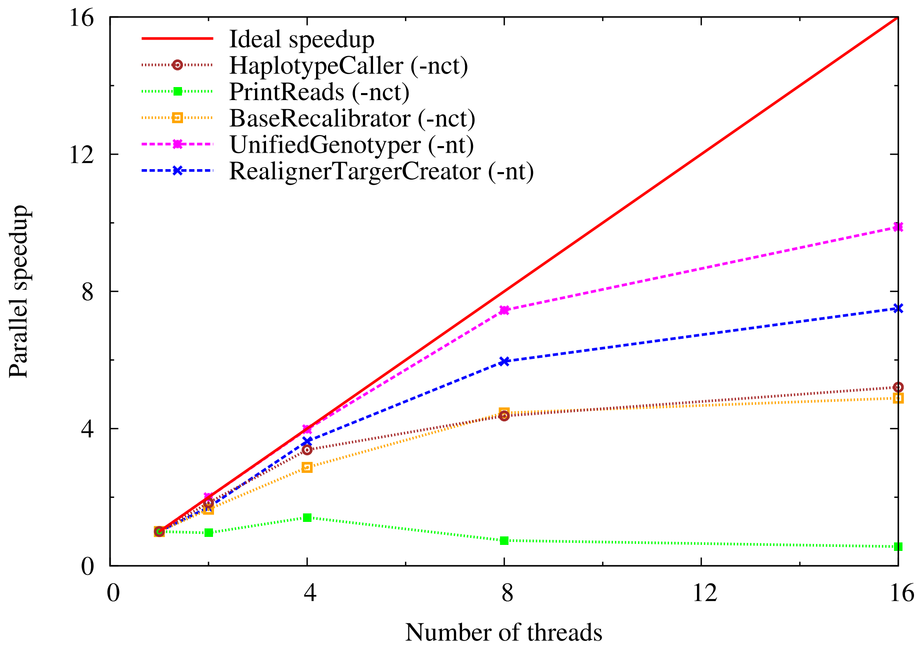


Figure 2.2: The parallel speedup (multithreading) of five GATK modules used in the Best Practices pipeline on a 16-core node with 94 GByte of RAM. The limited speedup prevents the efficient use of this node with more than a handful of CPU cores. Option -nt denotes data threads while option -nct denotes CPU threads (cfr. GATK manual).

processed by GATK. Additionally, the dbSNP database file used for the base quality score recalibration must be compatible with the exome being targeted.

2.2.3 Optimizations

In order to get the best performance out of the available resources, two crucial factors come into play. First, one needs to determine the optimal number of mappers (reducers) per node. This determines the number of map (reduce) tasks that will be executed concurrently on a worker node. Second, the user needs to select an appropriate map (reduce) task size. This determines the total number of map (reduce) tasks that will be executed. Both factors are described below.

To exploit parallelism in a workstation with one or more multi-core CPUs, one can either run a single instance of a tool with multithreading on all available CPU cores, or run multiple instances of that tool, each instance using only a fraction of the CPU cores. To illustrate the difference in performance, the parallel speedup for different GATK modules as a function of number of threads was benchmarked on a 16-core machine (dual socket Intel Xeon CPU E5-2670 @ 2.60GHz) with 94 GByte of RAM (Fig. 2.2). The benchmarks show that the maximum speedup gained by any of the GATK modules, using 16 threads, is less than 10, with one module exhibiting no speedup at all. Most modules show good scaling behavior up to 4 or 8 threads, but show only moderate reduction in runtime when the number of threads is increased further. It is thus more beneficial to start multiple instances of GATK, each instance using only a limited number of threads. Note that this is also true for Picard (which is single-threaded) and to a lesser extent, also for BWA. This concept is used in Halvade, which leads to better use of available resources and a higher overall parallel efficiency (see The optimal number of parallel tasks per node). On the other hand, the maximum number of parallel instances of a tool than can be run on a machine might be limited due to memory constraints.

A second important factor is the optimal task size, which in turn determines the total number of tasks. For the map phase, the task size is determined by the size of a FASTQ chunk. Very few, large chunks will lead to a high per-task runtime but an unevenly balanced workload, whereas many little files will result in a large task scheduling and tool initialization overhead. After extensive testing, we determined that a file size of about 60 MByte leads to the lowest runtime (see Influence of the map/reduce task size). Such chunk size is sufficiently big to define a meaningful task size, and small enough for a chunk to fit into a single HDFS block (default=64 MByte) which is entirely stored on a single worker node. If that worker node processes the corresponding map task, network traffic is avoided. Similarly, for the reduce phase, the task size is determined by the size of the chromosomal regions.

For data preparation, Picard can be replaced by elPrep. This tool combines all data preparation steps needed in the Best Practices pipeline. Whereas Picard requires file I/O for every preparation step, elPrep avoids file I/O by running entirely in memory and merges the computation of all steps in a single pass over the data. Using elPrep for data preparation again gives a significant speedup for this phase in Halvade.

2.2.3.1 The optimal number of parallel tasks per node

Halvade can run several parallel tasks (mappers/reducers) per node to increase overall performance. On nodes with a large number of CPU cores, it is often more efficient to run several instances of a tool, each instance using only a limited number of threads compared to the scenario of running a single instance of that tool using all available CPU cores. This is illustrated in Fig. 2.3 for the map phase (BWA-aln and BWA-sampe) on a 24-core machine of the Intel Big Data cluster using a small subset of the NA12878 dataset. Running four parallel tasks per node, each task using 6 CPU cores, significantly reduces the runtime compared to running only a single instance of BWA using 24 threads. In this case, this effect is largely due to BWA-sampe, which is single-threaded. Note that further increasing the number of parallel tasks per node no longer results in a reduction of runtime. When using only a few threads per tool instance, near optimal performance is already attained. Additionally, the maximum number of tool instances can be limited by the available memory of the worker nodes. Alignment tools require the reference genome and other data structures to be stored in memory and hence exhibit a memory footprint of a few GBytes.

Halvade can estimate the optimal number of tasks for both map and reduce phases on any cluster based on memory requirements of the tools and the parallel efficiency when running the tool with multithreading enabled.

2.2.3.2 Influence of the map/reduce task size

The load balancing of Halvade is determined by the size of the tasks of both map and reduce phases. To illustrate this, Fig. 2.4 shows the distribution of runtimes for all map tasks when using input chunks with a size of 60 MByte and 240 MByte, respectively. The average task execution time when using 240 MByte input chunks is about fourfold the average execution time when using 60 MByte input chunks. However, the distribution of execution times when using 240 MByte chunks is, expressed in absolute runtime, much wider than the distribution for 60 MByte chunks. In that case, even when every mapper is processing a roughly equal number of map tasks, load is not necessarily balanced. Additionally, because input chunks are stored on the HDFS in blocks of 64 MByte (default on Apache Hadoop), a chunk with a size of 64 MByte or smaller will be stored as a single HDFS block on a worker node in its entirety. MapReduce will try to schedule the map task on a worker node that actually contains the corresponding input chunk, thus avoiding (slow) network communication. Note that a single HDFS block, is, again by default, replicated on three worker nodes to prevent loss of data when a worker node fails. This provides extra scheduling flexibility to MapReduce as it can now choose from three worker nodes that can process the chunk without any required network communication. On the other hand, too small input chunks lead to small tasks sizes with a relatively large task scheduling and tool initialization overhead.

The same is true for the reduce tasks. If the genomic regions are too big, load balancing will be poor. On the other hand, selecting too small genomic regions will again suffer from increased overhead. Additionally, in order to compute accurate base quality score recalibration tables, at least 500 000 reads are required per genomic region (see Halvade accuracy assessment).

The runtime of Halvade is determined by the runtime of both map and reduce phases. In the MapReduce framework, no reduce task can start before all map tasks have finished. However, certain shuffle tasks *can* be started before all map tasks are done. This has the advantage that network communication, typically a bottleneck during data shuffling, is spread in time. However, this means that one or multiple task slots will no longer be used for read mapping but for data shuffling. Because the shuffle tasks can only finish once all map output has been determined, these shuffle tasks will sometimes be idle and thus waste valuable resources. In order to avoid this, Halvade is set to finish all map tasks prior to starting any shuffle & sort tasks.

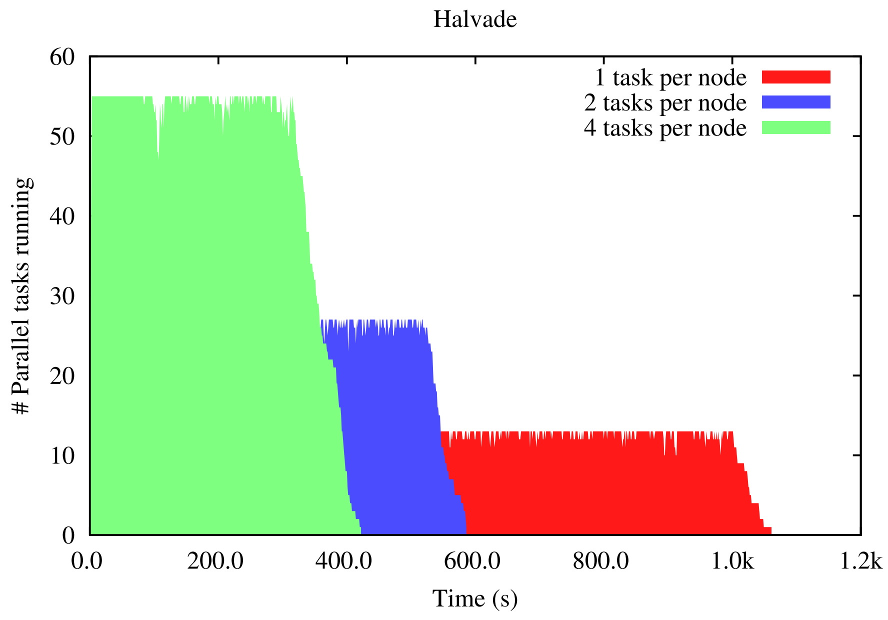


Figure 2.3: Runtime of the map phase (BWA-aln and BWA-sampe) using 1 task (24 CPU cores per task), 2 tasks (12 CPU cores per task) and 4 tasks (6 CPU cores per task) in parallel per node. Note that in all three cases, 14 worker nodes were used.

2.3 Results

Halvade was benchmarked on a whole genome human dataset (NA12878 from Illumina Platinum Genomes). The dataset consists of 1.5 billion 100 bp paired-end reads (50-fold coverage) stored in two 43 GByte compressed (gzip) FASTQ files. The software was benchmarked on two distinct computer clusters, an Intel-provided big-data cluster located in Swindon, U.K. and a commercial Amazon EMR cluster. Table 2.2 provides an overview of the runtime on these clusters. For these benchmarks, GATK version 3.1.1, BWA version 0.7.5a, BEDTools version 2.17.0, elPrep version 1.0, SAMtools version 0.1.19 and Picard version 1.112 were used. The dbSNP database and human genome reference found in the GATK hg19 resource bundle [1] were used. The reference genome was stripped of all alternate allele information and contains chromosomes 1 through 22, M, X and Y.

2.3.1 Intel Big Data cluster benchmark

This cluster consists of 15 worker nodes, each containing 24 CPU cores (dual-socket Intel Xeon CPU E5-2695 v2 @ 2.40GHz) and 62 GByte of RAM. The nodes each

[1] available to download at ftp://gsapubftp-anonymous@ftp.broadinstitute.org/bundle/hg19/

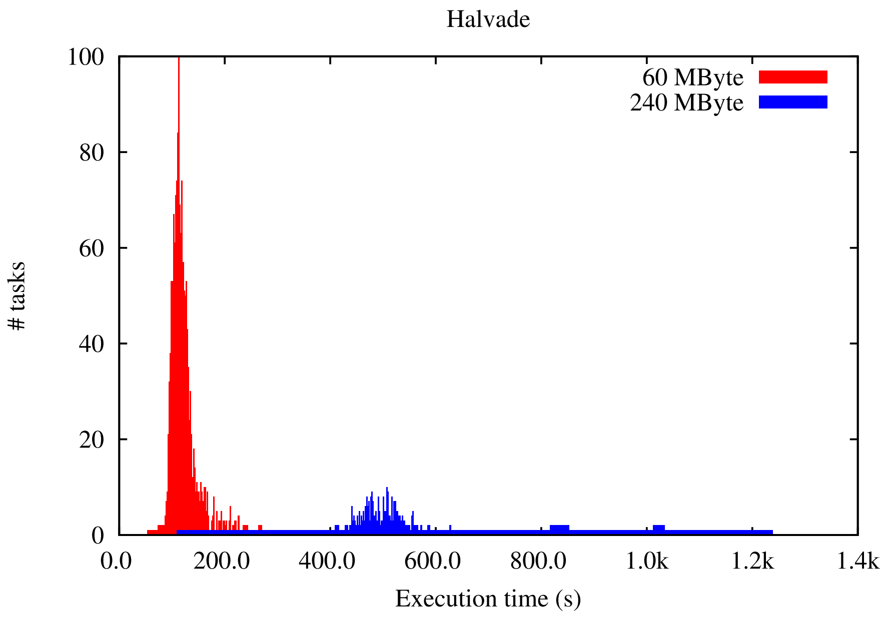


Figure 2.4: Distribution of the execution time of the map tasks using input chunks with a size of 60 MByte and 240 MByte.

dispose of four hard drives with a total capacity of 4 TByte, intended as HDFS storage and a single 800 GByte solid-state drive (SSD) that is intended for local storage during MapReduce jobs. The nodes are interconnected by a 10 Gbit/s Ethernet network. Cloudera 5.0.1b which supports MapReduce 2.3 was used. Initially, the input FASTQ files were present on a local disk of a single node. Using the Halvade Uploader, both files were decompressed, interleaved, compressed into separate files of about 60 MByte each (1552 chunks in total) and copied onto the local HDFS storage. This preprocessing step required ∼1.5 hours using 8 threads and can, in principle, overlap with the generation of the sequence data itself. For the chromosomal regions, a size of 2.5 Mbp was used, corresponding to 1261 reduce tasks in total.

Taking into account the memory requirements of the individual tools, the available RAM (62 GByte) and the available number of CPU cores (24) of a worker node, optimal performance on the Intel Big Data cluster was obtained using four parallel map or reduce tasks per node. This translates to 15.5 GByte of RAM and 6 CPU cores for each mapper/reducer. Fig. 2.5 shows the task execution over time. Load is well-balanced, indicative of an efficient use of resources during both phases and a good parallel efficiency. The scalability of Halvade was assessed by running the analysis pipeline with an increasing number of 1 to 15 nodes. As Cloudera reserves one slot for scheduling and job management, this corresponds to running 3 parallel tasks (1 node) to 59 parallel tasks (15 nodes) in total. Fig. 2.6 depicts the parallel

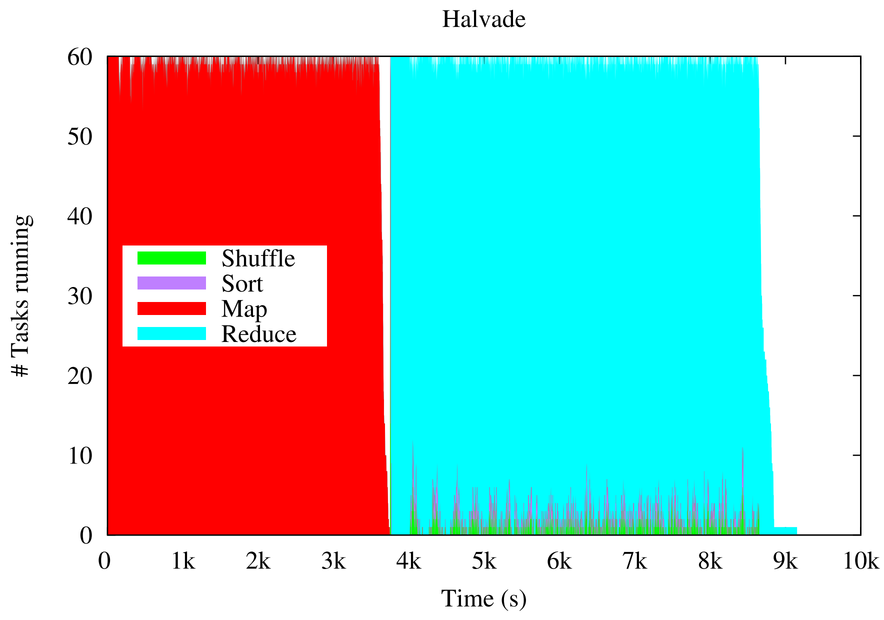


Figure 2.5: Task slot occupation of Halvade on the Intel Big Data cluster using 15 worker nodes (4 parallel tasks/node) on the complete NA12878 dataset.

speedup as a function of the number of parallel tasks and provides an accurate view of the scalability and efficiency. When using 59 tasks (15 nodes), we observe a speedup of a factor 18.11 compared to using 3 tasks (1 node). This corresponds to a parallel efficiency of 92,1%. In absolute terms, the runtime reduces from ∼48 hours (single node) to 2 hours 39 minutes.

It is important to note that Halvade already attains a significant speedup when applied to a single node (3 tasks and 18 CPU cores), compared to the scenario of running the multithreaded versions of the individual tools using all 24 CPU cores. Indeed, whereas Halvade requires ∼48 hours on a single node, ∼120 hours are required when Halvade is not applied (speedup of a factor 2.5). This is due to the limited multithreaded scaling behavior of certain tools or modules (see Methods). It is hence far more efficient to run multiple instances of e.g. GATK with a limited number of threads per instance than letting GATK make use of all available cores. Ultimately, Halvade achieves a ∼45-fold speedup when applied to 15 nodes (2 hours 39 minutes) compared to running the pipeline on a single node using only multithreading (120 hours).

2.3.2 Amazon EMR benchmark

Amazon Elastic Compute Cloud (Amazon EC2) provides, as a web service, a resizeable compute cluster in the cloud. MapReduce can be used on this compute cluster

Table 2.2: Runtime as a function of the number of parallel tasks (mappers/reducers) on the Intel Big Data cluster and Amazon EMR.

Cluster	# worker nodes	# parallel tasks	# CPU cores	runtime
Intel Big Data cluster	1	3	18	47h 59min
	4	15	90	9h 54min
	8	31	186	4h 50min
	15	59	354	2h 39min
Amazon EMR	1	4	32	38h 38min
	2	8	64	20h 19min
	4	16	128	10h 20min
	8	32	256	5h 13min
	16	64	512	2h 44min

with a service called Amazon EMR. This provides access to a Hadoop MapReduce cluster which can be chosen according to the requirements of the task at hand, e.g., the number of nodes and the node type. The EMR cluster was initialized with the Amazon Machine Images (AMI) v3.1.0, providing access to Hadoop MapReduce v2.4, which takes ∼5 minutes. One node is reserved for job scheduling and management while the remainder is used as worker nodes.

Using the Halvade Uploader, the input data was preprocessed and uploaded to S3, the Amazon cloud storage system, again as 1552 chunks. The uploading speed is highly dependent on internet network conditions, which varies greatly. Again, this step can overlap with the generation of the sequence data itself. As data stored on S3 is directly accessible by worker nodes, one can chose whether or not to copy these data to HDFS prior to starting the MapReduce job. According to [21], data can be copied between S3 and HDFS at a rate of 50 GByte per 19 minutes. However, for these benchmarks, data was read directly from S3, as copying would increase the overall runtime considerably.

For this benchmark, worker nodes of the type c3.8xlarge were used. Each node provides 32 CPU cores (Intel Xeon E5-2680 v2 @ 2.80GHz), 60 GByte of RAM and two 320 GByte SSDs which are available for both HDFS and intermediate data. To obtain optimal performance, Halvade again assigned four parallel tasks per node, with each task disposing of 8 CPU cores and 15 GByte of memory. The scalability was assessed by running Halvade with an increasing number of 1 to 16 worker nodes. When using 64 tasks (16 nodes), a speedup of a factor 14.16 is achieved compared to using 4 tasks (1 node) (see Fig. 2.6). This corresponds to a parallel efficiency of 88.5%. In absolute terms, the total runtime is reduced from 38 hours 38 minutes (4 tasks) to 2 hours 44 minutes (64 tasks) (see Table 2.2).

The runtime on Amazon EMR is slightly higher than that obtained using the Intel

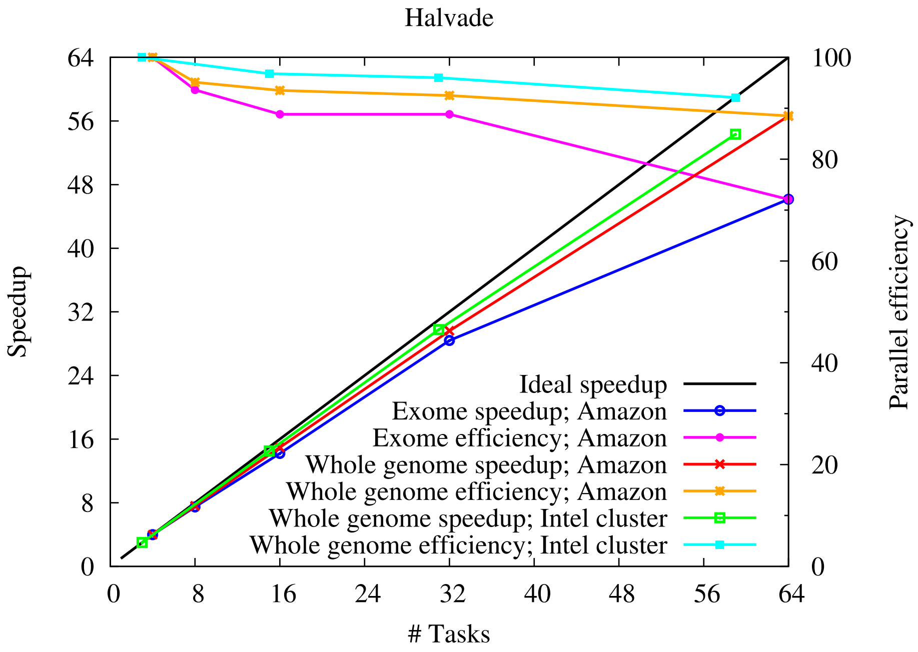


Figure 2.6: The speedup (primary y-axis) and parallel efficiency (secondary y-axis) of Halvade as a function of number of parallel tasks (cluster size) on both an Intel Big Data cluster and Amazon EMR.

Big Data cluster, even though a higher number of CPU cores was used. This is because the Intel Big Data cluster is configured with a persistent HDFS, whereas for the Amazon cluster, the HDFS is created on-the-fly when the MapReduce job starts. On the Intel Big Data cluster, we can therefore instruct the Halvade Uploader to copy data directly to the HDFS after which only limited network communication is required for map tasks to access the data. On Amazon, Halvade accesses the data straight from S3, which requires network communication and explains the increased runtime. With a total runtime of less than 3 hours, the financial cost of a whole genome analysis on Amazon EMR with 16 worker nodes amounts to 111.28 US dollar (based on the pricing of May 2014).

2.3.3 Exome sequencing analysis benchmark

To assess the performance of Halvade on an exome sequencing dataset (Illumina HiSeq NA12878), the same Amazon EMR cluster was used. The dataset consists of 168 million 100 bp paired-end reads stored in eight ∼1.6 GByte compressed (gzip) FASTQ files.

Using an Amazon EMR cluster with 8 worker nodes (32 parallel tasks), Halvade can call the variants in under one hour for a total cost of 19.64 US dollar (based on the pricing of May 2014). As the input for exome analysis is considerably smaller, the load balancing is more challenging as there are only 225 map tasks and 469 reduce tasks in total. A high parallel efficiency of about 90% is obtained when using 8 worker nodes; the efficiency drops to about 70% when using 16 worker nodes (see Fig. 2.6).

2.3.4 Comparison with Crossbow

Several modifications to the source code of Crossbow [12] were required to enable the software to run on the current version of Amazon EMR. Crossbow was benchmarked on Amazon EC2 c1.xlarge instances with default settings. These instances have 8 CPU cores (dual-socket Intel Xeon E5410 @ 2.33GHz), 7 GByte of RAM and four 420 GByte hard drives. As Amazon uses one node for job scheduling and management and the number of nodes available on demand is limited to 20, we ran benchmarks on 1, 10 and 19 *worker* nodes using a smaller dataset (ERR000589, human genome, 51 bp, $\sim$24 million paired-end reads). Table 2.3 shows an overview of the runtime and parallel speedup achieved with Crossbow. A parallel efficiency of about 58% was obtained using 19 worker nodes (152 CPU cores).

2.3.5 Halvade accuracy assessment

Halvade assumes that read mapping is parallel by read and that variant calling is parallel by genomic region. However, the VCF file obtained by a parallel Halvade run can differ slightly from that of a sequential run. To pinpoint the origin of this variation, we compared the SAM/BAM files of a parallel and sequential run at several points in the pipeline. To facilitate this procedure, we used the much smaller TAIR10 *Arabidopsis thaliana* genome [22] with a subset of the Seattle-0 dataset (1001 Genomes Project, available at http://1001genomes.org/data/SLU/SLUHenning2014/) as input. We identified three distinct causes of variation in the resulting VCF files.

The first, and biggest cause of variation originates from the read alignment step. Halvade splits the input FASTQ file in several smaller chunks to define parallel tasks in the map phase. Running BWA separately on these chunks leads, for certain reads, to a different alignment compared to running BWA on the original FASTQ file. To confirm this, 2 million reads were aligned twice: (1) stored in a single FASTQ file and (2) stored in four FASTQ files and aligned separately with four BWA instances. The resulting SAM files reveal that about 6% of the alignments differ. Most of these differences (80%) are alignments with same coordinate but with a different alignment score. The other 20% are reads that map to a different genomic position, most of which (90%) are mapped in repetitive regions. The other 10% represent reads with an alignment score of zero or close to zero and are flagged as 'unmapped'. We assume that BWA estimates the insert length of the fragments from which the paired-end reads were sequenced from a set of reads where both pairs map to non-repetitive regions. If multiple instances of BWA are used on different input chunks, the different instances use a different training set, and obtain a (slightly) different insert length. This would explain the variation. Even though other sources of variation exist (see further), the

Table 2.3: Parallel speedup obtained using Crossbow on Amazon EMR.

# worker nodes	# CPU cores	runtime	speedup
1	8	19h 24min	1.00
10	80	2h 28min	7.86
19	152	1h 46min	10.98

variation in alignment appears to be the biggest cause of variation in the resulting VCF file.

Second, during the reduce phase, reads that likely represent polymerase chain reaction (PCR) duplicates are marked by Picard or elPrep. When applying this procedure on separate SAM file chunks, the mark duplicates algorithm might mark a different read as 'duplicated' in case two reads map to the same coordinate and have the same quality score, compared to running this procedure on a single SAM file.

The third difference originates from computation of the base quality score recalibration (BQSR) tables. In Halvade, these statistics are computed individually per chromosomal region, using only the reads that map to that chromosomal region, instead of genome-wide (i.e., using all reads). This can result in slightly different recalibration metrics, yielding small differences in quality scores after recalibration. However, the authors of GATK have analyzed the residual root-mean-square error (RMSE) after recalibration when downsampling a dataset using the -L option of GATK [2]. They show that, when using at least 500 000 reads, the BQSR will produce near optimal results. The average number of reads that is used in each Halvade reduce task for whole genome analysis is over 1.2 million. This means that the BQSR done in Halvade will also produce near optimal recalibration. This is also ensured for exome sequencing by using less reduce tasks in total and thus increasing the number of reads per task.

Finally, to assess the downstream effects of this variation, the VCF file obtained with Halvade was compared to that obtained by a sequential run of the same pipeline, now again for the human NA12878 dataset (see Table 2.4). A total of 4.2 million variants were found both by Halvade and the reference run. This represents 99.1% of all variants found by the reference run. Halvade detects 25 336 unique variants (not found in the reference run), while 36 429 variants present in the reference VCF file were not discovered using Halvade. Fig. 2.7 shows that almost all of these differences correspond to variants with a very low 'variant confidence score', i.e., for which there is very little sequence evidence that the variant is a true positive.

A similar quality check was performed for the exome sequencing dataset. In that case, a reference VCF file was already provided, which was used to compare the output of Halvade against. Most of the variants found by Halvade (97.5%) were also present in the reference VCF. The different variants again has a low variant confidence score.

In conclusion, despite certain variation in SAM/BAM files that arises in certain steps of the Halvade pipeline, the resulting VCF file is reliable and has excellent cor-

[2]described in http://gatkforums.broadinstitute.org/discussion/44/base-quality-score-recalibration-bqsr

Table 2.4: Overlap and differences in variants in the VCF files obtained by a Halvade run and a sequential reference run.

# variants	Dataset intersection	% match
25336	Found only by Halvade run	0.6
36429	Found only by reference run	0.9
4210001	Found by both reference & Halvade run	99.1 / 99.4

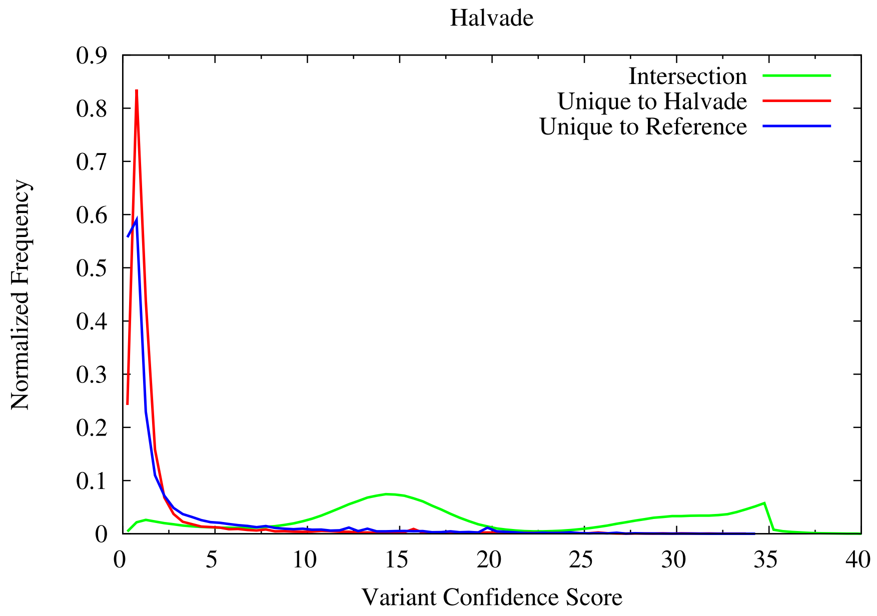


Figure 2.7: Normalized frequency as a function of variant confidence score of variants (a) both found by Halvade and a sequential reference run (b) only found by Halvade (c) only found by the reference run.

respondence to that obtained by a sequential execution of the pipeline.

2.4 Discussion and Conclusion

Especially for whole genome sequencing, the post-sequencing analysis (runtime of $\sim$12 days, single-threaded) is more time-consuming than the actual sequencing (several hours to a few days). Individual tools for mapping and variant calling are maturing and pipeline guidelines such as the GATK Best Practices recommendations are provided by their authors. However, existing tools currently have no support for multi-node execution. Even though some of them support multithreading, the parallel speedup that can be attained might be limited, especially when the number of CPU cores is high. As whole genome analysis is increasingly gaining attention, the need for a solution is apparent.

Halvade provides a parallel, multi-node framework for read alignment and variant calling that relies on the MapReduce programming model. Read alignment is then performed during the map phase, while variant calling is handled in the reduce phase. A variant calling pipeline based on the GATK Best Practices recommendations (BWA, Picard and GATK) has been implemented in Halvade and shown to significantly reduce the runtime. On both a Cloudera cluster (15 worker nodes, 360 CPU cores) and a commercial Amazon EMR cluster (16 worker nodes, 512 CPU cores), Halvade is able to process a 50-fold coverage whole genome dataset in under 3 hours, with a parallel efficiency of 92.1% and 88.5% respectively. To the best of our knowledge, these are the highest efficiencies reported to date. Like Halvade, MegaSeq supports a pipeline which is based on the GATK Best Practices recommendations. As the source code of MegaSeq is not publicly available, a direct comparison to Halvade is difficult. [15] estimate, based on benchmarks on 61 whole genomes, that 240 whole genomes could be processed in 50.3 hours, using a supercomputer with 17 424 CPU cores (AMD Magny-Cours @ 2.1 GHz). When rescaling this to 360 CPU cores @ 2.4 GHz, an average runtime of 8.9 hours is obtained for a single genome. Halvade processes such dataset in 2 hours and 39 minutes and thus provides for a more cost-effective way to minimize runtime. It should be noted that (a) MegaSeq used the GATK HaplotypeCaller whereas Halvade relied on the UnifiedGenotyper during benchmarking, (b) additional variant annotation was performed in MegaSeq, (c) a comparison based on the number of CPU cores and clock frequency alone has its limitations as also disk speed, available RAM, network speed and other hardware aspects may play a role.

To enable multi-node parallelization of sequencing pipelines, Halvade provides and manages parallel data streams to multiple instances of existing tools that run on the different nodes. No modifications to these tools are required; they can thus easily be replaced by newer versions. Functionality is provided to copy external files or databases that might by required by tools to the worker nodes. To achieve optimal performance on computer clusters with multi-core nodes, Halvade can be configured to run multiple parallel instances of a tool per node, each instance using a limited number of threads. This approach significantly increases the per-node performance as the multithreaded scaling behavior of certain tools is limited. Indeed, on single a 24-core node with three parallel tasks, Halvade already attains a speedup of 2.5 compared to a multithreaded execution of the same tools in a single tasks.

In Halvade, it is assumed that read alignment is parallel by read and that variant calling is parallel by genomic region. Certain tools, however, produce slightly different results when they operate on only part of the data. We have analyzed these sources of variation in detail. As for the accuracy of whole genome analysis, the variants found by Halvade match more than 99% with variants in the validation set created by a sequential run of the GATK Best Practices pipeline. Additionally, almost all the 1% different variants have a very low variant score.

In the implementation of the GATK Best Practices pipeline, the latest versions of BWA, Picard and GATK are supported. Both whole genome and exome analysis are supported. An RNA-seq variant calling pipeline is currently in development. Halvade is built with the Hadoop MapReduce 2.0 API and thus supports all distributions of Hadoop, including Amazon EMR and Cloudera. The Halvade source code available at http://bioinformatics.intec.ugent.be/halvade under GPL license.

References

[1] Zhang, J., R. Chiodini, A. Badr, and G. Zhang, 2011. The impact of next-generation sequencing on genomics. *Journal of genetics and genomics*, 38:95–109.

[2] Fonseca, N. A., J. Rung, A. Brazma, and J. C. Marioni, 2012. Tools for mapping high-throughput sequencing data. *Bioinformatics*, 28(24):3169–3177.

[3] Nielsen, R., J. S. Paul, A. Albrechtsen, and Y. S. Song, 2011. Genotype and SNP calling from next-generation sequencing data. *Nat Rev Genet*, 12(6):443–451.

[4] Van der Auwera, G. A., M. O. Carneiro, C. Hartl, R. Poplin, G. del Angel, et al., 2013. From FastQ Data to High-Confidence Variant Calls: The Genome Analysis Toolkit Best Practices Pipeline. *Current protocols in bioinformatics / editoral board, Andreas D. Baxevanis ... [et al.]*, 11(1110).

[5] Li, H. and R. Durbin, 2009. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics (Oxford, England)*, 25(14):1754–1760.

[6] Wysoker, A., 2009. Picard.

[7] McKenna, A., M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis, et al., 2010. The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. *Genome research*, 20(9):1297–1303.

[8] DePristo, M. A., E. Banks, R. Poplin, K. V. Garimella, J. R. Maguire, et al., 2011. A framework for variation discovery and genotyping using next-generation DNA sequencing data. *Nature genetics*, 43(5):491–498.

[9] Dean, J. and S. Ghemawat, 2008. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM*, 51(1):107–113.

[10] Schatz, M. C., 2009. CloudBurst: highly sensitive read mapping with MapReduce. *Bioinformatics*, 25:1363–1369.

[11] Pandey, R. V. and C. Schlötterer, 2013. DistMap: A Toolkit for Distributed Short Read Mapping on a Hadoop Cluster. *PLoS ONE*, 8:doi:10.1371/journal.pone.0072614.

[12] Langmead, B., M. C. Schatz, J. Lin, M. Pop, and S. S. L, 2009. Searching for SNPS with cloud computing. *Genome Biology*, 10:R134.

[13] Langmead, B., C. Trapnell, M. Pop, and S. L. Salzberg, 2009. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology*, 10:R25.

[14] Li, R., Y. Li, K. Kristiansen, and J. Wang, 2008. SOAP: short oligonucleotide alignment program. *Bioinformatics*, 24:713–714.

[15] Puckelwartz, M. J., L. L. Pesce, V. Nelakuditi, L. Dellefave-Castillo, J. R. Golbus, et al., 2014. Supercomputing for the parallelization of whole genome analysis. *Bioinformatics*, 30(11):1508–1513.

[16] Niemenmaa, M., A. Kallio, A. Schumacher, P. Klemel, E. Korpelainen, et al., 2012. Hadoop-BAM: Directly manipulating next generation sequencing data in the cloud. *Bioinformatics*, 28:876–877.

[17] Li, H., B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, et al., 2009. The Sequence Alignment/Map format and SAMtools. *Bioinformatics (Oxford, England)*, 25(16):2078–2079.

[18] Quinlan, A. R. and I. M. Hall, 2010. BEDTools: a flexible suite of utilities for comparing genomic features. *Bioinformatics*, 26(6):841–842.

[19] Van der Auwera, G. A., M. O. Carneiro, C. Hartl, R. Poplin, G. del Angel, et al., 2013. From FastQ Data to High-Confidence Variant Calls: The Genome Analysis Toolkit Best Practices Pipeline. *Current Protocols in Bioinformatics*, 43:11.10.1–11.10.33.

[20] Sherry, S. T., M. H. Ward, M. Kholodov, J. Baker, L. Phan, et al., 2001. dbSNP: the NCBI database of genetic variation. *Nucleic acids research*, 29(1):308–311.

[21] Deyhim, P., 2013. Best Practices for Amazon EMR. Technical report, Amazon Web Services Inc.

[22] Garcia-Hernandez, M., T. Z. Berardini, G. Chen, D. Crist, A. Doyle, et al., 2002. TAIR: a resource for integrated Arabidopsis data. *Funct Integr Genomics*, 2:339–253.

3
Performance analysis of a parallel, multi-node pipeline for DNA sequencing

In the previous chapter we analyzed the correctness and performance of Halvade. This showed that the performance is good when comparing task speedup, but when looking at the speedup compared to the single-core runtime we see a much lower efficiency. In this chapter, we further analyze the performance bottlenecks of Halvade and attempt to further increase the parallel efficiency.

⋆ ⋆ ⋆

Dries Decap, Joke Reumers, Charlotte Herzeel, Pascal Costanza and Jan Fostier.

Published in PPAM. 2014.

Abstract Post-sequencing DNA analysis typically consists of read mapping followed by variant calling and is very time-consuming, even on a multi-core machine. Recently, we proposed Halvade, a parallel, multi-node implementation of a DNA sequencing pipeline according to the GATK Best Practices recommendations. The MapReduce programming model is used to distribute the workload among different workers. In this paper, we study the impact of different hardware configurations on the performance of Halvade. Benchmarks indicate that especially the lack of good multithreading capabilities in the existing tools (BWA, SAMtools, Picard, GATK) cause suboptimal scaling behavior. We demonstrate that it is possible to circumvent this bottleneck by using multiprocessing on high-memory machines rather than using

multithreading. Using a 15-node cluster with 360 CPU cores in total, this results in a runtime of 1h 21min. Compared to a single-threaded runtime of ∼12 days, this corresponds to an overall parallel efficiency of 59%.

3.1 Introduction

The human DNA sequence was determined for the first time during the Human Genome Project, a $3 billion undertaking launched in 1990 and declared complete in 2003. Since then, the development of next-generation sequencing (NGS) platforms has tremendously increased the throughput at which genomic data can be produced, at a significantly lowered price. Nowadays, resequencing a human genome with a financial cost of under $1000 has become reality. Consequently, sequencing is becoming more and more standard practice, not only in academic research but also in clinical settings.

Several computational post-sequencing steps are required to transform raw sequencing data into a format that is usable for biological or clinical interpretation. In case of Illumina platforms, which currently accounts for about 90% of all sequencing data produced worldwide, raw data consists of many short fragments (so-called 'reads') with a length of 100-250 bp and an error rate of ∼1%. In resequencing applications, these reads are aligned to a reference genome ('read mapping') followed by the identification of differences between the reference genome and the aligned reads ('variant calling').

For both tasks, numerous tools have been described in literature. The Broad Institute has proposed Best Practices recommendations [1] for a DNA variant calling pipeline based on BWA [2] for read alignment, SAMtools [3]/Picard [4] for data preprocessing and GATK [5, 6] for variant calling. Especially for whole-genome datasets, this pipeline is very time-consuming with a single-core runtime of ∼12 days to process the NA12878 dataset (Illumina Platinum genomes, 1.5 billion paired-end reads, 100 bp, 50-fold coverage, human genome). Even when enabling multithreading support in the individual tools, the execution time for this dataset is still ∼5 days on a 24-core machine (dual socket Intel Xeon E5-2695 v2 @ 2.40GHz), indicative of poor scaling behavior.

To deal with this bottleneck, we recently proposed Halvade [7], a parallel, multi-node framework in which a variant calling pipeline has been implemented according to the GATK Best Practices recommendations. Halvade relies on the MapReduce programming model [8] to run multiple instances of existing tools (BWA, SAMtools/Picard, GATK) in parallel both across and within nodes on subsets of the data. Halvade is based on the simple observation that read mapping is parallel by read (i.e., aligning a certain read does not depend on the alignment of other reads) while variant calling is parallel by genomic region (i.e., variant calling in a certain genomic region does not depend on variant calling in other genomic regions). During the map phase, BWA is used to align reads to a reference genome in parallel, whereas data preprocessing

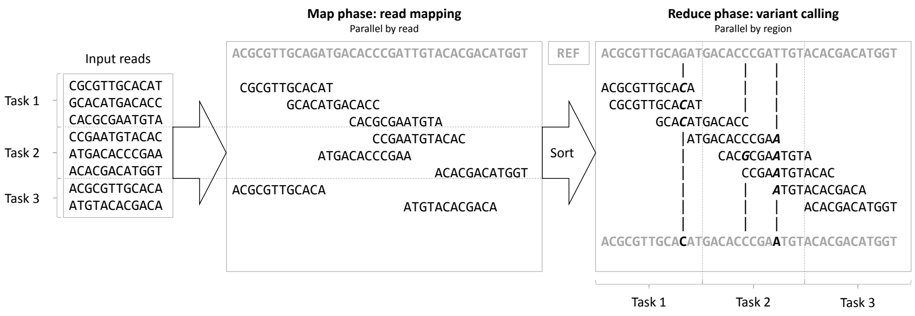


Figure 3.1: Halvade MapReduce implementation of a variant calling pipeline. During the map phase, input reads are aligned, in parallel, to a reference genome using BWA. Next, aligned reads are sorted according to aligned position using the MapReduce sorting functionality. In the reduce phase, variants are identified between the reference genome and aligned reads using GATK.

(SAMtools/Picard) and variant calling (GATK) are handled during the reduce phase by operating on different genomic regions in parallel. In between the map and reduce step, the aligned reads are sorted according to genomic position using the MapReduce sorting functionality. Fig. 3.1 presents an overview of Halvade. We refer to [7] for implementation details. Halvade is written in Java using the Hadoop MapReduce 2.0 API. The source code is available at http://bioinformatics.intec.ugent.be/halvade under GPL license.

In [7], it was demonstrated that Halvade strongly reduces runtime: on a 15-node cluster, each node containing 24 CPU cores and 64 GByte of RAM, the NA12878 dataset was processed in 2h 39min. Additionally, it was shown that the multi-node parallel efficiency of Halvade is excellent (around 90%), which means that the runtime is significantly reduced by using 15 nodes compared to using only a single node. However, significant performance loss can still be observed *within* each node. This can be seen from the overall performance: with a runtime of 2h 39min using 360 CPU cores (15 nodes $\times$ 24 cores/node), a speedup of $\sim$108 is obtained compared to a single-threaded runtime of $\sim$12 days. This corresponds to an overall parallel efficiency of about 30%, suggesting the presence of certain performance bottlenecks. Understanding the performance of a sequencing pipeline is a non-trivial matter. Certain components in the pipeline are very compute-intensive (e.g. read alignment) whereas other components (e.g. data preprocessing) are mostly data-intensive. Therefore, certain tools might be CPU bound whereas others might be limited by I/O bandwidth.

In order to better understand the influence of hardware configuration on the performance of sequencing pipelines, we have set up a range of benchmarks in order to identify possible bottlenecks. Specifically, in this paper, we study the influence on the total runtime of the amount of available RAM, the presence of NUMA domains, the type of network interconnection, the use of solid-state disks versus hard-disk drives and finally, the use of a distributed vs. centralized file system. We demonstrate that

Table 3.1: Runtime and parallel efficiency as a function of the number of tasks per node.

	map phase	reduce phase	total
1 task × 1 thread	14h 50min	15h 38min	30h 28min
	n/a	n/a	n/a
1 task × 24 threads	4h 28min	12h 3min	16h 31min
	13.84%	5.41%	7.69%
4 tasks × 6 threads	1h 21min	3h 6min	4h 27min
	45.78%	21.01%	28.53%
24 tasks × 1 thread	47min	55min	1h 42min
	78.80%	71.06%	74.67%

the use of high-memory machines and NUMA optimizations can further reduce the overall runtime whereas other hardware aspects have only limited influence. Additionally, load balancing in both map and reduce phases was improved. Ultimately, the combined effect of optimizations allows us to process the entire NA12878 dataset in 1h 21min, yielding an overall parallel efficiency of 59%, almost twice the efficiency reported in [7].

Even though the benchmarks in this work were obtained using Halvade, much of the analysis equally applies to alternative methods that implement sequencing pipelines in a distributed manner. Like Halvade, BigBWA [9] leverages Hadoop MapReduce to accelerate read mapping using BWA, but lacks variant calling functionality. The same functionality was recently also implemented on top of Spark by [10]. Both HugeSeq [11] and MegaSeq [12] implement a complete variant calling pipeline, however, parallelism during variant calling is limited to concurrent processing of entire chromosomes. In terms of functionality, Churchill [13] is similar to Halvade, however, whereas Halvade is MapReduce-based, Churchill relies on a combination of Bash and Python scripts to distribute the workload over the worker nodes using either the Sun Grid Engine (SGE) or Portable Batch System (PBS).

3.2 Dataset and tool versions

In all benchmarks, variant calling was performed on a whole-genome DNA sequencing dataset (NA12878, human genome, Illumina Platinum Genomes) or a subset thereof. The full dataset consists of 1.5 billion 100 bp paired-end reads (50-fold coverage) stored in two 43 GByte compressed (gzip) FASTQ files.

The DNA sequencing pipeline, used in Halvade and based on the Best Practices recommendations, consists of a number of existing tools. Reads are aligned using BWA-aln and BWA-sampe version 0.7.12-r1044. Reads are sorted according to genomic positions using the Hadoop framework. Data preprocessing (SAM to BAM conversion and read duplicate identification) is typically achieved using Picard. However, in all benchmarks, we used elPrep version 1.0 as a drop-in replacement for Picard as elPrep has a higher performance while producing identical output [14]. Indel realignment, base quality score recalibration (BQSR) and variant calling steps are all performed with GATK version 3.1.1. The dbSNP [15] database and human genome reference found in the GATK hg19 resource bundle [16] were used.

3.3 Single Node Benchmarks

As the runtime of the complete NA12878 dataset on a single node is impractically high, all benchmarks in this section were performed on a representative subset of 131 million paired-end reads (about 9% of the total number of reads). Benchmarks in this section were run on a single 24-core node (dual Intel E5-2680v3 @ 2.50GHz) with 512 GByte of RAM.

3.3.1 Influence of the number of tasks per node

When running Halvade, the number of parallel tasks (mappers/reducers) per node can have a big influence on performance. The number of tasks per node corresponds to the number of instances of the individual tools (BWA, GATK, etc.) that are being run concurrently on a machine. One scenario is to run only a single task and to use the multithreading functionality of the tools to make use of the available cores. An alternative scenario is to run multiple tasks in parallel on the same node, each task then using only a fraction of the available cores. Because of suboptimal multithreading scalability of certain individual tools, the choice in number of tasks can have a big impact on runtime. This is illustrated in Table 3.1 where the runtime is shown for three scenarios: (i) 1 task using 24 cores for multithreading; (ii) 4 tasks each using 6 cores for multithreading and (iii) 24 tasks without multithreading. The sequential runtime (single core) of the pipeline is $\sim$30.5h. When allowing the individual tools to run 24 threads on the same machine, the runtime reduces to $\sim$16.5h, resulting in a very low parallel efficiency of only 7.7%. This poor scaling can be observed in both map and reduce phase, but is especially pronounced in the reduce phase. It is caused partly by the lack of multithreading support in some of the tools used, e.g. BWA sampe and Picard. However, even the modules of GATK that do support multithreading exhibit poor scaling behavior. When moving from multithreading to multitasking as supported by Halvade, runtimes decrease significantly. Using 4 tasks with 6 threads each, runtime reduces to $\sim$4.5 h. When using 24 tasks without multithreading, a runtime of only 1h 42min is obtained, corresponding to a parallel efficiency of 74.7%. We observed an increased CPU utilization during pipeline execution when using 24 parallel tasks compared to using multithreading in 1 task.

On this type of node, optimal runtime is achieved when using a maximum number of tasks without multithreading. However, this is only possible because the node provides a sufficient amount of RAM. The memory bottleneck is caused by elPrep [14], which can be used as a drop-in replacement for Picard for data preprocessing. Even though elPrep has a significantly higher performance, elPrep instances require more memory, i.e., 16 GByte for elPrep versus 6 GByte for Picard. This means that running 24 parallel instances of elPrep requires 384 GByte RAM in total. For maximum performance, we recommend the use of elPrep; if insufficient memory is available, users can revert to Picard.

3.3.2 Influence of the presence of NUMA domains

Many recent systems make use of non-uniform memory access (NUMA) domains. Each NUMA domain contains a number of CPU cores and part of the RAM. Cores have faster access to memory that resides in the same NUMA domain (‘local’ access) and slower access to memory that is outside this domain (‘remote’ access). Files on disk that are accessed by a tool are typically buffered in memory by the Linux operating system in order to accelerate future accesses to the same file. If different processes are accessing the same file, this buffered copy of (part of) the file can be located in a different NUMA domain than that of the core accessing it, resulting in remote memory access. When such files are accessed frequently, this can result in degraded performance.

To avoid this, we make distinct copies of the reference file on disk, one copy per NUMA domain. The idea is that processes that belong to different NUMA domains access different copies of the file, while processes within the same NUMA domain access the same copy. This way, the original file contents will be buffered in each NUMA domain and all processes will have fast access to the locally buffered copy. We implemented this idea through the use of a wrapper script that makes the necessary copies of input files on local scratch space and then passes the correct input filename to each of the individual processes. The wrapper script relies on the ‘numactl’ utility to find out in which NUMA domain a certain process is running. Note that this approach increases the overall memory usage as the same data is buffered in memory multiple times.

Using 24 tasks on a single node and the entire NA12878 dataset, Fig. 3.2 shows the runtime of the different components (summed over all 24 tasks) of the pipeline with and without the use of the wrappers. For most components, the influence is only marginal, with the ScoreRecalibrator module from GATK being a notable exception. In that particular case, a reduction in runtime of 45% can be observed when using the wrapper script. The GATK ScoreRecalibration module relies intensively on the dbSNP database file (roughly 10 GByte) to generate recalibration tables. In this case, the improved NUMA data locality considerably improves runtime.

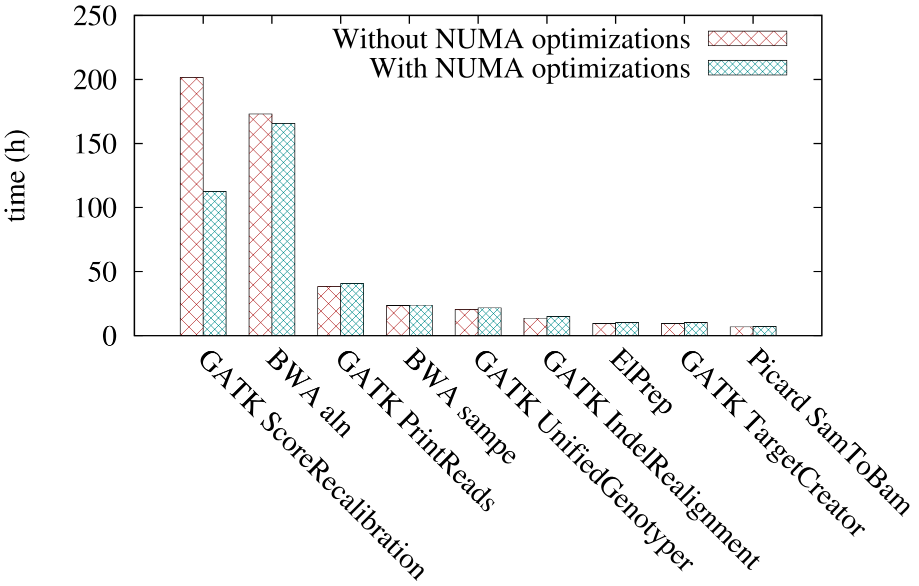


Figure 3.2: Comparison of the runtime (summed over all 24 parallel tasks) for each individual tool/module used in Halvade with and without optimized NUMA locality.

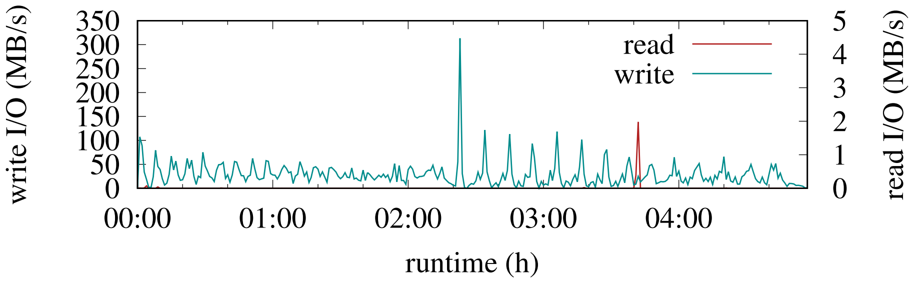


Figure 3.3: Disk I/O (scratch) observed on a worker node. Note that almost no data is actually being read from disk as data are still cached in memory.

3.4 Multi-node benchmarks

The benchmarks in this section were run on a big data cluster provided by Intel. This cluster consists of 15 nodes, each node contains a dual socket Intel E5-2695 @ 2.40GHz with 128 GByte of memory. This cluster was split in two, each containing 7 worker nodes. The first part had access to both a 200 GByte SSD and a 1 TByte HDD as intermediate storage and was used to assess the influence of the use of SSDs. The second part had access to a Lustre file system and a Hadoop setup with Intel's Hadoop Adapter for Lustre which was used to determine the influence of the used distributed file system.

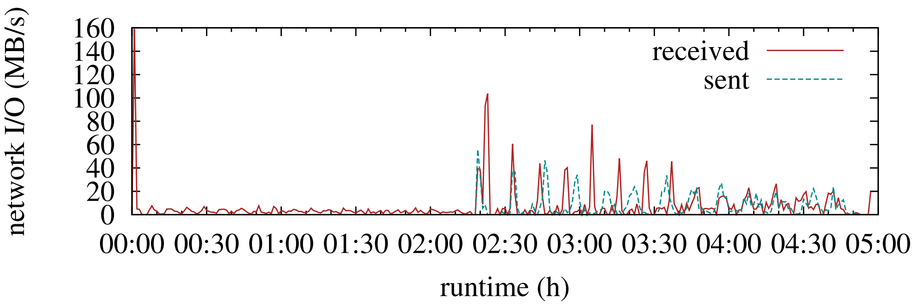


Figure 3.4: Network I/O observed on a single worker node.

3.4.1 Influence of the use of solid state disks

Many tools within Halvade rely on local disk I/O (scratch). This includes reading the reference genome and accessing the dbSNP database as well as writing and reading intermediate data generated by the different GATK modules as well as BWA-aln and BWA-sampe. We tested the performance difference between using solid state drives (SSD) and regular hard disk drives (HDD). Test results indicate only minimal differences in runtime. This is due to the relatively low overall disk usage during the execution of the Halvade job. The disk I/O volume was measured in intervals of one minute and converted to MB/s (see Fig. 3.3). With the exception of a peak during sorting phase, the disk I/O is well below 100 MB/s (averaged over one minute) which is well within the range of modern HDDs. During the entire job, volumes read from disk were very low, leading us to the conclusion that almost all data written to local disk was cached in memory by the operating system and again accessed from memory in the next step.

Due to their reduced latency, SSDs have an advantage over HDDs when lots of random access is required by an application. Even though during the execution of Halvade, tens of GBytes of intermediate key-value pairs are sorted on disk, performance is high even when using HDDs. This is due to the fact that Hadoop avoids random disk access as much as possible by buffering and sorting record chunks in memory and spilling them to disk only when they exceed a certain threshold.

3.4.2 Influence of the interconnection network

In between map and reduce phase, aligned reads are sorted according to genomic position. This parallel sorting step involves the movement of large volumes of data over the interconnection network. The network I/O volume was measured in intervals of one minute and again converted to MB/s (see Fig. 3.4). Again, as network I/O is below 100 MB/s, almost no performance benefit was observed by using an Infiniband interconnect over a 10 Gbit Ethernet network.

3.4.3 Influence of the file system

Traditionally, MapReduce relies on the Hadoop Distributed File System (HDFS) to read input and write final output data. In that case, data is stored on the local disks of the worker nodes in a distributed fashion. Alternatively, centralized file systems such as IBM's Generalized Parallel File System (GPFS) or the Intel Enterprise Edition for Lustre software can be used. In that case, data is stored on separate data nodes and transferred to the worker nodes through an interconnection network. As the pipeline is rather compute-intensive, all three systems were able to provide data to the worker nodes at a sufficiently high rate, hence almost no performance difference was observed. However, the use of Intel's Hadoop Adapter for Lustre included in Intel Enterprise Edition for Lustre software has two advantages. First, it decreases the time spent during the sort & shuffle phase compared with HDFS/GPFS. Second, Lustre uses less memory on the worker nodes. This can be important on nodes with limited memory capacity. For instance, on nodes equipped with 64 GByte of RAM running 4 Halvade tasks, we noticed that certain reduce tasks failed because of memory shortage. The cause of this is the difference in coverage over the different genomic regions and thus some tasks will have more reads to process. These reduce tasks had to be rescheduled causing an increase in runtime. On a 7-node cluster, the use of Intel's Hadoop Adapter for Lustre included in Intel Enterprise Edition for Lustre software decreased the runtime from 5h 27min (using HDFS) to 4h 48min on the same cluster.

3.5 Benchmark of NA12878 dataset on a 15-node cluster

Halvade was used to process the complete NA12878 dataset on a 15-node cluster, each node containing 24 CPU cores (dual-socket Intel E5-2680v3 @ 2.50GHz) with 512 GByte of RAM and three solid-state drives (SSD) of 400 GByte in RAID 0 to store intermediate data (local scratch). The nodes are interconnected through an FDR Infiniband network and access a GPFS storage through a second Infiniband network. Note that Lustre was not available on this cluster. Cloudera CDH 5.3 is deployed as a Hadoop distribution by HanythingOnDemand [17]. Halvade was configured to use 24 tasks per node, hence up to 360 tasks (24 tasks $\times$ 15 nodes) were run in parallel. NUMA optimizations were in place. On this cluster, Halvade completed read alignment and variant calling of the NA12878 dataset in 1h 31min. Compared to a single-threaded runtime of $\sim$12 days, this represents and overall speedup of a factor of $\sim$190 or a parallel efficiency of 53%.

We can now compare this result to previously reported results in [7]. A runtime of 2h 39min was reported on a comparable 15-node cluster (again with 360 CPU cores), however, in that case the nodes were equipped with only 64 GByte of memory. Therefore, it was optimal to run only 4 parallel tasks per node, i.e., 60 parallel tasks (4 tasks $\times$ 15 nodes) in total. As discussed before, the use of multithreading causing signifi-

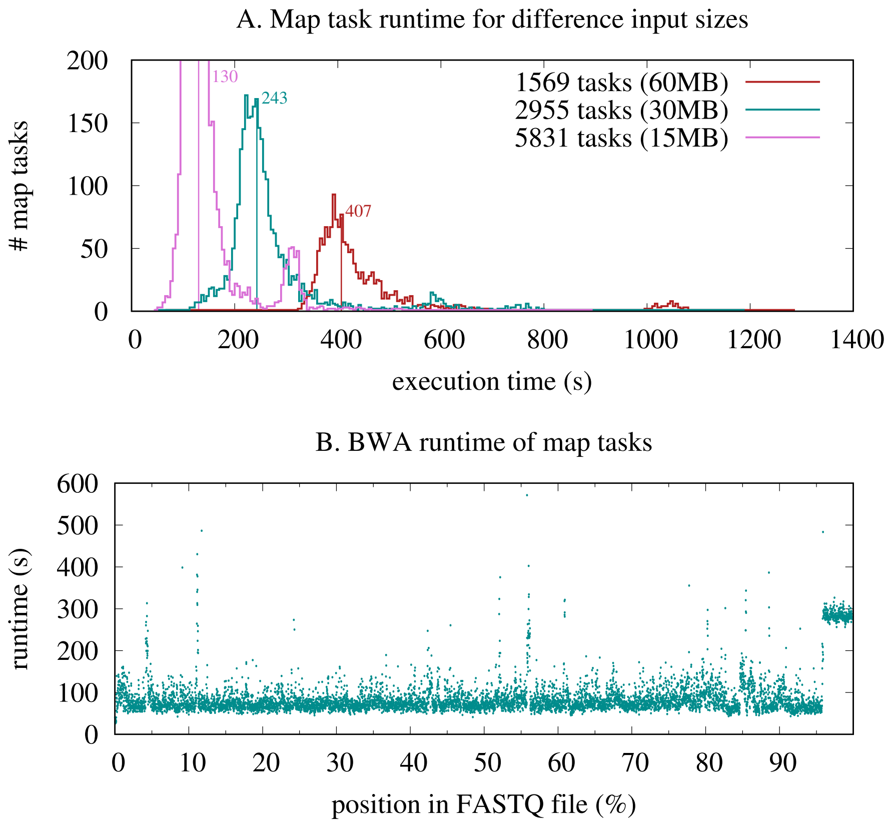


Figure 3.5: Panel A: distribution of runtime for map chunks of size 15, 30 and 60 MByte, respectively. Numbers indicate the median runtime. Panel B: runtime of map tasks as a of function of their position in the input FASTQ file.

cant loss of efficiency within each node, which largely accounts for the difference in runtime.

Unfortunately, the increased number of parallel tasks also increases the task scheduling overhead and makes it more difficult for the MapReduce framework to evenly distribute the workload among the different tasks. In the next sections, we investigate the underlying causes for load imbalance for the map and reduce phase respectively.

3.5.1 Load balance considerations: map phase

Aligning 1.5 billion input reads to the reference genome exhibits an abundance of parallelism, however, non-negligible load imbalance was observed during the map phase. We investigated whether the load balancing could be improved by generating smaller input chunks. Chunk sizes of 60 MByte, 30 MByte and 15 MByte result in a total of 1569, 2955 and 5831 input chunks respectively and lead to processing times (map phase only) of 2760 s, 2723 s and 2720 s, respectively. Even though a smaller chunk size effectively reduces idle time and thus improves load balancing, this effect is almost entirely neutralized by increased overhead, most notably the BWA instance initialization overhead.

Fig. 3.5A shows the distribution of execution times of map tasks for input chunk sizes of 60 MByte, 30 MByte and 15 MByte respectively. These sizes correspond to chunks containing respectively approximately 1 025 000, 525 000 and 275 000 reads. The execution time of most chunks follows a Gaussian-shaped distribution. This can be attributed to the well-known fact that the runtime to align individual reads exhibits large variability [18], depending on its sequence contents and quality scores. However, from Fig. 3.5A, it follows that the execution time of certain chunks is over three times the mean execution time. This phenomenon is consistently observed for all chunk sizes and cannot be attributed by the standard variation in read processing runtime.

Problematic input chunks appear to be spatially correlated to their position in the original input FASTQ file (see Fig. 3.5B). This is especially pronounced at the end of the FASTQ file. These chunks systematically contain more low-quality reads (see Fig. 3.6A and B), which take much longer to align. The fact that the input FASTQ file contains clusters of low-quality reads is likely caused by the fact that the NA12878 FASTQ file was generated from a sorted BAM file. Reads in the FASTQ file are then also sorted according to their aligned genomic position while unalignable reads occur at the end of the file. This explains the observed variation in Fig. 3.5B with hard-to-align chunks around distinct highly repetitive genomic regions and at the end of the FASTQ file. Simply shuffling the input FASTQ file prior to running Halvade significantly reduces task execution variability (see Fig. 3.6C). Note that shuffling FASTQ files that originate from BAM files is also recommended practice to prevent biasing the fragment insert size calculation during alignment [19].

3.5.2 Load balance considerations: reduce phase

Whereas the map phase suffers only from mild load imbalance, the reduce phase can exhibit a more severe imbalance. Again, even though genomic chunks are approximately equally sized, a large variability in task execution time can be observed (see Fig. 3.7A). In part, this variability can be explained by a different number of reads that map to a genomic chunk. It is well-known that biases during library preparation and sequencing result in an uneven distribution of reads across the genome: while certain regions are excessively sequenced, others lack coverage [20]. Fig. 3.7B shows the correlation between the number of reads that map to a genomic chunk and runtime. Clearly, while read coverage accounts for much of the runtime variability when using the GATK UnifiedGenotyper module to call variants, the more recent HaplotypeCaller module exhibits additional sources of runtime variability, making it difficult to balance the load. When using the HaplotypeCaller module, we observed that genomic chunks

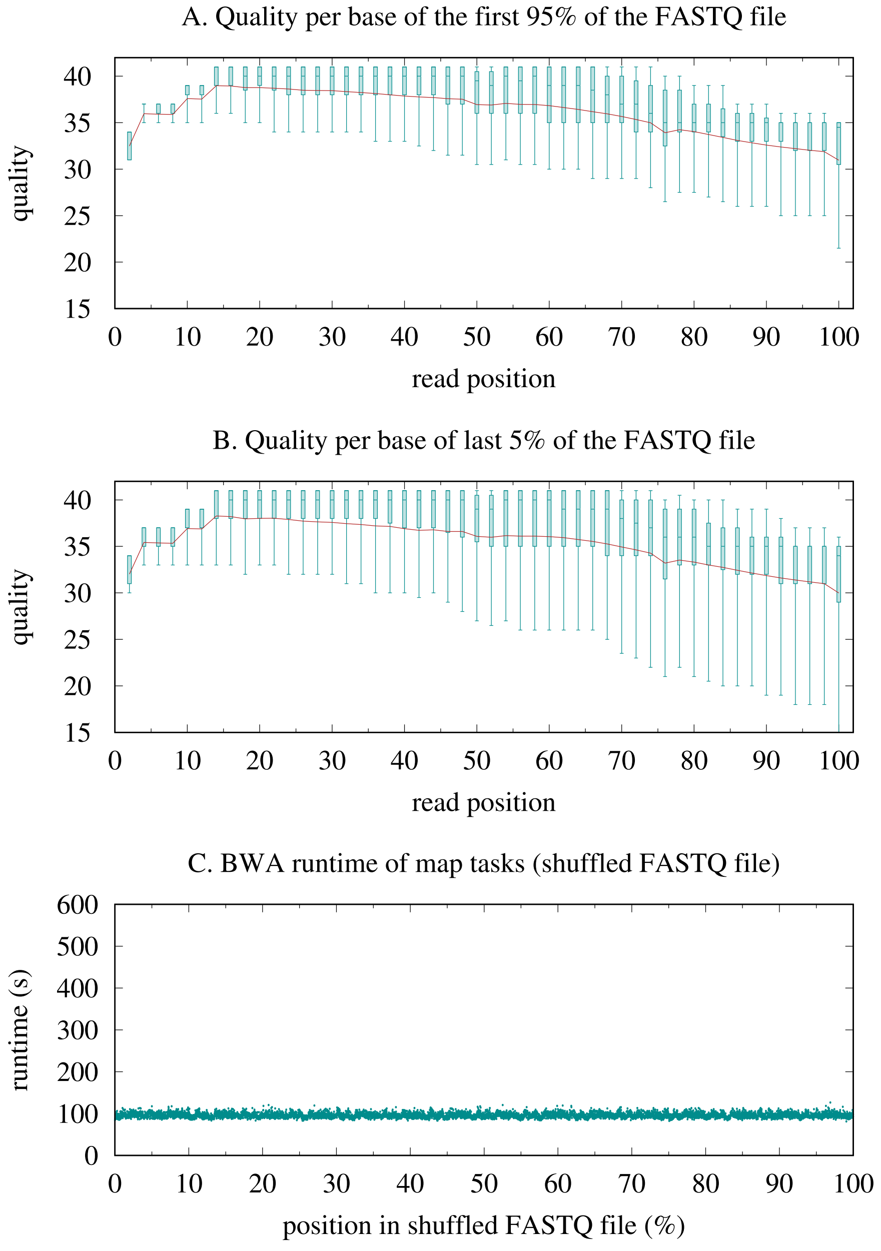


Figure 3.6: Panel A: average per-base quality scores in the first 95% of the FASTQ file. Panel B: per-base quality scores in the final 5% of the FASTQ file. Panel C: runtime of map tasks as a function of their position in the shuffled FASTQ file.

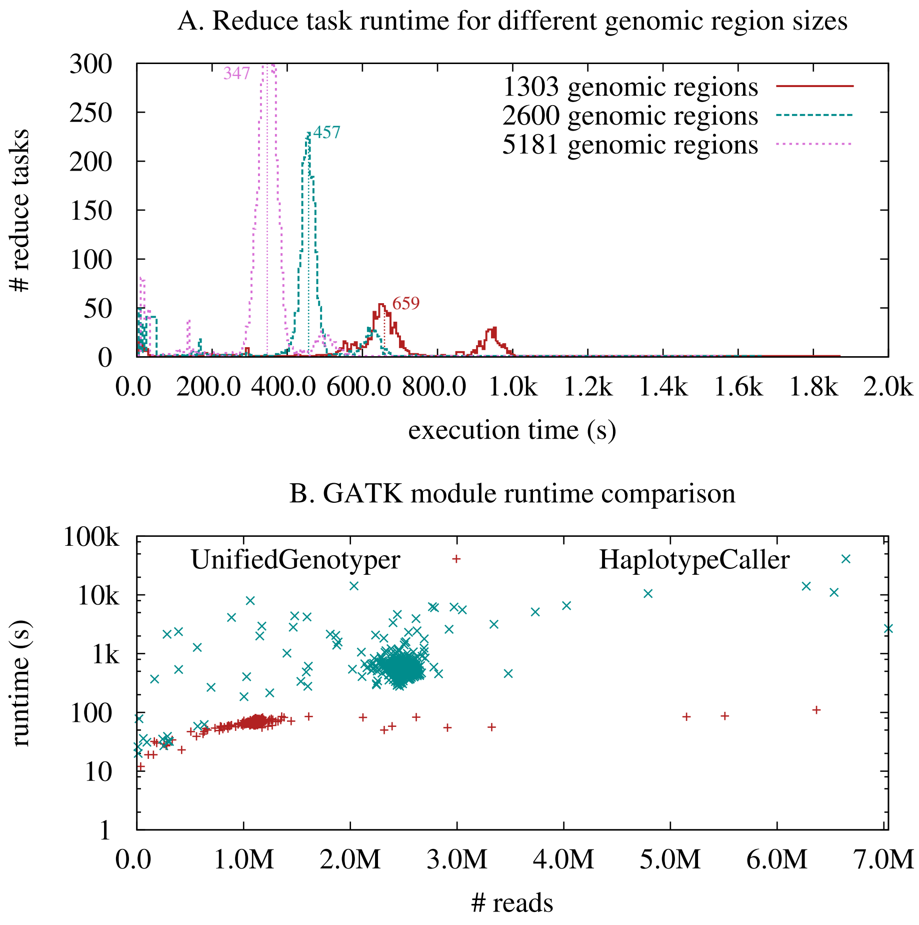


Figure 3.7: Panel A: distribution of runtime for reduce chunks for different genomic chunk sizes. Numbers indicate the median runtime. Panel B: reduce chunk runtime as a function of the number of aligned reads in the chunk for both the GATK UnifiedGenotyper and HaplotypeCaller modules.

with a highly non-uniform read coverage take significantly longer to process than genomic chunks with uniform read coverage, even though both chunks might have the same total number of aligned reads. In contrast, the runtime of the UnifiedGenotyper does not seem to be influenced by this, likely because the latter imposes a limit on the number of reads that is considered per position (default 250).

Again, we investigated whether smaller genomic regions improves load balancing. This effectively influences the number of reduce tasks as Halvade schedules one reduce task per genomic region. Unfortunately, reducing the size of the genomic regions does not result in a proportional decrease in runtime as is shown in Fig. 3.7A. With 1303 reduce tasks, the median task runtime is 659 s. Reducing the size of the genomic regions with a factor of four results in 5181 reduce tasks with a median task execution time of 347 s, i.e., more than half of the original median task runtime. Certain GATK modules have a fairly large overhead, regardless of the genomic region size that has to be processed. Additionally, an increased number of genomic regions also results in a higher number of interfaces between adjacent genomic regions. These interfaces require special care as (paired-end) reads that span such boundaries are duplicated in both regions. An increased number of interfaces hence results in more duplicated reads and hence, overhead. Finally, task scheduling and tracking overhead might also play a role.

Eventually, we obtained best performance when using only 647 genomic regions. This is slightly less than twice the number of parallel reduce tasks (360). The idea is that the biggest reduce tasks (in terms of number of aligned reads per genomic chunk) are executed first tasks as these are most likely to contain stragglers. When a reducer is finished with its first task, it can process a second, smaller task. We found that this way of working results in a fair load balance while keeping the GATK tool initialization overhead to a minimum. This improved load balancing strategy further reduced runtime with 10 minutes to 1h 21min or an overall parallel efficiency of 59%.

Fig. 3.8 shows the task execution history. A slight load imbalance can be observed in the map phase, however, this imbalance is largely compensated by the MapReduce framework by scheduling shuffle tasks that copy data between nodes as required for sorting. The simple load balancing strategy during the reduce phase can also be clearly observed: when the first reduce tasks finish processing their first genomic chunk they are allocated a second chunk. This causes a second wave of shuffle jobs to be triggered in order to move these data to the worker node that requires it. Even though this way of working does not result in a perfectly balanced load, we found that it resulted in the lowest overall runtime.

To confirm our findings we evaluated the parallel efficiency on two additional datasets. The first dataset is a whole genome DNA sequencing dataset (NA12877, human genome, Illumina Platinum Genomes). This dataset consists of 1.47 billion 100 bp paired-end reads stored in two compressed (gzip) FASTQ files. The single-threaded runtime on a 24-core machine (dual Intel E5-2680v3 @ 2.50GHz) with 512 GB of RAM is 10 days and 11 hours. With Halvade and all optimizations in place we

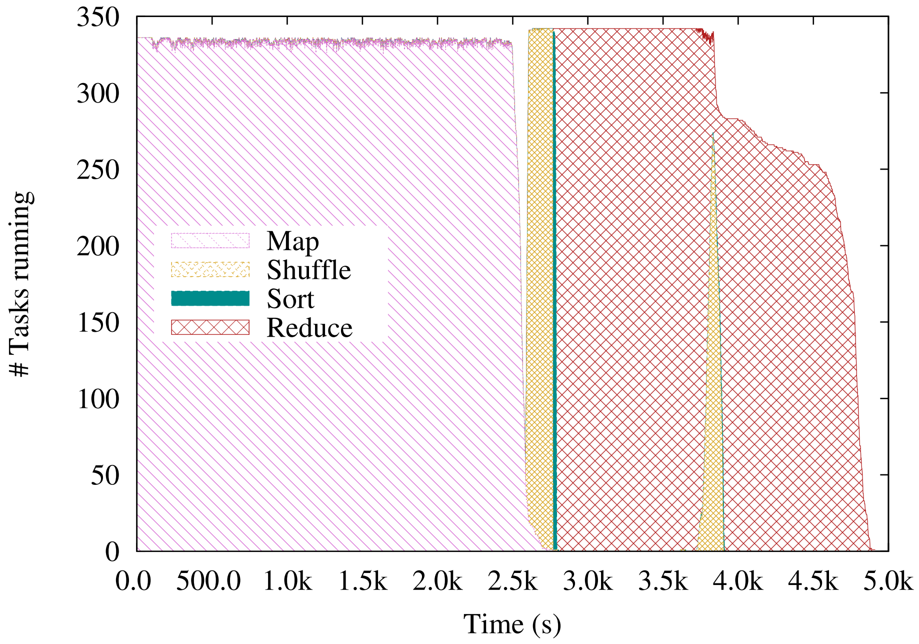


Figure 3.8: Execution of the NA12878 dataset on a 15-node cluster with improved load balancing. Each node runs 24 tasks in parallel. Note that certain tasks are used by MapReduce for task tracking and scheduling purposes.

achieve a runtime of 1h 11min on 15 nodes (360 processor cores), which again results in an overall parallel efficiency of 59%.

The third dataset originates from the same individual (NA12877) but has a higher sequencing depth of 200×, which amounts to approximately 5.97 billion 100 bp paired-end reads stored in 14 pairs of compressed (gzip) FASTQ files. For this dataset we *estimated* the single-threaded runtime, based on a linear extrapolation of the measured runtimes of each individual step on a subset of the reads. This leads to an estimated runtime of 43 days and 23 hours. This runtime is likely an underestimation of the actual runtime, as some components such as the sorting of SAM records have a super linear time complexity. On the same 15-node cluster with all optimizations in place, the runtime using Halvade is 5h 31min (measured on the entire dataset). This leads to an overall estimated parallel efficiency of 53%. This shows that even for very large datasets, Halvade can achieve very high efficiency.

3.6 Discussion and Conclusion

We investigated the impact of different hardware configurations on the runtime of Halvade, a parallel, multi-node framework that implements a variant calling pipeline according to the GATK Best Practices recommendations. Halvade relies on BWA for read mapping and GATK for variant calling.

Even though Halvade is primarily intended to allow for a multi-node parallelization of sequencing pipelines, Halvade can be used to significantly speed up post-sequencing analysis on a single node. This is because the overall parallel efficiency of the individual tools is very low: a speedup of approximately 2 is observed when moving from single-threaded execution to multithreaded execution on a 24-core machine. Part of this poor scaling behavior can be explained by the fact that BWA-sampe and Picard do not support multithreading, however, most of the GATK modules involved in the pipeline also do not exhibit good scaling behavior. By using Halvade on high-memory nodes, multithreading can be replaced by multitasking. The latter is far more efficient, which has also been shown in [21], and a speedup of $\sim$18 is obtained on a 24-core machine.

Additionally, having much memory in a system allows to hold a copy of buffered files in each of the NUMA domains. As such, CPU cores have access to a copy in the local NUMA domain, thus avoiding remote memory access. For the GATK ScoreRecalibrator module, this improves the runtime by nearly a factor of two.

Other hardware aspects, such as local disk speed (solid state drives vs. regular hard disk drives), speed of interconnection network (Infiniband vs. Ethernet networks) or file system (HDFS vs. GPFS) have only a minor influence on overall runtime. Even though a typical whole-genome dataset involves hundreds of GBytes of input data and a multiple thereof of intermediate data, the sequencing pipeline is mostly compute-intensive and hence, runtime is mostly influenced by the compute capacity of a node, rather than I/O speed.

Subsequently, Intel Enterprise Edition for Lustre software was investigated. The use of Intel's Hadoop Adapter for Lustre included in Intel Enterprise Edition for Lustre software simplifies the shuffle & sort which leads to better performance. Additionally, Lustre uses less memory which can be important when high-memory machines are not available.

Load balancing strategies for both map and reduce tasks were investigated. In both cases, we found significant variability in task runtime. For the read mapping, these could be primarily attributed to a FASTQ file that was generated from a sorted BAM file. Simply shuffling the input sequences resulted in a more uniform task execution time. For variant calling, most of the load imbalance is caused by a highly uneven distribution of reads across the genome. Even when choosing genomic chunk size based on the aligned number of reads, lots of runtime variability can be observed due to excessively highly covered regions within a chunk. Unfortunately, attempts to improve load balancing by choosing smaller task sizes are neutralized by increased overhead. A simple load balancing strategy with few tasks corresponding to large genomic regions appeared to yield best performance.

With all optimizations in place, Halvade is able to complete read alignment and variant calling of the complete NA12878 dataset in 1 hour and 21 minutes on a 15-node cluster, each node containing 24 CPU cores and 512 GByte of RAM. Compared to a single-threaded runtime of $\sim$12 days for this pipeline, this represents an overall speedup of a factor of $\sim$213 or a parallel efficiency of 59%.

Tools like BWA, SAMtools, Picard and GATK are widely adopted by the bioinformatics community, however, they were not designed for parallel execution. With the current status of multithreading performance in the available tools, it is best to configure Halvade to use as many tasks on a node as possible. In principle, a careful design of multithreaded software that is NUMA-aware and avoids false-sharing should lead to a single-node performance similar to that of Halvade. In fact, multithreading facilitates automated load balancing which is hard to achieve in a multi-process MapReduce settings.

It should be noted that highly parallel read mappers have been proposed that are based on UPC++ (e.g. the UPC++ implementation of CUSHAW3 [22] or merAligner [23]) or that are suitable for GPU (e.g. CUSHAW [24] or SOAP3 [25]) or Xeon Phi coprocessors (e.g. MICA [26]). Additionally, parallel variant calling pipelines have been proposed that do not rely on GATK. For example, SpeedSeq [27] relies on a custom implementation for variant calling with improved performance while ADAM [28] and Avocado [29] provide for a scalable variant calling framework natively implemented in Spark. With this in mind, it should be clear that newly proposed tools for read mapping and variant calling should not only be evaluated on their accuracy and single-core performance, but also on their scaling behavior.

References

[1] Van der Auwera, G. A., M. O. Carneiro, C. Hartl, R. Poplin, G. del Angel, et al., 2013. From FastQ Data to High-Confidence Variant Calls: The Genome Analysis Toolkit Best Practices Pipeline. *Current protocols in bioinformatics / editoral board, Andreas D. Baxevanis ... [et al.]*, 11(1110).

[2] Li, H. and R. Durbin, 2009. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics (Oxford, England)*, 25(14):1754–1760.

[3] Li, H., B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, et al., 2009. The Sequence Alignment/Map format and SAMtools. *Bioinformatics (Oxford, England)*, 25(16):2078–2079.

[4] Wysoker, A., 2009. Picard.

[5] McKenna, A., M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis, et al., 2010. The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. *Genome research*, 20(9):1297–1303.

[6] DePristo, M. A., E. Banks, R. Poplin, K. V. Garimella, J. R. Maguire, et al., 2011. A framework for variation discovery and genotyping using next-generation DNA sequencing data. *Nature genetics*, 43(5):491–498.

[7] Decap, D., J. Reumers, C. Herzeel, P. Costanza, and J. Fostier, 2015. Halvade: scalable sequence analysis with MapReduce. *Bioinformatics*, pages btv179+.

[8] Dean, J. and S. Ghemawat, 2008. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM*, 51(1):107–113.

[9] Abuín, J. M., J. C. Pichel, T. F. Pena, and J. Amigo, 2015. BigBWA: approaching the BurrowsWheeler aligner to Big Data technologies. *Bioinformatics*, pages btv506+.

[10] Al-Ars, Z. and H. Mushtaq, 2015. Scalability potential of BWA DNA mapping algorithm on apache spark. In *Proceedings of the 2nd Annual International Symposium on Information Management and Big Data - SIMBig 2015, Cusco, Peru, September 2-4, 2015.*, pages 85–88.

[11] Lam, H. Y. K., C. Pan, M. J. Clark, P. Lacroute, R. Chen, et al., 2012. Detecting and annotating genetic variations using the HugeSeq pipeline. *Nature Biotechnology*, 30(3):226–229.

[12] Puckelwartz, M. J., L. L. Pesce, V. Nelakuditi, L. Dellefave-Castillo, J. R. Golbus, et al., 2014. Supercomputing for the parallelization of whole genome analysis. *Bioinformatics*, 30(11):1508–1513.

[13] Kelly, B. J., J. R. Fitch, Y. Hu, D. J. Corsmeier, H. Zhong, et al., 2015. 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(1).

[14] Herzeel, C., P. Costanza, D. Decap, J. Fostier, and J. Reumers, 2015. elPrep: High-Performance Preparation of Sequence Alignment/Map Files for Variant Calling. *PLoS ONE*, 10(7):e0132868+.

[15] Sherry, S. T., M. H. Ward, M. Kholodov, J. Baker, L. Phan, et al., 2001. dbSNP: the NCBI database of genetic variation. *Nucleic acids research*, 29(1):308–311.

[16] Van der Auwera, G. A., 2013. Gatk resource bundle.

[17] Higgs, E., 2014. Hanything on demand.

[18] Herzeel, C., T. Ashby, P. Costanza, and W. De Meuter, 2013. Resolving Load Balancing Issues in BWA on NUMA Multicore Architectures. In *Parallel Processing and Applied Mathematics*, pages 227–236. Springer-Verlag.

[19] Van der Auwera, G. A., 2013. Revert a bam file to fastq format.

[20] Ross, M. G., C. Russ, M. Costello, A. Hollinger, N. J. Lennon, et al., 2013. Characterizing and measuring bias in sequence data. *Genome Biology*, 14(5):R51+.

[21] Kutlu, M. and G. Agrawal, 2014. PAGE: A Framework for Easy PArallelization of GEnomic Applications. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pages 72–81. IEEE.

[22] González-Domínguez, J., Y. Liu, and S. Bertil, 2016. Parallel and Scalable Short-Read Alignment on Multi-Core Clusters Using UPC++. *Plos ONE*, (6):1–15.

[23] Georganas, E., A. Bulu, J. Chapman, L. Oliker, D. Rokhsar, et al., 2015. meraligner: A fully parallel sequence aligner. In *IPDPS*, pages 561–570. IEEE Computer Society.

[24] Liu, Y., B. Schmidt, and D. L. Maskell, 2012. CUSHAW: a CUDA compatible short read aligner to large genomes based on the BurrowsWheeler transform. *Bioinformatics*, 28(14):1830–1837.

[25] Liu, C.-M., T. Wong, E. Wu, R. Luo, S.-M. Yiu, et al., 2012. SOAP3: ultra-fast GPU-based parallel alignment tool for short reads. *Bioinformatics*, 28(6):878–879.

[26] Chan, S.-H., J. Cheung, E. Wu, H. Wang, C.-M. Liu, et al., 2014. MICA: A fast short-read aligner that takes full advantage of Intel Many Integrated Core Architecture (MIC).

[27] Chiang, C., R. M. Layer, G. G. Faust, M. R. Lindberg, D. B. Rose, et al., 2015. SpeedSeq: ultra-fast personal genome analysis and interpretation. *Nat Meth*, 12(10):966–968.

[28] Massie, M., F. Nothaft, C. Hartl, C. Kozanitis, A. Schumacher, et al., 2013. Adam: Genomics formats and processing patterns for cloud scale computing. Technical Report UCB/EECS-2013-207, EECS Department, University of California, Berkeley.

[29] Nothaft, F., 2015. *Scalable Genome Resequencing with ADAM and avocado*. Master's thesis, EECS Department, University of California, Berkeley.

4 Halvade-RNA: parallel variant calling from transcriptomic data using MapReduce

Up until now we have discussed the DNA sequencing pipeline implemented in Halvade. In this chapter we introduce Halvade-RNA, which runs the RNA-seq pipeline in a similar manner.

⋆⋆⋆

Dries Decap, Joke Reumers, Charlotte Herzeel, Pascal Costanza and Jan Fostier.

Published in PLOS ONE 12(3). 2017. e0174575

Abstract Given the current cost-effectiveness of next-generation sequencing, the amount of DNA-seq and RNA-seq data generated is ever increasing. One of the primary objectives of NGS experiments is calling genetic variants. While highly accurate, most variant calling pipelines are not optimized to run efficiently on large data sets. However, as variant calling in genomic data has become common practice, several methods have been proposed to reduce runtime for DNA-seq analysis through the use of parallel computing. Determining the effectively expressed variants from transcriptomics (RNA-seq) data has only recently become possible, and as such does not yet benefit from efficiently parallelized workflows. We introduce Halvade-RNA, a parallel, multi-node RNA-seq variant calling pipeline based on the GATK Best Practices

recommendations. Halvade-RNA makes use of the MapReduce programming model to create and manage parallel data streams on which multiple instances of existing tools such as STAR and GATK operate concurrently. Whereas the single-threaded processing of a typical RNA-seq sample requires $\sim$28h, Halvade-RNA reduces this runtime to $\sim$2h using a small cluster with two 20-core machines. Even on a single, multi-core workstation, Halvade-RNA can significantly reduce runtime compared to using multithreading, thus providing for a more cost-effective processing of RNA-seq data. Halvade-RNA is written in Java and uses the Hadoop MapReduce 2.0 API. It supports a wide range of distributions of Hadoop, including Cloudera and Amazon EMR.

4.1 Introduction

Recently, a number of methods have been introduced to accelerate read mapping and variant calling through the use of parallel and distributed computing techniques: HugeSeq [1], MegaSeq [2], Churchill [3] and Halvade [4] implement a DNA-seq variant calling pipeline according to the Best Practices recommendations [5] for use with the GATK [6, 7] variant caller. These tools exploit the fact that read mapping is parallel by read, i.e., aligning one read is independent of the alignment of other reads, while variant calling is parallel by genomic region, i.e., variant calling in a certain genomic region is independent of variant calling in other regions. As such, the runtime to process whole genome or whole exome sequencing data sets is strongly reduced. Other parallel DNA-seq variant calling pipelines that do not rely on GATK include SpeedSeq [8] and ADAM [9].

Nowadays, RNA-seq datasets are becoming increasingly available. Even though primarily intended to identify transcripts and quantify expression, RNA-seq data can equally be used to call single nucleotide variants [10]. There are two main conceptual differences with DNA-seq based variant calling. First, the mapping step needs to be modified to avoid false positive variant calling at exon-exon junctions, by using a sample-specific genome index containing junction information. Second, as the assumptions on coverage depth and allelic balance used in DNA-seq based variant calling do not hold in RNA-seq, where coverage depth is dependent on transcript expression, and allele-specific expression can influence allelic balance, the variant caller should be adjusted accordingly. For that purpose, the GATK Best Practices recommendations have been adapted to involve two passes of STAR [11] for spliced read alignment, Picard (http://picard.sourceforge.net/) for data preprocessing and GATK for variant calling. Processing a typical RNA-seq sample using this pipeline on a single CPU core takes $\sim$28 hours. Enabling multithreading on a 20-core machine reduces runtime only by a factor of two, which indicates considerable loss of performance during execution.

Here we present Halvade-RNA, a parallel framework for variant calling from RNA-seq data that relies on the MapReduce programming model [12]. MapRe-

Table 4.1: RNA-seq variant calling pipeline used in this work.

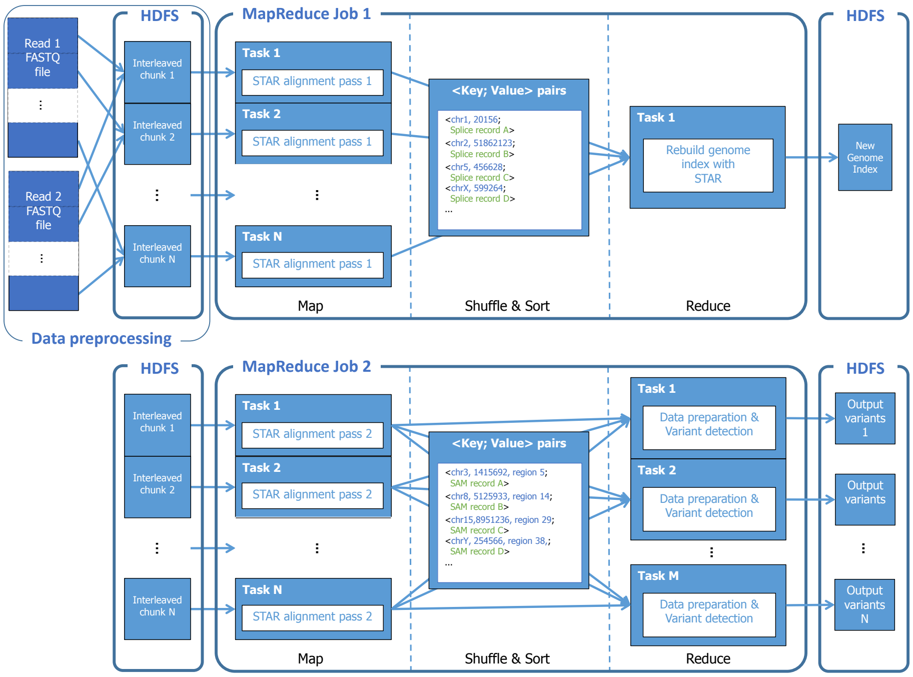
Step	Tool	Input	Output
Read mapping (1st pass)	STAR	FASTQ + index	SAM + splice junctions
Rebuild genome index	STAR	ref. genome + splice junctions	new index
Read mapping (2nd pass)	STAR	FASTQ + new index	SAM
Add read groups and sort	Picard	SAM	BAM
Mark duplicates	Picard	BAM	BAM
Split 'N' Trim	GATK	BAM	BAM
Indel realignment	GATK	BAM	BAM
Base quality score recalibration	GATK	BAM	BAM
Variant calling	GATK	BAM	VCF

duce has previously been used in bioinformatics for different applications [13–15]. Rather than relying on multithreading, Halvade-RNA runs several instances of the different tools involved (STAR, Picard, GATK) in parallel on subsets of the data, resulting in more efficient use of resources and thus lower cost of computing. In addition to reducing the runtime of a single RNA-seq sample, Halvade-RNA accelerates batch processing of multiple RNA-seq samples on any compute infrastructure on which Hadoop/MapReduce is installed, including public cloud platforms such as Amazon Web Services. To the best of our knowledge, this is the first framework to accelerate variant calling pipelines for RNA-seq data. The source is available at http://bioinformatics.intec.ugent.be/halvade under GPL license.

4.2 Materials and Methods

4.2.1 RNA-seq variant calling pipeline

In this work, we adopt the GATK RNA-seq variant calling pipeline as described in https://software.broadinstitute.org/gatk/guide/article?id=3891. Table 4.1 lists the different steps involved. In order to obtain accurate spliced read alignment, a two-step approach using the STAR aligner is used as first described in [16]. During the first pass, spliced alignment is performed without prior knowledge of splice sites. The Broad Institute chose to use an approach in which existing splice site information is not used in order to be independent of existing GTF or GFF files that are regularly updated. Although it is possible to use existing splice site information to improve the first pass, the Broad Institute did not validate this process, so we chose to stick with their suggestion. Identified splice junctions are then incorporated in a new genome index file which is subsequently used to guide the final alignments during the second pass. Next, Picard is used to add read group information, sort the aligned records ac-



Read 1 FASTQ file
Read 2 FASTQ file
Data preprocessing
HDFS
Interleaved chunk 1
Interleaved chunk 2
Interleaved chunk N
MapReduce Job 1
Task 1
STAR alignment pass 1
Task 2
Task N
<Key; Value> pairs
<chr1, 20156; Splice record A>
<chr2, 51862123; Splice record B>
<chr5, 456628; Splice record C>
<chrX, 599264; Splice record D>
Rebuild genome index with STAR
New Genome Index
Map
Shuffle & Sort
Reduce
MapReduce Job 2
STAR alignment pass 2
<chr3, 1415692, region 5; SAM record A>
<chr8, 5125933, region 14; SAM record B>
<chr15,8951236, region 29; SAM record C>
<chrY, 254566, region 38,; SAM record D>
Task M
Data preparation & Variant detection
Output variants 1
Output variants
Output variants N

Figure 4.1: Overview of the RNA-seq pipeline in Halvade-RNA. In the first job, reads are aligned in parallel in order to identify splice junctions and the reference genome index is rebuilt using this information. In the second job, final alignments are produced and after sorting and grouping the aligned reads by genomic region, the different Picard and GATK steps are executed in parallel.

cording to genomic position, mark read duplicates and convert the SAM file to binary BAM format. The GATK Split'N'Trim module is used to split reads into different exon segments and trim reads that overlap with intronic regions. Reads that contain short insertions or deletions (indels) are realigned to avoid false positive variant calls in later steps. Additionally, the per-base quality scores are recalibrated to accommodate certain batch artifacts. Finally, variants are called using the HaplotypeCaller and written to VCF file.

4.2.2 The RNA Pipeline in MapReduce

Because the RNA-seq variant calling pipeline involves two passes of the STAR aligner, it cannot be readily expressed in the Halvade MapReduce framework as implemented for DNA-seq variant calling [4]. Whereas the DNA-seq variant calling pipeline could be implemented using a single MapReduce job, two MapReduce jobs are required for Halvade-RNA. Fig. 4.1 provides an overview of the framework.

First, the input FASTQ files are interleaved (such that paired-end reads are adjacent to each other) and split into smaller file chunks. During the map phase of the first

MapReduce job, these input chunks are processed in parallel by multiple instances of the STAR aligner. To avoid loading the reference genome index from disk by each STAR instance individually, the genome index is first loaded in shared memory, after which all the STAR instances on this node can access this genome index from RAM. During this first alignment pass, the actual read alignments (i.e., the SAM records) are ignored and only splice junction information is retained. For each read that spans multiple exons, STAR produces a record containing junction information, such as genomic location and strand. The map tasks emit this information as intermediate <key, value> pairs, with the key holding tuples of integers containing the contig index and the position of the splice junction, and the value containing the STAR-generated string with splice junction information. When all map tasks are finished, all intermediate <key, value> pairs are sent to a single reducer and written to file. In the reduce task, this file is subsequently used by STAR to build a new genome index that incorporates this slice junction information. This is a purely sequential step, which should be kept as short as possible. We therefore configure STAR to build a sparse index, where only a fraction of the genomic locations are indexed. While using a sparse index results in a slightly higher runtime during the second alignment phase, the reduced runtime during index construction (5min for a typical RNA-seq sample, as opposed to approximately 30min for a dense index) ensures the lowest overall runtime.

The second MapReduce job is similar to the DNA-seq variant calling MapReduce implementation and we refer to [4] for more details. RNA-seq reads are again aligned in parallel during the map phase, using the newly constructed genome index. The map tasks emit <key, value> pairs where the value represents an actual SAM record and the key a composite structure that contains the genomic location to which the read aligns. In between map and reduce phases, the intermediate <key, value> pairs are sorted in parallel according to genomic location by the MapReduce framework in a highly efficient manner. This step replaces the sorting functionality otherwise achieved by Picard. The sorted SAM records are converted to BAM format using Hadoop-BAM [17] and partitioned according to a user-specified number of genomic regions.

During the reduce phase, remaining data preprocessing and variant calling steps are performed in parallel through the concurrent processing of multiple genomic regions. To achieve this, multiple instances of Picard and GATK are run in parallel, each instance operating on a distinct genomic region. These steps are similar for the DNA-seq variant calling pipeline. The only notable difference is the addition of the Split'N'Trim module. Called variants are written to VCF files, one VCF file per reduce task. Optionally, these partial VCF files can be merged into a single VCF file in a third, lightweight MapReduce job. It should be noted that Halvade-RNA can also be executed on existing BAM files. In that scenario, the first MapReduce job is skipped and the map phase of the second job is modified in order to partition the provided BAM file.

As RNA-seq analysis often involves the quantification of gene expression, Halvade-RNA provides the option to count the number of reads per exon. This is done by the FeatureCounts tool [18] and is run per reduce task, and thus in parallel per genomic region. Again, optionally, the counts per genomic region can be merged into a single file using an additional lightweight MapReduce job.

Finally we remark that the used tools are still often improved and features are added, so it is important to note that Halvade-RNA allows replacing the binaries of the tools with newer versions, assuming that the command line arguments remain the same. Similarly, new tools could easily be added by updating the source code and calling the new tools appropriately.

4.2.3 Benchmark setup

Halvade-RNA was benchmarked on 9 RNA-seq samples (SNU-1033, SNU-1041, SNU-1214, SNU-213, SNU-216, SNU-308, SNU-489, SNU-601, SNU-668) from the Cancer Cell Line Encyclopedia [19], each sample containing approximately 175 million 101 bp paired-end reads originating from a poly(A) selection experiment. The benchmarks were run using Halvade-RNA version 1.2.0, implemented using STAR version 2.4.0h1, Picard version 1.112, GATK version 3.4 (nightly-2015-05-12-gcdf54f8) and Java version 1.7.0, run on a Cloudera distribution based on Apache Hadoop (CDH) 5.0.0 with Hadoop version 2.3.0. Note that Halvade-RNA has been validated for compatibility with the more recent Hadoop version 2.6.0 (CDH version 5.10.0), Java version 1.8.0 and GATK version 3.7. We used a two-node cluster, each node containing 20 CPU cores (dual-socket Intel Xeon E5-2660 v3 @ 2.60GHz) and 128 GByte of RAM. The nodes were interconnected by an FDR Infiniband network. Halvade-RNA was configured to run 10 parallel instances of STAR per node (2 threads per instance) and 20 parallel instances (single-threaded) of Picard and GATK per node. This way, all available CPU cores are used.

Additionally, Halvade-RNA was compared with GNU parallel [20] for multi-sample processing of all 9 samples. Halvade-RNA was run with identical configuration as described above. GNU parallel was configured to run per node a single instance of STAR using all available CPU cores for multithreading, as STAR has very good multithreading capabilities but uses up to 40 GByte of RAM per instance. Because the BAM processing and variant calling steps achieve poor speedups with multithreading, GNU parallel was configured to run multiple parallel instances of GATK and Picard, each instance processing a different sample and using two threads (only for GATK).

4.3 Results & Discussion

4.3.1 Parallel Performance

Four cases were set up to assess the performance of Halvade-RNA: i) the original RNA-seq variant calling pipeline on a single CPU core, ii) the original RNA-seq

Table 4.2: Benchmarks of the RNA-seq variant calling pipeline per sample.

	Runtime for sample (speedup)	
	Classical	
Sample	1 node × single core	1 node × 20 cores
SNU-1033	26h 6min (n/a)	12h 53min (2.03×)
SNU-1041	27h 48min (n/a)	12h 51min (2.16×)
SNU-1214	34h 40min (n/a)	13h 38min (2.54×)
SNU-213	27h 11min (n/a)	12h 59min (2.09×)
SNU-216	27h 21min (n/a)	13h 1min (2.10×)
SNU-308	27h 48min (n/a)	13h 25min (2.07×)
SNU-489	27h 10min (n/a)	12h 30min (2.17×)
SNU-601	26h 48min (n/a)	12h 59min (2.07×)
SNU-668	25h 59min (n/a)	12h 7min (2.14×)
average	27h 52min (n/a)	12h 56min (2.16×)
	Halvade-RNA pipeline	
Sample	1 node × 20 cores	2 nodes × 20 cores
SNU-1033	3h 25min (7.65×)	2h 44min (9.56×)
SNU-1041	3h 3min (9.09×)	1h 48min (15.46×)
SNU-1214	3h 33min (9.77×)	2h 23min (14.56×)
SNU-213	2h 50min (9.61×)	1h 44min (15.66×)
SNU-216	2h 46min (9.87×)	1h 44min (15.81×)
SNU-308	2h 48min (9.95×)	1h 43min (16.26×)
SNU-489	3h 8min (8.69×)	2h 16min (11.98×)
SNU-601	2h 59min (9.01×)	2h 5min (12.91×)
SNU-668	2h 49min (9.24×)	1h 52min (13.97×)
average	3h 2min (9.18×)	2h 2min (13.72×)

variant calling pipeline on 20 CPU cores, with multithreading enabled in both STAR and GATK, iii) the Halvade-RNA pipeline on the same 20-core machine, and iv) the Halvade-RNA pipeline on two 20-core machines (Table 4.2). For the original pipeline, the obtained speedup using multithreading is only 2.16 on average (min: 2.03, max: 2.54). In contrast, Halvade-RNA shows a speedup of 9.18 on average (min: 7.65, max: 9.95) on the same node, indicating that Halvade-RNA is on average 4.25 times faster when identical compute resources are used. Halvade-RNA relies primarily on multitasking rather than multithreading, and measuring the average runtimes and speedups per phase of the pipeline (Table 4.3) shows that especially for the variant calling steps this proves to be more efficient. As such, Halvade-RNA not only reduces analysis time but also substantially reduces the financial cost for computing. Using two nodes, the average parallel speedup increases to 13.72 (min: 9.56, max: 16.26).

Note that similar results were obtained on a public cloud platform (Amazon EMR). Using a single node of the type r3.8xlarge, we obtain an average execution time of 3h 29min per sample (min: 3h 4min, max: 4h 28min). Using two nodes, the average runtime decreases to 2h 32min per sample (min: 2h 1min, max: 3h 46min). The

Table 4.3: Average runtimes per phase of the RNA-seq pipeline.

		Runtime (speedup) per phase	
Pipeline	No. of nodes and cores	Pass 1 map	Rebuild genome
Classical	1 node × single core	1h 19min (n/a)	4min (n/a)
	1 node × 20 cores	6min (14.24×)	2min (2.18×)
Halvade-RNA	1 nodes × 20 cores	14min (5.69×)	4min (1.02×)
	2 nodes × 20 cores	8min (9.29×)	4min (1.01×)
Pipeline	No. of nodes and cores	Pass 2 map	Variant calling steps
Classical	1 node × single core	3h 29min (n/a)	23h 1min (n/a)
	1 node × 20 cores	22min (9.69×)	12h 27min (1.85×)
Halvade-RNA	1 nodes × 20 cores	39min (5.29×)	2h 3min (11.21×)
	2 nodes × 20 cores	22min (9.49×)	1h 26min (16.04×)

Amazon nodes have only 16 CPU cores and an additional data transfer from central S3 storage to the local worker nodes storage is required which explains the slightly higher runtimes. The average financial cost per sample was 14.15 US dollar when using a single node and 20.48 US dollar when using two nodes (pricing of December 2016).

4.3.2 Multi-sample Throughput

Given the large variability in gene expression within one RNA-seq sample, certain genomic chunks can have significantly more aligned reads. For example, in our experiment, coverage depth between genomic chunks could vary as much as thousand fold. As a consequence, there is large variability in execution time during the reduce phases, making it more difficult to balance load than in the Halvade-DNA framework, as coverage depth is more uniform in DNA-seq data. Fig. 4.2 shows the distribution of the runtimes of the MapReduce tasks for sample SNU-668, other samples have similar distributions. The map phases of both MapReduce jobs show a clear peak, with the slight shift for the second map phase resulting from the use of a sparse genome index. The variant calling reduce tasks show a very wide range of runtimes, ranging from about a minute for the fastest task up to one hour for the slowest task. Furthermore, rebuilding the STAR genome (a sequential step) is a second source of losing computing resources.

In a realistic scenario, the available compute infrastructure will be used to process multiple samples. In that case, MapReduce jobs can be overlapped by using idle slots to start processing the next sample, even though the current sample is not yet fully completed. This way of working minimizes idle time and thus increases throughput. Table 4.4 lists the total runtimes for processing all 9 samples. As a reference to calculate the overall speedup, we use the sum of all runtimes on a single-threaded pipeline, which is ∼251 hours. The multi-sample batch processing script (GNU parallel) for the original pipeline processes the 9 samples in 40h 7min with 20 CPU cores on a sin-

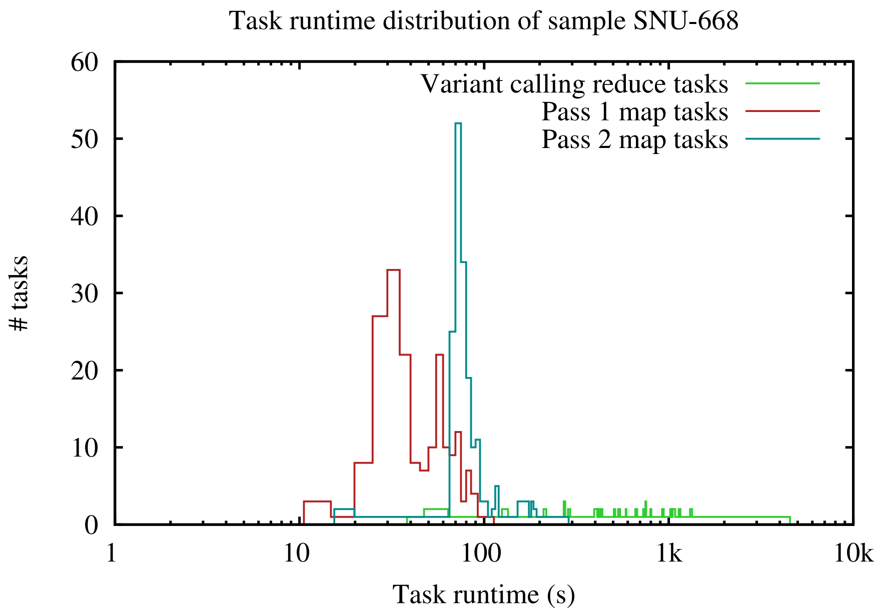


Figure 4.2: Runtime distribution of map and reduce tasks of both jobs. Note that the average runtime of the pass 2 map phase increases slightly due to the sparse index. Also note that the reduce phase of the first job is not displayed as this comprises only a single job.

Table 4.4: Runtime for the batch processing of all 9 RNA-seq samples.

Pipeline	No. of nodes and cores	Sum of per-sample runtime (speedup)
Classical	1 node × single core	250h 52min (n/a)
	1 node × 20 cores	116h 24min (2.16×)
	2 node × 20 cores	63h 21m (3.96×)
Halvade-RNA	1 nodes × 20 cores	27h 20min (9.18×)
	2 nodes × 20 cores	18h 17min (13.72×)

Pipeline	No. of nodes and cores	Batch processing runtime (speedup)
Classical	1 node × single core	250h 52min (n/a)
	1 node × 20 cores	40h 7min (6.25×)
	2 node × 20 cores	21h 3min (11.92×)
Halvade-RNA	1 nodes × 20 cores	22h 17min (11.26×)
	2 nodes × 20 cores	11h 48min(21.26×)

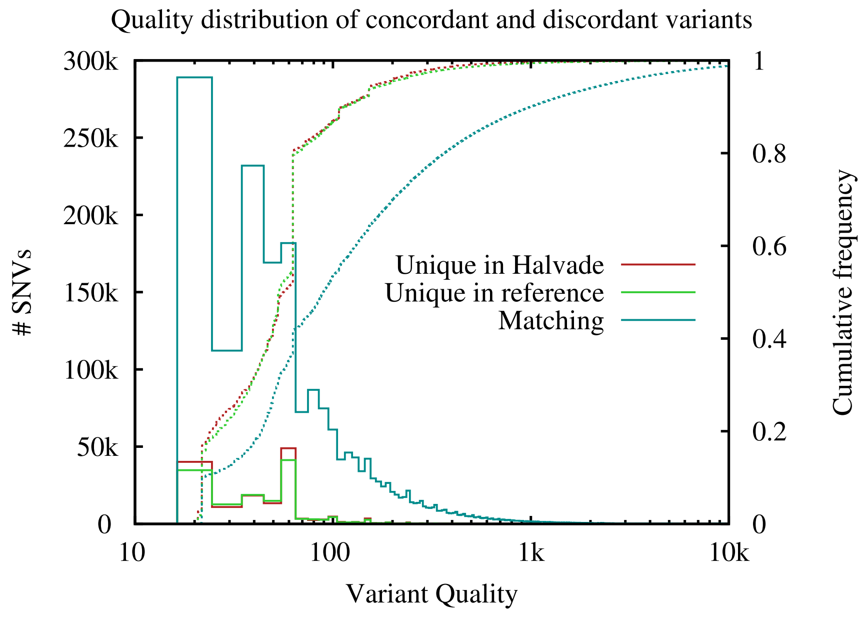


Figure 4.3: Comparison of variant Quality between Halvade and the reference pipeline. Shows the quality distribution of all variants taken from all 9 samples.

gle node. Using two nodes, we distribute the samples over the two nodes and run the sequential pipeline and the batch script for 4 and 5 samples per node (adjusted to run with 24GB and 4 cores per sample in the GNU parallel steps), resulting in a total runtime of 21h 3min. Using Halvade-RNA in batch mode, a total runtime of 22h 17min is obtained using a single node and a total runtime of 11h 48min on two nodes. This is respectively ∼4h and ∼6.5h faster compared to the sum of the per-sample runtimes. Clearly, the ability to avoid idle time again significantly increases the efficient use of compute resources.

4.3.3 Quality Assessment

Finally, we show the per-sample concordance in the variants called by the original sequential pipeline and Halvade-RNA (Table 4.5). On average, 93.8% of the variants found by the sequential pipeline are also called by Halvade-RNA. Variants that are called in either only the sequential pipeline or only the Halvade-RNA pipeline are supported by fewer reads and have a ∼8-fold lower average quality score compared with the overlapping variants, and are thus more likely to be filtered from a high-quality variant list. The normalized distribution of the variant quality for each of the three subsets, matching variants and variants unique to either Halvade or the original pipeline, is shown in Fig. 4.3. The origin of these discordant variants lies in the vari-

Table 4.5: Per sample overlap and average quality score.

Sample	Overlapping variants(%)	Avg. qual score overlapping variants	Avg. qual score Halvade-unique variants	Avg. qual score reference-unique variants
SNU-1033	93.9	651.6	80.3	92.4
SNU-1041	93.4	803.2	85.6	92.7
SNU-1214	93.4	741.3	82.6	92.7
SNU-213	94.3	612.7	74.0	87.2
SNU-216	94.2	660.8	83.7	94.1
SNU-308	93.4	522.7	71.5	71.8
SNU-489	94.4	773.4	81.9	98.3
SNU-601	93.3	742.1	94.4	116.4
SNU-668	93.9	671.0	73.9	88.0

Average per-sample variant quality score (QUAL) for i) variants called by both the single-threaded pipeline and Halvade-RNA on 2 nodes $\times$ 20 cores, ii) variants called only by Halvade-RNA ('Halvade-unique'), iii) variants called only by the single-threaded pipeline ('reference-unique').

ability during the read mapping step. Typically these variants are located in regions that are part of repeating patterns, causing the reads to align to multiple locations. Either in the parallelized or in the original sequential pipeline, variants residing in these regions have a high probability of being false positive calls.

4.4 Conclusion

We have implemented a parallelized variant calling pipeline for RNA-seq data using a MapReduce approach, and compared the efficiency and accuracy of the pipeline to the original sequential implementation. Running the original pipeline using a single core requires on average 27.9 hours per sample. When enabling multithreading on 20 CPU cores in STAR and GATK, the average runtime per sample decreases to 12.9 hours, largely due to the poor scaling behavior of GATK. In contrast, on the same node, when using Halvade-RNA configured to run 10 parallel instances of STAR (2 threads per instance) and 20 parallel instances (single threaded) of Picard and GATK, average runtime decreases to $\sim$3 hours, corresponding to a parallel speedup of 9.18 over sequential execution of the pipeline. Clearly, the use of Halvade on a single node strongly reduces average runtime and results in a more cost-effective use of the compute resources. On two nodes, the average runtime further decreases to $\sim$2 hours.

Obtaining a good load balance across parallel tasks is challenging for RNA-seq data, given the large variability in gene expression –and thus coverage depth– across genomic regions. To minimize idle time introduced by slower jobs, Halvade-RNA can be operated in batch mode. In that case, idle slots can be used to start processing the next sample even though the current sample is not yet fully completed. Processing all of the 9 samples in batch mode on the two-node cluster yields a total runtime of 11.8 hours. In comparison, running the pipeline in batch mode gives a runtime of 21h on a two-node cluster. Even though batch mode does not decrease the per-sample processing time below 2 hours, it considerably increases the overall throughput through a more efficient use of compute infrastructure.

On average, variants identified by Halvade-RNA and the sequential pipeline have a 93.8% overlap. The variability is almost exclusively caused by variability during read mapping. Variants called by either only Halvade-RNA or the sequential pipeline have a much lower number of supporting reads and hence correspond to low-confidence variants.

References

[1] Lam, H. Y. K., C. Pan, M. J. Clark, P. Lacroute, R. Chen, et al., 2012. Detecting and annotating genetic variations using the HugeSeq pipeline. *Nature Biotechnology*, 30(3):226–229.

[2] Puckelwartz, M. J., L. L. Pesce, V. Nelakuditi, L. Dellefave-Castillo, J. R. Golbus, et al., 2014. Supercomputing for the parallelization of whole genome analysis. *Bioinformatics*, 30(11):1508–1513.

[3] Kelly, B. J., J. R. Fitch, Y. Hu, D. J. Corsmeier, H. Zhong, et al., 2015. 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(1).

[4] Decap, D., J. Reumers, C. Herzeel, P. Costanza, and J. Fostier, 2015. Halvade: scalable sequence analysis with MapReduce. *Bioinformatics*, pages btv179+.

[5] Van der Auwera, G. A., M. O. Carneiro, C. Hartl, R. Poplin, G. del Angel, et al., 2013. From FastQ Data to High-Confidence Variant Calls: The Genome Analysis Toolkit Best Practices Pipeline. *Current protocols in bioinformatics / editoral board, Andreas D. Baxevanis ... [et al.]*, 11(1110).

[6] McKenna, A., M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis, et al., 2010. The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. *Genome research*, 20(9):1297–1303.

[7] DePristo, M. A., E. Banks, R. Poplin, K. V. Garimella, J. R. Maguire, et al., 2011. A framework for variation discovery and genotyping using next-generation DNA sequencing data. *Nature genetics*, 43(5):491–498.

[8] Chiang, C., R. M. Layer, G. G. Faust, M. R. Lindberg, D. B. Rose, et al., 2015. SpeedSeq: ultra-fast personal genome analysis and interpretation. *Nat Meth*, 12(10):966–968.

[9] Massie, M., F. Nothaft, C. Hartl, C. Kozanitis, A. Schumacher, et al., 2013. Adam: Genomics formats and processing patterns for cloud scale computing. Technical Report UCB/EECS-2013-207, EECS Department, University of California, Berkeley.

[10] Piskol, R., G. Ramaswami, and J. B. B. Li, 2013. Reliable identification of genomic variants from RNA-seq data. *American journal of human genetics*, 93(4):641–651.

[11] Dobin, A., C. A. Davis, F. Schlesinger, J. Drenkow, C. Zaleski, et al., 2012. STAR: ultrafast universal RNA-seq aligner. *Bioinformatics*, 29(1):bts635–21.

[12] Dean, J. and S. Ghemawat, 2008. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM*, 51(1):107–113.

[13] Schatz, M. C., 2009. CloudBurst: highly sensitive read mapping with MapReduce. *Bioinformatics*, 25:1363–1369.

[14] Zou, Q., X.-B. Li, W.-R. Jiang, Z.-Y. Lin, G.-L. Li, et al., 2013. Survey of MapReduce frame operation in bioinformatics. *Briefings in Bioinformatics*, 15(4):637–647.

[15] Zou, Q., Q. Hu, M. Guo, and G. Wang, 2015. Halign: Fast multiple similar dna/rna sequence alignment based on the centre star strategy. *Bioinformatics*, 31(15):2475–2481.

[16] Engström, P. G., T. Steijger, B. Sipos, G. R. Grant, A. Kahles, et al., 2013. Systematic evaluation of spliced alignment programs for RNA-seq data. *Nature Methods*, 10(12):1185–1191.

[17] Niemenmaa, M., A. Kallio, A. Schumacher, P. Klemel, E. Korpelainen, et al., 2012. Hadoop-BAM: Directly manipulating next generation sequencing data in the cloud. *Bioinformatics*, 28:876–877.

[18] Liao, Y., G. K. Smyth, and W. Shi, 2014. featureCounts: an efficient general purpose program for assigning sequence reads to genomic features. *Bioinformatics (Oxford, England)*, 30(7):923–930.

[19] Barretina, J., G. Caponigro, N. Stransky, K. Venkatesan, A. a. Margolin, et al., 2012. The Cancer Cell Line Encyclopedia enables predictive modelling of anticancer drug sensitivity. *Nature*, 483(7391):603–307.

[20] Tange, O., 2011. Gnu parallel - the command-line power tool. *;login: The USENIX Magazine*, 36(1):42–47.

5
WGS, WES and RNA variant comparison and analysis

In this chapter we introduce a scalable, cloud-based framework for variant (VCF) analysis. Using three samples for which RNA-seq, WES and WGS are available, we perform a comparative analysis of the variants called from these three sequencing technologies.

⋆⋆⋆

Abstract As sequencing of genomes and transcriptomes has become a commodity in clinical and genetics research, next-generation sequencing datasets are becoming increasingly larger. While analysis methods have matured along with the sequencing technology, there are still a few obstacles before the analysis of these data becomes a commodity as well. Firstly, read alignment and variant calling is very compute-intensive. Recently, several methods have been proposed to significantly reduce runtime for DNA-seq analysis through the use of parallel computing. The end point of most analysis pipelines is the variant calling stage, usually in the VCF file format. Clinical research questions need to be addressed by further analysis of these files, and as these are still in the range of GBytes, there is a need for a scalable framework for analyzing, comparing and visualizing large sets of VCF files from different omics experiments in an interactive manner.

We have previously introduced a modular, scalable DNA-seq and RNA-seq variant calling pipeline [1]. Halvade is built on top of MapReduce and is able to significantly reduce runtime for variant calling on multi-core and/or multi-node compute infras-

tructures. We demonstrate how resulting VCF files can be analyzed in a scalable, interactive, visual and reproducible manner by making use of Spark Notebooks. This approach was applied to a set of three cell line datasets from the Cancer Cell Line Encyclopedia (CCLE), for which matched whole-genome, whole-exome and RNA-seq data are available. Using the VCFs generated with Halvade-DNA and Halvade-RNA, we demonstrate that the identified variants called according to the Broad's Best Practices are highly concordant, and that discordant calls between DNA and RNA sequencing experiments can largely be attributed to RNA-editing and allele-specific gene expression.

5.1 Introduction

Rapidly declining costs result in an increasing number of applications for next-generation sequencing. Currently, it is possible to sequence the entire human genome for under 1000 USD. In turn, this results in a rapidly increasing volume of raw genomic data that has to be stored and analyzed. Post-sequencing analysis often involves aligning the reads to the reference genome ('read mapping') followed by the identification of differences between the reference genome and the sequenced individual ('variant calling'). These computational steps are very time-consuming: whereas the actual sequencing can be completed in one or two days, the computational steps can take more than two weeks, if executed sequentially (single core).

Recently, a number of methods have been introduced to accelerate the post-sequencing analysis phase through the use of parallel and distributed computing. Often, these tools rely on BWA [2] for read mapping and GATK [3] for variant calling. These tools exploit the fact that read mapping is parallel by read, i.e., aligning one read is independent of the alignment of other reads while variant calling is parallel by genomic region, i.e., variant calling in a certain genomic region is independent of variant calling in other regions. For example, BigBWA [4] leverages Hadoop MapReduce [5] to accelerate read mapping using BWA, but has no functionality for variant calling. A complete variant calling pipeline is implemented in both HugeSeq [6] and MegaSeq [7], however, parallelism during variant calling is limited to concurrent processing of entire chromosomes. Churchill [8] and Halvade [1] are both able to achieve a high degree of parallelism by splitting chromosomes into smaller regions that are independently processed. As such, both tools are able to process the NA12878 dataset (human genome, 50X coverage, 1.5 billion 100 bp reads) in less than two hours on a modest cluster. However, the tools different in the targeted platform: whereas Churchill relies on a combination of Bash and Python scripts to distribute the workload over the worker nodes using either the Sun Grid Engine (SGE) or Portable Batch System (PBS), Halvade is cloud-based by adopting the MapReduce programming model.

Most tools implement a DNA-seq variant calling pipeline according to the GATK Best Practices recommendations [9] for both whole-genome and whole-exome sequencing. However, RNA-seq datasets are becoming increasingly available (EN-

CODE, CCLE, 1000 genomes, etc.). Even though primarily intended to identify transcripts and quantify expression, RNA-seq data can equally be used to call genomic variants. Recently, a number of studies have been published in which variants obtained through an RNA-seq datasets are compared to variants called using either whole-genome sequencing [10, 11] or whole-exome sequencing. As is the case for DNA sequencing, the Broad Institute has published Best Practices recommendations for RNA sequencing as well [1]. In essence, the two pipelines differ in the fact that instead of relying on BWA for read mapping, the STAR aligner [12] is used to map RNA-seq reads to a reference genome. As part of this work, we adopted Halvade to accelerate a variant calling pipeline from RNA-seq data as well. Even though RNA-seq datasets are typically smaller than whole-genome DNA-seq datasets, the sequential processing (single core) of a typical RNA-seq sample takes more than a day. Enabling multithreading a 20-core machine (dual socket Intel Xeon E5-2660 v3 @ 2.60 Ghz) reduces runtime only by half, indicative a poor scaling behavior, especially in the GATK steps. Rather than relying on multithreading, Halvade-RNA [13] uses multiprocessing and data parallelization, starting several instances of the different tools (STAR, GATK, etc.) on subsets of a data to significantly reduce runtime. Additionally, Halvade-RNA can make use of several nodes in a compute cluster to further reduce runtime, if needed. To the best of our knowledge, Halvade-RNA is the first framework to accelerate RNA-seq pipelines. Halvade and Halvade-RNA accelerate batch processing of whole-genome, whole-exome and RNA-seq samples on any compute infrastructure on which Hadoop/MapReduce is installed, including publicly available cloud platforms such as Amazon.

The bulk of the computational burden lies in the steps that transform raw sequencing reads to genomic variants. During those steps, huge reductions in data volume are achieved: hundreds of GBytes of sequencing data to a few GBytes for the derived VCF files. However, an interactive, quasi real-time analysis and visualization of these variant files, often containing millions of variants, is not feasible on a simple desktop machine, especially when many files are involved. Currently, analysis, manipulation and visualization of VCF files is usually done through a number of utilities, operated from command-line, to filter, compare and visualize variants under study. As runtimes of the individual tools are non-negligible, they do not provide a productive, responsive or interactive working environment. This warrants the development of a scalable platform for the analysis, manipulation and visualization of resulting VCF files.

In this chapter, we adopt the Spark platform and Notebook technology to analyze and compare VCF files obtained from different omics experiments. Like MapReduce, the Spark platform enables distributed processing of large datasets. However, whereas MapReduce implements a single, three-stage programming model (map-sort-reduce), Spark implements a much wider range of operations to filter and manipulate large datasets in parallel. A second big difference is that while MapReduce relies on disk I/O to store intermediate data, Spark provides the possibility to cache data in memory.

[1]The pipeline is described at https://software.broadinstitute.org/gatk/guide/article?id=3891.

This feature is specifically beneficial when the same data is queried multiple times. Additionally, the use of a Notebook enables users to control the Spark cluster in an interactive manner. In contrast to well-defined, static workflows, such as the variant calling pipelines implemented in Halvade, researchers interested in the analysis of VCF files benefit a lot from an interactive and visual environment, unimpeded by scalability issues that may arise from dealing with large datasets. Notebooks run in a web browser and allow users to manipulate data (filtering, set operations, visualization, etc.) by issuing high-level commands. These commands are then processed by the Spark cluster and parallelized in an automated manner: the workload is partitioned among worker nodes that operate on the subset of data locally held in memory or on disk. Finally, Notebooks allow users to quickly visualize datasets. To this end, large datasets on the Spark cluster can be sampled and visualized in the web browser. We demonstrate that the Spark platform and Notebook technology is a viable alternative to a utility-based analysis from command line. To this end, we provide a proof-of-concept by jointly analyzing three samples for which whole-genome, whole-exome and RNA-seq data are available. Variants are called using Halvade and Halvade-RNA and resulting VCF files were loaded onto a single node, 32-core Spark cluster. From a Spark Notebook, we performed a three-way concordance and discordance analysis among VCF files. Even though the 9 annotated VCF files are sized 22.5 GBytes altogether, most of the analysis steps have a highly responsive, almost real-time behavior. The combination of Halvade(-RNA) and Spark Notebook technology provides a fully distributed analysis end-to-end framework, from raw molecular data to biological or clinical interpretation.

5.2 Materials and Methods

5.2.1 Preprocessing the CCLE data

The data used in this analysis comes from the Cancer Cell Line Encyclopedia (CCLE) project [14]. From the full manifest file of the TCGA CCLE Cell Line BAM files we selected the samples where an WES, WGS and RNA-seq BAM file is available from the same sample. These BAM files were downloaded with GeneTorrent version 3.8.7 and were converted back to FASTQ files. The WES samples consisted of approximately 90 million paired-end reads of 76 base pairs. The RNA data was sequenced using poly-A selection and is not strand specific. It shows a bit more variance in number of reads, between 160 million to 180 million 101 base pair long paired-end reads. The WGS samples consisted of approximately 1.5 billion 101 base pair long paired-end reads.

These were then used as input files for Halvade and Halvade-RNA. The read alignment and variant calling of these reads was done with Halvade [1] version 1.0.1, Halvade runs the Best Practices pipelines according to the Broad Institute [9] for both RNA-seq and DNA in a parallel way. Additionally, with Halvade we counted the

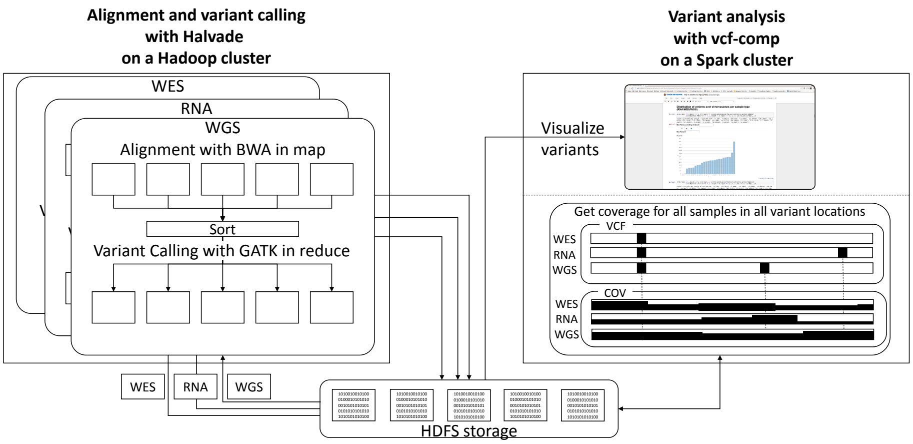

Alignment and variant calling with Halvade on a Hadoop cluster
WES
RNA
WGS
Alignment with BWA in map
Sort
Variant Calling with GATK in reduce
WES
RNA
WGS
HDFS storage
Variant analysis with vcf-comp on a Spark cluster
Visualize variants
Get coverage for all samples in all variant locations
VCF
WES
RNA
WGS
COV
WES
RNA
WGS

Figure 5.1: Overview of the analysis process described before. First Halvade processes RNA-seq, WES and WGS data and stores the output data on a shared file system. Next these variant data (VCF files) are preprocessed: the coverage for all three datasets is computed and also stored on the shared file system. As a final step, the data is analyzed in an interactive Spark Notebook with near real-time performance. All these tools are available to be used in the cloud.

number of read per exon for the RNA-seq datasets using featureCounts [15] from the Subread package. Halvade uses BWA version 0.7.5a, STAR version 2.4.0h1, Samtools [16] version 0.1.19-44428cd, Picard [17] version 1.112(1930), GATK [3] version nightly-2015-05-12-gcdf54f8 (this nightly build includes a bug fix for the SplitNCigars command), featureCounts version 1.4.6-p4, elPrep [18] version 2.31. For both DNA and RNA-seq pipelines, we used the HaplotypeCaller from GATK to call the variants. These variants were afterwards annotated with known variants from ClinVar [19] and common SNPs (both database downloaded on 3 June 2015) with snpSift [20] version 4.1k while gene effects were added with snpEff [21] version 4.1k. The analysis was performed using Spark Notebook [22] which uses Scala (version 2.10.4), Spark (version 1.4.1) and Hadoop 2.3.0 with Parquet support. To read and manipulate the variants we used ADAM [23] version 0.17.1.

5.2.2 Hadoop, Spark and Spark Notebook

In this chapter we adopted cloud-compatible technologies. For Halvade we relied on the Hadoop MapReduce framework to implement the GATK Best Practices pipeline for both RNA-seq and DNA-seq data. Hadoop MapReduce is accessible through Amazon Elastic MapReduce or similar publicly available services. Setting up an Amazon AWS cluster with a specific version of Hadoop pre-installed is relatively easy. This allows the use of Halvade when a local cluster is unavailable. In this project we based much of our implementation on the ADAM interface which allows the use of Spark. Spark is equally available on cloud environments such as Amazon AWS. However, using spark with the Spark shell or a bash command is not very practical as the process

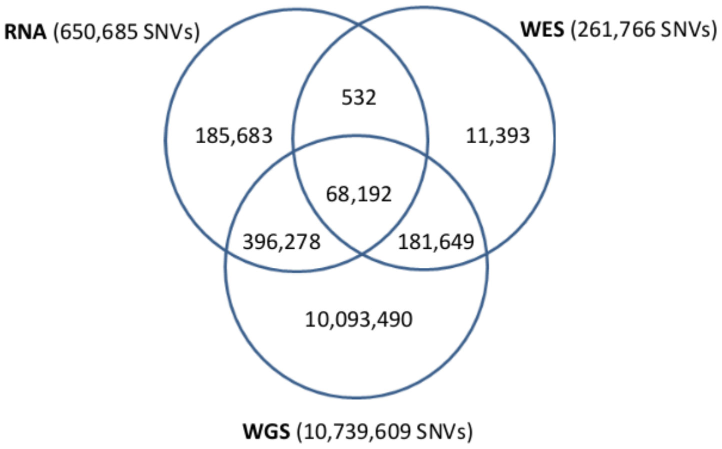


Figure 5.2: Overview of the distribution of the SNV counts for the different sequencing strategies. The numbers represent the sum of all three samples for each collection.

of variant analysis requires manual selection of the next set of parameters to filter on. With Spark Notebook we have access to the Spark environment within a graphical and interactive environment. This notebook can visualize large-scale datasets in a scalable manner thus providing for an interactive exploration of the data.

Because our analysis is based on the read coverage of the variant location in every sample, we preprocessed the variants in Spark to count the coverage for the three sequencing strategies for every variant. During this step, we use the BAM file that is obtained from the read alignment step and contains the aligned reads right after the BWA step. It does not include the indel realignment step so this might give some very low noise in the numbers. The notebook reads the variants called from the RNA-seq, WES and WGS datasets and keeps this list in memory. After this, each BAM file containing the aligned reads is read and the coverage of each location in the list is recorded and written to file. This preprocessing step speeds up the subsequent phases of analysis. During the next phase, again written in a Spark Notebook, the three samples are compared. In this case, the three sequencing strategies of the same sample. In order to keep an interactive environment we keep the variants in memory, including the corresponding calculated coverage numbers. This allows us to process the subsets in near-real time. A number of processing steps are available, including functions like filtering, base change counting and collecting annotation information. An overview of all tools and interactions is depicted in Fig. 5.1.

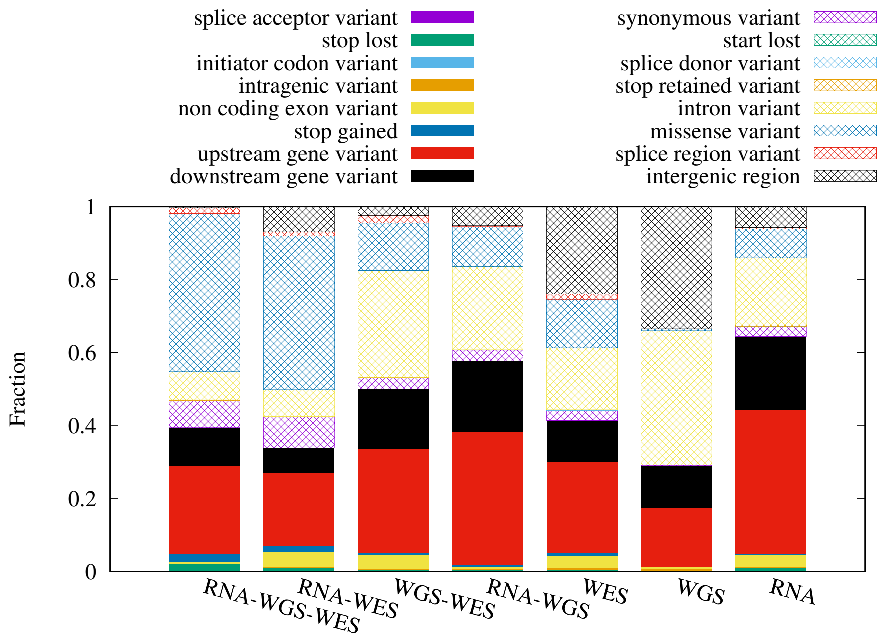


Figure 5.3: The fraction of each detected functional annotation for every subset.

5.3 Analysis and Discussion

We first filtered the VCF files for WGS, WES and RNA-seq to retain only Single Nucleotide Variants (SNVs). Indels were left out of this comparison as they are handled in a somewhat different manner. The retained SNVs were split into seven collections based on the overlap between RNA-seq, WGS and WES. This analysis is performed for each sample individually. No cross-sample analysis is made. Fig. 5.2 displays the aggregate number of variants for the three samples for every subset. We found that each sample individually showed similar numbers for every subgroup. The functional annotations for each of the seven subsets is displayed in Fig. 5.3. The functional annotation sometimes provides multiple possibilities for a single variant. In that case we selected the first reported annotation for every variant.

5.3.1 RNA-WGS-WES concordant variants

First we look at the collection of variants that shows concordance between the three sequencing strategies: RNA-seq, WES and WGS. The 68,192 variants in this collection correspond to 10.5% of the RNA-seq variants, 26.1% of the WES variants and 0.6% of the WGS variants. The variants found in this collection have high coverage in all three datasets. The fraction of variants with coverage greater than or equal to 5 in RNA-seq, WES and WGS datasets is respectively 83.1%, 90.5% and 100%. Due to the fact that these variants were called by all three datatypes with a decent coverage, we can consider these variants to be true variants. Next, we take a look at the snpEff functional annotations of these variants. We notice that 50.8% of these are either synonymous or non-synonymous variants. This corresponds to substitutions of bases in an exon of a gene coding for a protein. Synonymous variants are variants that do not alter the amino acid sequence, while non-synonymous variants do alter the amino acid sequence and consequently can have a biological impact on the organism. It makes sense that the majority of the variants found in this collection are variants found in exons, as these regions are primarily captured by WES. Another 42.2% consists of variants in the upstream gene, downstream gene and intron variants. These variants are in close proximity (respectively 5', 3' and intron regions of a gene) to the exon regions. In whole exome sequencing the location where reads are sequenced encapsulates the gene regions and thus can include these upstream and downstream regions.

5.3.2 RNA-WES concordant variants

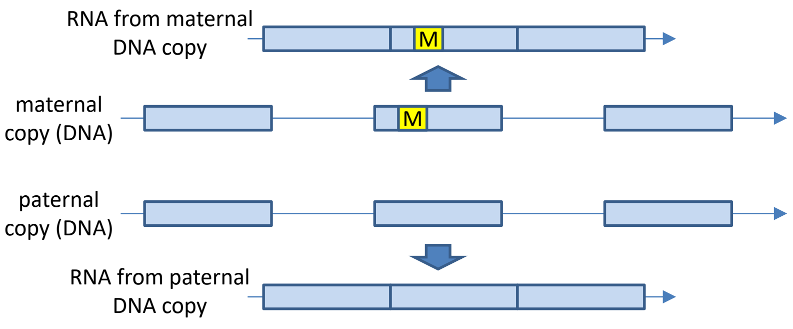
Next we consider the variants called by RNA-seq and WES but not by WGS. This is a small collection containing only 532 variants. Notable is that 32% of these variants have a coverage less than 5 in WGS which might explain why they were not picked up by that data type. The Ti/Tv ratio is a ratio of the number of transitions and transversions. Transitions are interchanges of two-ring purines (A and G) or one-ring pyrimidines (C and T). Transversions are mutations between purine and pyrimidine and exchange between one-ring and two-ring structures. The Ti/Tv ratio for WGS is expected to be approximately 2.1, while this is expected to be 3 or more for exome regions. If the Ti/Tv ratio is closer to that of a random substitution (0.5), low-quality data is implied [24]. The Ti/Tv ratio is 1.49 and is considerably different from the Ti/Tv ratio of 2.68 for the RNA, WES and WGS concordant set, which would be expected to be similar. This makes us believe that these variants are likely false positives.

If we look at the entire subset of RNA-seq and WES variants (68,192+532), we see that 99.2% are confirmed by WGS. This gives rise to the conclusion that if WGS were not available, 99.2% of the overlap between RNA-seq and WES are likely true positives.

5.3.3 WGS-WES concordant variants

There are 181 649 WES and WGS concordant variants. Looking at the matching coverage of all datasets for each of these variants, we see that there is a clear difference. All variants have a coverage of 5 or higher in the WGS dataset. The WES coverage shows that 83.4% have a coverage of 5 or higher whereas the RNA-seq dataset shows that only 3.5% have a coverage of 5 or higher and that 79.6% of the variants called have no coverage on those locations. This clearly shows that these variants are not called in the RNA-seq dataset because there simply is not enough coverage at those

Allele-specific expression



	both alleles expressed	only mutated allele expressed	only reference allele expressed
heterozygous variant	call in RNA and WGS/WES	call in RNA and WGS/WES	no call in RNA! call in WGS/WES

Figure 5.4: Example of allele-specific expression. Showing that allele-specific expression will not be called in RNA-seq when only the reference allele and not the mutation is expressed.

locations. Due to the lack of coverage in the RNA-seq dataset, the RNA-seq pipeline does not call about 74.9% of the variants called in WES. Or in other words, we miss about 60 000 variants per sample in RNA-seq in the exome due to low gene expression.

About 74% of the WGS and WES concordant variants are either upstream, downstream or intron variants and 16% are synonymous and non-synonymous variants. Again we see that most of these variants belong to regions that are very close or are contained in the exome. There are few synonymous and non-synonymous variants as many of these do have coverage in RNA-seq and were confirmed by RNA-seq and do not belong to this group.

Allele-specific expression

Next we attempt to identify why variants in this set are not called in RNA-seq even with a coverage higher than or equal to 5. We assume that one of two things can happen in this situation. Either the variant called by WES and WGS is a false positive or the variant is not called due to allele-specific expression. Allele-specific expression shows a (major) bias towards expressing one of two alleles. If the SNP is found in one allele but not in the other, this can be the reason why RNA-seq cannot detect it. If only the allele with the SNP is expressed or both alleles are expressed then this variant will also be detected in the RNA-seq sample and thus is not in this collection. However, if only the allele without the SNP is expressed, RNA-seq cannot call this

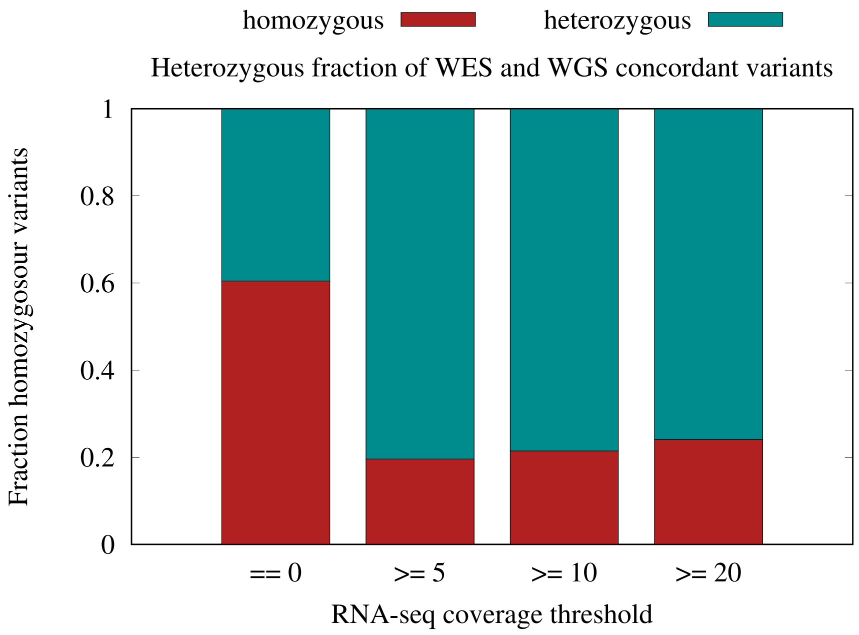


Figure 5.5: The homozygous fraction of the WGS-WES concordant variants set with increasing coverage threshold. The first bar represents the variants that have no coverage in the RNA-seq dataset.

SNP. This is shown in Fig. 5.4. This last situation corresponds to what we should find in the variants with RNA-seq coverage. If the SNP only occurs in one strand, then this variant should be a heterozygous call in the WES/WGS dataset. This means that not all reads covering this position have this SNP. To show this we look at the fraction of heterozygously called variants in this subgroup with and without coverage in the RNA-seq sample. We filter out the variants with no RNA-seq coverage, as well as make filtered groups where the RNA-seq coverage is bigger than at least 5, 10 or 20. The number of variants with a minimum coverage of 5 is only about 3.4% of the entire subcollection.

Fig. 5.5 shows a great increase in the fraction of heterozygous calls. About 39.5% of the variants without RNA-seq coverage are heterozygous calls. This is roughly 80% when looking at the variants that have a coverage of 5 or more in RNA-seq. This enrichment of heterozygous calls in WES/WGS is a strong indication of allele-specific expression. This explains why the variants that do have coverage in RNA-seq are not called in RNA-seq.

5.3.4 RNA-WGS concordant variants

Next we will discuss the RNA-seq and WGS concordant collection. Looking at the coverage of these variants in the WES dataset, we notice that only 1% of these have a coverage greater than or equal to 5. This explains why WES is unable to call these variants. A great deal of these variants (84%) are located in upstream and downstream gene regions or are classified as intron or intergenic variants. This explains the lack of coverage in the WES sample in those regions. Even though 13.8% of the variants are synonymous or non-synonymous, which are variants located in an exon region, they are probably not called due to lack of coverage. This shows that RNA-seq is able to call variants in the exome, that you would otherwise miss by only performing WES. Here, RNA-seq calls 20.9% additional variants that WES alone does not call.

5.3.5 WES-only variants

Looking at the variants only called by the WES pipeline we notice that 13.5% have a coverage greater than 5 in RNA-seq while over 87% of these variants have a coverage greater than 5 in the WGS dataset. This means that these variants are called by the WES pipeline but are not confirmed by WGS although there are at least 5 reads at that location. The lack of call in WGS together with the high coverage there excludes allele-specific expression as a cause for this. This indicates that these are most likely false positives called by WES or false negatives, i.e. actual variants that are not called by WGS. The low Ti/Tv ratio of 1.11 and the low homozygous fraction (35% of these variants) indicate that these are rather unexpected results for WES variants. This confirms our belief that these variants are most likely false positives. Previous studies have shown that the false-positive rate in WES-only data is much higher than in the WGS-only data, even in exon regions [25], which also confirms our findings.

5.3.6 WGS-only variants

The next collection we will discuss are the variants called only by the WGS pipeline. We notice that the majority of these variants are not called in the WES and the RNA-seq pipeline due to lack of coverage. Over 99% of these variants called only by WGS have a coverage of 2 or less in both the WES and RNA-seq datasets. While 99.7% of these variants have a coverage greater than or equal to 5 in the WGS dataset. This obvious difference in coverage is why these variants are called only by WGS. To explain this difference in coverage we look at the locations of these variants on the genome. This shows that over 98% of these variants are classified as intergenic region, intron variant, or upstream and downstream gene variants. These four regions explain the lack of WES and RNA-seq coverage, as these are not part of the exome.

5.3.7 RNA-only variants

Lastly, we look at the variants called only by the RNA-seq pipeline. The numbers of these variants change somewhat over the three samples: 45122, 58027 and 73884 variants for respectively hcc1954, hcc1143 and k-562. Looking at the coverage of these variant locations in the datasets, we notice that a significant fraction lacks coverage in WES (85.5% have less than 5 coverage). Unexpectedly, we see that only 71.1% of these have a coverage greater than or equal to 5 in the RNA-seq dataset itself while this is 98.4% for the WGS dataset. The lack of coverage in WES could be explained by the location of the variants, 83.7% are located in either upstream, downstream gene regions, introns or intergenic regions, but it does not explain why WGS does not call these. There are three reasons that could explain these variants; a) either we have low coverage in both WGS and WES and thus cannot call them there, b) they are false positives in the RNA-seq pipeline or c) they are variants caused by RNA editing. We know that 98.4% have a high coverage in WGS so we can assume that the first reason consists of only 1.6%. Similarly, we have counted the variants that are within the exome and have a coverage of at least 5 in the WES sample, this is approximately 3.9%. We believe the false positives is a larger number as the RNA-seq alignment is more complicated and will result in more errors regardless of the coverage in WES, i.e., we believe there are more false positives in this set including variants without coverage in WES. However, these false positives from RNA-seq and false negatives in the DNA sequencing samples have to be confirmed by doing wet-lab verification. As we do not know the fraction of false positives we decide to focus on the last reason, RNA editing.

RNA editing

RNA editing is a process where the cell makes post-transcriptional changes to the RNA molecule. This in turn changes the amino acid sequence which is crucial to homeostasis and development of the cell. This does not include splicing or other RNA processes like 5'-capping and 3'-polyadenylation. The editing may include insertions, deletions and base substitutions. As we have filtered out the insertions and deletions we will try to identify the base substitutions. From [26] we know that RNA editing in mammals consists mainly of A-to-I mutations. This is identified as A-to-G variants, as the Inosine is interpreted as Guanosine by the sequencing machinery. This means that we expect a higher than normal number of A-to-G variants. This is indeed the case. However, because our RNA-seq dataset was not strand specific, we cannot separate A-to-G and T-to-C variants.

Fig. 5.6 shows the fraction of A-to-G and T-to-C variants in both filtered and unfiltered RNA-only and RNA-WGS concordant variants. This shows that the fraction of A-to-G or T-to-C mutations is clearly higher in the RNA-only dataset than the RNA-WGS concordant variants. In the unfiltered sets the fraction increases from about 34% to approximately 77%. Even when filtering on a read depth of 20 or higher to eliminate the false positives, we see a similar situation.

The Alu family is a family of repetitive sequences in the human genome. The Alu elements have over one million copies throughout the genome. These elements have been important in explaining the evolution of primates. According to [27] many of the RNA editing mutations occur in the Alu sequences. The fact that Alu regions are part of repetitive sequences increases the possibility that reads are misaligned here.

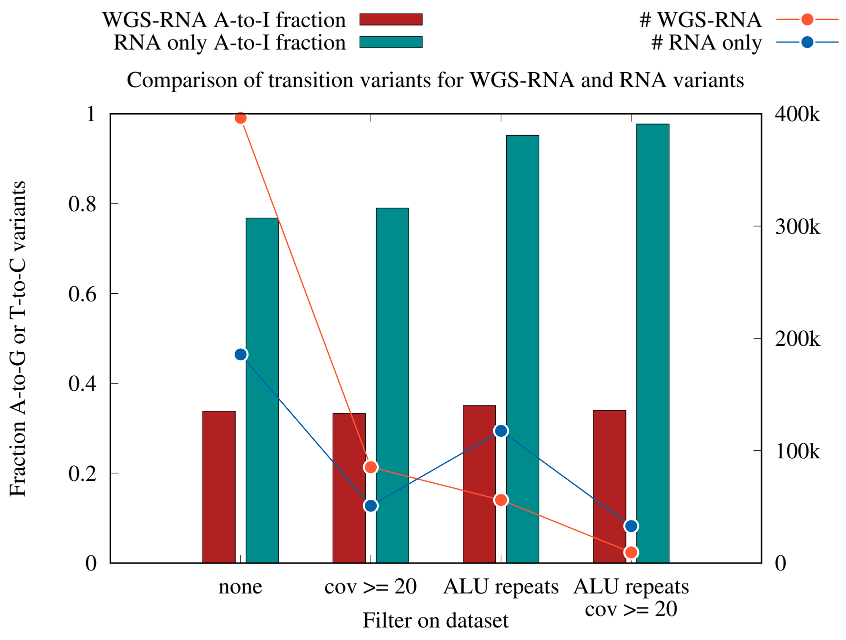


Figure 5.6: The homozygous fraction of subsets of the WGS-WES concordance set. Additionally the fraction of the variants per subset is given to give a better perspective on the numbers.

However, as stated in [27] it is statistically improbable for SNPs to cluster A-to-G substitutions almost exclusively in these regions and they believe these variants represent genuine RNA editing variants. They then showed that given these mutations in repeat regions, the aligned locations of the reads were still the best match. If we filter the RNA-only and RNA-WGS concordant variants on position with an Alu sequence, we see that the fraction of A-to-G and T-to-C mutations increases for the RNA-only variants but not for the RNA-WGS concordant set. If we go further and filter both on position in Alu sequences and minimum coverage of 20, we get an even higher fraction, approximately 97.7%. We see that the unfiltered set and filtered-on-coverage-20 set both give a higher count of RNA-WGS concordant variants. However, when filtering these sets on the ALU-repeat regions we see that the RNA-only variants have more variants in these regions. This shows that a big fraction of the RNA-only variants are in fact located in these Alu sequences. This makes us believe that we are in fact dealing with RNA editing mutations. This also explains why a great deal of these RNA-only variants cannot be called by WGS/WES sequencing.

Because WGS is typically more expensive than RNA-seq and WES, we tried to see if it is possible to detect RNA editing mutations when only RNA-seq and WES are available for a sample. To do this, there needs to be a way to separate the RNA-only variants from the RNA-WGS concordant variants when WGS is not available. We have found that there is no clear way of separating these two groups. When filtering the variants not called by WES on a given threshold for the coverage in the WES sample, we see that both RNA-only and RNA-WGS show a similar fraction of variants being filtered. This means that this scheme cannot be used to filter the RNA-only datasets. Similarly, we have tried to filter on Alu sequences but have found no clear distinction between the two groups. There were also no clear differences in quality, coverage or similar statistics.

5.4 Conclusions

With Halvade, Halvade-RNA and the Spark Notebooks developed for the variant comparison tool, we provide a cloud solution for the entire variant calling pipeline starting from alignment and ending with variant analysis. The Spark Notebook provides a way to achieve reproducible results. Using HDFS, the Halvade results are readily available to the variant analysis steps. We used the BAM files to get additional coverage information for all samples and used this data for the variant analysis steps. The VCF files were also annotated and to use this information ADAM had to be extended to handle this information. The analysis tool is able to compare several VCF files of several GBytes in size and perform near real-time responsiveness, while remaining scalable. As a proof of concept of our variant comparison tool we implemented a Spark Notebook that compares the variants of three sequencing strategies: RNA-seq, WES and WGS.

Using the Spark Notebook, we were able to determine that variants called by both RNA-seq and WES can be used confidently as over 99% of these are confirmed by the WGS sample. Looking at the variants called by WES and WGS but not RNA-seq we found that many variants here were not called due to lack of coverage (no expression). Additionally, we gave a good indication that variants where RNA-seq did have coverage were most likely not found in RNA-seq due to allele-specific expression. This shows that RNA-seq misses some variants that WES did call, however the same can be said for WES. We found a great deal of variants called by both RNA-seq and WGS but not WES. This was mostly due to lack of coverage in the WES sample, mostly located near regions close to the exome. The variants called only by WGS showed no coverage in both the RNA-seq and WES dataset. A small amount of variants were called only by WES, we surmised that these are most likely false positives. Lastly, we looked at the variants called only by RNA-seq. These variants showed that a high amount of A-to-G and T-to-C mutations were present, which are indicative of RNA editing. Which would also explain why these variants cannot be called by WGS. As an additional indication of RNA editing, we found that a great amount of these variants were located in Alu sequences. We also discovered some of these variants are likely false positives. The variants called only in the WGS pipeline represent 92.3% of all variants called by the three pipelines. An additional 5.9% of all variants were called by WGS and supported by either WES or RNA-seq or both. The remaining 1.8% of the variants or called either by RNA-seq or WES or both. A small fraction of 1.7% of all variants are called only from the RNA-seq dataset and is caused by RNA editing, where the mutation only occurs after transcription to RNA-seq and thus is not present in the DNA sequence itself. The last 0.1% of all variants are the WES-only variants, which was discussed before.

We believe that some additional improvements can be made, like adding stranded whole RNA-seq which would eliminate the ambiguity that is present in the A-to-I editing that we found in the RNA-seq only variants. We would also like to do wet-lab experiments to confirm our findings, e.g. RNA editing variants.

References

[1] Decap, D., J. Reumers, C. Herzeel, P. Costanza, and J. Fostier, 2015. Halvade: scalable sequence analysis with MapReduce. *Bioinformatics*, pages btv179+.

[2] Li, H. and R. Durbin, 2009. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics (Oxford, England)*, 25(14):1754–1760.

[3] McKenna, A., M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis, et al., 2010. The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. *Genome research*, 20(9):1297–1303.

[4] Abuín, J. M., J. C. Pichel, T. F. Pena, and J. Amigo, 2015. BigBWA: approaching the BurrowsWheeler aligner to Big Data technologies. *Bioinformatics*, pages btv506+.

[5] Dean, J. and S. Ghemawat, 2008. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM*, 51(1):107–113.

[6] Lam, H. Y. K., C. Pan, M. J. Clark, P. Lacroute, R. Chen, et al., 2012. Detecting and annotating genetic variations using the HugeSeq pipeline. *Nature Biotechnology*, 30(3):226–229.

[7] Puckelwartz, M. J., L. L. Pesce, V. Nelakuditi, L. Dellefave-Castillo, J. R. Golbus, et al., 2014. Supercomputing for the parallelization of whole genome analysis. *Bioinformatics*, 30(11):1508–1513.

[8] Kelly, B. J., J. R. Fitch, Y. Hu, D. J. Corsmeier, H. Zhong, et al., 2015. 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(1).

[9] Van der Auwera, G. A., M. O. Carneiro, C. Hartl, R. Poplin, G. del Angel, et al., 2013. From FastQ Data to High-Confidence Variant Calls: The Genome Analysis Toolkit Best Practices Pipeline. *Current Protocols in Bioinformatics*, 43:11.10.1–11.10.33.

[10] Shiraishi, Y., A. Fujimoto, M. Furuta, H. Tanaka, K.-i. Chiba, et al., 2014. Integrated analysis of whole genome and transcriptome sequencing reveals diverse transcriptomic aberrations driven by somatic genomic changes in liver cancers. *PloS one*, 9(12).

[11] Piskol, R., G. Ramaswami, and J. B. Li, 2013. Reliable Identification of Genomic Variants from RNA-Seq Data. *The American Journal of Human Genetics*, 93(4):641–651.

[12] Dobin, A., C. A. Davis, F. Schlesinger, J. Drenkow, C. Zaleski, et al., 2012. STAR: ultrafast universal RNA-seq aligner. *Bioinformatics*, 29(1):bts635–21.

[13] Decap, D., J. Reumers, C. Herzeel, P. Costanza, and J. Fostier, 2017. Halvade-rna: Parallel variant calling from transcriptomic data using mapreduce. *PLOS ONE*, 30.

[14] Barretina, J., G. Caponigro, N. Stransky, K. Venkatesan, A. A. Margolin, et al., 2012. The Cancer Cell Line Encyclopedia enables predictive modelling of anticancer drug sensitivity. *Nature*, 483(7391):603–307.

[15] Liao, Y., G. K. Smyth, and W. Shi, 2014. featureCounts: an efficient general purpose program for assigning sequence reads to genomic features. *Bioinformatics (Oxford, England)*, 30(7):923–930.

[16] Li, H., B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, et al., 2009. The Sequence Alignment/Map format and SAMtools. *Bioinformatics (Oxford, England)*, 25(16):2078–2079.

[17] Wysoker, A., 2009. Picard.

[18] Herzeel, C., P. Costanza, D. Decap, J. Fostier, and J. Reumers, 2015. elPrep: High-Performance Preparation of Sequence Alignment/Map Files for Variant Calling. *PLoS ONE*, 10(7):e0132868+.

[19] Landrum, M. J., J. M. Lee, M. Benson, G. Brown, C. Chao, et al., 2016. ClinVar: public archive of interpretations of clinically relevant variants. *Nucleic Acids Research*, 44(D1):D862–D868.

[20] Cingolani, P., V. Patel, M. Coon, T. Nguyen, S. Land, et al., 2012. Using drosophila melanogaster as a model for genotoxic chemical mutational studies with a new program, snpsift. *Frontiers in Genetics*, 3.

[21] Cingolani, P., A. Platts, M. Coon, T. Nguyen, L. Wang, et al., 2012. A program for annotating and predicting the effects of single nucleotide polymorphisms, snpeff: Snps in the genome of drosophila melanogaster strain w1118; iso-2; iso-3. *Fly*, 6(2):80–92.

[22] Petrella, A., 2014. Spark Notebook.

[23] Massie, M., F. Nothaft, C. Hartl, C. Kozanitis, A. Schumacher, et al., 2013. Adam: Genomics formats and processing patterns for cloud scale computing. Technical Report UCB/EECS-2013-207, EECS Department, University of California, Berkeley.

[24] Liu, Q., Y. Guo, J. Li, J. Long, B. Zhang, et al., 2012. Steps to ensure accuracy in genotype and SNP calling from Illumina sequencing data. *BMC genomics*, 13 Suppl 8(Suppl 8):S8+.

[25] Belkadi, A., A. Bolze, Y. Itan, A. Cobat, Q. B. Vincent, et al., 2015. Whole-genome sequencing is more powerful than whole-exome sequencing for detecting exome variants. *Proceedings of the National Academy of Sciences of the United States of America*, 112(17):5473–5478.

[26] Daneck, P., C. Nellaker, R. McIntyre, J. B. Buendia, S. Bumpstead, et al., 2012. High levels of RNA-editing site conservation amongst 15 laboratory mouse strains. *Genome Biology*, 13(4):R26+.

[27] Kim, D. D. Y., T. T. Y. Kim, T. Walsh, Y. Kobayashi, T. C. Matise, et al., 2004. Widespread RNA Editing of Embedded Alu Elements in the Human Transcriptome. *Genome Research*, 14(9):1719–1725.

6 Conclusions and Future Research

6.1 Discussion

Sequencing technology has seen an incredibly fast evolution. The cost of sequencing has been greatly reduced while the sequencing speed has increased considerably [1]. This means that more and more sequencing data is becoming available that needs to be processed. The speed at which sequencing technology has evolved has surpassed Moore's law [2], which means that the timely processing of this obtained data is becoming more and more difficult. Here we looked at variant calling, which processes the sequencing data and identifies variations with respect to a known reference genome. The Best Practices pipeline [3] proposed by the Broad Institute shows the required steps to perform variant calling starting from raw sequencing data when GATK is used as a variant caller. Tests show that this entire pipeline takes approximately 12 days using a single core or 5 days using a single 24-core machine for a human WGS sample. This processing time is slower than the sequencing itself. As this data is used in clinical settings or medicinal research, this step needs to be minimized in order to allow timely analysis of the sample in question. We implemented our own pipeline called Halvade, which is made for multi-node environments.

Based on the observation that read mapping is parallel by read while the variant calling and the preparation steps are parallel by region we use the MapReduce parallelization model [4]. Every map or reduce task starts an instance of the tool that performs the mapping, data preparation or variant calling. This allows us to use the very efficient parallel sort of the framework which is required between read mapping and variant calling. Similarly, the Hadoop MapReduce framework allows us to run the pipeline on multiple nodes with relative ease. The reads are chopped into smaller files, keeping paired-end reads together. These smaller files are used as input for the map tasks. Every map task streams the reads to the BWA alignment tool [5] and processes the output of this tool. This output data is parsed into <key, value> pairs, then grouped by genomic region and then sorted by position in that region. The number of genomic regions determines the number of reduce tasks that will be started. The reduce tasks stream the aligned reads to the required preparation tools and perform indel realignment and base quality score recalibration (BQSR). After these steps the read data is ready to be used for variant calling. The variant calling tool is called and the output is streamed to a file on HDFS. On a single node the runtime of the human whole genome sample is reduced from 5 days to approximately 2 days because multiple tasks run on a single system. The runtime is greatly reduced when using multiple nodes, on a 15-node machine the runtime is reduced to 2h 39min.

During the development of Halvade, Spark has become increasingly popular as an alternative framework. Spark is not limited to the MapReduce programming model and includes more functions to process data in parallel. Another advantage is that Spark attempts to keep the data in memory as much as possible. This allows faster access to the data and in turn should increase performance, especially when data is accessed iteratively. However, we maintain the use of the Hadoop MapReduce framework for Halvade. This decision is based on two observations. First, we use Hadoop more as a resource distribution framework. Multi-processing gives better performance than multithreading. We start a single instance per task and all tasks are distributed over the cluster by the framework. Secondly, the process is not iterative, which is where Spark would allow better performance. Because the input and output of the tools is not the limiting factor of the performance, we estimate that keeping the data in memory has very limited influence on the runtime.

The accuracy of the output of Halvade was compared to the output of the classical pipeline. We achieve more than 99% matching variants between the two. Additionally, almost all the variants that did not match had a very low variant confidence score. We investigated the source of this discordance. The biggest reason for this discordance is that read alignment occurs on only a part of the entire input. The first group of reads are used as a training set to determine the insert length. This changes slightly when a different training set is used. Marking PCR duplicates also gives rise to a very small change. Selecting which read is real and which are duplicates is based on a random factor. Lastly, the BQSR is based on a subset for every reduce task instead of the complete dataset. This again introduces a small change in the base quality scores. To achieve 100% concordance we would need to have global communication in read alignment, PCR duplicate marking and BQSR. However, this would severely influence the achieved performance and we estimate that the mismatching variants are of low quality and will be filtered out in high quality variant analysis anyway. We note that Halvade uses the tools described in the pipeline as is, this means that updates of these tools can easily be used in Halvade. Halvade can use a new version simply by replacing the binary file, provided that the arguments remain unchanged.

Halvade shows promising speedups when comparing the number of Hadoop containers that are run. Comparing 3 Hadoop containers (one node) with 59 Hadoop containers (15 nodes), we achieve a parallel efficiency of 92,1%. However, when using the single threaded runtime instead of containers as a baseline, the parallel efficiency is only about 30%. In an attempt to clarify this low parallel efficiency we tested Halvade on several systems to compare different hardware. We found that having sufficient memory influences the runtime the most, since this allows Halvade to run more tasks in parallel instead of relying on multithreading. Some tools show a low multithreading efficiency and perform better when started in parallel. Taking advantage of NUMA domains by using a separate copy of the reference data per domain also gives a modest performance improvement. Surprisingly, other aspects like local disk speed, speed of interconnection network or file system had only a minor influence on the runtime. We believe that this is partly caused by the Hadoop framework that tries to minimize the data transfers. Similarly, we believe that local disk speeds show so little influence because only a small fraction of the runtime of all the tools is spend in the input/output phases and often occurs simultaneously with the calculations of the tool.

Halvade will most likely be used in some situations where the data needs to be kept private, there are two options to realize this. The first is to have a private cluster or cloud. This means that the data does not leave the company and this ensures the safety of the data. The second is to use encryption with existing cloud solutions, e.g. Amazon AWS. These cloud solutions typically provide encryption when storing data but also when data is transferred between storage, compute node and the end-user. This encryption ensures the privacy of the data in every step, Halvade can then run in a virtual environment which decrypts the data, processes it, and the output is then encrypted again.

We also note that simply by defining the Java heap memory for the Halvade Uploader, some speedups can be gained for the preprocessing. Without this variable definition we had a runtime of about 1h 30min with 8 threads. Simply by defining the Java heap memory as 32 GByte we were able to reduce this time to 1h 10min, again with 8 threads. Not only does it increase speed, it also allows the use of more threads. When increasing the number of threads without defining the Java heap memory an out of memory error can occur. With 20 threads we can further reduce the runtime to 48min. We would expect an even lower runtime, but since it is a data intensive problem, it is limited by the hard disk drive I/O bandwidth and network connection speeds.

Next we investigated load balancing strategies for both the map and reduce phases. During the map phase, a big variation in task execution time is present. Some tasks take up to 20 times longer to finish than others, which means load balancing becomes more difficult. We found that the location of a read in the original input file has an influence on the alignment time. We believe that the reads were likely aligned right after sequencing, sorted by position, and then saved in that order. The reads that were not aligned were added at the end of the file. This explains why the last few tasks take considerably longer. When shuffling the file, the reads that were not aligned are not grouped together in the end and this means less variation in task runtime. However, we noticed that shuffling the input files has a very limited influence on the total map runtime. We believe this is caused by the increase in network activity because the reads in the input files are not sorted anymore. This means that every map task has to send a part of the aligned reads to (almost) every reduce task. For the reduce tasks we discovered that the execution time changes only slightly for all balancing strategies we tried, this left us with optimizing the number of reduce tasks. Using slightly less than twice the maximum parallel reduce containers as the total number of reduce tasks yields the best performance.

With all these optimizations we were able to reduce the runtime further to 1h 21min on a 15 node machine with 512 GByte of memory. This corresponds to a parallel efficiency of 59%, nearly twice of what we achieved before.

The Best Practices pipeline for both DNA-seq and RNA-seq are very similar. In Halvade we added the option to run this slightly different RNA-seq pipeline. This meant re-implementing the alignment step. The RNA-seq alignment is based on STAR and uses a two-pass alignment. This two-pass alignment includes the addition of splice site information to the existing reference genome to increase alignment accuracy. This was implemented in two MapReduce jobs, the first runs the first alignment step, collects the splice site information and then rebuilds the reference genome with the additional information in the reduce step. The second alignment is then run with the updated reference in the second MapReduce job and is followed first by sorting and then running the data preparation and variant calling steps in the reduce phase.

In our RNA-seq benchmarks we used 9 RNA-seq samples from the Cancer Cell Line Encyclopedia [6], the average runtime with the original pipeline on a single core is 27.9h and goes down to 12.9h when using 20 cores. This shows poor scaling behavior. With Halvade we were able to further reduce the runtime on a single node with 20 cores to $\sim$3h using 10 parallel tasks. This corresponds to a speedup of 9.18 compared to the sequential pipeline. Using two of these nodes the average runtime is reduced to $\sim$2h. We found that running all 9 samples in batch mode reduced the runtime to a total runtime of 11.8h. The parallel efficiency is considerably worse than when processing DNA-seq data. Obtaining a good load balance across parallel tasks is challenging for RNA-seq samples, as the reads per region is heavily influenced by the level of gene expression. A second cause is the two-step alignment with STAR, in between the two alignment steps, the found splice sites need to be added to the genome. In the sequential pipeline this is added on the fly. However, because all splice sites need to be used to rebuild the genome, Halvade has to first collect all this information from all tasks to do this. This process has to be done with a single task and is the cause of a very high performance loss. We believe that this can be improved when STAR supports adding this data to a reference already loaded in memory, which is not yet the case. However, when running multiple samples in batch this single task can overlap with other tasks from another sample which makes up for some of the performance loss.

The variants found by Halvade showed a 93.8% overlap with the original pipeline. The variability is almost exclusively caused by the difference attained by running the alignment on parts of the reads instead of all reads. However, the variants that did not match showed a lower number of reads supporting the call. Also, a high amount of these variants are located in regions with repeating patterns where we know read alignment is very difficult.

As an example use case we chose to compare variants called by three sequencing strategies on the same sample: WES, WGS and RNA-seq. Halvade can process WES and WGS, while Halvade-RNA can process the RNA-seq data. This gives us three Variant Call Format (VCF) files from the same source sample that can be compared and analyzed. To analyze this data, we implemented a VCF comparison tool using Spark Notebooks. Spark Notebooks have the advantage that they provide a way to achieve reproducible results. The access to HDFS through Spark allows very straightforward use of the VCF files in the Notebooks. We preprocessed these VCF files to get the corresponding coverage in all three samples for all variants found. This preprocessing step is somewhat time consuming which is why we store the data for later use. The next steps that use this data now have access to this coverage information. Lastly, we annotated the VCF files with SnpEff to be used in the analysis step. Using Spark allowed us to use the ADAM API, which enables access to VCF and SAM files in a Spark environment. This setup gave us a means to analyze VCF data in an iterative way. Using a single 32-core node we achieved near real time filtering of the VCF data ($\sim$22.5 GBytes on disk and nearly double in memory).

We observe that 99% of the variants called by both RNA-seq and WES are also confirmed by WGS and thus can be grouped as high confidence variants. The variants called by WGS only were caused by lack of coverage in the other two sequencing strategies. We found that RNA-seq calls variants confirmed by WGS but not WES, and similarly WES calls variants confirmed by WGS but not by RNA-seq. The lack of call in both cases is caused by different coverages in the respective locations.

This shows that both provide high confident variants that the other would miss. Lastly we see a high amount of variants called by RNA-seq that are not confirmed by WES nor WGS. These variants showed a very high amount of A-to-G and T-to-C mutations, which probably indicate RNA editing. Many of these variants were located in Alu sequences which again is an indication that these variants are caused by RNA editing. We would like to verify these results with wet-lab experiments.

The latest version of Halvade is available for download on http://bioinformatics.intec.ugent.be/halvade and runs on Hadoop MapReduce 2.0 or later (tested up to Hadoop 2.6) and Java 1.7 or later. The version of Java you should use depends on the GATK version. Since version 3.7 Java 1.8 is required while older versions use Java 1.7.

6.2 Future Work

Despite what we mentioned before, we do believe there is some merit in using Spark instead of MapReduce as a framework for Halvade. We estimate that the pipelines that are currently implemented would show limited or no improvement with Spark. However, we believe that a more complex pipeline would benefit from a framework that is not limited by only two functions, map and reduce. For one, it allows the communication or broadcasts to all nodes in a more efficient manner. As there is no strict map and reduce workflow, the programmer can easily adapt the workflow, this allows for a broadcast in between every step. This could provide a solution for the discordance caused by BQSR we had with Halvade. However, this would still lead to a performance drop as all tasks would need to sync in between these steps, and this always means that certain tasks will be idly waiting until all tasks are finished. However, steps like this might be required for more advanced pipelines.

Spark also allows a Resilient Distributed Datasets (RDD) to be written to the distributed file system. This way, the data is saved in a file per partition grouped in a folder. This can then again be read in Spark during later steps. This would eliminate the merge VCF step in Halvade and would allow us to read the data from there in the VCF comparison tool. However, often the files need to be annotated for analysis and this would require some additional effort to incorporate.

As a next step we would like to see an implementation of more complex pipelines. Somatic variant calling is an example of such a pipeline, this pipeline uses normal and tumor input files. In variant calling you typically expect to see SNPs fall into one of three bins of occurrence in the reads; 0%, 50%, or 100%, depending on whether they are heterozygous or homozygous. For somatic variant calling, certain things need to be taken into account. For example ploidy changes of a chromosome and that the pipeline uses both the tumor and normal BAM files for the variant calling step. This means both BAM files need to be prepared first. Next the tumor and normal samples are both used as input for the alignment tool. This means some coordination between all steps and input files is necessary.

Since single cell sequencing is becoming more important so it makes sense to provide an option to run Halvade using single cell sequencing data. The pipeline to process single cell RNA-seq (scRNA-seq) is very similar to our current implementation for the RNA-seq pipeline. An additional step would be to perform quality control for the input reads, as these are typically lower quality than other sequencing protocols. This could be added to the map phase before streaming the data to the alignment tool.

Halvade now supports merging a BAM file after the read alignment step, which allows the alignment of both files separately. However, the reduce step still includes some steps that should be run on both files, like indel realignment and BQSR. Using the current MapReduce approach, Halvade would align both files separately after which they are merged in a single reduce task, which would create a considerable performance drop. We believe that Spark can provide a better solution for somatic variant calling pipelines.

Another interesting avenue to pursue is a new way the aligned reads and variant data is represented in memory for the VCF comparison tool. This could allow more optimized filtering, searching and comparison of variant collections. Now, we depend on ADAM with an Avro representation of the variants. With this representation, we save duplicate variants on the same position together and thus store redundant information. Additionally, different data structures could be used that allow for better searching in intervals, which is often used when comparing VCF samples.

Because WGS is typically more expensive, we considered several ways to filter the RNA editing variants when only RNA-seq and WES data from a sample are available. However, we found that simply filtering on low WES coverage is not sufficient. This will include the variants that RNA-seq called but were then confirmed by WGS, which are not variants caused by RNA editing. We then looked at other filtering values like quality and functional annotation, but found no clear distinction that would filter out only the RNA editing variants. Even when using Alu regions as a filter, we found that a part of these RNA-seq variants were confirmed by WGS. We would like to further investigate how this can be done.

As stated before we would like to confirm our findings of the RNA-seq, WGS and WES comparison with wet-lab experiments. This includes confirming that the variants we considered to be caused by RNA editing are in fact RNA editing. To do this we need to select certain variants that would need to be confirmed. We would like to confirm high quality variants in the RNA-only collection. It would be interesting to confirm both high and low quality variants found in the RNA-seq, WES and WGS concordant collection. This would confirm our idea that variants called by all three strategies can be considered true variants. But it would also be interesting to verify the RNA-seq and WES concordant variants that are not confirmed by WGS: are these false positives by both WES and RNA-seq or are they false negatives in WGS?

References

[1] Metzker, M. L., 2010. Sequencing technologies - the next generation. *Nature reviews. Genetics*, 11(1):31–46.

[2] Wetterstrand, K., 2016. Dna sequencing costs: Data.

[3] Van der Auwera, G. A., M. O. Carneiro, C. Hartl, R. Poplin, G. del Angel, et al., 2013. From FastQ Data to High-Confidence Variant Calls: The Genome Analysis Toolkit Best Practices Pipeline. *Current protocols in bioinformatics / editoral board, Andreas D. Baxevanis ... [et al.]*, 11(1110).

[4] Dean, J. and S. Ghemawat, 2008. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM*, 51(1):107–113.

[5] Li, H. and R. Durbin, 2009. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics (Oxford, England)*, 25(14):1754–1760.

[6] Barretina, J., G. Caponigro, N. Stransky, K. Venkatesan, A. A. Margolin, et al., 2012. The Cancer Cell Line Encyclopedia enables predictive modelling of anti-cancer drug sensitivity. *Nature*, 483(7391):603–307.

A
Halvade Documentation

A.1 Introduction

Halvade is a Hadoop MapReduce implementation of the best practices pipeline from Broad Institute for whole genome and exome sequencing (DNA) as well as RNA-seq. Halvade will produce a VCF output file which contains the single nucleotide variants (SNVs) and additionally insertions and deletions (indels) in certain pipelines. This program requires Hadoop on either a local cluster with one or more nodes or an Amazon EMR cluster to run. As Hadoop is typically run on a linux cluster, this documentation only provides information for a linux setup. The new GATK 3.7 only works with Java v1.8, so this version of Java should be installed on every node. For older versions of GATK, Java v1.7 is required on every node.

Note: Halvade is available under the GNU license and provides a binary with all open source tools. However, since the GATK has its own license, which is available online here[1], the GATK binary is not provided in the bin.tar.gz file and needs to be added individually.

Halvade depends on existing tools to run the pipeline, these tools require additional data besides the raw sequenced reads. This data consists of the human genome reference FASTA file, some additional files created using the reference and the dbSNP database. The index file used in the BWA or STAR aligners are created with the tools itself. The FASTA index and dictionary files used by the GATK are created by sam-

[1]https://www.broadinstitute.org/gatk/about/#licensing

tools and Picard. The naming convention for these files and additional information to run Halvade is provided in this Halvade documentation.

A.1.1 Recipes

Two recipes have been created to run Halvade on WGS and RNA-seq data. These show all commands needed to run Halvade with the example data. The recipe for the DNA-seq pipeline can be found here[2]. The RNA-seq data is typically less big and therefore we chose to provide a recipe to run the Halvade rna pipeline on a single node Hadoop environment, which can be found here[3].

A.2 Installation

Every Halvade release is available at github[4]. Download and extract the latest release, currently v1.2.1:

```
wget https://github.com/biointec/halvade/releases/download/v1.2.1/
  Halvade_v1.2.1.tar.gz
tar -xvf Halvade_v1.2.1.tar.gz
```

The files that are provided in this package are the following:

```
example.config
runHalvade.py
halvade_bootstrap.sh
HalvadeWithLibs.jar
HalvadeUploaderWithLibs.jar
bin.tar.gz
```

A.2.1 Build from source

Halvade can also be built from the source files. To do this, you first need to clone the github repository and built the package with `ant` as follows:

```
git clone https://github.com/biointec/halvade.git
cd halvade/halvade/
ant
cd ../halvade_upload_tool/
ant
cd ../
```

This will build the two jar files in the respective *dist* subdirectories. The scripts and example configuration files can be found in the *scripts* directory, move the jar files to the script directory so the scripts have access to the jar file:

```
cp halvade/dist/HalvadeWithLibs.jar scripts/
cp halvade_upload_tool/dist/HalvadeUploaderWithLibs.jar scripts/
```

[2]https://github.com/biointec/halvade/wiki/Recipe:-DNA-seq-with-Halvade-on-a-local-Hadoop-cluster
[3]https://github.com/biointec/halvade/wiki/Recipe:-RNA-seq-with-Halvade-on-a-local-Hadoop-cluster
[4]https://github.com/biointec/halvade/releases

The next step is to download the human genome reference files and prepare them to use with Halvade.

A.3 The human genome reference

Halvade uses the genome reference FASTA file (`ucsc.hg19.fasta`), found in the GATK resource bundle, to build the index files for both BWA and STAR. The FASTA file comes with an index and a dictionary file. Additionally a full dbSNP file (version 138) is used when recalibrating the base scores for the reads. These files are all found in the GATK resource bundle which is available here[5]. This FTP site has a limited number of parallel downloads and might not load at these times. Here is how you download the files using the terminal in the current directory, it is advised to make a new directory for all reference files:

```
mkdir halvade_refs/
cd halvade_refs/
wget ftp://gsapubftp-anonymous@ftp.broadinstitute.org/bundle/hg19/
  ucsc.hg19.fasta.gz
wget ftp://gsapubftp-anonymous@ftp.broadinstitute.org/bundle/hg19/
  ucsc.hg19.fasta.fai.gz
wget ftp://gsapubftp-anonymous@ftp.broadinstitute.org/bundle/hg19/
  ucsc.hg19.dict.gz
mkdir dbsnp
cd dbsnp
wget ftp://gsapubftp-anonymous@ftp.broadinstitute.org/bundle/hg19/
  dbsnp_138.hg19.vcf.gz
wget ftp://gsapubftp-anonymous@ftp.broadinstitute.org/bundle/hg19/
  dbsnp_138.hg19.vcf.idx.gz
cd ../
```

Next we need to unzip all these files so they can be used in Halvade:

```
gunzip ucsc.hg19.fasta.gz && gunzip ucsc.hg19.fasta.fai.gz && gunzip
  ucsc.hg19.dict.gz && \
gunzip dbsnp/dbsnp_138.hg19.vcf.gz && gunzip dbsnp/dbsnp_138.hg19.vcf
  .idx.gz
```

The index files, the reference and the dbSNP file need to be uploaded to the HDFS server if a cluster with more than one node is used to run Halvade. Setting up Halvade is described in the following parts of the documentation.

A.3.1 BWA reference for WGS/WES data

The BWA aligner is used for the whole genome and exome sequencing pipelines. A BWT index of the reference FASTA file needs to be created to run BWA , which needs to be accessible by Halvade so BWA can be started correctly. The BWA binary is available in the *bin.tar.gz* archive, which is provided in every Halvade release.

```
tar -xvf bin.tar.gz
./bin/bwa index ucsc.hg19.fasta
```

[5]ftp://gsapubftp-anonymous@ftp.broadinstitute.org/bundle/2.8/hg19/

This process will create 5 files with the provided name as a prefix, this naming convention is important as Halvade finds this index by the FASTA prefix `ucsc.hg19`.

A.3.2 STAR reference for RNA-seq data

Note: The process to build the STAR index requires a minimum of 32 GBytes of RAM, make sure there is sufficient RAM memory.

The RNA-seq pipeline uses the STAR aligner to perform the read alignment step. Similarly to BWA, the STAR aligner requires an index of the reference FASTA file. Again, this can be created by using the STAR binary which is provided in the *bin.tar.gz* archive which is available in every Halvade release.

```
tar -xvf bin.tar.gz
mkdir ./STAR_ref/
./bin/STAR --genomeDir ./STAR_ref/ --genomeFastaFiles ucsc.hg19.fasta
   --runMode genomeGenerate --runThreadN 4
```

The shown command to build the STAR genome index uses 4 threads, this should be updated to reflect the number of cores available. After the STAR genome index has been created, the provided output folder will contain all files needed by STAR and in turn by Halvade.

A.4 Hadoop setup

Halvade runs on the Hadoop MapReduce framework, if Hadoop MapReduce version 2.0 or newer is already installed on your cluster, you can continue to the *Hadoop configuration* section to make sure the advised configuration is enabled. Halvade uses GATK, which requires a specific version of Java, version 3.7 requires Java v1.8. To make sure GATK works as expected the correct version of Java needs to be installed on every node in the cluster and set as the default Java instance, in Ubuntu use these commands:

```
sudo apt-get install openjdk-8-jre
sudo update-alternatives --config java
```

A.4.1 Single node

To run Hadoop on a single node, it is advised to install Hadoop in pseudo-distributed mode. The following instructions are based on this[6] tutorial and can be used for additional information. Hadoop requires ssh and rsync to run, to install these on your system, run these commands (on Ubuntu):

```
sudo apt-get install ssh rsync
```

[6]https://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-common/SingleCluster.html

Download and unzip the Hadoop distribution (here 2.7.2):

```
wget http://www.eu.apache.org/dist/hadoop/common/hadoop-2.7.2/hadoop
  -2.7.2.tar.gz
tar -xvf hadoop-2.7.2.tar.gz
```

To configure the Hadoop installation to run in pseudo-distributed mode edit these files as follows, creating the file or replacing the line if necessary:

`etc/hadoop/hadoop-env.sh`:

```
export JAVA_HOME=/your/java/bin/directory
```

`etc/hadoop/core-site.xml`:

```
<configuration>
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://localhost:9000</value>
    </property>
</configuration>
```

`etc/hadoop/hdfs-site.xml`:

```
<configuration>
  <property>
      <name>dfs.replication</name>
      <value>1</value>
  </property>
</configuration>
```

`etc/hadoop/mapred-site.xml`:

```
<configuration>
  <property>
      <name>mapreduce.framework.name</name>
      <value>yarn</value>
  </property>
</configuration>
```

`etc/hadoop/yarn-site.xml`:

```
<configuration>
  <property>
      <name>yarn.nodemanager.aux-services</name>
      <value>mapreduce_shuffle</value>
  </property>
</configuration>
```

Additionally we need to make sure that that the node can make a passwordless connection to localhost with ssh, check if `ssh localhost` works without a password. If this isn't the case run the following commands:

```
ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa
cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys
chmod 0600 ~/.ssh/authorized_keys
```

Now we need to format the NameNode and start the HDFS and Yarn services, do this as follows:

```
bin/hdfs namenode -format
sbin/start-dfs.sh
sbin/start-yarn.sh
bin/hdfs dfs -mkdir /user
bin/hdfs dfs -mkdir /user/<username>
```

Now Hadoop can be run from the `bin/hadoop` command and for ease of use this directory can be added to the `PATH` variable by adding this line to your `.bashrc` file:

```
export PATH=$PATH:/hadoop/install/dir/bin
```

After the *Hadoop configuration* has been updated to run Halvade optimally on your node, the services will need to be restarted. To restart the pseudo-distributed Hadoop environment run these commands:

```
sbin/stop-dfs.sh
sbin/stop-yarn.sh
sbin/start-dfs.sh
sbin/start-yarn.sh
```

A.4.2 Multi node

For the Hadoop installation on a multi node cluster, we refer to the manual given by Cloudera to install CDH 5 or later and configure the Hadoop cluster. You can find this detailed description online here[7].

A.4.3 Hadoop configuration

After Hadoop is installed, the configuration needs to be updated to run Halvade in an optimal environment. In Halvade, each task processes a portion of the input data. However, the execution time can vary to a certain degree. For this, the task timeout needs to be set high enough, in `mapred-site.xml` change this property to 30 minutes or more:

```
<property>
  <name>mapreduce.task.timeout</name>
  <value>1800000</value>
 </property>
```

The Yarn scheduler needs to know how many cores and how much memory is available on the nodes, this is set in `yarn-site.xml`. This is very important for the number of tasks that will be started on the cluster. In this example, nodes with 128 GBytes of memory and 24 cores are used. Because some of the tools used benefit from the hyperthreading capabilities of a CPU, the vcores is set to 48 if hyperthreading is available:

```
<property>
  <name>yarn.nodemanager.resource.memory-mb</name>
  <value>131072</value>
```

[7]http://www.cloudera.com/content/cloudera/en/documentation/cdh5/v5-0-0/CDH5-Installation-Guide/cdh5ig_cdh5_install.html

```
  </property>
 <property>
    <name>yarn.nodemanager.resource.cpu-vcores</name>
    <value>48</value>
  </property>
 <property>
    <name>yarn.scheduler.maximum-allocation-mb</name>
    <value>131072</value>
  </property>
 <property>
    <name>yarn.scheduler.minimum-allocation-mb</name>
    <value>512</value>
  </property>
 <property>
    <name>yarn.scheduler.maximum-allocation-vcores</name>
    <value>48</value>
  </property>
 <property>
    <name>yarn.scheduler.minimum-allocation-vcores</name>
    <value>1</value>
  </property>
```

After this, the configuration needs to be pushed to all nodes and certain running services restarted. On a multi node cluster the services running on different nodes need to be restarted after distributing the configuration files, these following commands assume a CDH 5 installation according to the guide shown before:

```
scp *-site.xml myuser@myCDHnode-<n>.mycompany.com:/etc/hadoop/conf.
   my_cluster/
```

On the ResourceManager run:

```
sudo service hadoop-yarn-resourcemanager restart
```

On every NodeManager run:

```
sudo service hadoop-yarn-nodemanager restart
```

On the JobHistory server run:

```
sudo service hadoop-mapreduce-historyserver restart
```

For the RNA-seq pipeline, the memory check needs to be disabled because Halvade uses multiple instances of the STAR aligner when aligning the reads. The genome index files are first loaded into shared memory so every instance can access this instead of loading the reference itself. However, due to the way Hadoop checks physical memory, which includes the shared memory, this check should be disabled. To do this, add these properties to the `yarn-site.xml` file.

```
<property>
   <name>yarn.nodemanager.vmem-check-enabled</name>
   <value>false</value>
 </property>
<property>
   <name>yarn.nodemanager.pmem-check-enabled</name>
   <value>false</value>
 </property>
```

A.4.4 Intel's Hadoop Adapter for Lustre

When using Lustre as the file system instead of HDFS, using Intel's adapter for Lustre will increase the performance of Halvade. To enable the Adapter for Lustre you need to change some configurations in your Hadoop installation. In `core-site.xml` you need to point to the location of Lustre and set the Lustre FileSystem class, if Lustre is mounted on `/mnt/lustre/`, add these to the file:

```
<property>
  <name>fs.defaultFS</name>
  <value>lustre:///</value>
 </property>
<property>
  <name>fs.lustre.impl</name>
  <value>org.apache.hadoop.fs.LustreFileSystem</value>
 </property>
<property>
  <name>fs.AbstractFileSystem.lustre.impl</name>
  <value>org.apache.hadoop.fs.LustreFileSystem$LustreFs</value>
 </property>
<property>
  <name>fs.root.dir</name>
  <value>/mnt/lustre/hadoop</value>
 </property>
```

Additionally, you need to set the Shuffle class in `mapred-site.xml`:

```
<property>
  <name>mapreduce.job.map.output.collector.class</name>
  <value>org.apache.hadoop.mapred.SharedFsPlugins$MapOutputBuffer</
   value>
 </property>
<property>
  <name>mapreduce.job.reduce.shuffle.consumer.plugin.class</name>
  <value>org.apache.hadoop.mapred.SharedFsPlugins$Shuffle</value>
 </property>
```

After adding these settings to the configuration, the files need to be pushed to all nodes again and all services restarted, see above. Additionally the jar containing Intel's Adapter for Lustre should be available on all nodes and added to the classpath of Hadoop. To do this you can find the directories that are currently in your hadoop classpath and add the jar to one of these on every node. To find the directories, run this command:

```
hadoop classpath
```

A.5 Uploading the references

The reference data needs to be available to all nodes in the cluster, which is why they should be available on the distributed file system. When running Halvade, the references will be copied to local scratch on every node when they need to be accessed to increase the performance of subsequent accessing of the file.

Note: The reference files shouldn't be uploaded to the distributed file system if a single node Hadoop environment is used. The tool would download them to local scratch to use. Instead we put the files on local scratch and add some additional files so that Halvade can find the correct references. Additionally, the `refdir` option should be set that points to the directory with all reference files when running Halvade. There are four files that are used to find the corresponding reference files and directories, these should be added to correspond with the reference names:

```
    touch ucsc.hg19.bwa_ref
    touch ucsc.hg19.gatk_ref
    touch STAR_ref/.star_ref
    touch dbsnp/.dbsnp
```

A.5.1 HDFS

The reference files need to be copied to the HDFS so that Halvade can distribute them to every node to be used locally. Here we will create a directory on HDFS where all the files will be collected, execute the following commands to do this:

```
hdfs dfs -mkdir -p /user/ddecap/halvade/ref/dbsnp
hdfs dfs -put ucsc.hg19.* /user/ddecap/halvade/ref/
hdfs dfs -put dbsnp/dbsnp_138.hg19.* /user/ddecap/halvade/ref/dbsnp/

# for the RNA pipeline copy the STAR ref:
hdfs dfs -put STAR_ref/ /user/ddecap/halvade/ref/
```

A.5.2 Amazon S3

To copy the files to Amazon AWS with the terminal the AWS Command Line Interface needs to be installed using this[8] documentation. If the bucket you want to use is called `halv_bucket`, execute the following commands:

```
aws s3 cp ./ s3://halv_bucket/user/ddecap/halvade/ref/ --include "
    ucsc.hg19.*"
aws s3 cp dbsnp/ s3://halv_bucket/user/ddecap/halvade/ref/dbsnp/ --
    include "dbsnp_138.hg19.*"

# for the RNA pipeline copy the STAR ref:
aws s3 cp STAR_ref/ s3://halv_bucket/user/ddecap/halvade/ref/ --
    recursive
```

A.5.3 GPFS & Lustre

Typically GPFS or Lustre are mounted on the directory on every node, the reference files simply need to be copied to that directory. If `/mnt/dfs` is the mounted distributed file system, execute the following commands:

[8]http://docs.aws.amazon.com/cli/latest/userguide/cli-chap-welcome.html

```
mkdir -p /mnt/dfs/halvade/ref/dbsnp
cp ucsc.hg19.* /mnt/dfs/halvade/ref/
cp -r dbsnp/dbsnp_138.hg19.* /mnt/dfs/halvade/ref/dbsnp/

# for the RNA pipeline copy the STAR ref:
cp -r STAR_ref/ /mnt/dfs/halvade/ref/
```

A.6 Halvade Preprocessing

The Halvade Uploader will preprocesses the FASTQ files, this will interleave the paired-end reads and split the files in pieces of 60MByte by default (this can be changed with the **-size** option). The Halvade Uploader will automatically upload these preprocessed files to the given output directory on either local scratch, GPFS, HDFS, Amazon S3 or any other distributed file system.

Note: This step is not required if the input file is an aligned BAM file.

A.6.1 Performance

For better performance it is advised to increase the Java heap memory for the Hadoop command, e.g. for 32GB:

```
export HADOOP_HEAPSIZE=32768
```

A.6.2 Synopsis

```
hadoop jar HalvadeUploaderWithLibs.jar -1 /dir/to/input.manifest -O /
    halvade/out/ -t 8
hadoop jar HalvadeUploaderWithLibs.jar -1 /dir/to/reads1.fastq -2 /
    dir/to/reads2.fastq -O /halvade/out/ -t 8
hadoop jar HalvadeUploaderWithLibs.jar -1 /dir/to/input.manifest -O
    s3://bucketname/halvade/out/ -profile /dir/to/credentials.txt -t 8
```

The manifest file contains per line a pair of files (reads 1 and reads 2) separated by a tab:

```
/path/to/file1_reads1.fq.gz /path/to/file1_reads2.fq.gz
/path/to/file2_reads1.fq.gz /path/to/file2_reads2.fq.gz
```

A.6.3 options

The options are shown in table A.1 and table A.2.

A.7 Example datasets

The input data for these pipelines typically consist of either 2 FASTQ files for paired-end reads or a BAM file containing already aligned reads.

Table A.1: Halvade Uploader required options.

-1 STR	Manifest/Input file. This string gives the absolute path of the manifest file or the first input FASTQ file. This manifest file contains a line per file pair, separated by a tab: **/dir/to/fastq1.fastq /dir/to/fastq2.fastq**. If this is equal to '-' then the fastq reads are read from standard input.
-O STR	Output directory. This string gives the directory where the output files will be put.

Table A.2: Halvade Uploader optional options.

-2 STR	Input file 2. This gives the second pair of paired-end reads in a FASTQ file.
- -dfs	Input on a DFS. This enables reading data from a distributed file system like HDFS and Amazon S3.
-i	Interleaved. This is used when one FASTQ input file is given, the input file is assumed to have both pairs of paired-end reads and the reads are interleaved.
- -lz4	Lz4 compression. This enables lz4 compression, this is faster than gzip but will require more disk space. The lz4 compression library needs to be enabled in the Hadoop distribution for this to work.
-p, - -profile STR	AWS profile. Gives the path of the credentials file used to access S3. This should have been configured when installing the Amazon EMR Command Line Interface. By default this is `~/.aws/credentials`.
-s, - -size INT	Size. This sets the maximum file size (in bytes) of each interleaved file [60MB].
–snappy	Snappy compression. This enables snappy compression, this is faster than gzip but will require more disk space. Snappy requires less disk space than lz4 and is comparable in compression speed. The snappy compression library needs to be enabled in the Hadoop distribution for this to work.
- -sse	Server side encryption. Turns on server side encryption (SSE) when transferring the data to the Amazon S3 storage.
-t INT	Threads. This sets the number of threads used to preprocess the input data. Performance will be limited if the heap memory isn't sufficient.

A.7.1 Whole genome sequencing sample

The whole genome sequencing sample is the NA12878 dataset, this dataset is typically used in similar benchmarks and papers. This dataset consists of 1.5 billion paired-end reads of 100 base pairs in length. This translates into a 50x coverage. Execute the following commands to download and preprocess the data:

```
wget ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR194/ERR194147/ERR194147_1.
  fastq.gz
wget ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR194/ERR194147/ERR194147_2.
  fastq.gz
export HADOOP_HEAPSIZE=32768
hadoop jar HalvadeUploaderWithLibs.jar -1 ERR194147_1.fastq.gz -2
  ERR194147_2.fastq.gz \
-O /user/ddecap/halvade/wgsin/ -t 16
```

A.7.2 RNA-seq sample

The RNA-seq example dataset is found in the **encode project** under the **SK-MEL-5** experiment. The **ENCSR201WVA** dataset provides both paired FASTQ files and aligned BAM files. In this example we will download a single replicate of the **ENCBS524EJL** bio sample available in paired FASTQ files. To download and preprocess the FASTQ files run these commands in the terminal:

```
wget https://www.encodeproject.org/files/ENCFF005NLJ/@@download/
  ENCFF005NLJ.fastq.gz
wget https://www.encodeproject.org/files/ENCFF635CQM/@@download/
  ENCFF635CQM.fastq.gz
export HADOOP_HEAPSIZE=32768
hadoop jar HalvadeUploaderWithLibs.jar -1 ENCFF005NLJ.fastq.gz -2
  ENCFF635CQM.fastq.gz \
-O /user/ddecap/halvade/rnain/ -t 16
```

A.8 Run Halvade

To run Halvade the GATK binary needs to be added to the `bin.tar.gz` file by executing the following commands:

```
tar -xvf bin.tar.gz
rm bin.tar.gz
cp GenomeAnalysisTK.jar bin/
tar -cvzf bin.tar.gz bin/*
```

Similar to the reference files, this file needs to be uploaded to the distributed file system if a cluster with more than one node is used. Run this command when using HDFS as distributed storage:

```
hdfs dfs -put bin.tar.gz /user/ddecap/halvade/
```

A.8.1 Configuration

Halvade is started using a python script `runHalvade.py`, this script reads the configuration from a file given in the first argument. This file contains all options you intend to give to the Halvade command. The configuration file uses a = character when a value needs to be provided to the option and the value should be quoted if it is a string. To add an options without arguments add a new line with just the option name. Commented lines, starting with #, are ignored by the script. The `example.config` file contains the most basic example, in which the necessary options are provided. The file looks like this:

```
N=5
M=128
C=24
B="/user/ddecap/halvade/bin.tar.gz"
D="/user/ddecap/halvade/ref/dbsnp/dbsnp_138.hg19.vcf"
R="/user/ddecap/halvade/ref/ucsc.hg19"
I="/user/ddecap/halvade/in/"
O="/user/ddecap/halvade/out/"
```

To run the RNA-seq pipeline two additional options need to be provided:

```
star="/user/ddecap/halvade/ref/STAR_ref/"
rna
```

If the nodes in the cluster have hyperthreading enabled, add the `smt` option to improve performance. To run the pipeline with a bam file as input, add the `bam` option.

Note: When running the Halvade job on a single node, it is not required to upload the reference files to the distributed file system. However, the input data should still be preprocessed with the Halvade Uploader tool and put on the distributed file system. Running Halvade in this setup, some additional files should be present to allow Halvade to find the references, these should have been added in a previous step. To show Halvade where to find the reference files, add the directory where the required files can be found like this:

```
refdir="/user/ddecap/halvade/ref/"
```

This folder is expected to be on local scratch or a mounted distributed file system so this doesn't require a prefix.

A.8.2 Run

When all desired configuration for Halvade have been added to the config file, simply run the following command to start Halvade:

```
python runHalvade.py
```

This will start Halvade, which in turn will start the necessary Hadoop jobs. The script will return the ID of the process (*PID*) which is used in the filenames to store the standard out and error logs, **halvade*PID*.stdout** and **halvade*PID*.stderr**. The output of Halvade will be a single VCF file which can be found in the subdirectory `merge` of the provided output directory.

A.8.3 Amazon AWS

To run Halvade on an Amazon EMR cluster, the AWS Command Line Interface needs to be installed, installation instructions can be found here[9]. To run Halvade on Amazon EMR, some additional configurations need to be added so the `runHalvade.py` script knows Halvade should be started on Amazon EMR. As the Halvade jar isn't available on every node yet, this needs to be uploaded to Amazon S3 first. Similarly, the **bootstrap** script, which creates the `halvade/` directory on the mounted SSD's for intermediate data, needs to be uploaded as well.

```
aws s3 cp  HalvadeWithLibs.jar  s3://halv_bucket/user/ddecap/halvade/
  ref/
aws s3 cp  halvade_bootstrap.sh  s3://halv_bucket/user/ddecap/halvade/
  ref/
```

To use Halvade on Amazon EMR an AMI version of 3.1.0 or newer should be used. Add the following EMR configuration to run Halvade on Amazon EMR:

```
emr_jar="s3://halv_bucket/user/ddecap/halvade/HalvadeWithLibs.jar"
emr_script="s3://halv_bucket/user/ddecap/halvade/halvade_bootstrap.sh
  "
emr_type="c3.8xlarge"
emr_ami_v="3.1.0"
tmp="/mnt/halvade/"
emr_s3logging="s3://halv_bucket/user/ddecap/halvade/logs/"
```

The `tmp` option is updated to point to the local SSD's on the Amazon EMR nodes, which are mounted in the `/mnt/` folder. The `emr_s3logging` argument is used to save all Hadoop master and task logs for debugging purposes.

Additionally, to run the script, the default EMR roles need to be created in order to work, run this command:

```
aws emr create-default-roles
```

A.9 Halvade Options

Any directory given in the command line option needs to be accessible by all nodes. This can be either on HDFS, GPFS, Amazon S3 or any other distributed file system. When using one node this can also be local scratch. If no prefix is used, HDFS will be used by default. However, the default file system can be changed with the `fs.defaultFS` configuration of Hadoop. When this is changed the directories can

[9]http://docs.aws.amazon.com/cli/latest/userguide/cli-chap-welcome.html

simply be given without any prefix, else a prefix `file:///` needs to be given for local scratch and mounted GPFS directories. For data stored on S3 when using the Amazon EMR service, the directories need to contain the bucket name as a prefix, e.g. `S3://bucketname/`. A script `runHalvade.py` is provided to gather all options in a simple config file which then calls Halvade with all provided options. The options are shown in table A.3, table A.4, table A.5, table A.6, table A.7 and table A.8.

Table A.3: Halvade required options.

-B STR	Binary location. This string gives the location where bin.tar.gz is located.
-D STR	DBSNP file location. This string gives the absolute filename of the DBSNP file, this file needs to be compatible with the reference FASTA file provided by the *-R* option.
-I STR	Input directory. The string points to the directory containing the preprocessed input or BAM file on the used file system.
-O STR	Output directory. This string points to the directory which will contain the output VCF file of Halvade.
-R STR	Reference Location. This string gives the prefix (without .fasta) of the absolute filename of the reference in FASTA format. The corresponding index files, built with BWA, needs to be in this directory having the same prefix as the reference FASTA file. The STAR genome index can be located in a different folder.
-M, - -mem INT	Memory size. This gives the total memory each node in the cluster has. The memory size is given in GB.
-N, - -nodes INT	Node count. This gives the total number of nodes in the local cluster or the number of nodes you want to request when using Amazon EMR. Amazon AWS has a limit of 20 nodes unless the nodes are reserved for an extended period of time.
-C, - -vcores INT	Vcores count. This gives the number of cores that can be used per node on the cluster (to enable simultaneous multithreading use the *- -smt* option).

Table A.4: Halvade optional options, part 1.

-A, - -justalign	Just align. This option is used to only align the data. The aligned reads are written to the output folder set with the *-O* option.

Table A.5: Halvade optional options, part 2.

--aln INT	Select Aligner. Sets the aligner used in Halvade. Possible values are 0 (bwa aln+sampe)[default], 1 (bwa mem), 2 (bowtie2), 3 (cushaw2). Note that these tools need to be present in the bin.tar.gz file.
--bam	Bam input. This option enables reading aligned BAM input, using this will avoid realigning. If a realignment is required, the data needs to be transformed to FASTQ files, shuffled and preprocessed for Halvade.
--bed STR	Bed region. This option uses a BED file to split the genome in genomic regions that will be processed by one reduce task. This is used when feature count is enabled and the bed region give the known gene boundaries to avoid counting double.
--CA STR=STR	Custom arguments. This options allows the tools run with Halvade to be run with additional arguments. The arguments are given in this form: toolname=extra arguments. All options must be correct for the tool in question, multiple arguments can be added by giving a quoted string and separating the arguments with a space. Possible tool names are bwa_aln, bwa_mem, bwa_sampe, star, elprep, samtools_view, bedtools_bdsnp, bedtools_exome, picard_buildbamindex, picard_addorreplacereadgroup, picard_markduplicates, picard_cleansam, gatk_realignertargetcreator, gatk_indelrealigner, gatk_baserecalibrator, gatk_printreads, gatk_combinevariants, gatk_variantcaller, gatk_variantannotator, gatk_variantfiltration, gatk_splitncigarreads.
--combine	Combine VCF. With this option Halvade will combine VCF files in the input directory and not perform variant calling if the relevant files are found. This is done by default after the variant calling.
--count	Count reads. This counts the reads per Halvade region, this is only used for debugging purposes.
--drop	Drop. Halvade will drop all paired-end reads where the pairs are aligned to different chromosomes.
--dryrun	Dry run. This will initialize Halvade, which calculates the task sizes and region sizes of the chromosomes, but Halvade will not execute the Hadoop jobs.
--fbed STR	Filter on bed. This option will enable the reads to be filtered on the given bed file before performing the GATK steps. This is typically used in an exome dataset where only reads in a known bed file are expected.

Table A.6: Halvade optional options, part 3.

--filter_dbsnp	Filter dbsnp. This flag turns on filtering of the dbSNP file before using it in the GATK. This can improve performance in some cases but typically the overhead of converting is too big.
--gff STR	GFF file. This sets the GFF file that will be used by Featurecounts to count the number of reads per exon.
-H, --haplotypecaller	HaplotypeCaller. With this option Halvade will use the HaplotypeCaller tool from GATK instead of the UnifiedGenotyper tool, which is used by default. This is the newer variant caller which is slower but more accurate.
- -id STR	Read Group ID. This string sets the Read Group ID which will be used when adding Read Group information to the intermediate results. [GROUP1]
--illumina	Convert Illumina scores. This Option forces Halvade to convert every base pair quality to the Illumina format.
-J STR	Java. This string sets the location of the Java binary, this file should be present on every node in the cluster. If this is not set Halvade with use the default Java. This can be used if the default Java is 1.6 and GATK requires version 1.7.
--keep	Keep intermediate files. This option enables all intermediate files to be kept in the temporary folder set by tmp. This allows the user to check the data after processing.
--lb STR	Read Group Library. This string sets the Read Group Library which will be used when adding Read Group information to the intermediate results. [LIB1]
--mapmem INT	Map Memory. This sets the memory available for the containers assigned for the map tasks.
--merge_bam	Merge BAM output. With this option set, Halvade will not perform variant calling but only read alignment. All alignments will be merged into 1 output BAM file.
--mpn INT	Maps per node. This overrides the number of map tasks that are run simultaneously on each node. Only use this when the number of map containers per node does not make sense for your cluster.
--elprep	elPrep. Use elPrep in the preprocessing steps, by default Picard is used which is a slower but requires less memory. ElPrep provides a more efficient execution of the preprocessing algorithms.
--pl STR	Read Group Platform. This string sets the Read Group Platform which will be used when adding Read Group information to the intermediate results. [ILLUMINA]

Table A.7: Halvade optional options, part 4.

--pu STR	Read Group Platform Unit. This string sets the Read Group Platform Unit which will be used when adding Read Group information to the intermediate results. [UNIT1]
--redistribute	Redistribute Cores. This is an optimization to better utilize the CPU cores at the end of the map phase, to improve load balancing. Only use when the cores per container is less than 4.
--redmem INT	Reduce Memory. This sets the memory available for the containers assigned for the reduce tasks.
--refdir STR	Reference directory. This sets the reference directory on local scratch, Halvade will use this directory to find existing references on each node. This directory needs to be accessible by all nodes, but can be a local disk or a network disk. Halvade finds the reference files by looking for files in the directory or subdirectory with these suffixes: .bwa_ref, .gatk_ref, .star_ref, .dbsnp. This folder is expected to be on local scratch or a mounted distributed file system so this doesn't require any prefix.
--remove_dups	Remove Duplicates. This will remove the found PCR duplicates in the corresponding step.
--report_all	Report all output. This option will give all VCF output records in the merged output file. By default the VCF record with the highest score will be kept if multiple records are found at the same location.
--rna	RNA pipeline. This options enables Halvade to run the RNA-seq pipeline instead of the default DNA pipeline. This option requires an additional argument *-S* which points to the STAR genome directory.
--rpn INT	Reduces per node. This overrides the number of reduce tasks that are run simultaneously on each node. Only use this when the number of reduce containers per node does not make sense for your cluster.
-S, --star STR	Star genome. This gives the directory of the STAR genome reference.
--scc INT	stand_call_conf. The value of this option will be used for the stand_call_conf when calling the GATK Variant Caller.
--sec INT	stand_emit_conf. The value of this option will be used for the stand_emit_conf when calling the GATK Variant Caller.
--single	Single-end reads. This option sets the input to be single-ended reads. By default, Halvade reads in paired-end interleaved FASTQ files.
--sm STR	Read Group Sample Name. This string sets the Read Group Sample Name which will be used when adding Read Group information to the intermediate results. [SAMPLE1]

Table A.8: Halvade optional options, part 5.

- -smt	Simultaneous multithreading. This option enables Halvade to use simultaneous multithreading on each node.
- -stargtf STR	GFF for STAR. This option point to the GFF/GTF file to be used when rebuilding the STAR genome, this can improve accuracy when finding splice sites.
- -tmp STR	Temporary directory. This string gives the location where intermediate files will be stored. This should be on a local disk for every node for optimal performance.
- -update_rg	Update read group. This forces the readgroup to be updated to the one provided by the options, even if the input is read from a BAM file with a read group present.
-v INT	Verbosity. This sets the verbosity level for debugging, default is [2].

A.10 Troubleshooting

A.10.1 Find the logs to solve Halvade errors

If Halvade doesn't finish due to an error, the error itself is printed in the output of the Hadoop command. However, more information can be found in the individual task stderr logs of the MapReduce job. The location of these log files is set in the MapReduce settings. Typically these are stored at `$yarn.log.dir/userlogs` or if the `YARN_LOG_DIR` environment is set under `$YARN_LOG_DIR/userlogs`. It's highly likely that all reduce tasks give a similar result, so look at the stderr log of any reduce task. This log will show where Halvade is running into problems. If it isn't clear from the log, try to run the last command, with the error, manually. The exact command should be printed in the log as an array of strings, run this command with the shown option.

A.10.2 Halvade with BAM input seems stuck at the MarkDuplicates step

If the stderr log of a reduce task shows it started the MarkDuplicates file but didn't finish in a considerable time. Then it is highly likely it is finding incorrect aligned reads and giving error messages that slow the process to the point where it seems stuck. If this is the case, look at the header file of the BAM file and the fasta dictionary file, if the contig order is different then this is the source of the problem. Halvade assigns the contig by index in the sequence dictionary when reading the bam files and this causes wrong matching between steps. To fix this, the sequence dictionary of the bam files needs to be reordered so that it's the same as the dictionary file. This can be done with the following command, after which the output file is used as input for Halvade.

```
java -jar picard.jar ReorderSam I=input.bam O=output.bam R=reference.
    fasta
```