

WARSAW UNIVERSITY OF TECHNOLOGY

ENGINEERING AND TECHNOLOGY

Information and Communications Technology

PhD Thesis

Distributed algorithms

and computational methods for scalable processing
of high-throughput sequencing data

Marek Wiewiórka, MA, BSc

Supervisor:

Tomasz Gambin, PhD, DSc, Associate Professor

Warsaw 2023

Acknowledgements

First and foremost, I wish to express my deepest gratitude to my supervisor and my friend, Prof. Tomasz Gamin, for supporting and guiding me throughout my pursuit of this doctorate. I would also like to thank my previous supervisors - Prof. Piotr Gawrysiak and Prof. Henryk Rybiński - for helping me take the first steps towards my Dissertation. My sincere thank you goes to Dr. Michał Okoniewski, who introduced me to bioinformatics, when I was on my PhD scholarship in Switzerland, and immensely assisted me with my first research paper. I am also grateful to all my Colleagues from the Institute of Computer Science at the Warsaw University of Technology, with whom I have had the privilege to work over the last years.

I want to thank Prof. Paweł Stankiewicz for his insightful comments and suggestions that made the content of this thesis more understandable.

Many of the ideas from this Dissertation would not come into life without Agnieszka Szmurło. I am truly grateful for our wonderful collaboration, all inspiring discussions, hours-long brainstorming sessions and then... starting from scratch all over again but in a different way – thanks a lot, Aga!

My special thanks goes to Rafał Małanij, who I can always count on and who brings tons of positive energy and „we-can-do” attitude to our small research team.

I am also grateful to all my GetInData Colleagues for their friendship, support and endless inspiration to tackle Big Data and Cloud Computing challenges.

I want to thank my lovely wife Dorota for being patient, supportive and helpful in numerous ways (many of which I did not even acknowledge, for sure!).

Last but not least, I want to thank my dearest Parents who planted a seed for my interest in science and engineering from the very first LEGO set they bought me, and who never stopped believing in me.

Warsaw, Poland, March 2023

Marek Wiewiórka

Abstract

Recent advances in high-throughput sequencing (HTS) have contributed to an unprecedented growth in the amount of generated multiomics datasets. Reaching the critical milestone of \$1,000 in the cost for a complete genome sequence of a single individual (whole genome sequencing, WGS) in the mid 2010s opened the door to many population-scale or national genome initiatives, such as Genomics England’s ‘100 000 Genomes Project’ or 1000 Polish Genomes.

Nonetheless, many popular bioinformatics methods for secondary and tertiary data analyses exhibit high computational complexity. To compound the situation, a great number of the existing genomic analysis tools and algorithms are intrinsically sequential and incapable of exploiting the power of distributed computing. In particular, truly scalable methods for common genomic operations, such as calculating depth of coverage, summarizing short reads in a form of pileup and joining datasets using interval intersections, have not been yet proposed.

Furthermore, very little studies have addressed the challenges of designing genomic cloud platforms for distributed processing and analysis of HTS data. Equally, the idea of unified declarative programming approaches for expressing genomic operations using Structured Query Language (SQL) has not gained enough traction either.

This Dissertation aims at bridging those gaps by presenting a Genomic Data Lakehouse concept together with the SeQuiLa project that implements novel scalable methods for the aforementioned computationally intensive genomic operations. It is organized in a series of six publications proceeded by an introduction outlining the HTS data analysis challenges and current advances.

Keywords: Big Data, Distributed computing, Cloud computing, High-throughput sequencing, Data Lakehouse architecture

Streszczenie

Najnowsze odkrycia w sekwencjonowaniu wysokoprzepustowym (HTS) przyczyniły się do bezprecedensowego wzrostu ilości generowanych danych multiomicznych. Przekroczenie w połowie pierwszej dekady XXI w. historycznego progu, jakim był koszt kompletnej sekwencji genomu człowieka (sekwencjonowanie całego genomu, WGS) poniżej tysiąca dolarów, otworzyło drzwi do wielu narodowych inicjatyw genomowych, takich jak 100,000 Genomes Project w Wielkiej Brytanii, czy 1000 Polskich Genomów.

Niemniej jednak, wiele popularnych metod bioinformatycznych do analizy drugo- i trzeciorzędowej wykazuje dużą złożoność obliczeniową. Większość istniejących narzędzi i algorytmów analizy genomicznej jest z natury sekwencyjna i nie jest w stanie w pełni wykorzystać możliwości rozproszonego modelu obliczeń, co czyni sytuację jeszcze trudniejszą. W szczególności, jak dotąd nie zaproponowano prawdziwie skalowalnych metod dla typowych operacji genomicznych, takich jak obliczanie głębokości pokrycia, podsumowywanie krótkich odczytów (ang. *pileup*) i łączenie zbiorów danych za pomocą przecięć przedziałowych.

Ponadto, znikoma liczba badań podejmuje temat wyzwań związanych z projektowaniem genomicznych platform chmurowych do rozproszonego przetwarzania i analizy danych pochodzących z HTS. Podobnie mało uwagi poświęcono idei wykorzystania zunifikowanego podejścia, realizującego deklaratywny paradymat programowania do wyrażania operacji genomicznych przy użyciu języka Structured Query Language (SQL).

Niniejsza rozprawa ma na celu wypełnienie tych luk poprzez przedstawienie koncepcji Genomicznej Platformy Danych typu Lakehouse oraz zaprezentowanie projektu SeQuiLa, implementującego nowatorskie skalowalne metody dla wyżej wymienionych, obliczeniowo wymagających, operacji genomicznych. Na poniższą pracę składa się seria sześciu publikacji poprzedzonych wstępem, w którym Autor opisuje wyzwania i najnowsze osiągnięcia w dziedzinie analizy danych genomicznych.

Słowa kluczowe: Big Data, obliczenia rozproszone, obliczenia chmurowe, sekwencjonowanie wysokoprzepustowe, architektura Data Lakehouse

Contents

1	Introduction	9
1.1	From MapReduce to Cloud Computing and Data Lakehouse era	10
1.2	HTS sequencing overview	13
1.2.1	Primary analysis	13
1.2.2	Secondary analysis	14
1.2.3	Tertiary analysis	15
1.3	Applications of Big Data techniques to HTS data analysis	15
1.4	Challenges leading to Genomic Data Lakehouse	20
1.5	Aim and Research Theses	23
1.6	Publications constituting this Dissertation	23
2	Main scientific contribution of the Author of the Dissertation	27
2.1	Genomic data distributed processing in data lakes [P1]	28
2.2	Genomic data querying in distributed data warehouses [P2]	29
2.3	Genomic Data Lakehouse	31
2.3.1	Distributed machine learning framework for genomic data[P3] . . .	31
2.3.2	Scalable range joins[P4]	31
2.3.3	Scalable depth of coverage and pileup calculations[P5], [P6] . . .	33
3	Scientific achievements	37
3.1	Source code repositories	37

3.2	Own research grants	38
3.3	Participation in research grants	38
3.4	Other research publications	38
3.5	Book chapters	39
3.6	Conference talks	39
3.7	Conference posters	40
3.8	Prizes and awards	40
4	Conclusions and future work	43
5	Bibliography	45
6	Copies of the publications constituting the Dissertation	77
6.1	SparkSeq: Fast, scalable and cloud-ready tool for the interactive genomic data analysis with nucleotide precision [P1]	79
6.2	Benchmarking distributed data warehouse solutions for storing genomic variant information [P2]	109
6.3	Scalable Framework for the Analysis of Population Structure Using the Next Generation Sequencing Data [P3]	125
6.4	SeQuiLa: an elastic, fast and scalable SQL-oriented solution for processing and querying genomic intervals [P4]	135
6.5	SeQuiLa-cov: A fast and scalable library for depth of coverage calculations [P5]	138
6.6	Cloud-native distributed genomic pileup operations [P6]	145
7	Co-Authors' statements	155

CHAPTER 1

Introduction

Since the unveiling of the double helix structure of deoxyribonucleic acid (DNA)[1] in 1953, scientists around the world have been developing methods for *DNA sequencing*, i.e. decoding the order of single nucleotide in various biological molecules at five key omics levels: genome, epigenome, transcriptome, proteome and metabolome[2]. Over the past 15 years, there has been exponential growth in sequencing capacity[3] especially thanks to the advances in High-throughput sequencing (HTS) technologies of the second (also called next-generation sequencing, NGS) and third generation[4]. Reaching the important milestone of \$1000[5] for full sequence of an individual genome (whole genome sequencing, WGS) in the mid 2010s has opened the door to many population-scale, *national or digital genome* initiatives such as Genomics England's '100 000 Genomes Project'[6], NIH All of Us Research Program[7] or 1000 Polish Genomes[8]. As long as the trend continues, the estimated storage space required just for human genomes by the year 2025 may reach the level of 2-40 exabytes[9].

Nonetheless, many studies [2, 10] have indicated the development of robust and scalable methods of HTS data analysis to be the key enabler of further advances in the genomics domain. Moreover, in the same publication, application of high-performance computing (HPC) clusters, graphic processing units (GPUs), field-programmable gate arrays (FPGAs), Big Data technologies and Cloud Computing have been proposed as

possible solutions. This is consistent with more recent studies[11], in which the authors have concluded that in the era with no Dennard scaling, along with reduced Moore’s Law, further exploitation of parallel computing adhering to Amdahl’s Law is the only viable remedy for achieving higher rates of performance improvement. Unfortunately, in many bioinformatics analyses the underlying algorithms and utilized file formats have not been planned for parallel or distributed execution model, making the shift towards Big Data programming paradigms challenging[12].

Consequently, this Dissertation aims at exploring the application of distributed computing methods for the design and implementation of scalable algorithms for efficient querying and analysing of HTS datasets. In particular, it presents new algorithmic strategies for partitioning datasets that store aligned sequences and novel distributed methods for depth of coverage, genomic pileup calculations and intervals intersections. These operations constitute the building blocks of the crucial bioinformatics pipelines including but not limited to: single nucleotide variants (SNVs), copy number variants (CNVs) based on read-depth methods[13] and structural variants (SVs) calling as well as variant annotation[14, 15, 16]. The application of *Infrastructure as Code* (IaC)[17, 18] approach to addressing the problem of reproducible and portable genomic scientific computing has been discussed as well. At the same time, this Dissertation also presents scalable computing methods in the much broader context of the cloud-based *Genomic Data Lakehouse* [19] architecture, where they naturally fit in. The combination of the above elements may help to make genomic distributed analysis in the cloud environment accessible to a wider group of researchers.

1.1 From MapReduce to Cloud Computing and Data Lakehouse era

The considerable growth of interest in application of dedicated techniques to large-scale data processing has been often associated with the popularization of MapReduce[20] distributed programming paradigm in the late 2000s. However, the real wave of its popularity has begun with the introduction of a production-level open source implementation of the whole Big Data ecosystem, i.e. Apache Hadoop[21]. It was built upon three layers:

- distributed file system, Hadoop Distributed File System (HDFS)[22], inspired by

the Google File System[23],

- computing resources manager – Yet Another Resource Negotiator (YARN)[24],
- data processing engine – Hadoop MapReduce Framework.

Apache Hadoop has revolutionized the way companies and researchers process and store large datasets in a form of *Data Lake*[25]. However, MapReduce programming paradigm (i.e. it was also a processing engine in Hadoop) was considered to be too low-level and too difficult both to reuse and maintain. Furthermore, it was not well suited for running *ad hoc* analysis – since even for common operations like e.g. projection, filtering or sorting custom code had to be written in languages such as Java [26]. The productivity of data analysts was seriously limited, as most of them were used to tabular data abstraction[27] and ability to query data using widely adopted declarative programming paradigm such as, based on relational algebra principles, Structured Query Language (SQL) [28]. Two pioneers in the Big Data revolution – Facebook and Yahoo! – almost at the same time proposed remedies in the form of two higher level languages in the spirit of SQL that could be compiled under the hood into Hadoop MapReduce jobs and executed on Hadoop-based computing cluster. Pig Latin[29] open-sourced by Yahoo! combines declarative data querying heavily inspired by SQL, support for User Defined Functions (UDFs) to accommodate specialized data processing tasks and procedural programming that resembles specifying a query execution plan (i.e. a dataflow graph). HiveQL[30] proposed by Facebook, on the other hand, is in-fact an SQL dialect that together with a table metadata catalog – Metastore builds up a fully-fledged distributed *Data Warehouse*[31] on top of a Hadoop cluster.

It is worth mentioning that there were also other attempts to increase adoption of Hadoop-based data processing among users interested in rapid application development. First group of projects including Happy, Dumbo and Pydoop[32] as notable examples tried to add support for MapReduce computations in Python, a language that is widely considered to have substantially lower barrier of entry[33] than Java or C++. On the other hand, Hadoop Streaming[34] included in the Hadoop distribution allows users to specify arbitrary executables or scripts written in any programming language in order to accomplish the map and reduce functions through the standard input and output.

Finally, the Apache Mahout and Cascading[35] projects aimed at streamlining machine

learning and data processing workflows development respectively by abstracting most of the MapReduce programming complexity and introducing a high-level Java application programming interface (API).

The Apache Spark[36, 37] distributed framework draws heavily on the experience of the aforementioned projects. It attempts to address out-of-the-box their shortcomings by providing end user with a high-level functional programming API for Scala, Java, Python and R languages with relational, SQL-oriented declarative way of querying data. Furthermore, it is shipped with additional modules that implement many popular graph and machine learning algorithms[38]. Last but not least, it relies on the in-memory computational model that makes it especially efficient in interactive data processing[39].

Current advances in Cloud Computing environments have simplified the deployment of Big Data workloads when compared to the on-premise Hadoop clusters, and can be summarized under the following three categories: Kubernetes-based[40, 41], fully managed Hadoop services[42] and serverless services[43] like e.g. Google Cloud Dataproc Serverless or Amazon Elastic Map Reduce (EMR) Serverless. Unlike Hadoop computing model that was designed to run on clusters of commodity hardware and rely on the *data locality*[44] concept, all of the above scenarios implement decoupling of compute and storage to support independent scalability of both layers, provide a better performance-cost ratio and better data availability[45].

The aforementioned developments in Big Data gave rise to new paradigms in data architectures such as *Data Lakehouse*[46, 47, 19]. It has been recently proposed to respond to the dominant in the industry two-tier data lake + data warehouse implementation pattern. Its main design principles can be summarised as follows:

- low-cost and efficient storage for very large-scale heterogeneous data, backed by cloud object storage systems (e.g. Azure Data Lake Store (ADLS)[48], Amazon Simple Storage Service (S3)[49], Google Cloud Storage (GCS)[50]) or other distributed file systems (e.g. HDFS),
- first-class support for machine learning and data science workloads with distributed DataFrame-like APIs, such as Koalas [51] project that has been recently merged into Apache Spark main codebase or Snowpark DataFrames available on the Snowflake platform[52],

- state-of-the-art SQL analytical queries performance with the general-purpose extensions to the existing SQL engines such as vectorized, native operators provided by the Photon[53] and Velox[54] projects or research domain-specific like ASTROIDE[55],
- open direct-access file formats such as Parquet, Optimized Row Columnar (ORC)[56], DeltaLake[57], Hudi[58] or Iceberg[59] to enable the above mentioned two types of data access patterns: Business Intelligence, which extracts a small amount of data, and Machine Learning, which usually processes large datasets.

1.2 HTS sequencing overview

Bioinformatics workflows for HTS analysis consist of a number of consecutive phases beginning with biological samples preparation, followed by sequencing, multistage data processing and final results interpretation. HTS enables versatile types of sequencing protocols including genome sequencing (DNA-seq), transcriptome profiling (RNA-seq), DNA-protein interaction assessment (ChIP-seq) and epigenome characterization (ChIP-seq, BS-seq, DNase-seq and FAIRE-seq)[60]. HTS analysis pipelines can differ substantially, depending on which omics level they operate, but from a high-level perspective they are traditionally split into three stages: *primary*, *secondary* and *tertiary* analysis[61]. Figure 1.1 and description below present the HTS data pipeline in the case of DNA-seq.

1.2.1 Primary analysis

The primary analysis is a pipeline stage during which genomic sequences are generated. This step takes place in a sequencing instrument (sequencer) that detects base pairs (bp) from the DNA molecule fragments in the *base calling* process. Depending on the technology used, the source of the measured signal can be synthesis, combinatorial probe anchor synthesis or ligation of a DNA fragment[63]. Moreover, each base call is augmented with a calculated quality score denoting the estimated probability that a given base pair has been wrongly detected. The typical length of short sequencing reads of the second-generation technology equipment, resulting from the base calling step, range from 50 to 300 bp. In the case of the third-generation sequencing machines, reads are much longer – 10–30 kbp genomic libraries are common[64]. The output of this stage is produced in

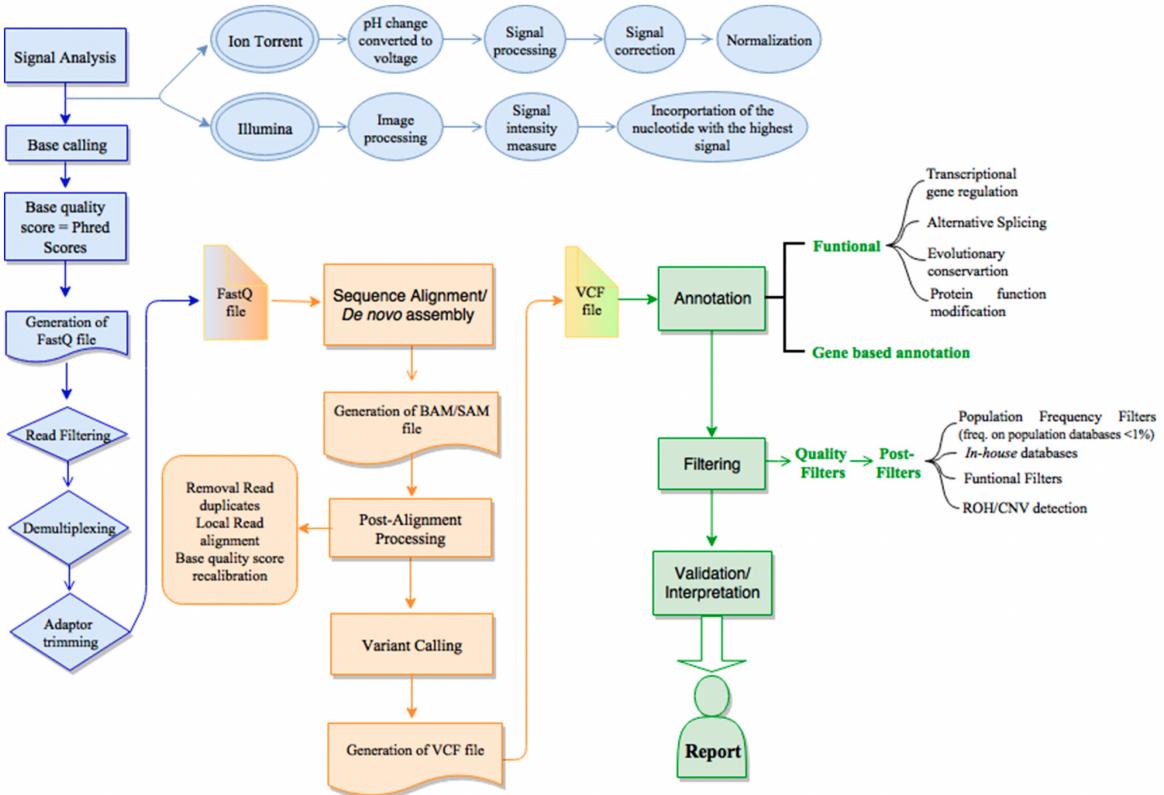


Figure 1.1: An overview of the next generation sequencing (HTS) data pipeline in the case of DNA-seq protocol. The HTS bioinformatics workflow is subdivided in the primary (blue), secondary (orange) and tertiary (green) analysis[62].

a form of a set of text files containing sequenced reads in the FASTQ[65] format. For each read, the following data are stored: (i) the raw sequence symbols; (ii) corresponding quality scores (encoded as American Standard Code for Information Interchange (ASCII) characters); (iii) record identifiers (in instrument-specific data format).

1.2.2 Secondary analysis

The secondary analysis stage encompasses pipelines producing two main categories of results: aligning reads with previously assembled reference sequence of the same species and optionally detecting various discrepancies between them or performing *de novo* assembly of the genome when there has not been one revealed yet[62]. In the latter case, a new reference sequence in a FASTA[66] file format is generated. However, more often the reference genome is already in place and then the study is focused on mapping reads to the genome by a dedicated class of tools called *read aligners* [67]. Typically, they produce the

output in the form of Sequence Alignment Map (SAM) or in its binary representation, BAM file format[68]. An alternative file format CRAM[69] offering lossless reference-based compression as well as optional lossy encoding of quality scores with 40-50% space saving has also been put forward. The final step of the secondary analysis, preceded by post-alignment preprocessing routines, like marking read duplicates and recalibrating base quality scores, is to perform various variant calling, including SNV, CNV and SV. This step results in producing a set of files in variant call format (VCF)[70].

Notably, read alignment, read deduplication, read *pileup* required among others for SNV calling and *depth of coverage* calculations crucial for CNV calling are the most computationally expensive[71, 15], especially in the case of high-coverage WGS samples required for diagnostics[72].

1.2.3 Tertiary analysis

The tertiary analysis is focused on drawing biological insights from the results of the secondary analysis combined with other data sources by applying various statistical and data science methods[73, 74]. In particular, variants of interest can be annotated to determine their impact on genes, transcripts and protein sequence, as well as regulatory regions[75, 76]. Therefore, at this stage of genomic analysis there is a great need for an easy, unified and declarative way of running *ad hoc* analytical queries joining different data sources at scale. However, currently tertiary data analysis is, in majority of cases, still performed with predefined *ad hoc* scripts, typically invoking multiple command line software tools for specific operations that are not suitable for Big Data processing[77].

1.3 Applications of Big Data techniques to HTS data analysis

Since the introduction of Hadoop MapReduce framework, a number of based on it Open Source projects attempted to increase the efficiency of both secondary and tertiary stages of the HTS data analysis[78]. Secondary analysis includes highly computational intensive steps like short reads mapping or variant calling, and this is why most of the efforts in the bioinformatics research community have been mainly focused on developing suitable Big Data strategies that address the growing needs of this stage of data analyses[79].

The list below presents recent notable projects that address the performance challenges of secondary analyses:

- ADAM[12, 80] is a genomic data processing system built on Apache Spark. It offers both a unified storage model based on Apache Avro[81] and Parquet[82] file formats as well as some extensions to SparkSQL enabling procedural, DataFrame oriented and SQL interfaces[37]. It implements distributed algorithms for higher level operations such as calculation depth of coverage and genomic intervals intersection.
- Canolli[83] is an Apache Spark based set of wrappers using Resilient Distributed Dataset (RDD) pipelining mechanism[84] over various bioinformatics tools such as bwa-mem[85], vt[86] or samtools.
- DECA[16] is a horizontally scalable implementation of the XHMM[87] algorithm using the ADAM framework and Apache Spark, that incorporates additional novel optimizations.
- Genome Analysis Toolkit (GATK)[15] is an actively developed structured programming framework designed for efficient and robust analysis of NGS data, SNV calling pipelines in particular, based on MapReduce programming paradigm gradually migrated to Apache Spark.

Nevertheless, integrative analysis of multi-omics and other data sources that is the subject of tertiary analysis is also considered computationally demanding[88, 74]. Along with the aforementioned paradigm shift towards declarative[89] way of genomic data processing, the tools can be further subdivided into three groups [90]. The first group, like SAMtools and BCFtools[91], requires programming (e.g. shell scripting) expertise to manipulate and query the aligned reads or variants using command line interface (CLI). They are designed for a single-node execution model only. With the development of the underlying HTSlib[92] library, they started supporting multi-threaded encoding and decoding of BAM (and SAM) files, offering performance that is comparable or, in some cases, better than sambamba [93].

The second category of tools aims at adding a clean abstraction over genomic collections by providing declarative SQL-like interfaces: Genome Query Language (GQL)[94], GORPipe[95] and more recent SamQL[96]. They neither support the distributed computing model nor are capable of higher level operations such as calculation depth of coverage

or genomic pileup. They are suitable only for querying genomic fragments in the form of mapped reads using a single node.

The third generation of tools is built upon Hadoop ecosystem technologies with the state-of-the-art support for distributed computing and declarative programming model:

- Signal Track Query Language (STQL)[97] aims only at analyzing signal tracks composed of a set of genomic intervals without support for reads or variants file formats. Queries can be expressed in HiveQL extended with some specific syntactic constructs like TRACK or BINS, that are compiled into MapReduce jobs by the Hive engine.
- GenoMetric Query Language (GMQL)[89, 77] similarly to STQL, provides the ability to query Genomic Data Model represented as *genomic regions* and *metadata*[74]. It implements a custom compiler that is able to translate queries expressed in GMQL language into a code specific to a given distributed computing engine, among others, Apache Spark. GMQL does not support bioinformatics file formats like FASTQ, BAM or VCF formats. It provides bindings for Python[98] and R [79] programming languages.
- GenAp[90] provides a custom strategy to Apache Spark’s Catalyst optimizer with an interval tree based implementation of range join faster than the one introduced by the ADAM project. It supports SQL interface and enables interoperability with the ADAM data formats.
- Glow[99], VariantSpark[100] and Hail[101] are Apache Spark based tools focusing on genome-wide association studies (GWAS). The main difference between them is the way they store cohort genotypes: `MatrixTable` distributed two-dimensional extension of a Table (Hail) or standard Spark `DataFrame` (Glow). Hail does not provide an SQL interface.

Table 1.1 summarizes all notable solutions that are addressing performance challenges of the secondary and tertiary analyses. Furthermore, the presented tools can be categorized at the highest level from two perspectives: implementational and functional.

From the technical point of view, they can be divided into the following groups:

- non-native distributed implementations around external single-node/single-thread tools (i.e. **wrappers**), where a computing framework is used mostly to distribute

(stream) the workload and encode/decode the bidirectional communication for operations like, for instance, short read mapping. Examples of such an approach are: DistMap[102], SEAL[103], SparkBWA[104], and Cannoli[83],

- **native** distributed tools implementing algorithms for steps in HTS data analysis pipelines that are novel or adjusted to the distributed computing models offered by frameworks like Hadoop MapReduce or Apache Spark, e.g. SeqPig[105], GATK[15] or SeQuiLa[P5, P6], DECA[16], and ADAM[12],
- **auxiliary** libraries/tools that aim at enabling distributed processing of complex bioinformatics file formats, e.g. Hadoop-BAM[106], disq[107], or ADAM[12].

From the functional perspective, we can distinguish the following subcategories:

- HTS **secondary** data analysis pipelines that seek to preserve results concordance with the existing best practices and recommendations[108], while at the same time improve the overall performance by distributing some steps of the workflows, e.g. SparkGA[109], SparkGA2[110], Halvade[111], Halvade-RNA[112],
- distributed tools for **tertiary** data analysis that encompasses exploratory data analysis, statistical analysis and machine learning, e.g. VariantSpark[100], Glow[99], Hail[101], SeQuiLa[P4] or MANGO[113].

SeQuiLa is one of the few tools in the above comparison that contributes to both secondary and tertiary analyses. Since it is fully compatible with Apache Spark DataFrame and SQL APIs, SeQuiLa can be easily integrated with all the other Apache Spark-based tools that share the same interfaces, including ADAM or Glow to enhance performance of sequence processing and EDA domains.

Tool	AM	VC	SP	DA	Engine	SQL	PP	Categories
SEAL[103]	+	-	+	-	MR	-	-	wrapper,secondary
DistMap[102]	+	-	-	-	MR	-	-	wrapper,secondary
BigBWA[114]	+	-	-	-	MR	-	-	wrapper,secondary
SparkBWA[104]	+	-	-	-	Spark	-	-	wrapper,secondary
PipeMEM[115]	+	-	-	-	Spark	-	-	wrapper,secondary
SeqPig[105]	-	+/-	+	+/-	MR	-	-	wrapper,tertiary

Hadoop-BAM[106]	-	-	+	-	MR/Spark	-	-	auxiliary
Halvade[111]	+	+	-	-	MR	-	+	wrapper,secondary
Halvade-RNA[112]	+	+	-	-	MR	-	+	wrapper,secondary
SparkSW[116]	+	-	-	-	Spark	-	-	native,secondary
DSA[117]	+	-	-	-	Spark	-	-	native,secondary
CloudSW[118]	+	-	-	-	Spark	-	-	native,secondary
disq[107]	-	-	+	-	Spark	-	-	auxiliary
Halvade somatic[119]	+	+	-	-	Spark	-	+	wrapper,secondary
VC@Scale[120]	-	+	-	-	Spark	-	+	wrapper,secondary
SparkGA[109]	+	+	-	-	Spark	-	+	wrapper,secondary
SparkGA2[110]	+	+	-	-	Spark	-	+	wrapper,secondary
SparkRA[121]	+	+	-	-	Spark	-	+	wrapper,secondary
Sparkhit[122]	+	-	+	-	Spark	-	-	wrapper,native
MANGO[113]	-	-	-	+/-	Spark	-	-	native,tertiary
DECA[16]	-	+	-	-	Spark	-	-	native,secondary
Cannoli[83]	+	+	+	-	Spark	-	-	wrapper,secondary
STQL[97]	-	-	-	+/-	MR	+	-	native,tertiary
GMQL[77, 98, 79]	-	-	-	+/-	Spark	-	-	native,tertiary
GenAp[90]	-	-	+/-	+	Spark	+	-	narive,tertiary
GATK[15]	-	+	+/-	-	MR/Spark	-	-	native,secondary
ADAM[12]	-	-	+	+	Spark	+/-	-	native/auxiliary, sec-ondary/tertiary
VariantSpark[100]	-	-	-	+	Spark	-	-	native,tertiary
Hail[101]	-	-	-	+	Spark	-	-	native,tertiary
Glow[99]	-	-	-	+	Spark	+	-	native,tertiary

SeQuiLa [P4, P5, P6]	-	+/-	+	+	Spark	+	-	native, secondary/tertiary
--------------------------------	---	-----	---	---	-------	---	---	----------------------------

Table 1.1: Big Data solutions utilized in the secondary and tertiary analysis of the HTS data. More comprehensive tool surveys can be found in many review studies[78, 123, 124, 125]. AM – alignment and mapping, VC – variant calling, SP – sequence processing (support for distributed analysis of FASTQ/BAM/VCF, including depth of coverage and pileup operations), DA – Exploratory Data Analysis (EDA, support for querying and joining different data sources, also using interval conditions) SQL – support for SQL, Spark – Apache Spark based, MR – MapReduce based, PP – end-to-end pipeline.

1.4 Challenges leading to Genomic Data Lakehouse

The above presented applications of the Big Data methods and technologies have already addressed many of the identified bottlenecks in the HTS data workflows, such as short read mapping or GWAS analyses. Also many improvements have been proposed to the existing variant calling pipelines optimizing their performance as a whole. However, there are still relatively few studies tackling the following challenges:

- designing novel or fine-grained optimization of the existing distributed algorithms for common operations: calculating depth of coverage, summarizing short reads in a form of pileup, joining datasets using interval intersections,
- designing and benchmarking genomic data warehousing,
- defining an unified programming approach for genomic data processing and analyses,
- proposing novel cloud approaches for cost-efficient, scalable and secure population scale genomic studies such as Genomic Data Lakehouse architecture,
- adapting IaC approach for provisioning genomic data platforms.

The ideas presented in this Dissertation evolved over time on par with the advances in Big Data technologies and paradigms. Initial studies[P1, P2] focused on the aforementioned two-tier architecture with the separation of data lake and data warehouse

components, whereas the recent SeQuiLa research papers concentrated on the unified cloud platform with SQL-first approach backed by open file formats and cost-efficient object storage[P4, P5, P6]. By that means, these publications set the foundation for Genomic Data Lakehouse platform architecture that adapted the concepts presented in the subsection 1.1 with SeQuiLa being its core component (figure 1.2). Its layered organization is inspired by the widely adopted *zone models*[126], *medallion architecture*[127], in particular and is composed of:

- the landing (raw) zone aimed at ingestion of sequencing data files containing short reads from the sequencing instruments (*Bronze* layer),
- the processing zone where alignment and variant calling pipelines are run at scale to deliver the results of the secondary analysis stored in open data formats like ADAM and ORC or DeltaLake (*Silver* layer),
- the exploratory and reporting zone that enables researchers to perform population studies using data science techniques as well as run *ad hoc* queries for further variant analysis (*Gold* layer).

To ensure cloud-agnostic architecture of the Genomic Data Lakehouse, it is deployed in a containerized way on the Kubernetes platform and cloud object storage systems that are natively supported by the Apache Spark engine. Such an approach also guarantees cost efficiency and, at the same time, high independent scalability of both storage and computing layers. Security follows recommended[19] hierarchy of system level permissions including: platform-level admin roles (i), the project-level admin roles (ii) and project-level user roles (iii) governed by the data access privileges and the data access needs.

Figure 1.2 visualizes dependencies between the published manuscripts, in which I proposed several novel approaches that resolve challenges stated above, in particular, methods presented in my research papers:

- [P1] is one of the first studies to assess the applicability of Apache Spark to HTS studies and confronting its performance with Hadoop MapReduce execution engine,
- [P5, P6] addresses the problem of non-satisfactory performance of the distributed depth of coverage and pileup algorithms when compared to the single node counterparts,

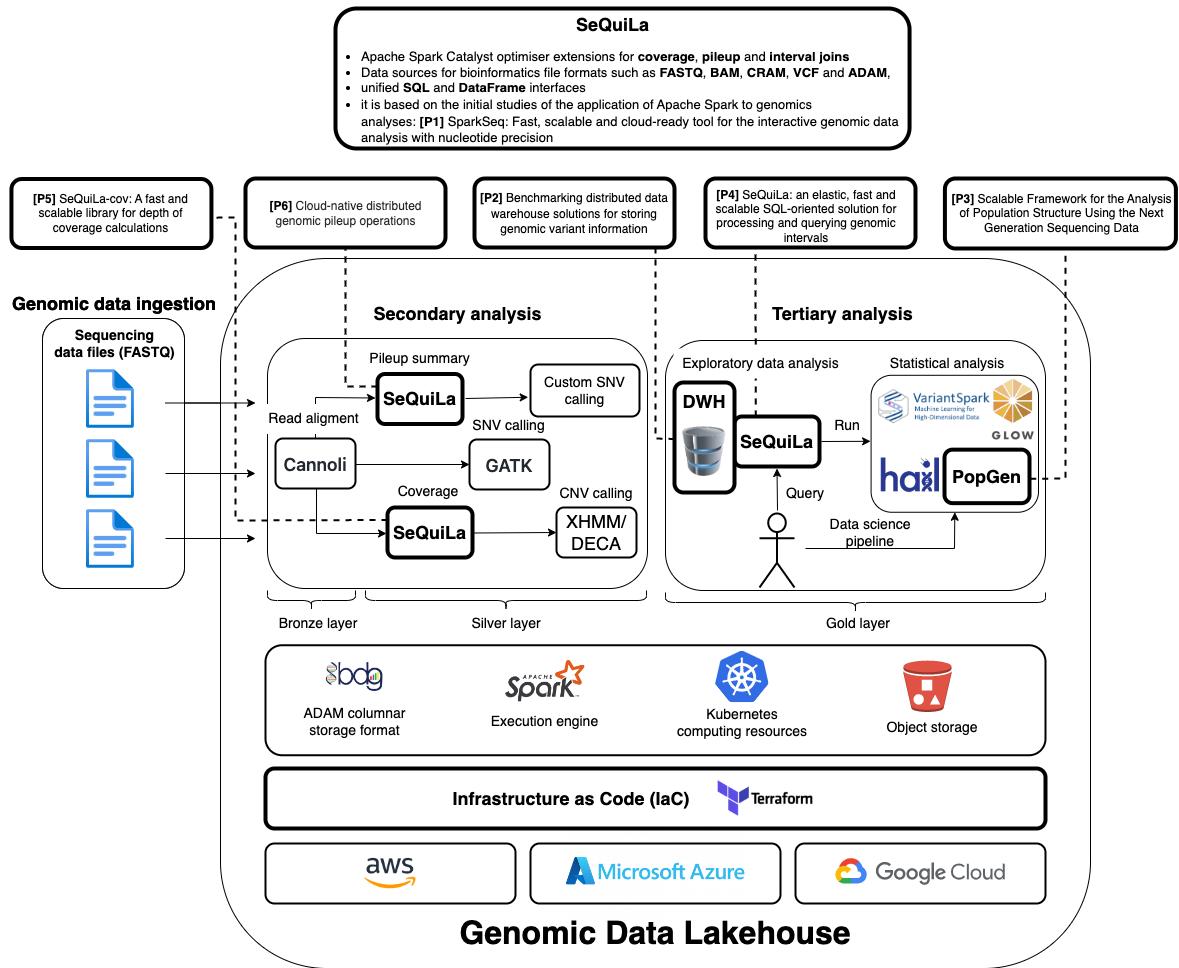


Figure 1.2: Genomic Data Lakehouse architecture overview, **bold rounded rectangles** symbolize Author's main contributions presented in this Dissertation.

- [P4] facilitates large-scale genomic ranges intersections by proposing optimized distributed join strategies,
- [P2] argues utility of genomic data warehouses for population scale studies,
- [P3] investigates areas of applicability and potential performance improvements of distributed machine learning algorithms in Apache Spark large scale genomic studies.

1.5 Aim and Research Theses

The main goal of this Dissertation was to design new distributed and scalable methods and algorithms that would help to improve the efficiency of the secondary and tertiary stages in the HTS data analysis, and also to contribute to the development of the Genomic Data Lakehouse paradigm.

This Dissertation develops and proves three main research theses:

- I Application of the novel distributed computing engines can substantially improve the performance of interactive alignment files ad hoc analyses in a Data Lake environment ([P1]).
- II Columnar data formats together with the modern SQL query engines and appropriate data modeling techniques enable building a distributed data warehousing platform that would outperform the existing single-node alternative solutions and can be used for population-scale studies ([P2]).
- III Building a scalable Genomic Data Lakehouse requires distributed algorithms designed and implemented for common bioinformatics operations such as interval intersections, calculation of depth of coverage, pileup summary and machine learning pipelines ([P3, P4, P5, P6]).

1.6 Publications constituting this Dissertation

This Dissertation presents the methods, architectures and algorithms that were previously published in peer-reviewed journals. It is based on a compilation of six research papers coauthored by the Author of this thesis:

- [P1] M.S. Wiewiórka, A. Messina, A. Pacholewska, S. Maffioletti, P. Gawrysiak, and M.J. Okoniewski. SparkSeq: Fast, scalable and cloud-ready tool for the interactive genomic data analysis with nucleotide precision. *Bioinformatics*, 30(18), 2014, **MNiSW list: 200 pts, Impact factor: 4.981, Contribution: 55%, Description of the contribution:** Conceptualization and formal analysis of the algorithms, implementation, preparing and running performance tests, performance tuning, manuscript writing.

[P2] M.S. Wiewiórka, D.P. Wysakowicz, M.J. Okoniewski, and T. Gambin. Benchmarking distributed data warehouse solutions for storing genomic variant information. *Database : the journal of biological databases and curation*, 2017, 2017, MNiSW list: 100 pts, Impact factor: 2.627, Contribution: 65%, Description of the contribution: Conceptualization, implementation, preparing and running performance tests, manuscript writing.

[P3] Anastasiia Hryhorzhevska, Marek Wiewiórka, Michał Okoniewski, and Tomasz Gambin. Scalable Framework for the Analysis of Population Structure Using the Next Generation Sequencing Data. In *Lecture Notes in Computer Science (including sub-series Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 10352 LNAI, pages 471–480. 2017, MNiSW list: 20 pts, Impact factor: –, Contribution: 20%, Description of the contribution: Conceptualization, preparing infrastructure for performance tests, manuscript writing.

[P4] Marek Wiewiórka, Anna Leśniewska, Agnieszka Szmurło, Kacper Stępień, Mateusz Borowiak, Michał Okoniewski, and Tomasz Gambin. SeQuiLa: an elastic, fast and scalable SQL-oriented solution for processing and querying genomic intervals. *Bioinformatics*, 35(12):2156–2158, June 2019, MNiSW list: 200 pts, Impact factor: 4.531, Contribution: 51%, Description of the contribution: Conceptualization and formal analysis of the algorithms, implementation, preparing and running performance tests, performance tuning, manuscript writing.

[P5] Marek Wiewiórka, Agnieszka Szmurło, Wiktor Kuśmirek, and Tomasz Gambin. SeQuiLa-cov: A fast and scalable library for depth of coverage calculations. *Giga-Science*, 8(8), August 2019, MNiSW list: 200 pts, Impact factor: 5.71, Contribution: 45%, Description of the contribution: Conceptualization and formal analysis of the algorithms, implementation, preparing and running performance tests, manuscript writing.

[P6] Marek Wiewiórka, Agnieszka Szmurło, Paweł Stankiewicz, and Tomasz Gambin. Cloud-native distributed genomic pileup operations. *Bioinformatics*, December 2022, MNiSW list: 200 pts, Impact factor: 6.931, Contribution: 45%, Description of the contribution: Conceptualization and formal analysis of the algorithms,

implementation, preparing and running performance tests, preparing IaC modules for cloud resource provisioning, performance tuning, manuscript writing.

CHAPTER 2

Main scientific contribution of the Author of the Dissertation

This chapter summarizes my main scientific contributions that fall into three categories: (i) evaluation of the distributed processing model for genomic studies, (ii) applicability of the data warehousing architecture for genomic variant analyses and (iii) design and implementation of the distributed methods for the development of the genomic data lakehouse concept.

My first three research papers[P1, P2, P3] were devoted to identifying the **opportunities** and **challenges** of introducing of the distributed computing model to genomic studies. In the three latter ones[P4, P5, P6], drawing on implementation experience from the initial studies, I proposed novel computational methods and algorithms to address some of the recognized shortcomings. Together, they gave rise to the SeQuiLa¹ project that was launched at the Institute of Computer Science at the Warsaw University of Technology in the late 2017. I have been one of its main contributors ever since.

SeQuiLa is a truly interdisciplinary project drawing on many fields of knowledge such as query optimization theory[128], distributed computing, bioinformatics and genomics. The ultimate goal of SeQuiLa is to provide researchers with an open, scalable and easy to use distributed computing environment for genomic studies at population scale. The project is still under active development and it has been gradually extended with new

¹<https://biodatageeks.github.io/sequila/>

features.

As its initiator, main developer and maintainer, my contribution to the SeQuiLa project consists of designing and implementing several distributed algorithms that optimize common time-consuming bioinformatics operations such as: range joins, depth of coverage and pileup calculations. The importance of these algorithms is briefly summarized below.

Intersection of genomic intervals is one of the most common operation utilized at various stages of variant calling and annotation pipelines. It is also a crucial element of most *ad hoc* bioinformatics analyses. Its aim is to determine the overlap between different intervals defined, for instance, as a set of sequencing reads or selected genomic features. This kind of operations is supported by several available software tools, including featureCounts[129], samtools[68] and GenomicRanges[130]. However popular and easy to use, they are still subject to performance limitations that make genome-wide analyses infeasible.

Pileup format was designed to provide the evidence of single-nucleotide variants or short insertions/deletions at given genomic positions. It is commonly used as an entry point to the well-established variant calling pipelines[131]. Further, it is also utilized in novel variant detection frameworks based on the neural networks[132] or on other methods like the binomial model, partial-order alignment, and de Bruijn graph local assembly[133] in the case of fast variant calling. Coverage position summary is also used for identification of somatic mutations and copy number variation[134].

Ultimately, a substantial effort has been put into making SeQuiLa a cloud-ready solution and developing additional mechanisms, following IaC best practices, for automatic deployments in all cloud environments.

2.1 Genomic data distributed processing in data lakes

[P1]

Publication [P1] with over 130 citations² at the time of preparing this Dissertation, alongside with the ADAM project that was published almost simultaneously, is among the earliest and the most highly cited studies in which using the Apache Spark engine was

²According to Google Scholar, <https://scholar.google.pl/scholar?q=sparkseq>

proposed for the HTS data analysis.

The proof-of-concept solution SparkSeq, that was developed for the purpose of the evaluation presented in this manuscript, allowed end users to perform, using Apache Spark RDD API, various types of basic analyses, such as reads counting, filtering and viewing from the BAM files stored on HDFS using the Hadoop-BAM[106] library. This study provided the following insights:

- SparkSeq methods implementation proved to be more than 9x faster than analogous available in SeqPig (MapReduce-based) in all 4 benchmarks,
- single-thread performance was tested against shell scripts using samtools and AWK[135],
- Apache Spark caching strategies, serialization and compression options were evaluated in terms of CPU-memory trade-offs and performance benefits for multi-pass algorithms,
- development of novel algorithms is a lot easier than with the low-level MapReduce approach.

Apache Spark evaluation confirmed its applicability to genomics analyses and, although this study did not result in a production-ready solution, it certainly set the foundation for further projects including SeQuiLa.

2.2 Genomic data querying in distributed data warehouses [P2]

Studies presented in the manuscript [P2] aimed at providing data architects and bioinformaticians with guidance as well as benchmark techniques that would help them assess performance of the designed distributed data warehousing platform for storing genomic variant information at population scale. This kind of solutions is targeted especially at large-scale, national sequencing initiatives, such as the already mentioned Genomics England's '100 000 Genomes Project'.

I proposed to use a novel testing methodology the basic idea of which was to prepare a reference star schema-based[136] data model (including additional materialized views[137]

implementing one-big table (OBT) optimization[138] for the elimination of distributed joins fact-dimensions tables) together with a data simulator and a set of predefined SQL queries that correspond with typical analyses such as calculating allele frequencies with geographical breakdowns, cumulative frequencies per genomic interval or various variant statistics across samples. The data generator component in order to preserve real-world statistical distributions of the variants combined information available in two databases, namely annotation details from dbNSFP[139] with allele frequencies from ExAC[140].

The benchmark framework I put forward was applied to evaluate performance of a number of distributed query engines, including Apache Hive, Prestodb[141], Apache Spark, Apache Impala[142]. In addition, their efficiency was compared against the single-node analytical database MonetDB[143] and the distributed On-Line Analytical Processing (OLAP) cube solution - Apache Kylin[144].

From this benchmark I was able to identify several key findings:

- Apache Spark (version 2.x) proved to be the most general purpose tool in the study, suitable for computationally intensive data processing but also applicable when interactivity is of great importance,
- both open columnar-oriented file formats evaluated – ORC and Parquet – offer great performance (although at the time of writing that manuscript Apache Spark did not support a native vectorized reader³, hence it underperformed in a number of test queries over tables stored in ORC),
- distributed joins minimization (or total elimination using OBT optimization) can substantially reduce query execution time.

All of the above conclusions were important from the perspective of my further work on adjusting Apache Spark for distributed genomic analyses.

³This feature was added in version 2.3.0: <https://spark.apache.org/docs/2.3.0/sql-programming-guide.html#orc-files>

2.3 Genomic Data Lakehouse

2.3.1 Distributed machine learning framework for genomic data[P3]

Publication [P3] presented a proof-of-concept tool for a distributed analysis of population structure using the ADAM project and the Apache Spark builtin machine learning library – MLlib[38]. The pipeline included a few machine learning methods for dimensionality reduction, clustering or classification such as: Principal Component Analysis (PCA), K-means, Gaussian Mixture, Support Vector Machine (SVM), Random Forests, and Decision Trees.

In this work, I contributed to the design and analysis of the proposed solution that confirmed a great applicability of distributed computing for feature pre-processing and engineering steps, even for studies involving more than a hundred thousand samples. On the other hand, experiments showed clearly that built-in implementations of dimensionality reduction and classification methods are not capable of handling genome-wide datasets, i.e. with samples represented as rows and millions of columns corresponding to the number of genomic variants under study.

This led to the conclusion that there was also a need for the specialized methods like the one proposed in VariantSpark[100] or REGENIE[145] algorithms, the distributed version of which was implemented recently in the Glow[99] project.

2.3.2 Scalable range joins[P4]

Range join (also known as region join, interval query) is an operation aiming at finding overlaps between datasets containing records with keys in a form of genomic intervals. A genomic interval r can be defined by the two coordinates, $start$ and end , on a given chromosome, i.e. $r = (r.chrom, r.start, r.end)$. Given a set of N intervals, $R = \{r_1, r_2, \dots, r_N\}$, for $N \gg 1$ and a query interval q , finding the subset S of R that intersects q can be represented as[146]:

$$S(q) = \{r \in R | (r.chrom = q.chrom \wedge r.start \leq q.end \wedge r.end \geq q.start)\} \quad (2.1)$$

That, in turn would also correspond to the *inner join* operation expressed in SQL as follows:

```

SELECT * FROM r JOIN q ON
q.chrom=r.chrom AND r.end >= q.start AND r.start <= q.end

```

There have been several efficient data structures proposed that optimize this operation in the case of a single-node execution model like: Augmented Interval Tree (AIT)[147], Nested Containment List (NCLList)[148], Augmented Interval List (AILList)[146], Interval Array (IA)[149], Implicit Interval Tree (IIT)[14] or Implicit Interval Tree with Interpolation Index (IITII)[150]. Distributed join implementations have been so far pursuing either of the two strategies: broadcast or shuffle-based[149].

In particular, the ADAM project offers two algorithms: a shuffle-based (sort-merge) and a shuffle-free (broadcasting smaller dataset as an IA-like data structure). Nonetheless, the authors of the GenAp manuscript confirmed that the modified, two-stage AIT broadcast algorithm (i.e. constructing an interval forest storing only rows identifiers instead of whole records) can be both substantially faster than sort-merge approach and, at the same time, able to address the problem of joining two relatively large datasets. In [P4] I proposed the following four main enhancements to the existing approaches:

- novel optimized, broadcast AIT-based join method (with time complexity $\mathcal{O}(\log n)$),
- novel rule-based mechanism for Apache Spark Catalyst that, depending on the estimated broadcast structure size, decides which join algorithm to use,
- fully declarative and ANSI SQL compliant interface that enables direct querying not only genomic datasets stored in ADAM but also BAM, CRAM and VCF files as well arbitrary DataFrames,
- the overall performance of the implemented algorithm was 1.7-4.2x better than other distributed solutions.

The broadcast method was generalized after the manuscript publication into the pluggable mechanism that enables specifying in the runtime other data structure designated for broadcasting, such as AILList, NCLList or IITII . This was designed to mitigate the problem of serializing large interval tree structures. However, unlike [149] that suggested using interval array structure (that guarantees only the best case $\Theta(\log n)$ because of additional linear search needed) we are able to preserve the time complexities of the underlying interval data structures.

2.3.3 Scalable depth of coverage and pileup calculations[P5], [P6]

Given a reference genome sequence S of symbols from the alphabet $\Sigma = \{A, C, G, T\}$ (the symbols in Σ represent the four nucleotides adenine, cytosine, guanine, and thymine), a set of all corresponding genomic positions G and a set of reads mapped to this reference genome R , then for each $g \in G$ we define a subset $R_g \subseteq R$ denoting sequenced fragments (reads) obtained from a sequencing experiment that cover a genomic position g as:

$$R_g = \{r \in R | (g.chrom = r.chrom \wedge g.pos \geq r.start \wedge g.pos \leq r.end)\} \quad (2.2)$$

a *depth of coverage* function can be represented as:

$$C(g) = |R_g| \quad (2.3)$$

where $|R_g|$ is the cardinality of a set R_g , i.e., the number of sequencing reads that overlap with the position g .

Equally, a *pileup* function in its simplest form can be defined as:

$$P(g) = \{(a_1, \dots, a_n) | r \in R_g \wedge n \in \{1, \dots, |R_g|\} \wedge a_n = r.seq(g)\} \quad (2.4)$$

where $r.seq$ denotes a sequence from the same alphabet as S and $r.seq(g)$ is an element in this sequence corresponding to a genomic position g in a read r . It is also straightforward to observe that:

$$C(g) = |P(g)| = |(a_1, \dots, a_n)| = |R_g| \quad (2.5)$$

and also that given a genome sequence S and $C(g)$ the reverse operation (i.e. deriving $P(g)$ from $C(g)$) is possible if all the reads covering a genomic position g have the same symbol in their sequences at respective positions corresponding the genomic position g , i.e.:

$$P(g) = \begin{cases} (a_n)_{n \in \{1, \dots, C(g)\}, a_n = S_g} & \text{if } \forall i \in \{1, \dots, C(g)\} r_i.seq(g) = S_g \\ (r_n.seq(g))_{n \in \{1, \dots, C(g)\}} | r \in R_g & \text{otherwise} \end{cases} \quad (2.6)$$

In either case additional filtering conditions on sequenced fragments can be imposed, e.g. basing on various quality metrics of a read or its flags[68]. Figure 2.1 presents calculation of depth of coverage and pileup. Optionally, a pileup summary ⁴ may include PHRED

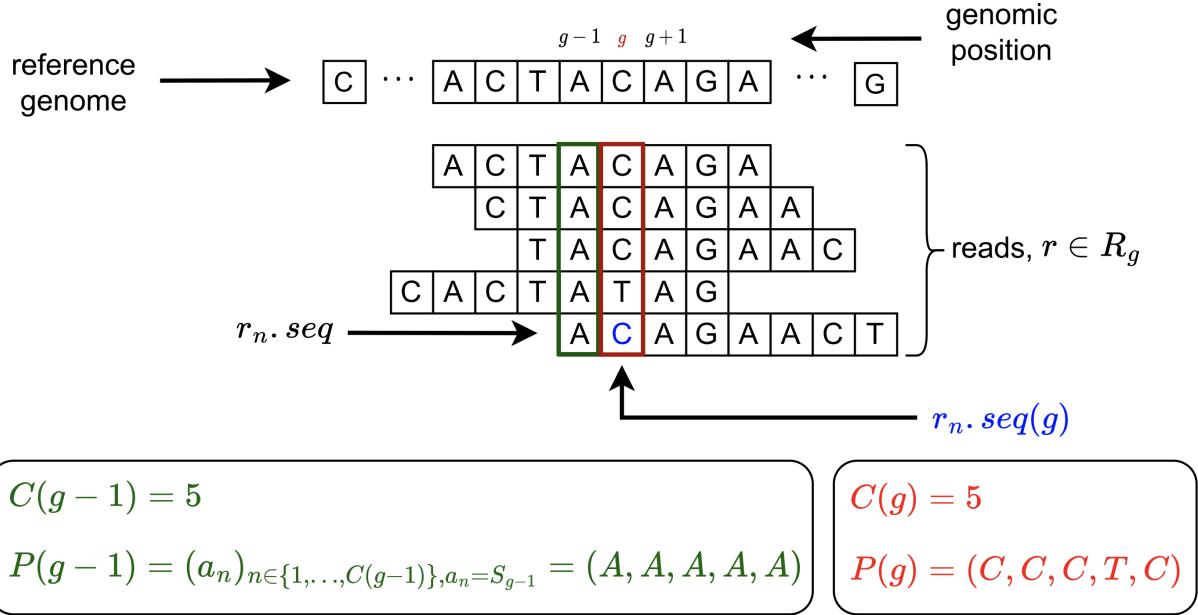


Figure 2.1: Example of depth of coverage and pileup calculations. Values of $C(g)$ and $P(g)$ functions are presented for the genomic position g and read fragments marked in red. Pileup for a genomic position $g - 1$ (marked in green) depicts how the value $P(g - 1)$ can be derived from the reference sequence S (as S_{g-1}) and the value of depth of coverage $C(g - 1)$ in the case when all the reads have a symbol S_{g-1} at a genomic position corresponding to $g - 1$. Consequently, all the elements of $P(g - 1)$ sequence are equal to S_{g-1} and the length of the sequence equals $C(g - 1)$.

quality scores for each of the base.

The depth of coverage algorithms can be divided into two categories:

- relying on determining the $P(g)$ values for all covered genomic positions and subsequently simply deriving $C(g)$ as the length of each pileup sequence (see equation 2.5). This approach is implemented, among others, in samtools[91], GATK[15] or sambamba[93],
- directly calculating $C(g)$ values using start and end positions of each read alignment for incrementing or decrementing interval counters. This event-based method has been initially proposed in bedtools[151] and more recently enhanced in mosdepth[152]. Neither of these utilities support parallel or distributed computation model – in the

⁴https://en.wikipedia.org/wiki/Pileup_format

case of mosdepth input file (i.e. BAM or CRAM) processing (BGZF blocks decompression) can be parallelized.

In [P5] I proposed a novel distributed method for calculating depth of coverage that generalizes the event-based approach. The scientific contribution can be summarized as follows:

- implementation of Apache Spark Catalyst custom execution strategy that handles depth of coverage calculations for both DataFrame and SQL APIs,
- an extension to the existing ANSI SQL compliant interface in a form of a table-valued function⁵,
- performance results confirmed that this algorithm is both scalable and up to two orders of magnitude faster than the existing single-threaded implementations and around 60x faster than the state-of-the art multi-threaded ones. It is also the fastest implementation among the tools supporting the distributed computing model.

In [P6], on the other hand, I enhanced the event-based algorithm to enable efficient distributed pileup computations. It is based on the observation expressed in equation 2.6. The key novelties introduced by this method are:

- a new algorithm that offers a drastically reduction of the computationally intensive read processing needed for materialization of the full pileup structure. It was possible thanks to the observation that, in most cases the majority of bases in the aligned sequencing reads are concordant with the reference sequence. This is why it is sufficient to use the information stored in Compact Idiosyncratic Gapped Alignment Report (CIGAR) strings, representing spliced alignment operations and MD tags⁶, encoding mismatched and deleted reference bases to construct a lean pileup structure holding only alternative alleles counts. This, in turn with the event-based coverage results can be efficiently used to render the full pileup summary,
- a new data partitions coalescing mechanism, which guarantees proper handling of reads overlapping more than one partition without the need of data shuffling,

⁵<https://biodatageeks.github.io/sequila/docs/api/sql/#sequila.SequilaSession.coverage>

⁶<https://samtools.github.io/hts-specs/SAMtags.pdf>

- a rewritten and optimized distributed event-based algorithm for depth of coverage calculations that enables single-pass over input data and lowers memory requirements since no intermediate data caching is required,
- implementation of Apache Spark Catalyst custom execution strategy that handles depth of coverage calculations for both DataFrame and SQL APIs,
- an extension to the existing ANSI SQL compliant interface in a form of a table-valued function⁷.

The performance results confirmed that this approach is by far the fastest pileup implementation in both single-node and distributed benchmarks.

Finally, in this manuscript, I have also emphasized the need of IaC[153] approach for cloud resources provisioning as crucial for ensuring portability and reproducibility of the SeQuiLa-based data processing workflows. There is still relatively very few publications offering ready-to-use modules and recipes for bioinformatics tools deployments in the cloud environments, and at the same time promoting Development, Security, Operations (DevSecOps)[154] principles. This approach may bring a number of benefits to the community, including:

- more secure cloud computing platforms provisioned using reviewed and opinionated IaC modules that additionally can be subject to automatic static code analysis tools (linters)[155],
- easy way of implementing FinOps[156] best practices for optimizing cloud spending as well as enabling better cost control e.g. through automatic resource labeling[157],
- unified and predictable change management process for application and infrastructure code.

In addition, a set of IaC modules⁸ for running SeQuiLa in all three main cloud platforms has been released together with the manuscript.

⁷<https://biodatageeks.github.io/sequila/docs/api/sql/#sequila.SequilaSession.pileup>

⁸<https://github.com/biodatageeks/sequila-cloud-recipes>

CHAPTER 3

Scientific achievements

The series of publications presented in this Dissertation is by far my original contribution of utmost importance. However, my scientific work also resulted in other notable achievements that are briefly summarised in the sections below. They include Open Source projects, research grants, other publications, conference appearances and a list of awards.

3.1 Source code repositories

The following git repositories have been created for the purpose of my research:

- [T1] – the main repository of the SeQuiLa project,
- [T2] – the Python PyPI package for SeQuiLa,
- [T3] – a set of Terraform[158] modules for the SeQuiLa cloud deployments,
- [T4] – the auxiliary source code for the benchmark methodology of distributed genomic data warehouses presented in[P2],
- [T5] – the source code for the proof-of-concept framework for the distributed analyses of the population structure presented in[P3],
- [T6] – the proof-of-concept SparkSeq solution for the distributed processing and analyses of the alignment files presented in [P1].

Git repositories [T1, T2, T2] are related to three SeQuiLa publications[P4, P5, P6].

3.2 Own research grants

- [O1] Sciex Fellowship no 12.289 – System „CLANG” — ClinicaL ANalysis in Genomics A Dedicated Application for Translational Biomedical Research - University of Zurich (07/2013-06/2014), **Budget:** 50,000 CHF, **Position:** Principal Investigator
- [O2] Microsoft Azure for Research - Towards an interactive secondary analysis of RNA sequencing data service in Widows Azure cloud with Apache Spark framework, 01/2014-01/2015, **Budget:** 40,000 USD, **Position:** Principal Investigator [159]
- [O3] National Science Centre Preludium 2014/13/N/ST6/01843 – Methods and algorithms of discovering novel phenomena in genomes and transcriptomes with the statistical analysis of nucleotide precision data from next generation sequencers, 03/2015-09/2018, **Budget:** 145,580 PLN, **Position:** Principal Investigator

3.3 Participation in research grants

- [O4] National Science Centre Opus 2014/13/B/NZ2/01248 – PerM-Cloud, Algorithms and methods of processing big genomic data repositories in computing cloud environment towards the personalized medicine, 03/2015-09/2018
- [O5] Polish budget funds for science Iuventus Plus IP2015/019874 – Simultaneous analysis of single nucleotide and structural variants from whole exome or targeted sequencing, 10/2016-10/2019

3.4 Other research publications

- [O6] Wiktor Kuśmirek, Agnieszka Szmurło, Marek Wiewiórka, Robert Nowak, and Tomasz Gamin. Comparison of kNN and k-means optimization methods of reference set selection for improved CNV callers performance. BMC Bioinformatics,

20(1):266, December 2019, **Description:** In this work apart from the main optimization challenge, there has been developed a prototype accelerated version of the tooling using a distributed depth of coverage computation method.

3.5 Book chapters

- [O7] Marek Wiewiora, Alicja Szabelska, Michał J. Okoniewski, Analysis of AmpliSeq RNA-Sequencing Enrichment Panels, Pattern Recognition and Machine Intelligence: 6th International Conference, PReMI 2015, Warsaw, Poland, June 30-July 3, 2015[160]
- [O8] Monika Szczerba, Marek Wiewiórka, Michał J. Okoniewski, Hneryk Rybiński, Scalable Cloud-Based Data Analysis Software Systems for Big Data from Next Generation Sequencing. In Nathalie Japkowicz and Jerzy Stefanowski, Big Data Analysis: New Algorithms for a New Society, volume 16, pages 263–283. Springer International Publishing, Cham, 2016[161]
- [O9] Michał J. Okoniewski, Rafał Płoski, Marek Wiewiórka, Urszula Demkow, Future Directions. In Clinical Applications for Next-Generation Sequencing, pages 281–294, Elsevier, 2016[162]

3.6 Conference talks

- [O10] Analysis of large genomic datasets with Big Bata tools, Big Data Technology Warsaw Summit, Warsaw, 2017
- [O11] Extending Apache Spark Catalyst optimizer, Data science summit, Warsaw, 2018
- [O12] Towards next generation, cloud-ready and open-source big data discovery platform, Big Data Technology Warsaw Summit, Warsaw, 2019
- [O13] Towards Enterprise Grade Data Discovery and Data Lineage, Big Data Technology Warsaw Summit, Warsaw, 2020

- [O14] MLOps implemented - how we combine the cloud open-source to boost data scientists work, Nordic Data Science and Machine Learning Summit, online, 2021
- [O15] From first contact to a full charge...How we built a Modern Data Platform in 4 months for a FinTech scale-up, DataMass, Gdańsk, 2022
- [O16] Genomic Data Lakehouse Architecture, 3rd Symposium of Genomics Platform at the Warsaw University of Technology, 2023

3.7 Conference posters

- [O17] Marek Wiewiórka, Dawid P. Wysakowicz, Michał J. Okoniewski , Tomasz Gambin, Benchmarking distributed data warehouse solutions for storing genomic variant information, The American Society of Human Genetics Annual Meeting 2016
- [O18] Marek Wiewiórka, Wiktor Kuśmirek, Michał J. Okoniewski, Tomasz Gambin, Automated parameter tuning for more accurate CNV calling in WES/WGS data, The American Society of Human Genetics Annual Meeting 2017
- [O19] Marek Wiewiórka, Agnieszka Szmurło, Tomasz Gambin, SeQuiLa: an elastic, fast and scalable SQL-oriented platform for processing and analyzing genomic data, The American Society of Human Genetics Annual Meeting 2018
- [O20] Marek Wiewiórka, Agnieszka Szmurło, Tomasz Gambin, High-level optimizations over query engines ensemble: Accelerating distributed genomic data science, The American Society of Human Genetics Annual Meeting 2019

3.8 Prizes and awards

- [O21] TransFormation.doc program winner (POIG.01.01.03-00-001/08) – Soft Skills and Entrepreneurship training at University of Lund, 11/2015
- [O22] Warsaw University of Technology Best paper award 2020 winner[163] for the manuscript: SeQuiLa: an elastic, fast and scalable SQL-oriented solution for processing and querying genomic intervals ([P4])

- [O23] Warsaw University of Technology Best paper award 2020 winner[163] for the manuscript: SeQuiLa-cov: A fast and scalable library for depth of coverage calculations ([P5])

CHAPTER 4

Conclusions and future work

This Dissertation presents several novel distributed algorithms and organization of the computational efforts for scalable processing of HTS data in the context of cloud data platform architectures. Throughout the thesis, I aimed at outlining both multidimensional challenges concerning design and implementation of solutions for genomic analyses at scale, as well as ideas and concepts to address them.

In my opinion, on one hand, a comprehensive approach that combines theoretical computer science perspective and real-world computing environment characteristics is absolutely crucial for developing high-performance systems. In the case of large-scale distributed applications, aspects such as inter-node communication, data compression, and I/O operations latency, may be equally important as theoretical time complexities and, as a whole, decisive for success of a given method. This is why I put a lot of effort into rigorous performance evaluation of all the presented algorithms, each time comparing them with the state-of-the-art single-node implementations. On the other hand, user-experience is something that cannot be neglected. Providing bioinformaticians and data scientists with programming interfaces that, within their communities, are considered to be *lingua franca* like SQL or Python languages, is in fact a critical factor for a software adoption. Finally, platforms maintainability together with portability and reproducibility of genomic data analyses are equally important. This is especially desirable in the cloud computing

context where, from DevSecOps and FinOps perspectives, resources provisioning should be fully automated to enable easy and secure environment setup.

The publications that constitute this Dissertation not only discuss extensively all the research theses presented in section 1.5 but also sketch the architecture of the Apache Spark-based Genomic Data Lakehouse using the SeQuiLa project. Nonetheless, it is worth summarizing the key research contributions of the Author:

- studies on applicability of Apache Spark framework for genomic analyses and implementation of the proof-of-concept tool – SparkSeq,
- proposing a benchmark methodology of evaluation and comparison of the distributed SQL query engines for storing and analysing variant information,
- design and implementation of three novel distributed algorithms, important from the perspective of genomics analyses, for the purpose of: finding genomic intervals intersections, computing depth of coverage and obtaining genomic pileup summary,
- being the initiator, main designer and developer of the SeQuiLa project,
- promoting IaC and DevSecOps principles for reproducible, more secure and more cost efficient genomic studies in the cloud.

The main focus of this Dissertation has been on the scalable and distributed computing methods using CPUs running only within Java Virtual Machine. However, there are at least two very promising extensions to this approach that are definitely worth being explored further: (i) heterogeneous computing with the intention to offload some of the most computationally intensive tasks over Java Native Interface or a similar interface to FPGAs[164] or GPUs[165, 166], (ii) a project inspired by Photon[53] and Velox[54] providing a similar native acceleration for genomics operations.

Lastly, one cannot forget about the data governance[167] aspects of genomic studies, especially population-scale, where automatic cataloging, security and robust data sharing mechanisms are extremely crucial for popularizing cloud computing genomic platforms.

CHAPTER 5

Bibliography

Original Author's Contribution

- [P1] M.S. Wiewiórka, A. Messina, A. Pacholewska, S. Maffioletti, P. Gawrysiak, and M.J. Okoniewski. SparkSeq: Fast, scalable and cloud-ready tool for the interactive genomic data analysis with nucleotide precision. *Bioinformatics*, 30(18), 2014. ISSN 14602059. doi: 10.1093/bioinformatics/btu343.
- [P2] M.S. Wiewiórka, D.P. Wysakowicz, M.J. Okoniewski, and T. Gambin. Benchmarking distributed data warehouse solutions for storing genomic variant information. *Database : the journal of biological databases and curation*, 2017, 2017. ISSN 17580463. doi: 10.1093/database/bax049.
- [P3] Anastasiia Hryhorzhevska, Marek Wiewiórka, Michał Okoniewski, and Tomasz Gambin. Scalable Framework for the Analysis of Population Structure Using the Next Generation Sequencing Data. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 10352 LNAI, pages 471–480. 2017. doi: 10.1007/978-3-319-60438-1_46. URL http://link.springer.com/10.1007/978-3-319-60438-1_46.
- [P4] Marek Wiewiórka, Anna Leśniewska, Agnieszka Szmurło, Kacper Stępień, Mateusz Borowiak, Michał Okoniewski, and Tomasz Gambin. SeQuiLa: an elastic, fast and scalable SQL-oriented solution for processing and querying genomic intervals. *Bioinformatics*, 35(12):2156–2158, June 2019. ISSN 1367-4803. doi: 10.

- 1093/bioinformatics/bty940. URL <https://academic.oup.com/bioinformatics/article/35/12/2156/5182295>.
- [P5] Marek Wiewiórka, Agnieszka Szmurło, Wiktor Kuśmirek, and Tomasz Gambin. SeQuiLa-cov: A fast and scalable library for depth of coverage calculations. *GigaScience*, 8(8), August 2019. ISSN 2047-217X. doi: 10.1093/gigascience/giz094. URL <https://academic.oup.com/gigascience/article/doi/10.1093/gigascience/giz094/5543653>.
- [P6] Marek Wiewiórka, Agnieszka Szmurło, Paweł Stankiewicz, and Tomasz Gambin. Cloud-native distributed genomic pileup operations. *Bioinformatics*, December 2022. ISSN 1367-4803. doi: 10.1093/bioinformatics/btac804. URL <https://academic.oup.com/bioinformatics/advance-article/doi/10.1093/bioinformatics/btac804/6900922>.

Source code repositories

- [T1] biodatageeks/sequila:1.1.0, January 2023. URL <https://zenodo.org/record/7581429>.
- [T2] biodatageeks/pysequila:0.4.1, January 2023. URL <https://zenodo.org/record/7581427>.
- [T3] biodatageeks/sequila-cloud-recipes:0.10.0, January 2023. URL <https://zenodo.org/record/7581435>.
- [T4] biodatageeks/variantsdwh: Paper release, March 2023. URL <https://zenodo.org/record/7699817>.
- [T5] biodatageeks/popgen: Paper release, March 2023. URL <https://zenodo.org/record/7699874>.
- [T6] biodatageeks/sparkseq: Paper release, March 2023. URL <https://zenodo.org/record/7699968>.

References

- [1] J. D. Watson and F. H. C. Crick. Molecular Structure of Nucleic Acids: A Structure for Deoxyribose Nucleic Acid. *Nature*, 171(4356):737–738, April 1953. ISSN 0028-0836, 1476-4687. doi: 10.1038/171737a0. URL <https://www.nature.com/articles/171737a0>.
- [2] Gaye Lightbody, Valeriia Haberland, Fiona Browne, Laura Taggart, Huiru Zheng, Eileen Parkes, and Jaine K Blayney. Review of applications of high-throughput sequencing in personalized medicine: barriers and facilitators of future progress in research and clinical application. *Briefings in Bioinformatics*, 20(5):1795–1811, September 2019. ISSN 1467-5463, 1477-4054. doi: 10.1093/bib/bby051. URL <https://academic.oup.com/bib/article/20/5/1795/5062275>.
- [3] Sara Goodwin, John D. McPherson, and W. Richard McCombie. Coming of age: ten years of next-generation sequencing technologies. *Nature Reviews Genetics*, 17(6):333–351, June 2016. ISSN 1471-0056, 1471-0064. doi: 10.1038/nrg.2016.49. URL <http://www.nature.com/articles/nrg.2016.49>.
- [4] Taishan Hu, Nilesh Chitnis, Dimitri Monos, and Anh Dinh. Next-generation sequencing technologies: An overview. *Human Immunology*, 82(11):801–811, November 2021. ISSN 01988859. doi: 10.1016/j.humimm.2021.02.012. URL <https://linkinghub.elsevier.com/retrieve/pii/S0198885921000628>.
- [5] Andreas von Bubnoff. Next-Generation Sequencing: The Race Is On. *Cell*, 132(5):

- 721–723, March 2008. ISSN 00928674. doi: 10.1016/j.cell.2008.02.028. URL <https://linkinghub.elsevier.com/retrieve/pii/S0092867408002766>.
- [6] Jan Smetana and Petr Brož. National Genome Initiatives in Europe and the United Kingdom in the Era of Whole-Genome Sequencing: A Comprehensive Review. *Genes*, 13(3):556, March 2022. ISSN 2073-4425. doi: 10.3390/genes13030556. URL <https://www.mdpi.com/2073-4425/13/3/556>.
- [7] Francis S. Collins and Harold Varmus. A New Initiative on Precision Medicine. *New England Journal of Medicine*, 372(9):793–795, February 2015. ISSN 0028-4793, 1533-4406. doi: 10.1056/NEJMp1500523. URL <http://www.nejm.org/doi/10.1056/NEJMp1500523>.
- [8] Elżbieta Kaja, Adrian Lejman, Dawid Sielski, Mateusz Sypniewski, Tomasz Gambin, Mateusz Dawidziuk, Tomasz Suchocki, Paweł Golik, Marzena Wojtaszewska, Magdalena Mroczek, Maria Stępień, Joanna Szyda, Karolina Lisiak-Teodorczyk, Filip Wolbach, Daria Kołodziejska, Katarzyna Ferdyn, Maciej Dąbrowski, Alicja Woźna, Marcin Żytkiewicz, Anna Bodora-Troińska, Waldemar Elikowski, Zbigniew J. Król, Artur Zaczyński, Agnieszka Pawlak, Robert Gil, Waldemar Wierzba, Paula Dobosz, Katarzyna Zawadzka, Paweł Zawadzki, and Paweł Sztromwasser. The Thousand Polish Genomes—A Database of Polish Variant Allele Frequencies. *International Journal of Molecular Sciences*, 23(9):4532, April 2022. ISSN 1422-0067. doi: 10.3390/ijms23094532. URL <https://www.mdpi.com/1422-0067/23/9/4532>.
- [9] Zachary D. Stephens, Skylar Y. Lee, Faraz Faghri, Roy H. Campbell, Chengxiang Zhai, Miles J. Efron, Ravishankar Iyer, Michael C. Schatz, Saurabh Sinha, and Gene E. Robinson. Big Data: Astronomical or Genomical? *PLOS Biology*, 13(7):e1002195, July 2015. ISSN 1545-7885. doi: 10.1371/journal.pbio.1002195. URL <https://dx.plos.org/10.1371/journal.pbio.1002195>.
- [10] Michael C Schatz, Ben Langmead, and Steven L Salzberg. Cloud computing and the DNA data race. *Nature Biotechnology*, 28(7):691–693, July 2010. ISSN 1087-0156, 1546-1696. doi: 10.1038/nbt0710-691. URL <http://www.nature.com/articles/nbt0710-691>.

- [11] John L. Hennessy and David A. Patterson. A new golden age for computer architecture. *Communications of the ACM*, 62(2):48–60, January 2019. ISSN 0001-0782, 1557-7317. doi: 10.1145/3282307. URL <https://dl.acm.org/doi/10.1145/3282307>.
- [12] Matt Massie, Frank Nothaft, Christopher Hartl, Christos Kozanitis, André Schumacher, Anthony D Joseph, and David A Patterson. Adam: Genomics formats and processing patterns for cloud scale computing. *University of California, Berkeley Technical Report, No. UCB/EECS-2013*, 207:2013, 2013.
- [13] Min Zhao, Qingguo Wang, Quan Wang, Peilin Jia, and Zhongming Zhao. Computational tools for copy number variation (CNV) detection using next-generation sequencing data: features and perspectives. *BMC Bioinformatics*, 14(S11):S1, September 2013. ISSN 1471-2105. doi: 10.1186/1471-2105-14-S11-S1. URL <https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-14-S11-S1>.
- [14] Heng Li and Jiazen Rong. Bedtk: finding interval overlap with implicit interval tree. *Bioinformatics*, 37(9):1315–1316, June 2021. ISSN 1367-4803. doi: 10.1093/BIOINFORMATICS/BTAA827. URL <https://academic.oup.com/bioinformatics/article/37/9/1315/5910546>. Publisher: Oxford Academic.
- [15] Aaron McKenna, Matthew Hanna, Eric Banks, Andrey Sivachenko, Kristian Cibulskis, Andrew Kernytsky, Kiran Garimella, David Altshuler, Stacey Gabriel, Mark Daly, and Mark A. DePristo. The Genome Analysis Toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Research*, 20(9):1297–1303, September 2010. ISSN 1088-9051. doi: 10.1101/gr.107524.110.
- [16] Michael D. Linderman, Davin Chia, Forrest Wallace, and Frank A. Nothaft. DECA: Scalable XHMM exome copy-number variant calling with ADAM and Apache Spark. *BMC Bioinformatics*, 20(1):1–8, October 2019. ISSN 14712105. doi: 10.1186/S12859-019-3108-7/TABLES/2. URL <https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-019-3108-7>. Publisher: BioMed Central Ltd.
- [17] Peter Vaillancourt, Bennett Wineholt, Brandon Barker, Plato Deliyannis, Jackie Zheng, Akshay Suresh, Adam Brazier, Rich Knepper, and Rich Wolski. Reproducible

- and Portable Workflows for Scientific Computing and HPC in the Cloud. In *Practice and Experience in Advanced Research Computing*, pages 311–320, New York, NY, USA, July 2020. ACM. ISBN 978-1-4503-6689-2. doi: 10.1145/3311790.3396659. URL <https://dl.acm.org/doi/10.1145/3311790.3396659>.
- [18] Carl Boettiger. An introduction to Docker for reproducible research. *ACM SIGOPS Operating Systems Review*, 49(1):71–79, January 2015. ISSN 0163-5980. doi: 10.1145/2723872.2723882. URL <https://dl.acm.org/doi/10.1145/2723872.2723882>.
- [19] Edmon Begoli, Ian Goethert, and Kathryn Knight. A Lakehouse Architecture for the Management and Analysis of Heterogeneous Data for Biomedical Research and Mega-biobanks. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 4643–4651. IEEE, December 2021. ISBN 978-1-66543-902-2. doi: 10.1109/BigData52589.2021.9671534. URL <https://ieeexplore.ieee.org/document/9671534/>.
- [20] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, January 2008. ISSN 0001-0782, 1557-7317. doi: 10.1145/1327452.1327492. URL <https://dl.acm.org/doi/10.1145/1327452.1327492>.
- [21] Tom White. *Hadoop: the definitive guide*. O'Reilly, Beijing, third edition edition, 2012. ISBN 978-1-4493-1152-0.
- [22] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The Hadoop Distributed File System. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10, Incline Village, NV, USA, May 2010. IEEE. ISBN 978-1-4244-7152-2. doi: 10.1109/MSST.2010.5496972. URL <http://ieeexplore.ieee.org/document/5496972/>.
- [23] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 29–43, Bolton Landing NY USA, October 2003. ACM. ISBN 978-1-58113-757-6. doi: 10.1145/945445.945450. URL <https://dl.acm.org/doi/10.1145/945445.945450>.
- [24] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth

- Seth, Bikas Saha, Carlo Curino, Owen O’Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. Apache Hadoop YARN: yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*, pages 1–16, Santa Clara California, October 2013. ACM. ISBN 978-1-4503-2428-1. doi: 10.1145/2523616.2523633. URL <https://dl.acm.org/doi/10.1145/2523616.2523633>.
- [25] James Dixon. Pentaho, Hadoop, and data lakes, October 2010. URL: <https://jamesdixon.wordpress.com/2010/10/14/pentahohadoop-and-data-lakes>, 2010.
- [26] Alan F. Gates, Olga Natkovich, Shubham Chopra, Pradeep Kamath, Shravan M. Narayananamurthy, Christopher Olston, Benjamin Reed, Santhosh Srinivasan, and Utkarsh Srivastava. Building a high-level dataflow system on top of Map-Reduce: the Pig experience. *Proceedings of the VLDB Endowment*, 2(2):1414–1425, August 2009. ISSN 2150-8097. doi: 10.14778/1687553.1687568. URL <https://dl.acm.org/doi/10.14778/1687553.1687568>.
- [27] Jeremy Kepner and Hayden Jananthan. *Mathematics of big data: Spreadsheets, databases, matrices, and graphs*. MIT Press, 2018.
- [28] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, June 1970. ISSN 0001-0782, 1557-7317. doi: 10.1145/362384.362685. URL <https://dl.acm.org/doi/10.1145/362384.362685>.
- [29] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1099–1110, Vancouver Canada, June 2008. ACM. ISBN 978-1-60558-102-6. doi: 10.1145/1376616.1376726. URL <https://dl.acm.org/doi/10.1145/1376616.1376726>.
- [30] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Ning Zhang, Suresh Antony, Hao Liu, and Raghotham Murthy. Hive - a petabyte scale data warehouse using Hadoop. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, pages 996–1005, Long Beach, CA, USA, 2010. IEEE. ISBN 978-1-4244-5445-7. doi: 10.1109/ICDE.2010.5447738. URL <http://ieeexplore.ieee.org/document/5447738/>.

- [31] W. H. Inmon. The data warehouse and data mining. *Communications of the ACM*, 39(11):49–50, November 1996. ISSN 0001-0782, 1557-7317. doi: 10.1145/240455.240470. URL <https://dl.acm.org/doi/10.1145/240455.240470>.
- [32] Simone Leo and Gianluigi Zanetti. Pydoop: a Python MapReduce and HDFS API for Hadoop. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing - HPDC '10*, page 819, Chicago, Illinois, 2010. ACM Press. ISBN 978-1-60558-942-8. doi: 10.1145/1851476.1851594. URL <http://portal.acm.org/citation.cfm?doid=1851476.1851594>.
- [33] Sebastian Bassi. A Primer on Python for Life Science Researchers. *PLoS Computational Biology*, 3(11):e199, November 2007. ISSN 1553-7358. doi: 10.1371/journal.pcbi.0030199. URL <https://dx.plos.org/10.1371/journal.pcbi.0030199>.
- [34] Mengwei Ding, Long Zheng, Yanchao Lu, Li Li, Song Guo, and Minyi Guo. More convenient more overhead: the performance evaluation of Hadoop streaming. In *Proceedings of the 2011 ACM Symposium on Research in Applied Computation*, pages 307–313, Miami Florida, November 2011. ACM. ISBN 978-1-4503-1087-1. doi: 10.1145/2103380.2103444. URL <https://dl.acm.org/doi/10.1145/2103380.2103444>.
- [35] Ronald C Taylor. An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics. *BMC Bioinformatics*, 11(S12):S1, December 2010. ISSN 1471-2105. doi: 10.1186/1471-2105-11-S12-S1. URL <https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-11-S12-S1>.
- [36] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 10)*, 2010.
- [37] Michael Armbrust, Reynold S. Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K. Bradley, Xiangrui Meng, Tomer Kaftan, Michael J. Franklin, Ali Ghodsi, and Matei Zaharia. Spark SQL. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1383–1394, New York, NY, USA, May 2015. ACM. ISBN 978-1-4503-2758-9. doi: 10.1145/2723372.2742797. URL <https://dl.acm.org/doi/10.1145/2723372.2742797>.

- [38] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, Doris Xin, Reynold Xin, Michael J. Franklin, Reza Zadeh, Matei Zaharia, and Ameet Talwalkar. MLLib: Machine Learning in Apache Spark. 2015. doi: 10.48550/ARXIV.1505.06807. URL <https://arxiv.org/abs/1505.06807>. Publisher: arXiv Version Number: 1.
- [39] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. Apache Spark: a unified engine for big data processing. *Communications of the ACM*, 59(11):56–65, October 2016. ISSN 0001-0782, 1557-7317. doi: 10.1145/2934664. URL <https://dl.acm.org/doi/10.1145/2934664>.
- [40] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. Borg, Omega, and Kubernetes. *Communications of the ACM*, 59(5):50–57, April 2016. ISSN 0001-0782, 1557-7317. doi: 10.1145/2890784. URL <https://dl.acm.org/doi/10.1145/2890784>.
- [41] Diogo Castro, Prasanth Kothuri, Piotr Mrowczynski, Danilo Piparo, and Enric Tejedor. Apache Spark usage and deployment models for scientific computing. *EPJ Web of Conferences*, 214:07020, September 2019. ISSN 2100-014X. doi: 10.1051/epjconf/201921407020.
- [42] Inès Krissaane, Carlos De Niz, Alba Gutiérrez-Sacristán, Gabor Korodi, Nneka Ede, Ranjay Kumar, Jessica Lyons, Arjun Manrai, Chirag Patel, Isaac Kohane, and Paul Avillach. Scalability and cost-effectiveness analysis of whole genome-wide association studies on Google Cloud Platform and Amazon Web Services. *Journal of the American Medical Informatics Association*, 27(9):1425–1430, September 2020. ISSN 1067-5027. doi: 10.1093/jamia/ocaa068.
- [43] Dariusz Mrozek, Krzysztof Stępień, Piotr Grzesik, and Bożena Małysiak-Mrozek. A Large-Scale and Serverless Computational Approach for Improving Quality of NGS Data Supporting Big Multi-Omics Data Analyses. *Frontiers in Genetics*, 12:1078, July 2021. ISSN 16648021. doi: 10.3389/FGENE.2021.699280/BIBTEX. Publisher: Frontiers Media S.A.

- [44] Zhenhua Guo, Geoffrey Fox, and Mo Zhou. Investigation of Data Locality in MapReduce. In *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pages 419–426, Ottawa, Canada, May 2012. IEEE. ISBN 978-1-4673-1395-7 978-0-7695-4691-9. doi: 10.1109/CCGrid.2012.42. URL <http://ieeexplore.ieee.org/document/6217449/>.
- [45] Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica. Disk-locality in datacenter computing considered irrelevant. In *Proceedings of the 13th USENIX conference on hot topics in operating systems*, HotOS’13, page 12, USA, 2011. USENIX Association. URL http://www.usenix.org/events/hotos11/tech/final_files/Ananthanarayanan.pdf. Number of pages: 1 Place: Napa, California.
- [46] Michael Armbrust, Ali Ghodsi, Reynold Xin, and Matei Zaharia. Lakehouse: a new generation of open platforms that unify data warehousing and advanced analytics. In *Proceedings of CIDR*, 2021.
- [47] Ahmed A. Harby and Farhana Zulkernine. From Data Warehouse to Lakehouse: A Comparative Review. In *2022 IEEE International Conference on Big Data (Big Data)*, pages 389–395, Osaka, Japan, December 2022. IEEE. ISBN 978-1-66548-045-1. doi: 10.1109/BigData55660.2022.10020719. URL <https://ieeexplore.ieee.org/document/10020719/>.
- [48] Raghu Ramakrishnan, Baskar Sridharan, John R. Douceur, Pavan Kasturi, Balaji Krishnamachari-Sampath, Karthick Krishnamoorthy, Peng Li, Mitica Manu, Spiro Michaylov, Rogério Ramos, Neil Sharman, Zee Xu, Youssef Barakat, Chris Douglas, Richard Draves, Shrikant S. Naidu, Shankar Shastry, Atul Sikaria, Simon Sun, and Ramarathnam Venkatesan. Azure Data Lake Store: A Hyperscale Distributed File Service for Big Data Analytics. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 51–63, Chicago Illinois USA, May 2017. ACM. ISBN 978-1-4503-4197-4. doi: 10.1145/3035918.3056100. URL <https://dl.acm.org/doi/10.1145/3035918.3056100>.
- [49] Mayur R. Palankar, Adriana Iamnitchi, Matei Ripeanu, and Simson Garfinkel. Amazon S3 for science grids: a viable solution? In *Proceedings of the 2008 international workshop on Data-aware distributed computing*, pages 55–64, Boston MA USA,

June 2008. ACM. ISBN 978-1-60558-154-5. doi: 10.1145/1383519.1383526. URL <https://dl.acm.org/doi/10.1145/1383519.1383526>.

- [50] Ekaba Bisong. Google Cloud Storage (GCS). In *Building Machine Learning and Deep Learning Models on Google Cloud Platform*, pages 25–33. Apress, Berkeley, CA, 2019. ISBN 978-1-4842-4469-2 978-1-4842-4470-8. doi: 10.1007/978-1-4842-4470-8_4. URL http://link.springer.com/10.1007/978-1-4842-4470-8_4.
- [51] Pramod Singh. Manage Data with PySpark. In *Machine Learning with PySpark*, pages 15–37. Apress, Berkeley, CA, 2022. ISBN 978-1-4842-7776-8 978-1-4842-7777-5. doi: 10.1007/978-1-4842-7777-5_2. URL https://link.springer.com/10.1007/978-1-4842-7777-5_2.
- [52] Adam Morton. Developing Applications in Snowflake. In *Mastering Snowflake Solutions*, pages 201–219. Apress, Berkeley, CA, 2022. ISBN 978-1-4842-8028-7 978-1-4842-8029-4. doi: 10.1007/978-1-4842-8029-4_10. URL https://link.springer.com/10.1007/978-1-4842-8029-4_10.
- [53] Alexander Behm, Shoumik Palkar, Utkarsh Agarwal, Timothy Armstrong, David Cashman, Ankur Dave, Todd Greenstein, Shant Hovsepian, Ryan Johnson, Arvind Sai Krishnan, Paul Leventis, Ala Luszczak, Prashanth Menon, Mostafa Mokhtar, Gene Pang, Sameer Paranjpye, Greg Rahn, Bart Samwel, Tom Van Bussel, Herman Van Hovell, Maryann Xue, Reynold Xin, and Matei Zaharia. Photon: A Fast Query Engine for Lakehouse Systems. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 2326–2339, June 2022. ISSN 07308078. doi: 10.1145/3514221.3526054. URL <https://dl.acm.org/doi/10.1145/3514221.3526054>. Publisher: Association for Computing Machinery ISBN: 9781450392495.
- [54] Pedro Pedreira, Orri Erling, Masha Basanova, Kevin Wilfong, Laith Sakka, Krishna Pai, Wei He, and Biswapedesh Chattopadhyay. Velox. *Proceedings of the VLDB Endowment*, 15(12):3372–3384, August 2022. ISSN 21508097. doi: 10.14778/3554821.3554829. URL <https://dl.acm.org/doi/10.14778/3554821.3554829>. Publisher: VLDB Endowment PUB4722.
- [55] Mariem Brahem, Laurent Yeh, and Karine Zeitouni. Efficient astronomical query processing using spark. In *Proceedings of the 26th ACM SIGSPATIAL International*

- Conference on Advances in Geographic Information Systems*, pages 229–238, Seattle Washington, November 2018. ACM. ISBN 978-1-4503-5889-7. doi: 10.1145/3274895.3274942. URL <https://dl.acm.org/doi/10.1145/3274895.3274942>.
- [56] Avrilia Floratou, Umar Farooq Minhas, and Fatma Özcan. SQL-on-Hadoop: full circle back to shared-nothing database architectures. *Proceedings of the VLDB Endowment*, 7(12):1295–1306, August 2014. ISSN 2150-8097. doi: 10.14778/2732977.2733002. URL <https://dl.acm.org/doi/10.14778/2732977.2733002>.
- [57] Michael Armbrust, Tathagata Das, Liwen Sun, Burak Yavuz, Shixiong Zhu, Mukul Murthy, Joseph Torres, Herman van Hovell, Adrian Ionescu, Alicja Łuszczak, Michał Świtakowski, Michał Szafrański, Xiao Li, Takuya Ueshin, Mostafa Mokhtar, Peter Boncz, Ali Ghodsi, Sameer Paranjpye, Pieter Senster, Reynold Xin, and Matei Zaharia. Delta lake. *Proceedings of the VLDB Endowment*, 13(12):3411–3424, August 2020. ISSN 2150-8097. doi: 10.14778/3415478.3415560. URL <https://dl.acm.org/doi/10.14778/3415478.3415560>. Publisher: VLDB Endowment.
- [58] Overview | Apache Hudi. URL <https://hudi.apache.org/docs/overview>.
- [59] Paras Jain, Peter Kraft, Conor Power, Tathagata Das, Ion Stoica, and Matei Zaharia. Analyzing and comparing lakehouse storage systems. *CIDR*, January 2023. URL <https://www.cidrdb.org/cidr2023/papers/p92-jain.pdf>.
- [60] Sara Goodwin, John D. McPherson, and W. Richard McCombie. Coming of age: ten years of next-generation sequencing technologies. *Nature Reviews Genetics*, 17(6):333–351, June 2016. ISSN 1471-0056, 1471-0064. doi: 10.1038/nrg.2016.49. URL <http://www.nature.com/articles/nrg.2016.49>.
- [61] Gavin R Oliver, Steven N Hart, and Eric W Klee. Bioinformatics for Clinical Next Generation Sequencing. *Clinical Chemistry*, 61(1):124–135, January 2015. ISSN 0009-9147, 1530-8561. doi: 10.1373/clinchem.2014.224360. URL <https://academic.oup.com/clinchem/article/61/1/124/5611448>.
- [62] Rute Pereira, Jorge Oliveira, and Mário Sousa. Bioinformatics and Computational Tools for Next-Generation Sequencing Analysis in Clinical Genetics. *Journal of Clinical Medicine*, 9(1):132, January 2020. ISSN 2077-0383. doi: 10.3390/jcm9010132. URL <https://www.mdpi.com/2077-0383/9/1/132>.

- [63] Jose Garrido-Cardenas, Federico Garcia-Maroto, Jose Alvarez-Bermejo, and Francisco Manzano-Agugliaro. DNA Sequencing Sensors: An Overview. *Sensors*, 17(3):588, March 2017. ISSN 1424-8220. doi: 10.3390/s17030588. URL <http://www.mdpi.com/1424-8220/17/3/588>.
- [64] Shanika L. Amarasinghe, Shian Su, Xueyi Dong, Luke Zappia, Matthew E. Ritchie, and Quentin Gouil. Opportunities and challenges in long-read sequencing data analysis. *Genome Biology*, 21(1):30, December 2020. ISSN 1474-760X. doi: 10.1186/s13059-020-1935-5. URL <https://genomebiology.biomedcentral.com/articles/10.1186/s13059-020-1935-5>.
- [65] Peter J. A. Cock, Christopher J. Fields, Naohisa Goto, Michael L. Heuer, and Peter M. Rice. The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic Acids Research*, 38(6):1767–1771, April 2010. ISSN 0305-1048, 1362-4962. doi: 10.1093/nar/gkp1137. URL <https://academic.oup.com/nar/article-lookup/doi/10.1093/nar/gkp1137>.
- [66] W R Pearson and D J Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences*, 85(8):2444–2448, April 1988. ISSN 0027-8424, 1091-6490. doi: 10.1073/pnas.85.8.2444. URL <https://pnas.org/doi/full/10.1073/pnas.85.8.2444>.
- [67] S P Pfeifer. From next-generation resequencing reads to a high-quality variant data set. *Heredity*, 118(2):111–124, February 2017. ISSN 0018-067X, 1365-2540. doi: 10.1038/hdy.2016.102. URL <https://www.nature.com/articles/hdy2016102>.
- [68] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, and R. Durbin. The Sequence Alignment/Map format and SAM-tools. *Bioinformatics*, 25(16):2078–2079, August 2009. ISSN 1367-4803. doi: 10.1093/bioinformatics/btp352.
- [69] Markus Hsi-Yang Fritz, Rasko Leinonen, Guy Cochrane, and Ewan Birney. Efficient storage of high throughput DNA sequencing data using reference-based compression. *Genome Research*, 21(5):734–740, May 2011. ISSN 1088-9051. doi: 10.1101/gr.114819.110. URL <http://genome.cshlp.org/lookup/doi/10.1101/gr.114819.110>.

- [70] P. Danecek, A. Auton, G. Abecasis, C. A. Albers, E. Banks, M. A. DePristo, R. E. Handsaker, G. Lunter, G. T. Marth, S. T. Sherry, G. McVean, R. Durbin, and 1000 Genomes Project Analysis Group. The variant call format and VCFtools. *Bioinformatics*, 27(15):2156–2158, August 2011. ISSN 1367-4803, 1460-2059. doi: 10.1093/bioinformatics/btr330. URL <https://academic.oup.com/bioinformatics/article-lookup/doi/10.1093/bioinformatics/btr330>.
- [71] Samuel Valentini, Tarcisio Fedrizzi, Francesca Demichelis, and Alessandro Romanel. PaCBAM: fast and scalable processing of whole exome and targeted sequencing data. *BMC Genomics*, 20(1):1018, December 2019. ISSN 1471-2164. doi: 10.1186/s12864-019-6386-6. URL <https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12864-019-6386-6>.
- [72] Cliff Meldrum, Maria A. Doyle, and Richard W. Tothill. Next-generation sequencing for cancer diagnostics: a practical perspective. *The Clinical Biochemist. Reviews*, 32(4):177–195, November 2011. ISSN 1838-0212.
- [73] Sara Pidò, Pietro Crovari, and Franca Garzotto. Modelling the bioinformatics tertiary analysis research process. *BMC Bioinformatics*, 22(S13):452, September 2021. ISSN 1471-2105. doi: 10.1186/s12859-021-04310-5. URL <https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-021-04310-5>.
- [74] Marco Masseroli, Abdulrahman Kaitoua, Pietro Pinoli, and Stefano Ceri. Modeling and interoperability of heterogeneous genomic big data for integrative processing and querying. *Methods*, 111:3–11, December 2016. ISSN 10462023. doi: 10.1016/j.ymeth.2016.09.002. URL <https://linkinghub.elsevier.com/retrieve/pii/S1046202316303012>. Publisher: Academic Press.
- [75] Charles A. Steward, Alasdair P. J. Parker, Berge A. Minassian, Sanjay M. Sisodiya, Adam Frankish, and Jennifer Harrow. Genome annotation for clinical genomic diagnostics: strengths and weaknesses. *Genome Medicine*, 9(1):49, December 2017. ISSN 1756-994X. doi: 10.1186/s13073-017-0441-1. URL <https://genomemedicine.biomedcentral.com/articles/10.1186/s13073-017-0441-1>.
- [76] William McLaren, Laurent Gil, Sarah E. Hunt, Harpreet Singh Riat, Graham R. S.

- Ritchie, Anja Thormann, Paul Flicek, and Fiona Cunningham. The Ensembl Variant Effect Predictor. *Genome Biology*, 17(1):122, December 2016. ISSN 1474-760X. doi: 10.1186/s13059-016-0974-4. URL <http://genomebiology.biomedcentral.com/articles/10.1186/s13059-016-0974-4>.
- [77] Marco Masseroli, Arif Canakoglu, Pietro Pinoli, Abdulrahman Kaitoua, Andrea Gulino, Olha Horlova, Luca Nanni, Anna Bernasconi, Stefano Perna, Eirini Stamoulakatou, and Stefano Ceri. Processing of big heterogeneous genomic datasets for tertiary analysis of Next Generation Sequencing data. *Bioinformatics*, 35(5):729–736, March 2019. ISSN 1367-4803. doi: 10.1093/BIOINFORMATICS/BTY688. URL <https://academic.oup.com/bioinformatics/article/35/5/729/5067860>. Publisher: Oxford Academic.
- [78] Aisling O'Driscoll, Jurate Daugelaite, and Roy D. Sleator. Big data, Hadoop and cloud computing in genomics. *Journal of Biomedical Informatics*, 46(5):774–781, October 2013. ISSN 15320464. doi: 10.1016/j.jbi.2013.07.001. URL <https://linkinghub.elsevier.com/retrieve/pii/S1532046413001007>.
- [79] Simone Pallotta, Silvia Cascianelli, and Marco Masseroli. RGMQL: scalable and interoperable computing of heterogeneous omics big data and metadata in R/Bioconductor. *BMC Bioinformatics*, 23(1):1–28, December 2022. ISSN 14712105. doi: 10.1186/S12859-022-04648-4/FIGURES/8. URL <https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-022-04648-4>. Publisher: BioMed Central Ltd.
- [80] Frank Austin Nothaft, Matt Massie, Timothy Danford, Zhao Zhang, Uri Laserson, Carl Yeksigian, Jey Kottalam, Arun Ahuja, Jeff Hammerbacher, Michael Linderman, Michael J. Franklin, Anthony D. Joseph, and David A. Patterson. Rethinking data-intensive science using scalable analytics systems. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2015-May:631–646, May 2015. ISSN 07308078. doi: 10.1145/2723372.2742787. URL <https://dl.acm.org/doi/10.1145/2723372.2742787>. Publisher: Association for Computing Machinery ISBN: 9781450327589.
- [81] Deepak Vohra. Apache Avro. In *Practical Hadoop Ecosystem*, pages 303–

323. Apress, Berkeley, CA, 2016. ISBN 978-1-4842-2198-3 978-1-4842-2199-0. doi: 10.1007/978-1-4842-2199-0_7. URL http://link.springer.com/10.1007/978-1-4842-2199-0_7.
- [82] Deepak Vohra. Apache Parquet. In *Practical Hadoop Ecosystem*, pages 325–335. Apress, Berkeley, CA, 2016. ISBN 978-1-4842-2198-3 978-1-4842-2199-0. doi: 10.1007/978-1-4842-2199-0_8. URL http://link.springer.com/10.1007/978-1-4842-2199-0_8.
- [83] Michael L Heuer, Frank Austin Nothaft, and Walter Blair. biodatageeks/cannoli: cannoli-parent-spark3_2.12-1.0, January 2023. URL <https://zenodo.org/record/7581150>.
- [84] Walter Blair, Leonardo De Melo Joao, Larry Davis, and Paul Anderson. Streamlining the Genomics Processing Pipeline via Named Pipes and Persistent Spark Satasets. In *2017 IEEE 17th International Conference on Bioinformatics and Bioengineering (BIBE)*, pages 35–38, Washington, DC, October 2017. IEEE. ISBN 978-1-5386-1324-5. doi: 10.1109/BIBE.2017.8251262. URL <https://ieeexplore.ieee.org/document/8251262/>.
- [85] Heng Li. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv preprint arXiv:1303.3997*, 2013.
- [86] Adrian Tan, Gonçalo R. Abecasis, and Hyun Min Kang. Unified representation of genetic variants. *Bioinformatics*, 31(13):2202–2204, July 2015. ISSN 1367-4811, 1367-4803. doi: 10.1093/bioinformatics/btv112. URL <https://academic.oup.com/bioinformatics/article/31/13/2202/196142>.
- [87] Menachem Fromer, Jennifer L. Moran, Kimberly Chambert, Eric Banks, Sarah E. Bergen, Douglas M. Ruderfer, Robert E. Handsaker, Steven A. McCarroll, Michael C. O’Donovan, Michael J. Owen, George Kirov, Patrick F. Sullivan, Christina M. Hultman, Pamela Sklar, and Shaun M. Purcell. Discovery and Statistical Genotyping of Copy-Number Variation from Whole-Exome Sequencing Depth. *The American Journal of Human Genetics*, 91(4):597–607, October 2012. ISSN 00029297. doi: 10.1016/j.ajhg.2012.08.005. URL <https://linkinghub.elsevier.com/retrieve/pii/S000292971200417X>.

- [88] Dariusz R Augustyn, Łukasz Wycislik, and Dariusz Mrozek. Perspectives of using Cloud computing in integrative analysis of multi-omics data. *Briefings in Functional Genomics*, March 2021. ISSN 2041-2649. doi: 10.1093/bfgp/elab007.
- [89] Marco Masseroli, Pietro Pinoli, Francesco Venco, Abdulrahman Kaitoua, Vahid Jalili, Fernando Palluzzi, Heiko Muller, and Stefano Ceri. GenoMetric Query Language: a novel approach to large-scale genomic data management. *Bioinformatics*, 31(12):1881–1888, June 2015. ISSN 1367-4803, 1460-2059. doi: 10.1093/bioinformatics/btv048. URL <https://academic.oup.com/bioinformatics/article-lookup/doi/10.1093/bioinformatics/btv048>.
- [90] Christos Kozanitis and David A. Patterson. GenAp: A distributed SQL interface for genomic data. *BMC Bioinformatics*, 17(1):1–8, February 2016. ISSN 14712105. doi: 10.1186/S12859-016-0904-1/TABLES/2. URL <https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-016-0904-1>. Publisher: BioMed Central Ltd.
- [91] Petr Danecek, James K Bonfield, Jennifer Liddle, John Marshall, Valeriu Ohan, Martin O Pollard, Andrew Whitwham, Thomas Keane, Shane A McCarthy, Robert M Davies, and Heng Li. Twelve years of SAMtools and BCFtools. *GigaScience*, 10(2), January 2021. ISSN 2047-217X. doi: 10.1093/gigascience/giab008. URL <https://academic.oup.com/gigascience/article/doi/10.1093/gigascience/giab008/6137722>.
- [92] James K Bonfield, John Marshall, Petr Danecek, Heng Li, Valeriu Ohan, Andrew Whitwham, Thomas Keane, and Robert M Davies. HTSlip: C library for reading/writing high-throughput sequencing data. *GigaScience*, 10(2), January 2021. ISSN 2047-217X. doi: 10.1093/gigascience/giab007. URL <https://academic.oup.com/gigascience/article/doi/10.1093/gigascience/giab007/6139334>.
- [93] Artem Tarasov, Albert J. Vilella, Edwin Cuppen, Isaac J. Nijman, and Pjotr Prins. Sambamba: fast processing of NGS alignment formats. *Bioinformatics*, 31(12):2032–2034, June 2015. ISSN 1367-4803. doi: 10.1093/bioinformatics/btv098.
- [94] Christos Kozanitis, Andrew Heiberg, George Varghese, and Vineet Bafna. Using Genome Query Language to uncover genetic variation. *Bioinformatics*, 30(1):1–8,

January 2014. ISSN 1367-4803. doi: 10.1093/BIOINFORMATICS/BTT250. URL <https://academic.oup.com/bioinformatics/article/30/1/1/234445>. Publisher: Oxford Academic.

- [95] Hákon Guðbjartsson, Guðmundur Fr. Georgsson, Sigurjón A. Guðjónsson, Ragnar Þór Valdimarsson, Jóhann H. Sigurðsson, Sigmar K. Stefánsson, Gísli Másson, Gísli Magnússon, Vilmundur Pálmason, and Kári Stefánsson. GORpipe: a query tool for working with sequence data based on a Genomic Ordered Relational (GOR) architecture. *Bioinformatics*, 32(20):3081–3088, October 2016. ISSN 1367-4803, 1460-2059. doi: 10.1093/bioinformatics/btw199. URL <https://academic.oup.com/bioinformatics/article-lookup/doi/10.1093/bioinformatics/btw199>.
- [96] Christopher T. Lee and Manolis Maragkakis. SamQL: a structured query language and filtering tool for the SAM/BAM file format. *BMC Bioinformatics*, 22(1):1–8, December 2021. ISSN 14712105. doi: 10.1186/S12859-021-04390-3/FIGURES/2. URL <https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-021-04390-3>. Publisher: BioMed Central Ltd.
- [97] Xinjie Zhu, Qiang Zhang, Eric Dun Ho, Ken Hung-On Yu, Chris Liu, Tim H. Huang, Alfred Sze-Lok Cheng, Ben Kao, Eric Lo, and Kevin Y. Yip. START: a system for flexible analysis of hundreds of genomic signal tracks in few lines of SQL-like queries. *BMC Genomics*, 18(1):749, December 2017. ISSN 1471-2164. doi: 10.1186/s12864-017-4071-1. URL <http://bmngenomics.biomedcentral.com/articles/10.1186/s12864-017-4071-1>.
- [98] Luca Nanni, Pietro Pinoli, Arif Canakoglu, and Stefano Ceri. PyGMQL: scalable data extraction and analysis for heterogeneous genomic datasets. *BMC Bioinformatics*, 20(1):560, December 2019. ISSN 1471-2105. doi: 10.1186/s12859-019-3159-9. URL <https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-019-3159-9>.
- [99] Karen Feng, Henry Davidge, Williambrandler, Kiavash Kianfar, T. Alex, Mah-Databricks, Boris Boutkov, Brian Cajes, Ahir Reddy, Amir Kermany, Cameron Smith, Douglas Moore, Frank Austin Nothaft, Herman Van Hovell, Joseph Bradley, Leland,

Michael L Heuer, and Dim De Grave. projectglow/glow: v1.2.1, April 2022. URL <https://zenodo.org/record/6474169>.

- [100] Arash Bayat, Piotr Szul, Aidan R O'Brien, Robert Dunne, Brendan Hosking, Yatish Jain, Cameron Hosking, Oscar J Luo, Natalie Twine, and Denis C Bauer. VariantSpark: Cloud-based machine learning for association study of complex phenotype and large-scale genomic data. *GigaScience*, 9(8), August 2020. ISSN 2047-217X. doi: 10.1093/gigascience/giaa077.
- [101] Hail Team. Hail, December 2022. URL <https://zenodo.org/record/7439252>.
- [102] Ram Vinay Pandey and Christian Schlötterer. DistMap: A Toolkit for Distributed Short Read Mapping on a Hadoop Cluster. *PLoS ONE*, 8(8):e72614, August 2013. ISSN 1932-6203. doi: 10.1371/journal.pone.0072614. URL <https://dx.plos.org/10.1371/journal.pone.0072614>.
- [103] Luca Pireddu, Simone Leo, and Gianluigi Zanetti. Seal: A distributed short read mapping and duplicate removal tool. *Bioinformatics*, 27(15):2159–2160, August 2011. ISSN 13674803. doi: 10.1093/BIOINFORMATICS/BTR325.
- [104] José M. Abuín, Juan C. Pichel, Tomás F. Pena, and Jorge Amigo. SparkBWA: Speeding Up the Alignment of High-Throughput DNA Sequencing Data. *PLOS ONE*, 11(5):e0155461, May 2016. ISSN 1932-6203. doi: 10.1371/journal.pone.0155461. URL <https://dx.plos.org/10.1371/journal.pone.0155461>.
- [105] André Schumacher, Luca Pireddu, Matti Niemenmaa, Aleksi Kallio, Eija Korpelainen, Gianluigi Zanetti, and Keijo Heljanko. SeqPig: simple and scalable scripting for large sequencing data sets in Hadoop. *Bioinformatics*, 30(1):119–120, January 2014. ISSN 1460-2059. doi: 10.1093/bioinformatics/btt601.
- [106] Matti Niemenmaa, Aleksi Kallio, André Schumacher, Petri Klemelä, Eija Korpelainen, and Keijo Heljanko. Hadoop-BAM: directly manipulating next generation sequencing data in the cloud. *Bioinformatics*, 28(6):876–877, March 2012. ISSN 1460-2059. doi: 10.1093/bioinformatics/bts054.
- [107] Tom White, Michael L. Heuer, Louis Bergelson, Chris Norman, Daniel Gómez-

Sánchez, and tedsharpe. biodatageeks/disq: disq-0.3.8, January 2023. URL <https://doi.org/10.5281/zenodo.7581194>.

- [108] Geraldine A. Van der Auwera. From FastQ Data to High [U+2010] Confidence Variant Calls: The Genome Analysis Toolkit Best Practices pipeline. *Current Protocols in Bioinformatics*, 43(1), October 2013. ISSN 1934-3396, 1934-340X. doi: 10.1002/0471250953.bi1110s43. URL <https://onlinelibrary.wiley.com/doi/10.1002/0471250953.bi1110s43>.
- [109] Hamid Mushtaq, Frank Liu, Carlos Costa, Gang Liu, Peter Hofstee, and Zaid Al-Ars. SparkGA. In *Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*, pages 148–157, New York, NY, USA, August 2017. ACM. ISBN 978-1-4503-4722-8. doi: 10.1145/3107411.3107438. URL <https://dl.acm.org/doi/10.1145/3107411.3107438>.
- [110] Hamid Mushtaq, Nauman Ahmed, and Zaid Al-Ars. SparkGA2: Production-quality memory-efficient Apache Spark based genome analysis framework. *PLOS ONE*, 14(12):e0224784, December 2019. ISSN 1932-6203. doi: 10.1371/journal.pone.0224784. URL <https://dx.plos.org/10.1371/journal.pone.0224784>.
- [111] Dries Decap, Joke Reumers, Charlotte Herzeel, Pascal Costanza, and Jan Fostier. Halvade: scalable sequence analysis with MapReduce. *Bioinformatics*, 31(15):2482–2488, August 2015. ISSN 1367-4803. doi: 10.1093/bioinformatics/btv179. URL <https://academic.oup.com/bioinformatics/article-lookup/doi/10.1093/bioinformatics/btv179>.
- [112] Dries Decap, Joke Reumers, Charlotte Herzeel, Pascal Costanza, and Jan Fostier. Halvade-RNA: Parallel variant calling from transcriptomic data using MapReduce. *PLOS ONE*, 12(3):e0174575, March 2017. ISSN 1932-6203. doi: 10.1371/journal.pone.0174575.
- [113] Alyssa Kramer Morrow, George Zhixuan He, Frank Austin Nothaft, Eric Tongching Tu, Justin Paschall, Nir Yosef, and Anthony Douglas Joseph. Mango: Exploratory Data Analysis for Large-Scale Sequencing Datasets. *Cell Systems*, 9(6):609–613.e3, December 2019. ISSN 2405-4712. doi: 10.1016/J.CELS.2019.11.002. Publisher: Cell Press.

- [114] José M. Abuín, Juan C. Pichel, Tomás F. Pena, and Jorge Amigo. BigBWA: approaching the Burrows–Wheeler aligner to Big Data technologies. *Bioinformatics*, page btv506, August 2015. ISSN 1367-4803, 1460-2059. doi: 10.1093/bioinformatics/btv506. URL <https://academic.oup.com/bioinformatics/article-lookup/doi/10.1093/bioinformatics/btv506>.
- [115] Lingqi Zhang, Cheng Liu, and Shoubin Dong. PipeMEM: A Framework to Speed Up BWA-MEM in Spark with Low Overhead. *Genes*, 10(11):886, November 2019. ISSN 2073-4425. doi: 10.3390/genes10110886. URL <https://www.mdpi.com/2073-4425/10/11/886>.
- [116] Guoguang Zhao, Cheng Ling, and Donghong Sun. SparkSW: Scalable Distributed Computing System for Large-Scale Biological Sequence Alignment. In *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 845–852, Shenzhen, China, May 2015. IEEE. ISBN 978-1-4799-8006-2. doi: 10.1109/CCGrid.2015.55. URL <http://ieeexplore.ieee.org/document/7152568/>.
- [117] Bo Xu, Changlong Li, Hang Zhuang, Jiali Wang, Qingfeng Wang, Jinhong Zhou, and Xuehai Zhou. DSA: Scalable Distributed Sequence Alignment System Using SIMD Instructions. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 758–761, Madrid, Spain, May 2017. IEEE. ISBN 978-1-5090-6611-7. doi: 10.1109/CCGRID.2017.74. URL <http://ieeexplore.ieee.org/document/7973775/>.
- [118] Bo Xu, Changlong Li, Hang Zhuang, Jiali Wang, Qingfeng Wang, and Xuehai Zhou. Efficient Distributed Smith-Waterman Algorithm Based on Apache Spark. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pages 608–615, Honolulu, CA, USA, June 2017. IEEE. ISBN 978-1-5386-1993-3. doi: 10.1109/CLOUD.2017.83. URL <http://ieeexplore.ieee.org/document/8030640/>.
- [119] Dries Decap, Louise de Schaetzen van Brienen, Maarten Larmuseau, Pascal Costanza, Charlotte Herzeel, Roel Wuyts, Kathleen Marchal, and Jan Fostier. Halvade somatic: Somatic variant calling with Apache Spark. *GigaScience*, 11, January 2022. ISSN 2047-217X. doi: 10.1093/gigascience/giab094.

- [120] Tanveer Ahmad, Zaid Al Ars, and H Peter Hofstee. VC@Scale: Scalable and high-performance variant calling on cluster environments. *GigaScience*, 10(9), September 2021. ISSN 2047-217X. doi: 10.1093/gigascience/giab057.
- [121] Zaid Al-Ars, Saiyi Wang, and Hamid Mushtaq. SparkRA: Enabling Big Data Scalability for the GATK RNA-seq Pipeline with Apache Spark. *Genes*, 11(1):53, January 2020. ISSN 2073-4425. doi: 10.3390/genes11010053. URL <https://www.mdpi.com/2073-4425/11/1/53>.
- [122] Liren Huang, Jan Krüger, and Alexander Sczyrba. Analyzing large scale genomic data on the cloud with Sparkhit. *Bioinformatics*, 34(9):1457–1465, May 2018. ISSN 1367-4803. doi: 10.1093/BIOINFORMATICS/BTX808. URL <https://academic.oup.com/bioinformatics/article/34/9/1457/4747885>. Publisher: Oxford Academic.
- [123] Runxin Guo, Yi Zhao, Quan Zou, Xiaodong Fang, and Shaoliang Peng. Bioinformatics applications on Apache Spark. *GigaScience*, August 2018. ISSN 2047-217X. doi: 10.1093/gigascience/giy098.
- [124] Jamie J. Alnasir and Hugh P. Shanahan. The application of Hadoop in structural bioinformatics. *Briefings in Bioinformatics*, 21(1):96–105, January 2020. ISSN 14774054. doi: 10.1093/BIB/BBY106. URL <https://academic.oup.com/bib/article/21/1/96/5162997>. Publisher: Oxford Academic.
- [125] Andrea Manconi, Matteo Gnocchi, Luciano Milanesi, Osvaldo Marullo, and Giuliano Armano. Framing Apache Spark in life sciences. *Heliyon*, page e13368, February 2023. ISSN 24058440. doi: 10.1016/j.heliyon.2023.e13368. URL <https://linkinghub.elsevier.com/retrieve/pii/S2405844023005753>.
- [126] Corinna Giebler, Christoph Groger, Eva Hoos, Holger Schwarz, and Bernhard Mitschang. A Zone Reference Model for Enterprise-Grade Data Lake Management. In *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC)*, pages 57–66, Eindhoven, Netherlands, October 2020. IEEE. ISBN 978-1-72816-473-1. doi: 10.1109/EDOC49727.2020.00017. URL <https://ieeexplore.ieee.org/document/9233155/>.

- [127] Ron L’Esteve. Databricks. In *The Azure Data Lakehouse Toolkit*, pages 83–139. Apress, Berkeley, CA, 2022. ISBN 978-1-4842-8232-8 978-1-4842-8233-5. doi: 10.1007/978-1-4842-8233-5_3. URL https://link.springer.com/10.1007/978-1-4842-8233-5_3.
- [128] Surajit Chaudhuri. An overview of query optimization in relational systems. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems - PODS ’98*, pages 34–43, Seattle, Washington, United States, 1998. ACM Press. ISBN 978-0-89791-996-8. doi: 10.1145/275487.275492. URL <http://portal.acm.org/citation.cfm?doid=275487.275492>.
- [129] Yang Liao, Gordon K. Smyth, and Wei Shi. featureCounts: an efficient general purpose program for assigning sequence reads to genomic features. *Bioinformatics*, 30(7):923–930, April 2014. ISSN 1367-4811, 1367-4803. doi: 10.1093/bioinformatics/btt656. URL <https://academic.oup.com/bioinformatics/article/30/7/923/232889>.
- [130] Michael Lawrence, Wolfgang Huber, Hervé Pagès, Patrick Aboyoun, Marc Carlson, Robert Gentleman, Martin T. Morgan, and Vincent J. Carey. Software for Computing and Annotating Genomic Ranges. *PLOS Computational Biology*, 9(8):e1003118, 2013. ISSN 1553-7358. doi: 10.1371/JOURNAL.PCBI.1003118. URL <https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1003118>. Publisher: Public Library of Science.
- [131] Heng Li. A statistical framework for SNP calling, mutation discovery, association mapping and population genetical parameter estimation from sequencing data. *Bioinformatics*, 27(21):2987–2993, November 2011. ISSN 1367-4811, 1367-4803. doi: 10.1093/bioinformatics/btr509. URL <https://academic.oup.com/bioinformatics/article/27/21/2987/217423>.
- [132] Ruibang Luo, Chak-Lim Wong, Yat-Sing Wong, Chi-Ian Tang, Chi-Man Liu, Chi-Ming Leung, and Tak-Wah Lam. Exploring the limit of using a deep neural network on pileup data for germline variant calling. *Nature Machine Intelligence*, 2(4):220–227, April 2020. ISSN 2522-5839. doi: 10.1038/s42256-020-0167-4. URL <https://www.nature.com/articles/s42256-020-0167-4>.

- [133] Yadong Liu, Tao Jiang, Yan Gao, Bo Liu, Tianyi Zang, and Yadong Wang. Psi-Caller: A Lightweight Short Read-Based Variant Caller With High Speed and Accuracy. *Frontiers in Cell and Developmental Biology*, 9:731424, August 2021. ISSN 2296-634X. doi: 10.3389/fcell.2021.731424. URL <https://www.frontiersin.org/articles/10.3389/fcell.2021.731424/full>.
- [134] Daniel C. Koboldt, Qunyuan Zhang, David E. Larson, Dong Shen, Michael D. McLellan, Ling Lin, Christopher A. Miller, Elaine R. Mardis, Li Ding, and Richard K. Wilson. VarScan 2: Somatic mutation and copy number alteration discovery in cancer by exome sequencing. *Genome Research*, 22(3):568–576, March 2012. ISSN 1088-9051. doi: 10.1101/gr.129684.111. URL <http://genome.cshlp.org/lookup/doi/10.1101/gr.129684.111>.
- [135] Alfred V Aho, Brian W Kernighan, and Peter J Weinberger. *The AWK programming language*. Addison-Wesley Longman Publishing Co., Inc., 1987.
- [136] Christopher Adamson. *Mastering data warehouse aggregates: solutions for star schema performance*. John Wiley & Sons, 2012.
- [137] Amit Shukla, Prasad Deshpande, Jeffrey F Naughton, and others. Materialized view selection for multidimensional datasets. In *VLDB*, volume 98, pages 488–499, 1998.
- [138] Star Schema vs. OBT for Data Warehouse Performance | Blog | Fivetran. URL <https://www.fivetran.com/blog/star-schema-vs-obt>.
- [139] Xiaoming Liu, Chunlei Wu, Chang Li, and Eric Boerwinkle. dbNSFP v3.0: A One-Stop Database of Functional Predictions and Annotations for Human Nonsynonymous and Splice-Site SNVs. *Human Mutation*, 37(3):235–241, March 2016. ISSN 10597794. doi: 10.1002/humu.22932. URL <https://onlinelibrary.wiley.com/doi/10.1002/humu.22932>.
- [140] Exome Aggregation Consortium, Monkol Lek, Konrad J. Karczewski, Eric V. Minikel, Kaitlin E. Samocha, Eric Banks, Timothy Fennell, Anne H. O'Donnell-Luria, James S. Ware, Andrew J. Hill, Beryl B. Cummings, Taru Tukiainen, Daniel P. Birnbaum, Jack A. Kosmicki, Laramie E. Duncan, Karol Estrada, Fengmei Zhao,

James Zou, Emma Pierce-Hoffman, Joanne Berghout, David N. Cooper, Nicole Deffaix, Mark DePristo, Ron Do, Jason Flannick, Menachem Fromer, Laura Gauthier, Jackie Goldstein, Namrata Gupta, Daniel Howrigan, Adam Kiezun, Mitja I. Kurki, Ami Levy Moonshine, Pradeep Natarajan, Lorena Orozco, Gina M. Peloso, Ryan Poplin, Manuel A. Rivas, Valentin Ruano-Rubio, Samuel A. Rose, Douglas M. Ruderfer, Khalid Shakir, Peter D. Stenson, Christine Stevens, Brett P. Thomas, Grace Tiao, Maria T. Tusie-Luna, Ben Weisburd, Hong-Hee Won, Dongmei Yu, David M. Altshuler, Diego Ardiissino, Michael Boehnke, John Danesh, Stacey Donnelly, Roberto Elosua, Jose C. Florez, Stacey B. Gabriel, Gad Getz, Stephen J. Glatt, Christina M. Hultman, Sekar Kathiresan, Markku Laakso, Steven McCarroll, Mark I. McCarthy, Dermot McGovern, Ruth McPherson, Benjamin M. Neale, Aarno Palotie, Shaun M. Purcell, Danish Saleheen, Jeremiah M. Scharf, Pamela Sklar, Patrick F. Sullivan, Jaakko Tuomilehto, Ming T. Tsuang, Hugh C. Watkins, James G. Wilson, Mark J. Daly, and Daniel G. MacArthur. Analysis of protein-coding genetic variation in 60,706 humans. *Nature*, 536(7616):285–291, August 2016. ISSN 0028-0836, 1476-4687. doi: 10.1038/nature19057. URL <http://www.nature.com/articles/nature19057>.

- [141] Raghav Sethi, Martin Traverso, Dain Sundstrom, David Phillips, Wenlei Xie, Yutian Sun, Nezih Yeginbasi, Haozhun Jin, Eric Hwang, Nileema Shingte, and Christopher Berner. Presto: SQL on Everything. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1802–1813. IEEE, April 2019. ISBN 978-1-5386-7474-1. doi: 10.1109/ICDE.2019.00196. URL <https://ieeexplore.ieee.org/document/8731547/>.

- [142] MKABV Bittorf, Taras Bobrovitsky, CCACJ Erickson, Martin Grund Daniel Hecht, MJIJL Kuff, Dileep Kumar Alex Leblang, NLIPH Robinson, David Rorke Silvius Rus, JRDTS Wanderman, and Milne Michael Yoder. Impala: A modern, open-source sql engine for hadoop. In *Proceedings of the 7th biennial conference on innovative data systems research*, 2015.

- [143] Peter A. Boncz, Martin L. Kersten, and Stefan Manegold. Breaking the memory wall in MonetDB. *Communications of the ACM*, 51(12):77–85, December 2008. ISSN 0001-0782, 1557-7317. doi: 10.1145/1409360.1409380. URL <https://dl.acm.org/doi/10.1145/1409360.1409380>.

- [144] Roberto Tardío, Alejandro Maté, and Juan Trujillo. A New Big Data Benchmark for OLAP Cube Design Using Data Pre-Aggregation Techniques. *Applied Sciences*, 10(23):8674, December 2020. ISSN 2076-3417. doi: 10.3390/app10238674. URL <https://www.mdpi.com/2076-3417/10/23/8674>.
- [145] Joelle Mbatchou, Leland Barnard, Joshua Backman, Anthony Marcketta, Jack A. Kosmicki, Andrey Ziyatdinov, Christian Benner, Colm O'Dushlaine, Mathew Barber, Boris Boutkov, Lukas Habegger, Manuel Ferreira, Aris Baras, Jeffrey Reid, Goncalo Abecasis, Evan Maxwell, and Jonathan Marchini. Computationally efficient whole-genome regression for quantitative and binary traits. *Nature Genetics*, 53(7):1097–1103, July 2021. ISSN 1061-4036, 1546-1718. doi: 10.1038/s41588-021-00870-7. URL <http://www.nature.com/articles/s41588-021-00870-7>.
- [146] Jianglin Feng, Aakrosh Ratan, and Nathan C. Sheffield. Augmented Interval List: A novel data structure for efficient genomic interval search. *Bioinformatics*, 35(23):4907–4911, December 2019. ISSN 14602059. doi: 10.1093/BIOINFORMATICS/BTZ407. Publisher: Oxford University Press.
- [147] Thomas H. Cormen, Charles Eric Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. The MIT Press, Cambridge, Massachusett, fourth edition edition, 2022. ISBN 978-0-262-36750-9. OCLC: 1305060400.
- [148] Alexander V. Alekseyenko and Christopher J. Lee. Nested Containment List (NCLList): A new algorithm for accelerating interval query of genome alignment and interval databases. *Bioinformatics*, 23(11):1386–1393, June 2007. ISSN 13674803. doi: 10.1093/BIOINFORMATICS/BTL647.
- [149] Frank Austin Nothaft. *Scalable systems and algorithms for genomic variant analysis*. PhD thesis, UC Berkeley, 2017.
- [150] Mike Lin. mwiewior/iitii: 1.0.0, February 2023. URL <https://zenodo.org/record/7674122>.
- [151] Aaron R. Quinlan and Ira M. Hall. BEDTools: a flexible suite of utilities for comparing genomic features. *Bioinformatics*, 26(6):841–842, March 2010. ISSN 1367-4811, 1367-4803. doi: 10.1093/bioinformatics/btq033. URL <https://academic.oup.com/bioinformatics/article/26/6/841/244688>.

- [152] Brent S. Pedersen and Aaron R. Quinlan. Mosdepth: quick coverage calculation for genomes and exomes. *Bioinformatics*, 34(5):867–868, March 2018. ISSN 1367-4803. doi: 10.1093/BIOINFORMATICS/BTX699. URL <https://academic.oup.com/bioinformatics/article/34/5/867/4583630>. Publisher: Oxford Academic.
- [153] Michele Guerriero, Martin Garriga, Damian A. Tamburri, and Fabio Palomba. Adoption, Support, and Challenges of Infrastructure-as-Code: Insights from Industry. In *2019 IEEE International Conference on Software Maintenance and Evolution (IC-SME)*, pages 580–589. IEEE, September 2019. ISBN 978-1-72813-094-1. doi: 10.1109/ICSME.2019.00092. URL <https://ieeexplore.ieee.org/document/8919181/>.
- [154] Roshan N. Rajapakse, Mansooreh Zahedi, M. Ali Babar, and Haifeng Shen. Challenges and solutions when adopting DevSecOps: A systematic review. *Information and Software Technology*, 141:106700, January 2022. ISSN 09505849. doi: 10.1016/j.infsof.2021.106700. URL <https://linkinghub.elsevier.com/retrieve/pii/S0950584921001543>.
- [155] Sofia Reis, Rui Abreu, Marcelo d’Amorim, and Daniel Fortunato. Leveraging Practitioners’ Feedback to Improve a Security Linter. In *37th IEEE/ACM International Conference on Automated Software Engineering*, pages 1–12, Rochester MI USA, October 2022. ACM. ISBN 978-1-4503-9475-8. doi: 10.1145/3551349.3560419. URL <https://dl.acm.org/doi/10.1145/3551349.3560419>.
- [156] Jon Bryant. Driving into the Cloud: What is Finops? *ITNOW*, 64(3):54–55, August 2022. ISSN 1746-5702, 1746-5710. doi: 10.1093/combul/bwac097. URL <https://academic.oup.com/itnow/article/64/3/54/6672552>.
- [157] JR Storment and Mike Fuller. *Cloud FinOps*. " O'Reilly Media, Inc.", 2023.
- [158] Yevgeniy Brikman. *Terraform: Up and Running*. " O'Reilly Media, Inc.", 2022.
- [159] BANK.pl – Portal finansowy | Polskie politechniki z grantami Microsoft Azure - BANK.pl - Portal finansowy, June 2014. URL <https://bank.pl/polscie-politechniki-z-grantami-microsoft-azure/?id=44547&catid=358>.
Section: Finanse i gospodarka.

- [160] Marek S. Wiewiora, Alicja Szabelska, and Michal J. Okoniewski. Analysis of AmpliSeq RNA-Sequencing Enrichment Panels. In Marzena Kryszkiewicz, Sanghamitra Bandyopadhyay, Henryk Rybinski, and Sankar K. Pal, editors, *Pattern Recognition and Machine Intelligence*, volume 9124, pages 495–500. Springer International Publishing, Cham, 2015. ISBN 978-3-319-19940-5 978-3-319-19941-2. doi: 10.1007/978-3-319-19941-2_47. URL http://link.springer.com/10.1007/978-3-319-19941-2_47. Series Title: Lecture Notes in Computer Science.
- [161] Monika Szczerba, Marek S. Wiewiórka, Michał J. Okoniewski, and Henryk Rybiński. Scalable Cloud-Based Data Analysis Software Systems for Big Data from Next Generation Sequencing. In Nathalie Japkowicz and Jerzy Stefanowski, editors, *Big Data Analysis: New Algorithms for a New Society*, volume 16, pages 263–283. Springer International Publishing, Cham, 2016. ISBN 978-3-319-26987-0 978-3-319-26989-4. doi: 10.1007/978-3-319-26989-4_11. URL http://link.springer.com/10.1007/978-3-319-26989-4_11. Series Title: Studies in Big Data.
- [162] Michał Okoniewski, Rafał Płoski, Marek Wiewiora, and Urszula Demkow. Future Directions. In *Clinical Applications for Next-Generation Sequencing*, pages 281–294. Elsevier, 2016. ISBN 978-0-12-801739-5. doi: 10.1016/B978-0-12-801739-5.00015-5. URL <https://linkinghub.elsevier.com/retrieve/pii/B9780128017395000155>.
- [163] Znamy wyniki I edycji konkursu Best Paper / 2020 / Wyniki konkursów / Konkursy / Strona główna - Uczelnia Badawcza - Politechnika Warszawska. URL <https://badawcza.pw.edu.pl/Konkursy/Wyniki-konkursow/2020/Znamy-wyniki-I-edycji-konkursu-Best-Paper>.
- [164] Maria Xekalaki, Juan Fumero, Athanasios Stratikopoulos, Katerina Doka, Christos Katsakioris, Constantinos Bitsakos, Nectarios Koziris, and Christos Kotselidis. Enabling Transparent Acceleration of Big Data Frameworks Using Heterogeneous Hardware. *Proceedings of the VLDB Endowment*, 15(13):3869–3882, September 2022. ISSN 2150-8097. doi: 10.14778/3565838.3565842. URL <https://dl.acm.org/doi/10.14778/3565838.3565842>.
- [165] Dalton Lunga, Jonathan Gerrand, Lexie Yang, Christopher Layton, and Robert Stewart. Apache Spark Accelerated Deep Learning Inference for Large Scale Satellite

Image Analytics. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 13:271–283, 2020. ISSN 1939-1404, 2151-1535. doi: 10.1109/JSTARS.2019.2959707. URL <https://ieeexplore.ieee.org/document/8949817/>.

- [166] Samyuktha Muralidharan, Savita Yadav, Jungwoo Huh, Sanghoon Lee, and Jongwook Woo. Scalable Prediction Models for Airbnb Listing in Spark Big Data Cluster using GPU-accelerated RAPIDS. *Journal of Information and Communication Convergence Engineering*, 20(2):96–102, June 2022. doi: 10.6109/JICCE.2022.20.2.96. URL <https://doi.org/10.6109/JICCE.2022.20.2.96>.
- [167] Kerina Jones, Helen Daniels, Sharon Heys, Arron Lacey, and David V Ford. Toward a Risk-Utility Data Governance Framework for Research Using Genomic and Phenotypic Data in Safe Havens: Multifaceted Review. *Journal of Medical Internet Research*, 22(5):e16346, May 2020. ISSN 1438-8871. doi: 10.2196/16346. URL <https://www.jmir.org/2020/5/e16346>.

CHAPTER 6

Copies of the publications constituting the Dissertation

- [P1] M.S. Wiewiórka, A. Messina, A. Pacholewska, S. Maffioletti, P. Gawrysiak, and M.J. Okoniewski. SparkSeq: Fast, scalable and cloud-ready tool for the interactive genomic data analysis with nucleotide precision. *Bioinformatics*, 30(18), 2014, **Page: 79**
- [P2] M.S. Wiewiórka, D.P. Wysakowicz, M.J. Okoniewski, and T. Gambin. Benchmarking distributed data warehouse solutions for storing genomic variant information. *Database : the journal of biological databases and curation*, 2017, **Page: 109**
- [P3] Anastasiia Hryhorzhevska, Marek Wiewiórka, Michał Okoniewski, and Tomasz Gambin. Scalable Framework for the Analysis of Population Structure Using the Next Generation Sequencing Data. In *Lecture Notes in Computer Science (including sub-series Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 10352 LNAI, pages 471–480. 2017, **Page: 125**
- [P4] Marek Wiewiórka, Anna Leśniewska, Agnieszka Szmurło, Kacper Stępień, Mateusz Borowiak, Michał Okoniewski, and Tomasz Gambin. SeQuiLa: an elastic, fast and scalable SQL-oriented solution for processing and querying genomic intervals. *Bioinformatics*, 35(12):2156–2158, June 2019, **Page: 135**
- [P5] Marek Wiewiórka, Agnieszka Szmurło, Wiktor Kuśmirek, and Tomasz Gambin.

SeQuiLa-cov: A fast and scalable library for depth of coverage calculations. *Giga-Science*, 8(8), August 2019, **Page:** [138](#)

- [P6] Marek Wiewiórka, Agnieszka Szmurło, Paweł Stankiewicz, and Tomasz Gambin. Cloud-native distributed genomic pileup operations. *Bioinformatics*, December 2022, **Page:** [145](#)

Genome analysis

Advance Access publication May 19, 2014

SparkSeq: fast, scalable and cloud-ready tool for the interactive genomic data analysis with nucleotide precisionMarek S. Wiewiórka^{1,*}, Antonio Messina², Alicja Pacholewska^{3,4}, Sergio Maffioletti², Piotr Gawrysiak¹ and Michał J. Okoniewski⁵¹Institute of Computer Science, Warsaw University of Technology, Warsaw, Poland, ICS 00-665 Warsaw (MW, PG),²Grid Computing Competence Center-GC3, University of Zurich, 8057 Zürich (SM, AM), ³Swiss Institute of Equine Medicine, Vetsuisse Faculty, University of Bern and ALP-Haras, 3001 Bern (AP), ⁴Institute of Genetics, Vetsuisse Faculty, University of Bern, Bern, 3001 Bern (AP) and ⁵Functional Genomics Center Zurich, CH-8057 Zurich, Switzerland

Associate Editor: Inanc Birol

Downloaded from https://academic.oup.com/bioinformatics/article/30/18/2652/2475607 by guest on 01 May 2021

ABSTRACT

Summary: Many time-consuming analyses of next-generation sequencing data can be addressed with modern cloud computing. The Apache Hadoop-based solutions have become popular in genomics **because of** their scalability in a cloud infrastructure. So far, most of these tools have been used for batch data processing rather than interactive data querying.

The SparkSeq software has been created to take advantage of a new MapReduce framework, Apache Spark, for next-generation sequencing data. SparkSeq is a general-purpose, flexible and easily extendable library for genomic cloud computing. It can be used to build genomic analysis pipelines in Scala and run them in an interactive way. SparkSeq opens up the possibility of customized *ad hoc* secondary analyses and iterative machine learning algorithms. This article demonstrates its scalability and overall fast performance by running the analyses of sequencing datasets. Tests of SparkSeq also prove that the use of cache and HDFS block size can be tuned for the optimal performance on multiple worker nodes.

Availability and implementation: Available under open source Apache 2.0 license: <https://bitbucket.org/mwiewiorka/sparkseq/>.

Contact: marek.wiewiorka@gmail.com

Supplementary information: Supplementary data are available at *Bioinformatics* online.

Received on December 17, 2013; revised on April 18, 2014; accepted on May 12, 2014

In recent years, next-generation sequencing (NGS) has become the essential technology that is producing precise insight into the genomes and transcriptomes of living organisms. A standard sequencer run produces hundreds gigabase pairs of short reads. The experimental results (fastq or alignment files) can be regarded as big data, in particular with the high number of biological samples. This can be less overwhelming when lab techniques of preselection are applied or when the data are aggregated. Computationally, the reduction of the datasets is done by counting the reads in genes or finding genome variants. The resulting count tables or variant lists are then much smaller than the original BAM files and in most cases can be processed in

memory. However, aggregation and summarization always result in the loss of information on such phenomena as novel expression regions, alternative splicing or low coverage areas. Sequencing can pinpoint various properties of genomes and transcriptomes (Frazee *et al.*, 2014; Lesniewska and Okoniewski, 2011) with a nucleotide precision, but often the size of datasets is prohibitive to study such details. The nucleotide-level analysis can be a driving force for the many new and additional applications of sequencing such as linc-RNA discovery, verification of gene and UTR boundaries in the species with imprecise annotation or studies of alternative splicing.

The developments in computer science, especially in the area of distributed and cloud computing, are trying to keep pace with the rapidly growing amount of experimental data. There are several applications already available (Langmead *et al.*, 2010; Schumacher *et al.*, 2014; Taylor, 2010). Currently there are many initiatives within the IT and bioinformatics community that can be used for creating a useful and scalable system for sequencing data analysis. One of the most frequently used parallel and distributed programming models is MapReduce, which has its open-source implementation—Apache Hadoop (Borthakur, 2007). As initially it was not possible to analyze NGS data stored in Hadoop Distributed File System (HDFS) without conversion to file formats supported by Apache Hadoop, a library Hadoop-BAM was developed for direct access and manipulation of formats commonly used in bioinformatics: Binary Alignment/Map format (BAM), FASTQ and Variant Call format (VCF) (Niemenmaa *et al.*, 2012).

Many analytical tools that process data within the Hadoop ecosystem have been developed recently in the open-source community: new high-level languages, database engines and application frameworks. They use the same Apache Hadoop execution model, primarily designed and optimized for one-pass batch processing of on-disk data but not for interactive and in-memory *ad hoc* data exploration. A good example of the successful combination of Hadoop-BAM and the Apache Pig language is SeqPig, an extension for processing of sequencing data (Schumacher *et al.*, 2014) similar to standard samtools (Li *et al.*, 2009). However, using high-level dataflow languages such as Apache Pig has one drawback: in many cases, data processing with them is far from optimal, as it is harder to do fine-grained code optimizations.

*To whom correspondence should be addressed.

A completely new MapReduce paradigm implementation, Apache Spark (Zaharia *et al.*, 2012), addresses both of those problems. It introduces a new Application Programming Interface (API), suitable for both high-level data processing workflow definition as well as low-level code tuning, using the concise Scala language. Apache Spark includes also a new storage primitive: resilient distributed datasets. It lets the users cache precomputed data in memory and/or disk across queries. Apache Spark has also much lower launching overheads, which is important for running *ad hoc* queries. Another advantage of Apache Spark is its REPL (read-eval-print-loop) environment: Spark-shell.

To study the utility of Apache Spark in the genomic context, we created SparkSeq. SparkSeq is a general-purpose tool for RNA and DNA sequencing analyses, tuned for processing in the cloud big alignment data with nucleotide precision. It combines Picard Java Development Kit for Sequence Alignment/Map format (SAM) (Picard SAM JDK) via Hadoop-BAM library and Apache Spark to introduce versatile sequencing analyses in MapReduce environment. It currently implements operations on many alignment files at a time, falling into three categories: filtering of reads, summarizing genomic features and statistical analyses. It can be run directly with Scala, or in R using the RSparkSeq package. The scalability study done with the prototype has proven that by using big enough computational infrastructure, it is possible to handle the constantly growing number of alignment files with base-pair resolution in a manageable time. Much shorter processing time due to scalability can lead to more interactive analysis with a deeper biological insight for the genomics researchers. The computing infrastructure for the tests included cluster nodes equipped with 8-core CPU and 16 GB RAM running Linux Ubuntu 12.04 virtual machines with Apache Hadoop 1.2.1, Apache Spark 0.8.0 and Scala 2.9.3 installed. Scalability and performance has been evaluated by increasing the number of worker nodes from 1 to 10 (8 to 80 cores in total) and comparing this with samtools and SeqPig on the coverage/pileup function of a multi-sample dataset. In separate tests, such parameters as HDFS block size and the use of caching have been examined to find the optimal processing time. Implemented analysis pipelines include the calculation of read coverage for all the nucleotides in the reference genome and the counts of reads within all exons, that can be processed them by the standard statistical methods for RNA sequencing (Anders *et al.*, 2013). Testing of SparkSeq has been done mainly using two datasets: one 30 GB BAM file from multiple-amplicon sequencing experiment and 32 BAM files (~40 GB) from a whole-transcript RNA-sequencing experiment.

The results presented in Figure 1 show that SparkSeq outperforms SeqPig in terms of speed (8.4–9.15 times) and that finding the coverage for all the nucleotides scales up well. The cache experiments described in detail in the Supplementary File show that using a fast data serializer (like KryoSerializer) instead of the Java built-in one together with LZF or Snappy compression can speed up multi-pass data querying up to 110 times and reduce memory consumption by a factor of 13. HDFS block size adjustment can also result in a performance boost.

The tests done on SparkSeq clearly prove the utility of Apache Spark/Hadoop-BAM-based solution for high-performance analysis of sequencing alignment files. The efficiency is comparable

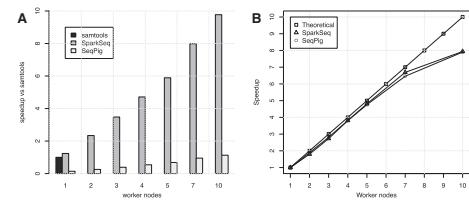


Fig. 1. Speedup (A) and scalability (B) of SparkSeq and SeqPig on 8 BAM files RNA-sequencing dataset (total 9 GB) and a different number of worker nodes. The benchmark consists of the filtering out duplicates and reads with low quality or gaps in CIGAR string, then calculating the coverage/pileup for all the samples. samtools used as a baseline

with standards (samtools) on a standalone machine and it offers the added value of horizontal scalability. Similar implementations can be done for other types of analyses and other type of genomic alignment files. The benefit for biology researchers will be the speeding up of the long analyses and getting even the nucleotide precision of multi-sample results in non-prohibitive time. This will enable unsupervised searches for novel genomics phenomena in many samples in parallel, as well as optimizing standard analyses by running them many times, tuning their parameters in an interactive way.

ACKNOWLEDGEMENTS

The authors thank Christian Panse, Riccardo Murri, Tyanko Aleksiev and Martin Ryan for the valuable discussions.

Funding: Scientific Exchange Programme NMS-CH (grant no. 12.289) and in part by the Swiss National Science Foundation (grant 310000-116502).

Conflicts of Interest: none declared.

REFERENCES

- Anders,S. *et al.* (2013) Count-based differential expression analysis of RNA sequencing data using R and Bioconductor. *Nat. Protoc.*, **8**, 1765–1786.
- Borthakur,D. (2007) *The Hadoop Distributed File System: Architecture and Design*. Hadoop Project Website 11 (2007): 21.
- Frazee,A.C. *et al.* (2014) Differential expression analysis of RNA-seq data at single-base resolution. *Biostatistics*, **15**, 413–426.
- Langmead,B. *et al.* (2010) Cloud-scale RNA-sequencing differential expression analysis with Myrna. *Genome Biol.*, **11**, R83.
- Leśniewska,A. and Okoniewski,M.J. (2011) rnaSeqMap: a Bioconductor package for RNA sequencing data exploration. *BMC Bioinformatics*, **12**, 200.
- Li,H. *et al.* (2009) The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, **25**, 2078–2079.
- Niemennmaa,M. *et al.* (2012) Hadoop-BAM: directly manipulating next generation sequencing data in the cloud. *Bioinformatics*, **28**, 876–877.
- Schumacher,A. *et al.* (2014) Seqpig: simple and scalable scripting for large sequencing data sets in hadoop. *Bioinformatics*, **30**, 119–120.
- Taylor,R.C. (2010) An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics. *BMC Bioinformatics*, **11** (Suppl. 12), S1.
- Zaharia,M. *et al.* (2012) Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: *Proceedings of the 9th USENIX Conference*. San Jose.

SparkSeq - a fast, scalable, cloud-ready tool for interactive genomics data analysis with nucleotide precision

Supplementary material

April 18, 2014

Contents

1 Performance comparison of samtools, SeqPig and SparkSeq	2
1.1 Tests description	2
1.2 Software versions	3
1.3 Hardware infrastructure	3
1.4 Apache Hadoop, Pig and Spark parameters	4
1.5 Code listings test 1	4
1.5.1 samtools	4
1.5.2 SparkSeq	5
1.5.3 SeqPig	5
1.6 Test 1 results	5
1.7 Code listings test 2	5
1.7.1 samtools	5
1.7.2 SparkSeq	6
1.7.3 SeqPig	6
1.8 Test 2 results	6
1.9 Code listings test 3	6
1.9.1 samtools	6
1.9.2 SparkSeq	7
1.9.3 SeqPig	7
1.10 Test 3 results	7
1.11 Code listings test 4	7
1.11.1 samtools	7
1.11.2 SparkSeq	7
1.11.3 SeqPig	7
1.12 Test 4 results	8
1.13 Tests results discussion	8
2 HDFS block size evaluation	9
3 SparkSeq scalability	9
4 SparkSeq cache strategies evaluation	9
5 Comparison of samtools, SeqPig and SparkSeq features	10
6 Getting started with SparkSeq	12
6.1 Dependencies	12
6.2 Installation	12
6.3 Basics	12
6.4 Project documentation	13

7 Multisample analyses tutorial	13
7.1 Creating SparkSeqAnalysis object	14
7.2 Counting and displaying	14
7.3 Sorting	15
7.4 Filtering	16
7.4.1 Filter and undo	16
7.4.2 Quality filtering	16
7.4.2.1 Using base-pair qualities	16
7.4.2.2 Using the alignment quality	16
7.4.3 Filtering with the alignment and reference names	16
7.4.4 Filtering on read name and length	17
7.4.5 Filtering on CIGAR Strings	17
7.4.6 Filtering using the alignment (SAM) flags	17
7.4.7 Combining filters	17
7.4.8 Generic filter	18
7.4.9 Selecting samples	18
7.5 Computing the coverage function and counts of reads in genomic regions	18
7.5.1 Nucleotide-level coverage function	18
7.5.2 Exon-level counts	19
7.5.3 Gene-level counts	19
7.6 Grouping	19
7.7 Joining multiple SparkSeqAnalysis objects	20
7.7.1 Full outer join	20
7.7.2 Inner join	21
7.8 Junction reads analysis	21
7.9 Using cache	22
7.10 Saving results	23
7.11 Setting up complete data processing pipelines	23
7.12 Optimization hints	24
8 RSparkSeq	25
8.1 Introduction	25
8.2 Installation instructions	25
8.3 Getting started	26
8.3.1 Establishing connection to SparkSeq	26
8.3.2 Creating RSparkSeqAnalysis object	26
8.3.3 Computing feature counts	26
8.3.3.1 Genes	26
8.3.3.2 Exons	26
A Data sets used in tests	27
A.1 RNA-sequencing experiment	27
A.2 Multi-amplicon experiment	27
B Additional literature	28

1 Performance comparison of samtools, SeqPig and SparkSeq

1.1 Tests description

In order to compare performance of SeqPig and SparkSeq with samtools 4 test were conducted on 2 large BAM files (12GB and 30GB) and one dataset containing 8 samples (9GB in total, BAM files from `./Fam1/Case` directory A.1):

- Test 1 - filter out reads using 4 conditions, compute base coverage for 8 samples at once and construct „coverage vector“ for each position, e.g. for chr1:67086 it would be the following vector: (6, 13, 3, 6, 8, 8, 8, 8) where each value corresponds to base coverage in a sample for a given genome position. Test 1 was the most compound case (combination of test 2 and 4 – base coverage computation and filtering reads) that can be considered as an indicator of how well the tools perform in the close to real cases of multiple BAM files queries (Figure 1). Sample output of all 4 methods:

- Samtools: `chr1:67086 8 8 6 13 3 8 8 6`

- SeqPig: chr1 67086 (8),(6),(8),(6),(3),(13),(8),(8)
- SparkSeq: ((chr1,67086), ArrayBuffer(6, 13, 3, 6, 8, 8, 8, 8))

SeqPig's ReadPileup and BinReadPileup implementations are not yet tested enough and it was not possible to run them using whole datasets since the following errors were reported:

- ReadPileup Caught error from UDF: fi.alto.seqpig.pileup.ReadPileup [BUG or bad data?? Could not find matching MD deletion after finding CIGAR deletion!]
- BinReadPileup ERROR org.apache.pig.tools.pigstats.SimplePigStats - ERROR: Unexpected data type [I found in stream. Note only standard Pig type is supported when you output from UDF/LoadFunc]

Besides, according to the preliminary tests coverage computations done using SeqPig ReadRefPositions was approximately 10% faster than analogous ones using ReadPileup. The only drawback of ReadRefPositions implementation is that it does not support correct CIGAR strings parsing and this is why it was decided to run the tests using only reads that were aligned to only one region without any INDELS or gaps (i.e. CIGAR == '101M').

- Test 2 - compute base coverage for all genome coordinates (Figure 2);
- Test 3 - calculate the total number of short reads in a BAM file without any filtering (Figure 3);
- Test 4 - calculate the number of short reads in a BAM file that were mapped, were not duplicates and have mapping quality greater than 19 (Figure 4);

In the tests 2-4 performance and speedup were compared to the computation time of samtools (one node, single thread) which was the baseline. In the case of 1st test samtools processing pipeline consisted of 3 samtools processes, 2 awk processes and 1 lzop processes executing in parallel - this could be compared to running an 6-thread application.

1.2 Software versions

- Operating system: Linux Ubuntu 12.04 LTS
- JDK: OpenJDK 64-Bit 1.6.0_27
- Scala: 2.9.3
- Apache Hadoop: 1.2.1
- Apache Pig: 0.11.1
- SeqPig: 0.5
- Hadoop-BAM: 0.6
- Apache Spark: 0.8.0
- samtools: 0.1.19 (single-threaded)
- Elasticcluster 1.0.3 (a user-friendly command line tool to create, manage and setup computing clusters hosted on cloud infrastructures, <http://gc3-uzh-ch.github.io/elasticcluster/>)
- hadoop-lzo 0.4.20-SNAPSHOT <https://github.com/twitter/hadoop-lzo> (in the 1st test case)

1.3 Hardware infrastructure

Tests were run on IaaS cloud provided by Grid Computing Competence Center at University of Zurich (<http://www.gc3.uzh.ch/>). Each of the instances used in the test was of m1.xlarge flavour (8CPUs and 16GB RAM). The HDFS file system used the node-local disk space. Compute nodes were equipped with a 1Gbit/s Ethernet interfaces, and the network backbone run at 10Gbit/s.

1.4 Apache Hadoop, Pig and Spark parameters

In all the tests we set the following Apache Hadoop parameters:

- dfs.block.size = 64MB (default)
- mapred.tasktracker.map.tasks.maximum = 8
- mapred.tasktracker.reduce.tasks.maximum = 4
- mapred.child.java.opts = -Xmx1536m

In all the tests we set the following Apache Spark parameters:

- spark.executor.memory = 10g
- spark.rdd.compress = true
- mapred.spark.serializer = org.apache.spark.serializer.KryoSerializer¹

In all the tests we set the following Apache Pig parameters:

- pig.exec.mapPartAgg = true;
- pig.exec.mapPartAgg.minReduction = 5;
- pig.schematuple = on;

In the test 4 we have turned on lzo compression for intermediate job results in Pig which due LZO's licence (GPL) is incompatible with that of Hadoop (Apache) and therefore it cannot be bundled with it. It was built from source separately on the cluster and the following parameters was additionally set:

- pig.tmpfilecompression true
- pig.tmpfilecompression.codec lzo

Also the samtools/Unix scripts used lzo compression to store intermediate results between computing coverage separately for each sample and joining the results. It was the test in which SeqPig could benefit the most from the compression of the intermediate results as the processing consisted of 2 MapReduce jobs. After running a few tests it was decided not to set parameters turning on data compression between the mapper and the reducer (which is suggested SeqPig Authors), since using gzip codec slowed down data processing by approximately 15%:

¹ SET mapred.compress.map.output true
2 SET mapred.output.compression.codec org.apache.hadoop.io.compress.GzipCodec

1.5 Code listings test 1

1.5.1 samtools

```
1 time for f in *.bam ; do samtools view -q 19 -F 260 -h $f | awk '($6 ~/101M/) || ($1 ~ /^@/)' |  
2 samtools view -bS - | samtools mpileup - | awk '{print $1":"$2"\t"$4}' |lzop > ${f%.}lzo ;done  
3  
4 time join -i -a 1 -a 2 <(lzop -d < Sample_25_sort.lzo |sort -f) <(lzop -d < Sample_38_sort.lzo|sort -f) |  
5 join -i -a 1 -a 2 -<(lzop -d < Sample_39_sort.lzo|sort -f) |  
6 join -i -a 1 -a 2 -<(lzop -d < Sample_42_sort.lzo|sort -f) |  
7 join -i -a 1 -a 2 -<(lzop -d < Sample_44_sort.lzo|sort -f) |  
8 join -i -a 1 -a 2 -<(lzop -d < Sample_45_sort.lzo|sort -f) |  
9 join -i -a 1 -a 2 -<(lzop -d < Sample_47_sort.lzo|sort -f) |  
10 join -i -a 1 -a 2 -<(lzop -d < Sample_53_sort.lzo|sort -f) >cov_vectors.txt
```

¹Kryo is a fast and efficient object graph serialization framework for Java (<https://github.com/EsotericSoftware/kryo>)

1.5.2 SparkSeq

```
1 import pl.elka.pw.sparkseq.seqAnalysis.SparkSeqAnalysis
2 import pl.elka.pw.sparkseq.conversions.SparkSeqConversions
3 import pl.elka.pw.sparkseq.util.SparkSeqRegType
4 import org.apache.spark.storage.StorageLevel
5
6 val rootPath = "hdfs://hadoop-name001:9000/RA0/"
7 val fileSplitSize = 64
8 val pathFam1 = rootPath+fileSplitSize.toString+"MB/condition_9/Fam1"
9 val caseIdFam1 = Array(38,39,42,44,45,47,53)
10 val testSuff = "_sort.bam"
11 val seqAnalysisCase = new SparkSeqAnalysis(sc, pathFam1 + "/Case/Sample_25" + testSuff, 25, 1, 8)
12 for (i < caseIdFam1) {
13   val path = pathFam1 + "/Case/Sample_" + i.toString + testSuff
14   seqAnalysisCase.addBAM(sc, path, i, 1)
15 }
16 seqAnalysisCase.filterDuplicateReadFlag(_ == false)
17 seqAnalysisCase.filterUnmappedFlag(_ == false)
18 seqAnalysisCase.filterMappingQuality(_ > 19)
19 seqAnalysisCase.filterCigarString(_ == "101M")
20 val baseCov = seqAnalysisCase.getCoverageBase()
21 val groupBaseCov = baseCov.map(r=>( SparkSeqConversions.stripSampleID(r._1),r._2))
22 .groupByKey()
23 .map(r=>((SparkSeqConversions.idToCoordinates(r._1),r._2)))
24 groupBaseCov.saveAsTextFile("hdfs://hadoop-name001:9000/seqpig/sparkseq2.txt")
```

1.5.3 SeqPig

```
1 SET default_parallel $parallel
2 SET pig.exec.mapPartAgg true;
3 SET pig.exec.mapPartAgg.minReduction 5;
4 SET pig.schematuple on;
5 SET pig.tmpfilecompression true
6 SET pig.tmpfilecompression.codec lzo
7
8 DEFINE ReadUnmapped SAMFlagsFilter('HasSegmentUnmapped');
9 DEFINE IsDuplicate fi.alto.seqpig.filter.SAMFlagsFilter('IsDuplicate');
10
11 S25 = load 'hdfs://hadoop-name001:9000/RA0/64MB/condition_9/Fam1/Case/Sample_25_sort.bam' using BamLoader();
12 S25 = foreach S25 generate 25 as id,*;
13 S38 = load 'hdfs://hadoop-name001:9000/RA0/64MB/condition_9/Fam1/Case/Sample_38_sort.bam' using BamLoader();
14 S38 = foreach S38 generate 38 as id,*;
15 S39 = load 'hdfs://hadoop-name001:9000/RA0/64MB/condition_9/Fam1/Case/Sample_39_sort.bam' using BamLoader();
16 S39 = foreach S39 generate 39 as id,*;
17 S42 = load 'hdfs://hadoop-name001:9000/RA0/64MB/condition_9/Fam1/Case/Sample_42_sort.bam' using BamLoader();
18 S42 = foreach S42 generate 42 as id,*;
19 S44 = load 'hdfs://hadoop-name001:9000/RA0/64MB/condition_9/Fam1/Case/Sample_44_sort.bam' using BamLoader();
20 S44 = foreach S44 generate 44 as id,*;
21 S45 = load 'hdfs://hadoop-name001:9000/RA0/64MB/condition_9/Fam1/Case/Sample_45_sort.bam' using BamLoader();
22 S45 = foreach S45 generate 45 as id,*;
23 S47 = load 'hdfs://hadoop-name001:9000/RA0/64MB/condition_9/Fam1/Case/Sample_47_sort.bam' using BamLoader();
24 S47 = foreach S47 generate 47 as id,*;
25 S53 = load 'hdfs://hadoop-name001:9000/RA0/64MB/condition_9/Fam1/Case/Sample_53_sort.bam' using BamLoader();
26 S53 = foreach S53 generate 53 as id,*;
27 SCASEUN= UNION S25, S38, S39, S42, S44, S45, S47, S53;
28 SCASEFILTER = FILTER SCASEUN BY not ReadUnmapped(flags) AND not IsDuplicate(flags) AND mapqual > 19 AND cigar =='101M';
29 SCASE = FOREACH SCASEFILTER GENERATE id, read, flags, refname, start, cigar, basequal, mapqual;
30 REFPOS = FOREACH SCASE GENERATE id,ReadRefPositions(read, flags, refname, start,
31   cigar, basequal);
32 flatset = FOREACH REFPOS GENERATE $0,flatten($1);
33 grouped = GROUP flatset BY ($0, $1, $2);
34 base_counts = FOREACH grouped GENERATE group.id, group.chr, group.pos, COUNT(flatset) as cov;
35 grouped2 = GROUP base_counts BY ($1,$2);
36 base_counts2 = FOREACH grouped2 GENERATE group.chr, group.pos, base_counts.cov;
37 store base_counts2 into 'hdfs://hadoop-name001:9000/seqpig/input.basecounts44_21_compr_allgz';
```

1.6 Test 1 results

1.7 Code listings test 2

1.7.1 samtools

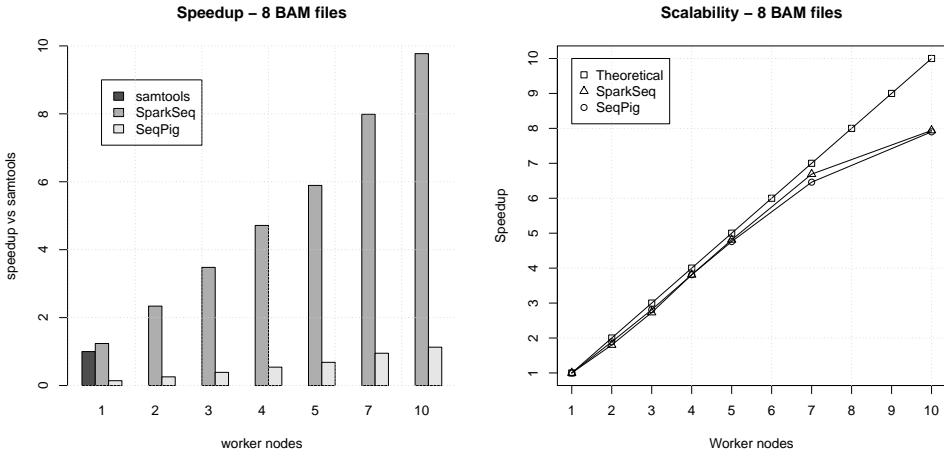


Figure 1: Filtering out reads using 4 conditions and computing base coverage, constructing „coverage vector“ for each position for 8 BAM files dataset (approx. 9GB) and a different number of worker nodes with samtools, SeqPig and SparkSeq. Speedup SparkSeq vs SeqPig: 8.40 up to 9.15

```
1 samtools mpileup allBig.bam |awk '{print $1":"$2":"$4}' > allBig.txt
```

1.7.2 SparkSeq

```
1 import pl.elka.pw.sparkseq.seqAnalysis.SparkSeqAnalysis
2 val seqAnalysis = new SparkSeqAnalysis(sc,
3   "hdfs://hadoop-name001:9000//seqpig/allBig.bam",1,1,1)
4 val out=seqAnalysis.getCoverageBase()
5 out.saveAsTextFile("hdfs://hadoop-name001:9000//seqpig/out1_1.txt")
```

1.7.3 SeqPig

```
1 set pig.exec.mapPartAgg true;
2 set pig.exec.mapPartAgg.minReduction 5;
3 set pig.schematuple on;
4 DEFINE ReadUnmapped SAMFlagsFilter('HasSegmentUnmapped');
5 A = load 'hdfs://hadoop-name001:9000//seqpig/allBig.bam' using BamLoader('yes');
6 A = FOREACH A GENERATE read, flags, refname, start, cigar, basequal, mapqual;
7 A = FILTER A BY not ReadUnmapped(flags);
8 RefPos = FOREACH A GENERATE ReadRefPositions(read, flags, refname, start,
9   cigar, basequal), mapqual;
10 flatset = FOREACH RefPos GENERATE flatten($0);
11 grouped = GROUP flatset BY ($0, $1);
12 base_counts = FOREACH grouped GENERATE group.chr, group.pos, COUNT(flatset);
13 store base_counts into 'hdfs://hadoop-name001:9000//seqpig/input.basecounts1_1';
```

1.8 Test 2 results

1.9 Code listings test 3

1.9.1 samtools

```
1 samtools view -c allBig.bam
```

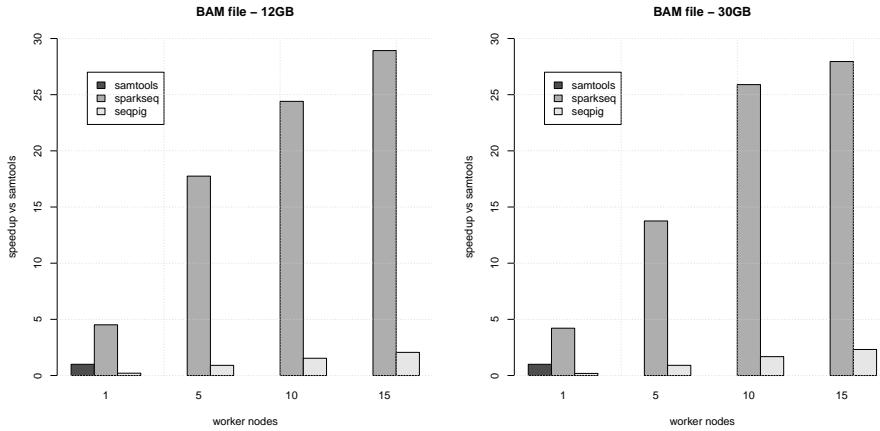


Figure 2: Computing of base level coverage for an approx. 12 and 30GB BAM files dataset and a different number of worker nodes with samtools, SeqPig and SparkSeq. Comparative speedup SparkSeq vs SeqPig: 12.02 up to 22.61 for 30GB dataset, and 14.04 up to 21.66 for 12GB dataset.

1.9.2 SparkSeq

```

1 import pl.elka.pw.sparkseq.seqAnalysis.SparkSeqAnalysis
2 val seqAnalysis = new SparkSeqAnalysis(sc,
3   "hdfs://hadoop-name001:9000//seqpig/allBig.bam", 1, 1, 1)
4 seqAnalysis.bamFile.map(r=>r._2).count()

```

1.9.3 SeqPig

```

1 set pig.exec.mapPartAgg true;
2 set pig.exec.mapPartAgg.minReduction 5;
3 set pig.schematuple on;
4 A = load 'hdfs://hadoop-name001:9000//seqpig/allBig.bam' using BamLoader('yes');
5 B = group A all;
6 C = foreach B generate COUNT(A);
7 dump C;

```

1.10 Test 3 results

1.11 Code listings test 4

1.11.1 samtools

```

1 samtools view -F 1028 -q 19 -c allBig.bam

```

1.11.2 SparkSeq

```

1 import pl.elka.pw.sparkseq.seqAnalysis.SparkSeqAnalysis
2 val seqAnalysis = new SparkSeqAnalysis(sc,
3   "hdfs://hadoop-name001:9000//seqpig/allBig.bam", 1, 1, 1)
4 seqAnalysis.bamFile.map(r=>r._2)
5   .filter(r=> r._2.get.getReadUnmappedFlag==false &&
6   r._2.get.getDuplicateReadFlag==false && r._2.get.getMappingQuality>19 ).count

```

1.11.3 SeqPig

```

1 set pig.exec.mapPartAgg true;
2 set pig.exec.mapPartAgg.minReduction 5;

```

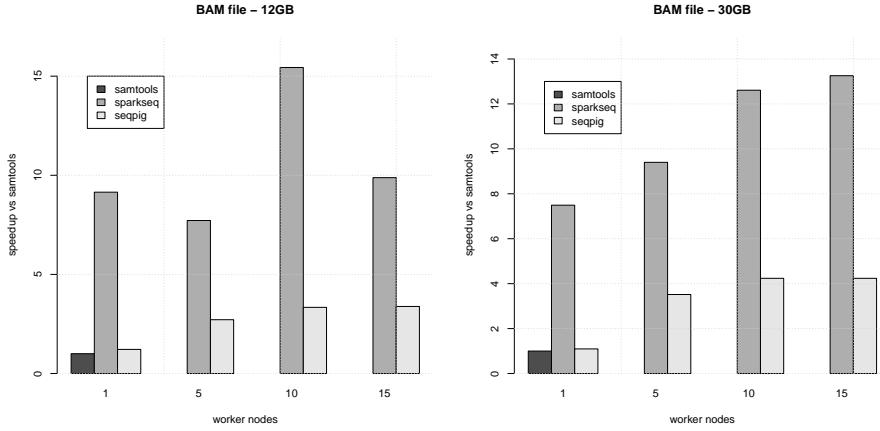


Figure 3: Calculating the total number of short reads for an approx. 12 and 30GB BAM files dataset and a different number of worker nodes with samtools, SeqPig and SparkSeq. Comparative speedup SparkSeq vs SeqPig: 3.13 up to 6.84 for 30GB dataset, and 2.92 up to 7.52 for 12GB dataset.

```

3 set pig.schematuple on;
4 DEFINE ReadUnmapped SAMFlagsFilter('HasSegmentUnmapped');
5 DEFINE IsDuplicate fi.alto.seqpig.filter.SAMFlagsFilter('IsDuplicate');
6 A = load 'hdfs://hadoop-name001:9000//seqpig/allBig.bam' using BamLoader('yes');
7 B = FILTER A BY not ReadUnmapped(flags) AND not IsDuplicate(flags) AND mapqual > 19;
8 C = group B all;
9 D = foreach C generate COUNT(B);
10 dump D;

```

1.12 Test 4 results

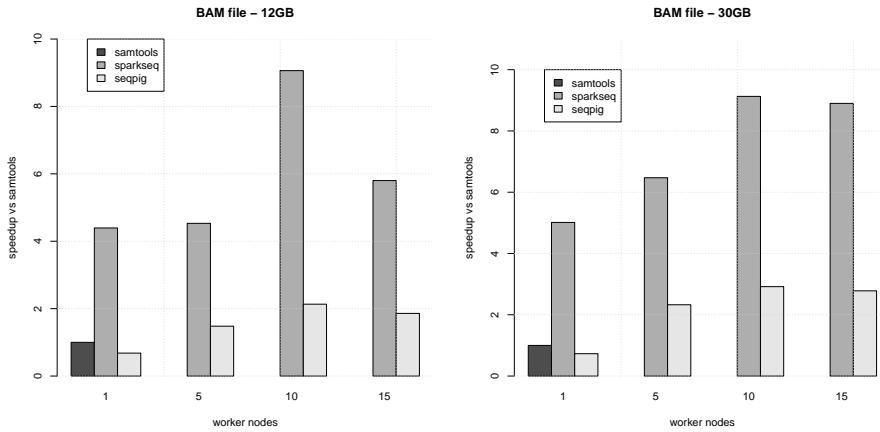


Figure 4: Calculating the number of short reads that are mapped, not duplicates and have mapping quality >19 for an approx. 12 and 30GB BAM files dataset and a different number of worker nodes with samtools, SeqPig and SparkSeq. Comparative speedup SparkSeq vs SeqPig: 3.20 up to 6.87 for 30GB dataset, and 3.12 up to 6.45 for 12GB dataset.

1.13 Tests results discussion

Test 1 and 2 are clearly the most computationally intensive ones and hence are the best proofs of scalability of SparkSeq and SeqPig. In case of Test 3 and Test 4 the speedup is mostly limited by the overall IO-throughput

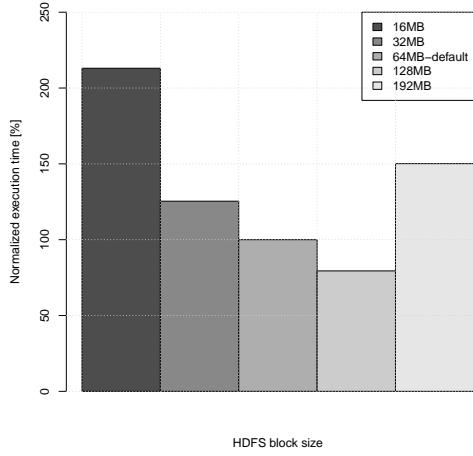


Figure 5: The impact of HDFS block size selection

and the fact that even computation time using one node is quite low (and the computation starting overheads are high when compared to the total time, so the percentage of the time that can be reduced by distributed computations is relatively low). Besides, the BAM file format uses a centralized header that may lead to block contentions in some cases while accessed concurrently. In the Test 1 the scalability characteristic of both SeqPig and SparkSeq on up to 7 nodes is close to linear. Than it shows below linear returns as due to the size of the dataset some of the computing resources were not fully utilized all the time. Comparison to samtools/Unix-based pipelines has demonstrated that SparkSeq is equally fast using a single node, and on multiple nodes offers a very good scalability. This makes it a good choice for both types of computing : using "domestic" hardware (e.g. laptops, medium-size desktops) and large clusters or servers equipped with many CPUs. In all the tests SparkSeq was significantly faster than SeqPig.

2 HDFS block size evaluation

In another round of tests the impact of HDFS block size was evaluated. It is a parameter the value of which should be adjusted in line with throughput of the underlying storage performance. We proved that HDFS block size selection in case of NGS data formats can also result in large performance boost — in tested hardware infrastructure setting parameter to 128MB (instead of default 64MB) cut processing time by more than 20% (Figure 5). This was the optimal value — increasing it further has led to serious performance decrease.

3 SparkSeq scalability

Testing of SparkSeq has been also done using 32 BAM files (approx. 40GB) from an RNA-sequencing experiment. The results presented in the Figure 6 are showing that finding the coverage for all the nucleotides scales close to linearly with the adding of the worker nodes. It shows again that computationally intensive tasks can benefit from the MapReduce model the most. In the case of exon coverage computation, the speedup up to 4 worker nodes is very good, then it becomes I/O-bound due to the fact that non-local shared storage with a measured throughput of approx. 800MB/s was used to store the virtual disks of all the virtual machines. This issue could be solved by either using a more performant shared storage or by using local storage only for backing up the disks of the virtual machines. It is also far less computationally intensive task and cluster overheads play here a greater role. Besides, the header contentions are of less importance here, since worker processes access concurrently 32 BAM file headers not just one as in the above comparisons.

4 SparkSeq cache strategies evaluation

Cache experiments shows clearly that using fast data serialized (like KryoSerializer) instead of Java built-in together LZF or Snappy compression can speed up multi-pass data querying up 80-110 times and reduce memory

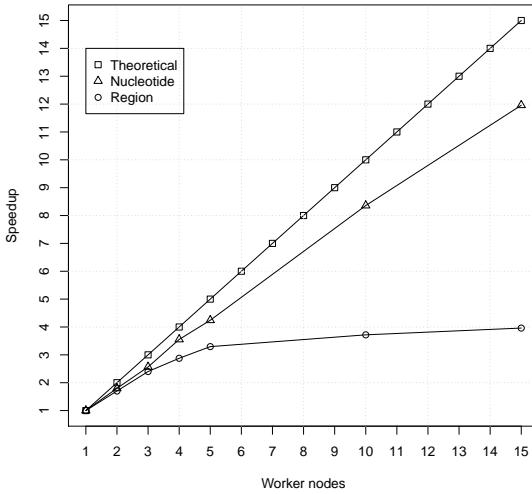


Figure 6: SparkSeq scalability for exon and nucleotide-level analysis

consumption approx. 13 times (Figure 7). Abbreviations used: Raw-size of Scala RDD without serialization or data compression, MEM-memory, J-JavaSerializer, K-KryoSerializer, L-LZF Compression, S-Snappy compression, N-no serialization or no compression, e.g. MEM-K-N means in memory data cache, objects serialized with KryoSerializer and without any compression.

SparkSeq can use the same storage level settings as Apache Spark², by default it is `MEMORY_ONLY` as it seems to be most CPU-efficient one.

5 Comparison of samtools, SeqPig and SparkSeq features

Feature	samtools	SeqPig	SparkSeq
programming language	C/shell	Java/Pig	Scala
vertical scalability	Partial	Yes	Yes
horizontal scalability	No	Yes	Yes
local/MapReduce mode	Yes/No	Yes/Yes	Yes/Yes
control structures	No	Yes when embeded in Python,Groovy,etc.	Yes
Turing complete	No	Yes when extended with Java UDFs	Yes
regular expressions	Yes when used with awk,etc.	Yes	Yes
multisample analysis	Partial	Partial	Yes
reads filtering	Partial	Yes	Yes
pileup	Yes	Yes	Partial
SNP/INDELS calling	Yes	No	No
base coverage	Yes	Yes	Yes
feature counts	No	No	Yes
junctions analysis	No	Partial	Yes

²More information can be found at <http://spark.incubator.apache.org/docs/latest/scala-programming-guide.html#rdd-persistence>.

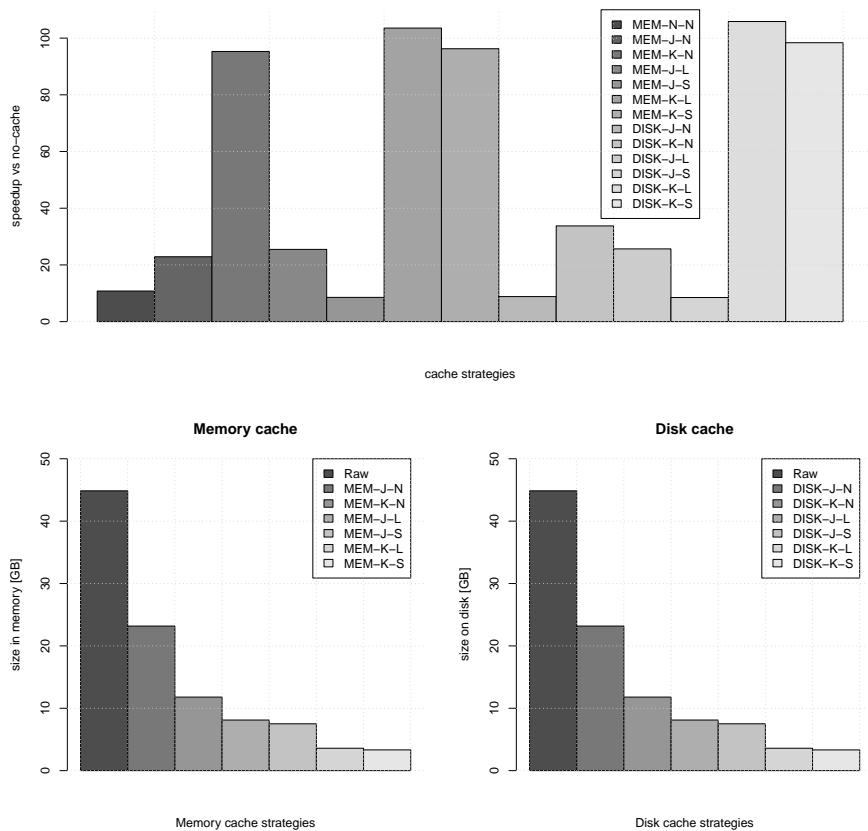


Figure 7: Apache Spark cache strategies comparison

6 Getting started with SparkSeq

Currently the only supported operating systems are Linux 64bit and Mac OS X.

6.1 Dependencies

In order to build SparkSeq you need to have the following dependencies installed:

- Java Virtual Machine >= 1.6
- Scala compiler >= 2.10
- SBT - Simple Build Tool
- Apache Spark >= 0.9.0

the rest of dependencies will be automatically downloaded by sbt during the build process.

6.2 Installation

1. If you are using development Apache Spark version that is not available via Maven repositories you need to publish compiled Jars to your local repository:

```
1 cd ${SPARK_HOME}  
2 sbt/sbt publish-local
```

2. Clone the most recent version from Bitbucket repository:

```
1 git clone https://bitbucket.org/mwiewiorka/sparkseq.git  
2 cd sparkseq
```

3. If you also wish to read data from Hadoop's HDFS, you will also need to edit **build.sbt** file (from sparkseq-core directory) and choose hadoop-client for your version of HDFS, e.g. if you would like to use version 1.2.1, the configuration should be like this:

```
1 val DEFAULT_HADOOP_VERSION = "1.2.1"
```

Please make sure that you use the same version of hadoop-client library for building Apache Spark and SparkSeq. Here you can find some information on how to configure this in Apache Spark.

4. Build SparkSeq

```
1 sbt package
```

5. Run unit test

```
1 sbt test
```

If all tests are passed you are ready to go:-)

6.3 Basics

The simplest way to work with SparkSeq is to use Apache Spark's interactive shell. Before launching you should add paths to SparkSeq jarfile and its dependencies to **ADD_JARS**, **SPARK_CLASSPATH** and **SPARK_JAVA_OPTS** environment variables, e.g.(using Bash syntax):

```
1 export ADD_JARS="${ADD_JARS}, /opt/hadoop-classpath/hadoop-bam-6.1-SNAPSHOT.jar,/opt/hadoop-classpath/picard-1.93.jar,  
2 /opt/hadoop-classpath/sam-1.93.jar,/opt/hadoop-classpath/variant-1.93.jar,/opt/hadoop-classpath/tribble-1.93.jar,  
3 /opt/hadoop-classpath/commons-jexl-2.1.1.jar,/opt/hadoop-classpath/sparkseq_2.10-0.1.jar"  
4  
5 export SPARK_CLASSPATH="${SPARK_CLASSPATH}: /opt/scala-2.10.3/lib/scala-library.jar:/opt/scala-2.10.3/lib/scala-compiler.jar:  
6 /opt/hadoop-classpath/hadoop-bam-6.1-SNAPSHOT.jar:/opt/hadoop-classpath/picard-1.93.jar:/opt/hadoop-classpath/sam-1.93.jar:  
7 /opt/hadoop-classpath/variant-1.93.jar:/opt/hadoop-classpath/tribble-1.93.jar:/opt/hadoop-classpath/commons-jexl-2.1.1.jar:
```

```

8 /opt/hadoop-classpath/sparkseq_2.10-0.1.jar"
9
10 export SPARK_JAVA_OPTS="-Dspark.rdd.compress=true -Dspark.serializer=org.apache.spark.serializer.KryoSerializer
11 -Dspark.kryoserializer.buffer.mb=25"

```

Now you can start Apache Spark's REPL locally by e.g. typing:

```

1 cd ${SPARK_HOME}/bin
2 MASTER=local [4] SPARK_MEM=4g ./spark-shell

```

If you have your Apache Spark cluster up and running then you need to set only the **MASTER** variable to URI of your Apache Spark master node in the cluster.

At that point you can run your first analysis with SparkSeq using Apache Spark's REPL-calculate, to compute how many nucleotides in a BAM file have read coverage greater or equal to 10:

```

1 import pl.elka.pw.sparkseq.seqAnalysis.SparkSeqAnalysis
2 val seqAnalysis = new SparkSeqAnalysis(sc, "pathToYourBAM", 1, 1, 1)
3 seqAnalysis.getCoverageBase().filter(p=>(p._2>=10)).count()

```

Saving results to a text file, e.g.:

For more information on available data transformations (like map, groupByKey, etc.) and actions please consult:
Apache Spark's Documentation.

6.4 Project documentation

Latest version of code examples, Windows Azure cloud deployment guide, Ansible³ playbooks to automate prerequisites installation and API documentation can be found on the project's wiki (<https://bitbucket.org/mwiewiora/sparkseq/wiki/Home>)

7 Multisample analyses tutorial

This tutorial presents how to perform various operations on your BAM files using SparkSeq.
We assume that you have the following components of your system up and running:

- Apache Spark cluster (either in standalone mode or using resource manager like Mesos or YARN) or Apache Spark local mode (More info here)
- SparkSeq compiled or downloaded from the unofficial Maven repository (please pay attention to HDFS version compatibility) and appropriate variables added to SPARK_CLASSPATH, ADD_JARS and SPARK_JAVA_OPTS environment (More info here)
- Apache Spark's REPL connected to a cluster or ran locally (More info here)
- HDFS NameNode(s) and DataNode(s)

This tutorial uses an RNA sequencing dataset with alignments limited to the chromosome Y, freely available at DERFinder's github. It should be copied this dataset to our local Hadoop cluster and form the following directory structure:

```

1 spark@sparkseq:/sandbox$ hadoop fs -lsr /BAM/64MB/derfinder/chrom_Y | cut -c61-
2
3 /BAM/64MB/derfinder/chrom_Y/F
4 /BAM/64MB/derfinder/chrom_Y/F/orbFrontalF23_Y.bam
5 /BAM/64MB/derfinder/chrom_Y/F/orbFrontalF2_Y.bam
6 /BAM/64MB/derfinder/chrom_Y/F/orbFrontalF33_Y.bam
7 /BAM/64MB/derfinder/chrom_Y/F/orbFrontalF40_Y.bam
8 /BAM/64MB/derfinder/chrom_Y/F/orbFrontalF55_Y.bam
9 /BAM/64MB/derfinder/chrom_Y/F/orbFrontalF56_Y.bam
10 /BAM/64MB/derfinder/chrom_Y/M
11 /BAM/64MB/derfinder/chrom_Y/M/orbFrontalF11_Y.bam

```

³<https://github.com/ansible/ansible>

```

12 /BAM/64MB/derfinder/chrom_Y/M/orbFrontalF1_Y.bam
13 /BAM/64MB/derfinder/chrom_Y/M/orbFrontalF32_Y.bam
14 /BAM/64MB/derfinder/chrom_Y/M/orbFrontalF3_Y.bam
15 /BAM/64MB/derfinder/chrom_Y/M/orbFrontalF42_Y.bam
16 /BAM/64MB/derfinder/chrom_Y/M/orbFrontalF43_Y.bam
17 /BAM/64MB/derfinder/chrom_Y/M/orbFrontalF47_Y.bam
18 /BAM/64MB/derfinder/chrom_Y/M/orbFrontalF53_Y.bam
19 /BAM/64MB/derfinder/chrom_Y/M/orbFrontalF58_Y.bam

```

Yet another directory should be created for storing various BED files that can be downloaded from the project's page:

```

1 spark@sparkseq:/sandbox$ hadoop fs -lsr /BAM/64MB/aux | cut -c61-
2
3 /BAM/64MB/aux/Homo_sapiens.GRCh37.74_exons_chr_merged_id_st.bed
4 /BAM/64MB/aux/Homo_sapiens.GRCh37.74_exons_chr_sort_uniq.bed
5 /BAM/64MB/aux/Homo_sapiens.GRCh37.74_genes_chr_merged_swap.bed

```

If you are not sure what the SparkSeq's syntax is, you can always find more information at <http://mwiewiora.bitbucket.org/#package>

7.1 Creating SparkSeqAnalysis object

The most important SparkSeq's class for performing multisample analyses is SparkSeqAnalysis.

An object of this class stores all the information about the samples and references them.

It features a simple API and together with Apache Spark's transformations make SparkSeq a powerful yet easy tool for RNA/DNA analyses.

So let's create a SparkSeqAnalysis object and attach all the BAM files from /BAM/64MB/derfinder/chrom.Y/M/ directory. We first initialize an object using the first sample and the rest of them we can very easily add using addBAM method. Let's run Apache Spark's REPL and set the maximum memory per worker machine to 2GB (as the default 512MB may be insufficient in some cases):

```

1 spark@sparkseq:~/spark-0.9.0-incubating/bin$ SPARK_MEM=2g ./spark-shell

```

and then create a SparkSeqAnalysis object:

```

1 import pl.elka.pw.sparkseq.seqAnalysis.SparkSeqAnalysis
2 import pl.elka.pw.sparkseq.conversions.SparkSeqConversions
3 import pl.elka.pw.sparkseq.util.SparkSeqRegType._
4
5 val seqAnalysis = new SparkSeqAnalysis(sc, "/BAM/64MB/derfinder/chrom_Y/M/orbFrontalF1_Y.bam", 1, 1, 1)
6 val caseId = Array(11, 32, 3, 42, 43, 47, 53, 58)
7 for(i<-caseId)
8   seqAnalysis.addBAM(sc, "/BAM/64MB/derfinder/chrom_Y/M/orbFrontalF"+i.toString+"_Y.bam", i, 1)

```

Then we can list all the samples has been attached using listSamples method:

```

1 scala> seqAnalysis.listSamples()
2 /BAM/64MB/derfinder/chrom_Y/M/orbFrontalF1_Y.bam
3 /BAM/64MB/derfinder/chrom_Y/M/orbFrontalF11_Y.bam
4 /BAM/64MB/derfinder/chrom_Y/M/orbFrontalF32_Y.bam
5 /BAM/64MB/derfinder/chrom_Y/M/orbFrontalF3_Y.bam
6 /BAM/64MB/derfinder/chrom_Y/M/orbFrontalF42_Y.bam
7 /BAM/64MB/derfinder/chrom_Y/M/orbFrontalF43_Y.bam
8 /BAM/64MB/derfinder/chrom_Y/M/orbFrontalF47_Y.bam
9 /BAM/64MB/derfinder/chrom_Y/M/orbFrontalF53_Y.bam
10 /BAM/64MB/derfinder/chrom_Y/M/orbFrontalF58_Y.bam

```

7.2 Counting and displaying

Let's start with some easy counting of samples and reads. We will use getSamples method to obtain all samples, getReads to get all reads and getSampleReads(11) to get all reads from the sample '11':

```

1 scala> seqAnalysis.getSamples.count
2 res2: Long = 9
3
4 scala> seqAnalysis.getReads.count
5 res3: Long = 925231
6
7 scala> seqAnalysis.getSampleReads(11).count
8 res5: Long = 120493

```

To get all reads info in a way similar to samtools view (first 5 reads):

```

1 scala> seqAnalysis.viewSampleReads(11,5)
2
3 011211_fca_:6:1102:10430:158969:0:1    16     Y     2655174 50     100M   *     0     0
4
5 011211_fca_:6:1308:17172:193560:0:1    16     Y     2658568 50     100M   *     0     0
6
7 011211_fca_:6:2201:7222:62493:0:1    16     Y     2658568 50     100M   *     0     0
8
9 011211_fca_:6:2202:14432:150553:0:1    16     Y     2658568 50     100M   *     0     0
10
11 011211_fca_:6:1301:11508:181776:0:1    272    Y     2675692 3      100M   *     0     0

```

To get the read sequence only you may combine the methods in a following way:

```

1 scala> seqAnalysis.getSampleReads(11).take(5).map(r=> r._2.getReadString).foreach(println)
2
3 CACTTCGCTGCAGAGTACCGAAGCGGGATCTCGGGAAGCAAAGTCAATTCTCGGCAGCATCTCGCCTCCGACGAGGTGCGATACTTATAATTGGG
4 CTCCCAGATGCATATATTACAGGACGGGGTGTGTGAGGGAGACTCACTTGGGGTGGCAGTCACAAAAATTAAATAAGACACGAGATTTTATTATTTT
5 CTCCAGATGCATATATTACAGGACGGGGTGTGTGAGGGAGACTCACTTGGGGTGGCAGTCACAAAAATTAAATAAGACACGAGATTTTATTATTTT
6 CTCCAGATGCATATATTACAGGACGGGGTGTGTGAGGGAGACTCACTTGGGGTGGCAGTCACAAAAATTAAATAAGACACGAGATTTTATTATTTT
7 GAATGCTCCAGTTTGCCCCATTCACTGATATTGGCTGTGGTTGTATAAATAGCTTATTGAAATACATCCATCAACACTAGTTATTG

```

To get the average read length of all reads from all samples:

```

1 scala> seqAnalysis.getAvgReadLength
2 res1: Double = 100.45402715646146

```

7.3 Sorting

The following command can be used to sort all reads from all samples by sampleID and alignment start and then display the first 5 reads:

```

1 scala> seqAnalysis.sortReadsByAlign().take(5).foreach(println)
2
3 ((1,(chrY,2675709)),300811_fcB_:1:1108:2837:104706:0:1 101b aligned read.)
4 ((1,(chrY,2675709)),300811_fcB_:1:2101:14014:45631:0:1 101b aligned read.)
5 ((1,(chrY,2675754)),300811_fcB_:1:1305:7608:74961:0:1 101b aligned read.)
6 ((1,(chrY,2675923)),300811_fcB_:1:2304:9858:81704:0:1 101b aligned read.)
7 ((1,(chrY,2676173)),300811_fcB_:1:2303:10751:139532:0:1 101b aligned read.)

```

To sort the reads from a specific sample by alignment start:

```

1 scala> seqAnalysis.sortSampleReadsByAlign(11).take(5).foreach(println)
2
3 ((11,(chrY,2655174)),011211_fca_:6:1102:10430:158969:0:1 100b aligned read.)
4 ((11,(chrY,2658568)),011211_fca_:6:1308:17172:193560:0:1 100b aligned read.)
5 ((11,(chrY,2658568)),011211_fca_:6:2201:7222:62493:0:1 100b aligned read.)
6 ((11,(chrY,2658568)),011211_fca_:6:2202:14432:150553:0:1 100b aligned read.)
7 ((11,(chrY,2675692)),011211_fca_:6:1301:11508:181776:0:1 100b aligned read.)

```

To implement custom sorting, e.g. to find reads with the greatest number of CIGAR operators (i.e. to sort descending by CIGARString length):

```

1 scala> seqAnalysis.getSampleReads(11)
2 .map(r=>(r._2.getCigarLength, r._2.getCigarString, r._2.getReadName)))
3 .sortByKey(false)
4 .take(5)
5 .foreach(println)
6
7 (11,(73M1I6M2D2M1I6M1D4M1I6M,011211_fcA_:6:1201:20258:60417:0:1))
8 (11,(73M1I6M2D2M1I6M1D4M1I6M,011211_fcA_:6:1201:8410:133984:0:1))
9 (11,(73M1I6M2D2M1I6M1D4M1I6M,011211_fcA_:6:1208:14033:185503:0:1))
10 (11,(73M1I6M2D2M1I6M1D4M1I6M,011211_fcA_:6:2104:3708:73930:0:1))
11 (11,(73M1I6M2D2M1I6M1D4M1I6M,011211_fcA_:6:2206:9709:108114:0:1))

```

7.4 Filtering

7.4.1 Filter and undo

SparkSeqAnalysis class provides a great range of methods that can facilitate process of filtering out reads that do not meet our requirements. All filters follow *lazy evaluation* principle - they are not executed until they are really needed (so until some of Apache Spark's actions such as count, reduce, collect, etc is not called).

Filters can be combined by calling one method after another, similarly as combining conditions using 'logical and' operator. If you need more sophisticated combinations of logical operators you can use a generic *filterReads* method that allows to create any conditions on sampleID and reads.

If you wish to undo all the filtering you can call *undoFilter* method.

Currently up to **2** conditions per method are supported. In order to set eg. 3 conditions on start of alignment you can just call the same method twice: 1. with 2 conditions and the 2. with the remaining condition. Please note that calling filter method for the n-th time is equivalent to combining conditions using 'and' operator.

7.4.2 Quality filtering

7.4.2.1 Using base-pair qualities

To filter out all reads that have the mean base quality < 30:

```

1 import pl.elka.pw.sparkseq.statisticalTests.SparkSeqStats
2 seqAnalysis.filterBaseQualities(SparkSeqStats.mean(_)>=30).count
3 res7: Long = 925176

```

To filter out reads that have at least 2 base with quality < 30:

```

1 scala> seqAnalysis.filterBaseQualities(_.sortBy(r=>(-r)).takeRight(2).head >= 30).count
2 res9: Long = 849744

```

Please remember to **undo** all the filtering before running each example:

```

1 scala> seqAnalysis.undoFilter

```

7.4.2.2 Using the alignment quality

To keep only reads with alignment quality greater or equal to 30:

```

1 scala> seqAnalysis.filterMappingQuality(_ >= 30).count
2 res22: Long = 669871

```

7.4.3 Filtering with the alignment and reference names

To get only the reads that have start alignments between 2655174 and 2685880 in chrY:

```

1 scala> seqAnalysis.filterReferenceName(_ == "chrY")
2 scala> seqAnalysis.filterAlignmentStart(_ >= 2655174 && _ <= 2685880).count
3 res1: Long = 934

```

To get only the reads that are fitting into the interval [2655174,2700500] on chromosome Y:

```

1 scala> seqAnalysis.filterReferenceName(_ == "chrY")
2 scala> seqAnalysis.filterAlignmentStart(_ >= 2655174)
3 scala> seqAnalysis.filterAlignmentEnd(_ <= 2700500).count
4 res1: Long = 967

```

7.4.4 Filtering on read name and length

To get a read with a specific ID you can use the following method:

```
1 scala> seqAnalysis.filterReadName(_ == "011211_fcA_:6:1102:10430:158969:0:1")
2 scala> seqAnalysis.viewReads()
3 011211_fcA_:6:1102:10430:158969:0:1    16      Y      2655174 50      100M    *      0      0
```

7.4.5 Filtering on CIGAR Strings

To get only the reads with CIGAR String equal to “100M“ and then to display 2 of them:

```
1 scala> seqAnalysis.filterCigarString(_ == "100M").count
2 res21: Long = 425718
3
4 scala> seqAnalysis.viewReads(2)
5 011211_fcA_:6:1102:10430:158969:0:1    16      Y      2655174 50      100M    *      0      0
6
7 011211_fcA_:6:1308:17172:193560:0:1    16      Y      2658568 50      100M    *      0      0
```

To do some more sophisticated filtering you can also take advantage of regular expressions:

```
1 scala> val cigRegex = "^[0-9]+[0-9]+D[0-9]+M$".r
2 scala> seqAnalysis.filterCigarString(cigRegex.pattern.matcher(_).matches)
3 scala> seqAnalysis.viewReads(2)
4
5 300811_fcB_:1:1302:12119:168462:0:1    16      Y      2712173 50      29M1D72M    *      0      0
6
7 300811_fcB_:1:2204:12824:4029:0:1    16      Y      2719503 50      27M4D74M    *      0      0
```

7.4.6 Filtering using the alignment (SAM) flags

To filter out read duplicates (non-primary alignments) :

```
1 scala> seqAnalysis.filterDuplicateReadFlag(_ == false).count
2 res45: Long = 925231
```

To filter out unmapped reads:

```
1 scala> seqAnalysis.filterUnmappedFlag(_ == false).count
2 res1: Long = 925231
```

To filter out reads using other SAM flags (or their combinations) you can use *filterFlags* method and SAM flags calculator to find the necessary integer representation, e.g. ‘not primary alignment’= 256 :

```
1 scala> seqAnalysis.filterFlags(_ != 256).count
2 res4: Long = 829036
```

7.4.7 Combining filters

Filters methods can be combined in the the following ways: either in a single method using various logical operators (currently up to 2 conditions supported) or by calling methods several times or by mixing both approaches:

```
1 scala> seqAnalysis.filterAlignmentStart(_ >= 2655174 && _ < 2685880).count
2 res1: Long = 934
3
4 scala> seqAnalysis.undoFilter
5
6 scala> seqAnalysis.filterAlignmentStart(_ >= 2655174)
7 scala> seqAnalysis.filterAlignmentStart(_ < 2685880).count
8 s4: Long = 934
```

7.4.8 Generic filter

There is also a generic filter method that makes it possible to set conditions (currently up to 2) on any BAMRecord property as well as sampleID(_._1 refers to sampleID whereas _._2 to read object):

```
1 scala> seqAnalysis.filterReads(_.getReferenceName == "Y" && _.1==1).count
2 res16: Long = 105691
```

Using this generic method you can implement any of the above specialized filters.

7.4.9 Selecting samples

To filter the reads from a specific sample:

```
1 scala> seqAnalysis.filterSample(_ == 32).count
2 res1: Long = 99955
```

7.5 Computing the coverage function and counts of reads in genomic regions

The coverage with reads, as a function defined on all the nucleotides of the reference genome as well as read counts for genomic regions (e.g. exons or genes) are the typical summarizations in a sequencing experiment.

You can do this for the regions of interest, describing them in a BED-like, tab-delimited file format that contains the following columns:

```
1 chr    regStart regEnd  strand  upperRegionID   regionID
```

where regionID **must** be globally unique. RegionID **must** contain exactly 15 character, start with letters, end with digits and padded with zeroes.

For instance:

```
1 chr1    11869   12227   .      ENSG00000223972 ENSE00002234944
2 chr1    11872   12227   .      ENSG00000223972 ENSE00002234632
3 chr1    11874   12227   .      ENSG00000223972 ENSE00002269724
```

Precomputed BED-like files for human genes and exons according to Ensembl can be downloaded from [here](#).

7.5.1 Nucleotide-level coverage function

Coverage matrix is a generalization of a “coverage vector” and GenomicArray, which is explained here. It can be computed for all the positions in all samples or for some particular regions of interest. The following listings shows how to calculate, view (region) and save (all results):

```
1 scala> val baseCov = seqAnalysis.getCoverageBase()
2 scala> seqAnalysis.viewBaseCoverage("chrY",2675720,2675728)
3 Feature          Sample_1  Sample_3  Sample_11 Sample_32 Sample_42 Sample_43 Sample_47 Sample_53 Sample_58
4 =====
5 chrY,2675720    2       2       7       0       0       19      0       0       0
6 chrY,2675721    2       2       7       0       0       19      0       0       0
7 chrY,2675722    2       2       7       0       0       19      0       0       0
8 chrY,2675723    2       2       7       0       0       19      0       0       0
9 chrY,2675724    2       2       7       0       0       19      0       0       0
10 chrY,2675725   2       2       7       0       0       19      0       0       0
11 chrY,2675726   2       2       7       0       0       19      0       0       0
12 chrY,2675727   2       2       7       3       0       19      0       0       0
13 chrY,2675728   2       2       7       3       0       19      0       0       0
14
15 scala> seqAnalysis.saveBaseCoverageToFile("test_base.txt")
```

If need to calculate the coverage for a specific chromosome, or a region you can run the following commands:

```

1 scala> seqAnalysis.getCoverageBaseRegion("chrY",2600000,2700000)
2 scala> seqAnalysis.viewBaseCoverage("chrY",2675720,2675728)
3
4 Feature          Sample_1  Sample_3  Sample_11 Sample_32 Sample_42 Sample_43 Sample_47 Sample_53 Sample_58
5 =====
6 chrY,2675720    2        2        7        0        0        19       0        0        0
7 chrY,2675721    2        2        7        0        0        19       0        0        0
8 chrY,2675722    2        2        7        0        0        19       0        0        0
9 chrY,2675723    2        2        7        0        0        19       0        0        0
10 chrY,2675724   2        2        7        0        0        19       0        0        0
11 chrY,2675725   2        2        7        0        0        19       0        0        0
12 chrY,2675726   2        2        7        0        0        19       0        0        0
13 chrY,2675727   2        2        7        3        0        19       0        0        0
14 chrY,2675728   2        2        7        3        0        19       0        0        0

```

7.5.2 Exon-level counts

To calculate counts for 4 sample exons you can use *getCoverageRegion* method. The **unionMode** parameter is analogous to ‘union’ of htseq-count.

```

1 scala> val genExonsMapB = sc.broadcast(SparkSeqConversions.BEDFileToHashMap(sc, "/BAM/64MB/aux/Homo_sapiens.GRCh37.74_exons_chr_sort_uniq.bed"))
2 scala> seqAnalysis.getCoverageRegion(genExonsMapB, unionMode=true)
3 scala> val exonArray = Array("ENSE00000652498","ENSE00000652501","ENSE00000652502","ENSE00000652503")
4 scala> seqAnalysis.viewExonCoverage(exonArray)
5
6 Feature          Sample_1  Sample_3  Sample_11 Sample_32 Sample_42 Sample_43 Sample_47 Sample_53 Sample_58
7 =====
8 ENSE00000652498 186      251      135      168      131      140      100      198      149
9 ENSE00000652501 148      211      136      156      130      83       140      230      171
10 ENSE00000652502 207     383      173      198      83       129      188      176      169
11 ENSE00000652503 215     384      182      265      93       136      177      161      135
12
13 scala> seqAnalysis.saveFeatureCoverageToFile("test_exon.txt",Exon)

```

7.5.3 Gene-level counts

To view counts for 4 sample genes you can use also *getCoverageRegion* method. **unionMode** parameter is analogous to ‘union’ of htseq-count.

```

1 scala> val genesMapB = sc.broadcast(SparkSeqConversions.BEDFileToHashMap(sc, "/BAM/64MB/aux/Homo_sapiens.GRCh37.74_genes_chr_merged_swap.bed"))
2 scala> seqAnalysis.getCoverageRegion(genesMapB, unionMode=true)
3 scala> val geneArray = Array("ENSG00000012817","ENSG00000067048","ENSG00000067646","ENSG00000092377")
4 scala> seqAnalysis.viewGeneCoverage(geneArray)
5
6 Feature          Sample_1  Sample_3  Sample_11 Sample_32 Sample_42 Sample_43 Sample_47 Sample_53 Sample_58
7 =====
8 ENSG00000012817 8720     9103     8585     7375     5387     5835     7669     8175     7258
9 ENSG00000067048 6321     8647     5325     5598     3468     4140     4376     6827     4335
10 ENSG00000067646 1305    1419     1610     853      966      535      848      1043     920
11 ENSG00000092377 75      48      31      61      29      10      31      26      42
12
13 scala> seqAnalysis.saveFeatureCoverageToFile("test_gene.txt",Gene)

```

7.6 Grouping

To get a coverage vector for a specific genome position across samples you can do the following:

```

1 scala> val seqBaseCov = seqAnalysis.getCoverageBaseRegion("chrY",2600000,2700000)
2 scala> seqBaseCov.map(r=>(SparkSeqConversions.stripSampleID(r._1),r._2))
3 .groupByKey()
4 .map(r=>(SparkSeqConversions.idToCoordinates(r._1),r._2))
5 .take(10)
6 .foreach(println)
7
8 ((chrY,2658591),ArrayBuffer(3))
9 ((chrY,2677791),ArrayBuffer(10, 10, 6, 5, 4, 3, 5))
10 ((chrY,2655261),ArrayBuffer(1))
11 ((chrY,2694913),ArrayBuffer(10))
12 ((chrY,2677365),ArrayBuffer(6, 21, 5, 16, 1))
13 ((chrY,2676565),ArrayBuffer(6))
14 ((chrY,2692535),ArrayBuffer(3))
15 ((chrY,2695925),ArrayBuffer(2))

```

```

16 ((chrY,2677473),ArrayBuffer(11, 7, 24, 7, 17, 2, 8))
17 ((chrY,2686485),ArrayBuffer(1))

```

7.7 Joining multpile SparkSeqAnalysis objects

In this section we will show how to do joins in SparkSeq in order to define more complex analyses using multiple SparkSeqAnalysis objects. If you are not familiar with the difference between full outer join and inner join, please consult Wikipedia.

In the previous examples we have operated only on the male samples. To demonstrate how to operate on 2 SparkSeqAnalysis objects we will create another two for storing male and female samples separately:

```

1 import pl.elka.pw.sparkseq.seqAnalysis.SparkSeqAnalysis
2 import pl.elka.pw.sparkseq.conversions.SparkSeqConversions
3 import pl.elka.pw.sparkseq.util.SparkSeqRegType._
4
5 val seqAnalysisMale = new SparkSeqAnalysis(sc,"/BAM/64MB/derfinder/chrom_Y/M/orbFrontalF1_Y.bam",1,1,1)
6 val maleId = Array(11, 32, 3, 42, 43, 47, 53, 58)
7 for(i<-maleId)
8     seqAnalysisMale.addBAM(sc,"/BAM/64MB/derfinder/chrom_Y/M/orbFrontalF"+i.toString+"_Y.bam",i,1)
9
10 val seqAnalysisFemale = new SparkSeqAnalysis(sc,"/BAM/64MB/derfinder/chrom_Y/F/orbFrontalF2_Y.bam",2,1,1)
11 val femaleId = Array(23, 33, 40, 55, 56)
12 for(i<-femaleId)
13     seqAnalysisFemale.addBAM(sc,"/BAM/64MB/derfinder/chrom_Y/F/orbFrontalF"+i.toString+"_Y.bam",i,1)

```

And a quick check if all the samples are correctly attached:

```

1 scala> seqAnalysisMale.listSamples
2 /BAM/64MB/derfinder/chrom_Y/M/orbFrontalF1_Y.bam
3 /BAM/64MB/derfinder/chrom_Y/M/orbFrontalF11_Y.bam
4 /BAM/64MB/derfinder/chrom_Y/M/orbFrontalF32_Y.bam
5 /BAM/64MB/derfinder/chrom_Y/M/orbFrontalF3_Y.bam
6 /BAM/64MB/derfinder/chrom_Y/M/orbFrontalF42_Y.bam
7 /BAM/64MB/derfinder/chrom_Y/M/orbFrontalF43_Y.bam
8 /BAM/64MB/derfinder/chrom_Y/M/orbFrontalF47_Y.bam
9 /BAM/64MB/derfinder/chrom_Y/M/orbFrontalF53_Y.bam
10 /BAM/64MB/derfinder/chrom_Y/M/orbFrontalF58_Y.bam
11
12 scala> seqAnalysisFemale.listSamples
13 /BAM/64MB/derfinder/chrom_Y/F/orbFrontalF2_Y.bam
14 /BAM/64MB/derfinder/chrom_Y/F/orbFrontalF23_Y.bam
15 /BAM/64MB/derfinder/chrom_Y/F/orbFrontalF33_Y.bam
16 /BAM/64MB/derfinder/chrom_Y/F/orbFrontalF40_Y.bam
17 /BAM/64MB/derfinder/chrom_Y/F/orbFrontalF55_Y.bam
18 /BAM/64MB/derfinder/chrom_Y/F/orbFrontalF56_Y.bam
19
20 scala> seqAnalysisMale.getSamples.count
21 res7: Long = 9
22
23 scala> seqAnalysisFemale.getSamples.count
24 res8: Long = 6

```

Now we will do the coverage function summarization as described in the previous section on those 2 SparkSeqAnalysis objects separately
in order to get a coverage vector across all the samples from the group:

```

1 scala> val seqBaseCovMale= seqAnalysisMale.getCoverageBaseRegion("chrY",2708570,2708870)
2     .map(r=>(SparkSeqConversions.stripSampleID(r._1),r._2) ).groupByKey()
3
4 scala> val seqBaseCovFemale= seqAnalysisFemale.getCoverageBaseRegion("chrY",2708570,2708870)
5     .map(r=>(SparkSeqConversions.stripSampleID(r._1),r._2) ).groupByKey()

```

7.7.1 Full outer join

To join coverage vectors of males and females by position even if there is no coverage for it in one of the groups you can use a full outer join. For position (chrY,2708650) there are no reads mapped to this position in any female sample and this is why we get empty ArrayBuffer. The commands are as follows:

```

1 scala> val outerJoin = seqBaseCovMale.cogroup(seqBaseCovFemale)
2 scala> outerJoin.filter(r=>r._2.length>=3).sortByKey()
3 .map(r=>(SparkSeqConversions.idToCoordinates(r._1),r._2)).take(10).foreach(println)
4
5 ((chrY,2708650),(ArrayBuffer(ArrayBuffer(2, 12, 1)),ArrayBuffer()))
6 ((chrY,2708651),(ArrayBuffer(ArrayBuffer(2, 12, 1)),ArrayBuffer(ArrayBuffer(2))))
7 ((chrY,2708652),(ArrayBuffer(ArrayBuffer(12, 1, 2)),ArrayBuffer(ArrayBuffer(2))))
8 ((chrY,2708653),(ArrayBuffer(ArrayBuffer(1, 12, 2)),ArrayBuffer(ArrayBuffer(2))))
9 ((chrY,2708654),(ArrayBuffer(ArrayBuffer(12, 10, 1)),ArrayBuffer(ArrayBuffer(2))))
10 ((chrY,2708655),(ArrayBuffer(ArrayBuffer(10, 1, 12)),ArrayBuffer(ArrayBuffer(2))))
11 ((chrY,2708656),(ArrayBuffer(ArrayBuffer(1, 10, 12)),ArrayBuffer(ArrayBuffer(2))))
12 ((chrY,2708657),(ArrayBuffer(ArrayBuffer(10, 12, 1)),ArrayBuffer(ArrayBuffer(2))))
13 ((chrY,2708658),(ArrayBuffer(ArrayBuffer(1, 12, 1, 10)),ArrayBuffer(ArrayBuffer(2))))
14 ((chrY,2708659),(ArrayBuffer(ArrayBuffer(1, 1, 12, 4, 10)),ArrayBuffer(ArrayBuffer(2)))

```

7.7.2 Inner join

To join coverage vectors if and only if the both groups have at least one element in a vector for a given genomic position you can do the inner join operation:

```

1 scala> val innerJoin = seqBaseCovMale.join(seqBaseCovFemale)
2 scala> innerJoin.filter(r=>r._2.length>=6).sortByKey()
3 .map(r=>(SparkSeqConversions.idToCoordinates(r._1),r._2)).take(10).foreach(println)
4
5 ((chrY,2708700),(ArrayBuffer(8, 7, 10, 6, 27, 1, 1, 3),ArrayBuffer(1, 4, 4, 5, 3, 4)))
6 ((chrY,2708701),(ArrayBuffer(8, 27, 7, 1, 3, 6, 12, 1),ArrayBuffer(5, 4, 4, 1, 4, 3)))
7 ((chrY,2708702),(ArrayBuffer(7, 27, 8, 1, 1, 6, 3, 12),ArrayBuffer(4, 5, 4, 4, 1, 3)))
8 ((chrY,2708703),(ArrayBuffer(12, 1, 3, 1, 27, 6, 7, 8),ArrayBuffer(4, 3, 6, 4, 4, 1)))
9 ((chrY,2708704),(ArrayBuffer(7, 27, 1, 3, 6, 12, 8, 1),ArrayBuffer(4, 6, 3, 4, 1, 5)))
10 ((chrY,2708705),(ArrayBuffer(3, 2, 27, 7, 13, 1, 6, 8),ArrayBuffer(1, 4, 4, 6, 4, 5)))
11 ((chrY,2708706),(ArrayBuffer(1, 6, 7, 13, 3, 8, 27, 2),ArrayBuffer(5, 6, 6, 4, 5, 1)))
12 ((chrY,2708707),(ArrayBuffer(27, 2, 6, 8, 13, 7, 3, 1),ArrayBuffer(4, 6, 5, 9, 5, 1)))
13 ((chrY,2708708),(ArrayBuffer(8, 7, 27, 3, 2, 1, 13, 6),ArrayBuffer(5, 4, 5, 6, 9, 1)))
14 ((chrY,2708709),(ArrayBuffer(27, 2, 8, 7, 6, 1, 13, 3),ArrayBuffer(4, 6, 9, 5, 1, 5)))

```

7.8 Junction reads analysis

To get junction reads count you need to initialize SparkSeqJunctionAnalysis object and pass SparkSeqAnalysis object as an input parameter:

```

1 scala> import pl.elka.pw.sparkseq.seqAnalysis.SparkSeqAnalysis
2 scala> import pl.elka.pw.sparkseq.conversions.SparkSeqConversions
3 scala> import pl.elka.pw.sparkseq.util.SparkSeqRegType.-
4 scala> import pl.elka.pw.sparkseq.junctions.SparkSeqJunctionAnalysis
5
6 scala> val seqAnalysis = new SparkSeqAnalysis(sc,"/BAM/64MB/derfinder/chrom_Y/M/orbFrontalF1_Y.bam",1,1,1)
7 scala> val caseId = Array(11, 32, 3, 42, 43, 47, 53, 58)
8 scala> for(i<-caseId)
9   seqAnalysis.addBAM(sc,"/BAM/64MB/derfinder/chrom_Y/M/orbFrontalF"+i.toString+"_Y.bam",i,1)
10
11 scala> val juncAnalysis = new SparkSeqJunctionAnalysis(seqAnalysis)
12 scala> juncAnalysis.getJuncReadsCounts()
13 scala> juncAnalysis.viewJuncReadsCounts(20)
14 SampleID ChrName StartPos EndPos Count
15 =====
16 3 chrY 2722788 2733104 186
17 11 chrY 2722714 2733030 169
18 3 chrY 2722713 2733029 157
19 11 chrY 2722789 2733105 140
20 32 chrY 2710283 2712117 120
21 58 chrY 2710277 2712111 119
22 3 chrY 2710282 2712116 119
23 1 chrY 2710282 2712116 116
24 1 chrY 2733258 2734805 116
25 1 chrY 2722713 2733029 114
26 3 chrY 2733208 2734755 112
27 11 chrY 2710278 2712112 106
28 53 chrY 2710226 2712060 105
29 1 chrY 2710277 2712111 104
30 11 chrY 2710283 2712117 103
31 3 chrY 2712235 2713623 102
32 1 chrY 2722788 2733104 100
33 3 chrY 2733258 2734805 94

```

34	3	chrY	2713704	2722560	94
35	32	chrY	2722714	2733030	94

7.9 Using cache

If one is going to query results of computations several times, it may be a good idea to cache them in memory or/and disk. Apache Spark provide a few caching strategies that are described in section Apache Spark RDD persistence. Here we demonstrate using two of them and show the performance benefits. In either case we need to follow the same procedure:

1. Mark RDD as cached and optionally choose cache storage level (using cache or persist method with storage level as a parameter)
2. Populate the cache by running the query for the first time or by running some other Apache Spark's action like e.g. count
3. Do the analyses
4. Optionally purge the cache using 'unpersist' method

To cache results in memory:

```

1  scala> val seqBaseCovMale= seqAnalysisMale.getCoverageBase
2  .map(r=>(SparkSeqConversions.stripSampleID(r._1),r._2) ).groupByKey()
3  scala> val posStart = SparkSeqConversions.chrToLong("chrY") + 2708701
4  scala> val posEnd = SparkSeqConversions.chrToLong("chrY") + 2708708
5  scala> seqBaseCovMale.filter(r=>r._1 >=posStart && r._1<=posEnd).collect.foreach(println)
6  14/03/28 11:10:34 INFO spark.SparkContext: Job finished: collect at <console>:26, took 7.4827817 s
7  (110002708701,ArrayBuffer(27, 8, 12, 3, 7, 1, 1, 6))
8  (110002708700,ArrayBuffer(1, 3, 1, 6, 27, 8, 10, 7))
9  (110002708703,ArrayBuffer(1, 3, 6, 8, 7, 27, 1, 12))
10 (110002708705,ArrayBuffer(13, 3, 6, 8, 1, 2, 27, 7))
11 (110002708702,ArrayBuffer(1, 8, 6, 27, 12, 3, 1, 7))
12 (110002708704,ArrayBuffer(1, 6, 12, 8, 27, 3, 1, 7))
13 (110002708707,ArrayBuffer(2, 27, 6, 1, 13, 7, 8, 3))
14 (110002708706,ArrayBuffer(2, 1, 13, 8, 27, 3, 6, 7))
15 (110002708708,ArrayBuffer(8, 7, 1, 3, 27, 6, 2, 13))
16
17
18 scala> val seqBaseCovMale= seqAnalysisMale.getCoverageBase
19 map(r=>(SparkSeqConversions.stripSampleID(r._1),r._2) ).groupByKey().cache
20 scala> seqBaseCovMale.count //populate memory cache
21 scala> seqBaseCovMale.filter(r=>r._1 >=posStart && r._1<=posEnd).collect.foreach(println)
22 14/03/28 11:11:37 INFO spark.SparkContext: Job finished: collect at <console>:26, took 0.2592477 s
23 (110002708701,ArrayBuffer(27, 8, 12, 3, 7, 1, 1, 6))
24 (110002708700,ArrayBuffer(1, 3, 1, 6, 27, 8, 10, 7))
25 (110002708703,ArrayBuffer(1, 3, 6, 8, 7, 27, 1, 12))
26 (110002708705,ArrayBuffer(13, 3, 6, 8, 1, 2, 27, 7))
27 (110002708702,ArrayBuffer(1, 8, 6, 27, 12, 3, 1, 7))
28 (110002708704,ArrayBuffer(1, 6, 12, 8, 27, 3, 1, 7))
29 (110002708707,ArrayBuffer(2, 27, 6, 1, 13, 7, 8, 3))
30 (110002708706,ArrayBuffer(2, 1, 13, 8, 27, 3, 6, 7))
31 (110002708708,ArrayBuffer(8, 7, 1, 3, 27, 6, 2, 13))
32
33 scala> seqBaseCovMale.filter(r=>r._1 >=posStart+5 && r._1<=posEnd+5).collect.foreach(println)
34 14/03/28 11:13:11 INFO spark.SparkContext: Job finished: collect at <console>:26, took 0.2585521 s
35 (110002708705,ArrayBuffer(13, 3, 6, 8, 1, 2, 27, 7))
36 (110002708713,ArrayBuffer(3, 6, 2, 8, 13, 27, 1, 7))
37 (110002708707,ArrayBuffer(2, 27, 6, 1, 13, 7, 8, 3))
38 (110002708712,ArrayBuffer(13, 7, 3, 27, 8, 2, 6, 11))
39 (110002708706,ArrayBuffer(2, 1, 13, 8, 27, 3, 6, 7))
40 (110002708709,ArrayBuffer(7, 1, 8, 27, 13, 3, 6, 2))
41 (110002708708,ArrayBuffer(8, 7, 1, 3, 27, 6, 2, 13))
42 (110002708711,ArrayBuffer(1, 2, 7, 3, 8, 13, 6, 27))
43 (110002708710,ArrayBuffer(2, 13, 1, 3, 7, 6, 27, 8))

```

To cache results on disk only:

```

1  scala> import org.apache.spark.storage.StorageLevel
2  scala> val seqBaseCovMale= seqAnalysisMale.getCoverageBase.map(r=>(SparkSeqConversions.stripSampleID(r._1),r._2) )
3  .groupByKey().persist(StorageLevel.DISK_ONLY)
4  scala> seqBaseCovMale.count //populate memory cache
5  scala> seqBaseCovMale.filter(r=>r._1 >=posStart+5 && r._1<=posEnd+5).collect.foreach(println)
6  14/03/28 12:09:38 INFO spark.SparkContext: Job finished: collect at <console>:27, took 0.602364 s

```

```

7 (110002708705, ArrayBuffer(13, 3, 6, 8, 1, 2, 27, 7))
8 (110002708713, ArrayBuffer(3, 6, 2, 8, 13, 27, 1, 7))
9 (110002708707, ArrayBuffer(2, 27, 6, 1, 13, 7, 8, 3))
10 (110002708712, ArrayBuffer(13, 7, 3, 27, 8, 2, 6, 1))
11 (110002708706, ArrayBuffer(2, 1, 13, 8, 27, 3, 6, 7))
12 (110002708709, ArrayBuffer(7, 1, 8, 27, 13, 3, 6, 2))
13 (110002708708, ArrayBuffer(8, 7, 1, 3, 27, 6, 2, 13))
14 (110002708711, ArrayBuffer(1, 2, 7, 3, 8, 13, 6, 27))
15 (110002708710, ArrayBuffer(2, 13, 1, 3, 7, 6, 27, 8))
16
17 seqBaseCovMale.unpersist

```

Reading cached data from disk is obviously slower than from memory (0.6s vs 0.26s in this case) but it is still much faster then recompute everything from the scratch (7.5s)

7.10 Saving results

To summarize the feature saving methods:

Base coverage:

```

1 scala> val baseCov = seqAnalysis.getCoverageBase()
2 scala> seqAnalysis.saveBaseCoverageToFile("test_base.txt")

```

Exon counts:

```

1 scala> val genExonsMapB = sc.broadcast(SparkSeqConversions.BEDFileToHashMap(sc,
2      "/BAM/64MB/aux/Homo_sapiens.GRCh37.74_exons_chr_sort_uniq.bed"))
3 scala> seqAnalysis.getCoverageRegion(genExonsMapB, unionMode=true)
4 scala> seqAnalysis.saveFeatureCoverageToFile("test_exon.txt", Exon)

```

Gene counts:

```

1 scala> val genesMapB = sc.broadcast(SparkSeqConversions.BEDFileToHashMap(sc,
2      "/BAM/64MB/aux/Homo_sapiens.GRCh37.74_genes_chr_merged_swap.bed"))
3 scala> seqAnalysis.getCoverageRegion(genesMapB, unionMode=true)
4 seqAnalysis.saveFeatureCoverageToFile("test_gene.txt", Gene)

```

You can of course use the Apache Spark's way of saving results to file by calling 'saveAsTextFile' or 'saveAsObjectFile'(details). Note that that for using either of these methods you need to specify a path to a file located on your HDFS storage while running an Apache Spark cluster.

7.11 Setting up complete data processing pipelines

Data processing pipelines may consist of a combination of:

- alignment data filtering
- feature building - getting the information on counts, coverage function, exon junctions in given genomic regions.
- analysis of features - statistics, sanity checks, post-processing

The building blocks from each group may be selected according to the needs of the particular analysis.

Examples of the pipelines are:

- Typical pipeline for finding target genes with differential RNA expression or alternative splicing according to a fixed annotation.

Filtering: filtering out the reads with low quality (eg PHRED>30) and large, biologically unjustified (artefactual) alignment gaps (gap in CIGAR > 5000bp).

Feature building: Getting the counts of reads for all the genes and exons in the annotation.

```

1 import pl.elka.pw.sparkseq.seqAnalysis.SparkSeqAnalysis
2 import pl.elka.pw.sparkseq.conversions.SparkSeqConversions
3 import pl.elka.pw.sparkseq.util.SparkSeqRegType._
4
5 /*initiate a SparkSeqJunctionAnalysis object and attach samples to it*/
6 val seqAnalysis = new SparkSeqAnalysis(sc,"/BAM/64MB/derfinder/chrom_Y/M/orbFrontalF1_Y.bam",1,1,1)
7 val caseId = Array(11, 32, 3, 42, 43, 47, 53, 58)
8 for(i<-caseId)
9   seqAnalysis.addBAM(sc,"/BAM/64MB/derfinder/chrom_Y/M/orbFrontalF"+i.toString+"_Y.bam",i,1)
10 /*filter out reads with low quality, i.e. PHRED > 30*/
11 seqAnalysis.filterMappingQuality(_ > 30)
12 /*CIGAR regular expression for filtering out alignment gaps > 5000bp */
13 val cigRegex=
14   "^([0-9]+[MEQISXDHP])+((0*([0-9]{1,3})[1-4][0-9]{3}))?N([0-9]+M[0-9]*[EQISXDHP])*{0,3}[0-9]*[MEQISXDHP]*$".r
15 seqAnalysis.filterCigarString(cigRegex.pattern.matcher(_).matches)
16 /* get feature counts */
17 /*exons*/
18 val genExonsMapB = sc.broadcast(SparkSeqConversions.BEDFileToHashMap(sc,
19 "/BAM/64MB/aux/Homo_sapiens.GRCh37.74_exons_chr_sort_uniq.bed"))
20 val exonCounts = seqAnalysis.getCoverageRegion(genExonsMapB, unionMode=true)
21 /*genes*/
22 val genesMapB = sc.broadcast(SparkSeqConversions.BEDFileToHashMap(sc,
23 "/BAM/64MB/aux/Homo_sapiens.GRCh37.74_genes_chr_merged_swap.bed"))
24 val genesCount = seqAnalysis.getCoverageRegion(genesMapB, unionMode=true)

```

Analysis: DESeq or edgeR according (*Anders et al, Nat Prot 2013*), finding genes with splicing as those that have high variance of its exons median coverage, finding genes with splicing as those that have the high or significant splicing index (*Gardina et al, Gen Biology, 2006*). Gene/exon counts can be easily imported into R environment using RSparkSeq package.

- Analysis of splicing in RNA sequencing using exon junctions (“very simplified cufflinks”)

Filtering: filtering out low quality and large gap alignments as above

Feature building: finding the exon junctions, counting the junctions by genomic location

Analysis: excluding the junctions with low count (eg. 10) finding the genes with high variance of counts of junctions; finding the junctions not hooked on canonical splice start and stop sites.

```

1 import pl.elka.pw.sparkseq.seqAnalysis.SparkSeqAnalysis
2 import pl.elka.pw.sparkseq.conversions.SparkSeqConversions
3 import pl.elka.pw.sparkseq.util.SparkSeqRegType._
4 import pl.elka.pw.sparkseq.junctions.SparkSeqJunctionAnalysis
5
6 /*initiate a SparkSeqJunctionAnalysis object and attach samples to it*/
7 val seqAnalysis = new SparkSeqAnalysis(sc,"/BAM/64MB/derfinder/chrom_Y/M/orbFrontalF1_Y.bam",1,1,1)
8 val caseId = Array(11, 32, 3, 42, 43, 47, 53, 58)
9 for(i<-caseId)
10   seqAnalysis.addBAM(sc,"/BAM/64MB/derfinder/chrom_Y/M/orbFrontalF"+i.toString+"_Y.bam",i,1)
11 /*filter out reads with low quality, i.e. PHRED > 30*/
12 seqAnalysis.filterMappingQuality(_ > 30)
13 /*CIGAR regular expression for filtering out alignment gaps > 5000bp */
14 val cigRegex=
15   "^([0-9]+[MEQISXDHP])+((0*([0-9]{1,3})[1-4][0-9]{3}))?N([0-9]+M[0-9]*[EQISXDHP])*{0,3}[0-9]*[MEQISXDHP]*$".r
16 seqAnalysis.filterCigarString(cigRegex.pattern.matcher(_).matches)
17 /*get junction reads*/
18 val juncAnalysis = new SparkSeqJunctionAnalysis(seqAnalysis)
19 val juncReadsCount = juncAnalysis.getJuncReadsCounts().filter(j=>(j._2 > 10))
20 val juncResults = juncReadsCount
21 .map(j=>(SparkSeqConversions.splitSampleID(j._1._1),SparkSeqConversions.splitSampleID(j._1._2),j._2) )
22 .map(j=>(j._1._1,SparkSeqConversions.idToCoordinates(j._1._2),
23   SparkSeqConversions.idToCoordinates(j._2._2)),j._3 ) )

```

7.12 Optimization hints

- First of all, read carefully Apache Spark Tuning guide;
- Use numeric IDs. Do conversions to Strings or Tuples only for the results presentation;
- Filter as much and as early in your data processing as possible;
- While using position predicates convert your value to Long, not the other way round, e.g.:

```

1 filter(r=>r._1 == position ))

```

not:

```

1 filter(r=>SparkSeqConversions.idToCoordinates(r._1) == ("chrY",2708711) )



---


1 scala> val seqCovBase= seqAnalysis.getCoverageBase.map(r=>(SparkSeqConversions.stripSampleID(r._1),r._2)).cache
2 scala> seqCovBase.count
3 res3: Long = 7156008
4
5 scala> val position=SparkSeqConversions.coordinatesToId(("chrY",2708711))
6 scala> seqCovBase.filter(r=>r._1==position).count
7 14/03/28 17:20:02 INFO spark.SparkContext: Job finished: count at <console>:24, took 0.5758662 s
8 res18: Long = 8
9
10 scala> seqCovBase.filter(r=>SparkSeqConversions.idToCoordinates(r._1)==("chrY",2708711)).count
11 14/03/28 17:20:21 INFO spark.SparkContext: Job finished: count at <console>:22, took 0.7433719 s
12 res20: Long = 8

```

With small dataset the computation time difference is negligible but it is still 30% in this simple case.

8 RSparkSeq

8.1 Introduction

The goal of RSparkSeq⁴ package is to bring the power of SparkSeq to R-project environment.

8.2 Installation instructions

1. Make sure that rJava, jvmr and multicore R packages are installed.
Besides, all SparkSeq dependencies should be met.

If you come across any error while installing rJava package, first of all check if JAVA_LD_LIBRARY_PATH and LD_LIBRARY_PATH are set correctly, e.g. like this (exact path JVM and R may be different in your system):

```

1 export JAVA_LD_LIBRARY_PATH=$JAVA_LD_LIBRARY_PATH:/usr/lib/jvm/java-6-oracle/lib:/usr/lib/jvm/java-6-oracle/jre/lib/:
2 /usr/lib/jvm/java-6-oracle/jre/lib/amd64/server/
3 export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$JAVA_LD_LIBRARY_PATH:/usr/lib/R/lib

```

2. Get the SparkSeq “fat jar” from project’s Maven repo if you use HDFS in 1.2.1 version. If have other HDFS version you need to prepare it by yourself:

```

1 git clone https://bitbucket.org/mwiewiora/sparkseq.git
2 cd sparkseq/sparkseq-core

```

Edit build.sbt file and set your HDFS version, e.g. for Hadoop-2.2.0 it would look like this:

```

1 val DEFAULT_HADOOP_VERSION = "2.2.0"

```

and then you are ready to create your assembly:

```

1 sbt assembly

```

3. Find the path where jvmr package is installed, run R and enter these commands:

```

1 library(jvmr)
2 path.package("jvmr")
3 [1] "/home/spark/R/x86_64-pc-linux-gnu-library/3.0/jvmr"

```

4. Copy assembly jar from point 2. into *java* directory in *jvmr* package path.
5. Clone RSparkSeq repository and install the package:

⁴<https://bitbucket.org/mwiewiora/rsparkseq/>

```

1 git clone https://bitbucket.org/mwiewiorka/rsparkseq.git
2 R CMD build rsparkseq
3 R CMD INSTALL RSparkSeq_0.01.0.tar.gz

```

8.3 Getting started

RSparkSeq is right now under development but some functionalities such as connecting do SparkSeq, creating SparkSeqAnalysis objects and feature(gene, exon) have been already implemented.

8.3.1 Establishing connection to SparkSeq

To establish a connection to SparkSeq running of top of Apache Spark deployed using Apache Mesos you can create a RSparkContext object as follows:

```

1 rcont<-RSparkContext(master="mesos://sparkseq001.cloudapp.net:5050",
2   executor="/frameworks/spark/0.9.0/spark-0.9.0-incubating-hadoop_1.2.1-bin.tar.gz",
3   sparkJar="sparkseq-core-assembly-0.1-SNAPSHOT.jar",sparkMesosCoarse=TRUE, debug=TRUE)

```

8.3.2 Creating RSparkSeqAnalysis object

To create a RSparkSeqAnalysis object and attach samples from SparkSeq multisample analysis tutorial7 you can run the following code:

```

1 seqAnalysis<-RSparkSeqAnalysis(rcont,"/BAM/64MB/derfinder/chrom_Y/M/orbFrontalF1_Y.bam",1,1,1,debug=TRUE)
2 samplesID<-c(11, 32, 3, 42, 43, 47, 53, 58)
3 for(i in samplesID) addBAMfile(seqAnalysis,c(
4   paste("/BAM/64MB/derfinder/chrom_Y/M/orbFrontalF",as.character(i) , "_Y.bam",sep=""),i)
5 )

```

8.3.3 Computing feature counts

8.3.3.1 Genes

```

1 regionHashMap(seqAnalysis) <-" /BAM/64MB/aux/Homo_sapiens.GRCh37.74_genes_chr_merged_swap.bed"
2 genes<-geneCounts(seqAnalysis)
3 > genes[1:5,]
4   Feature Sample_1 Sample_3 Sample_11 Sample_32 Sample_42 Sample_43 Sample_47 Sample_53 Sample_58
5 ENSG00000012817    8720    9103    8585    7375    5387    5835    7669    8175    7258
6 ENSG00000067048   6321    8647    5325    5598    3468    4140    4376    6827    4335
7 ENSG00000067646   1354    1499    1691    889     977     558     872     1128    1004
8 ENSG00000092377   75      48      31      61      29      10      31      26      42
9 ENSG00000099715   291     418     158     229     42      128     125     145     303

```

8.3.3.2 Exons

```

1 regionHashMap(seqAnalysis) <-" /BAM/64MB/aux/Homo_sapiens.GRCh37.74_exons_chr_sort_uniq.bed"
2 exons<-exonCounts(seqAnalysis)
3 > exons[1:5,]
4   Feature Sample_1 Sample_3 Sample_11 Sample_32 Sample_42 Sample_43 Sample_47 Sample_53 Sample_58
5 ENSE00000652498    186     251     135     168     131     140     100     198     149
6 ENSE00000652501    148     211     136     156     130     83      140     230     171
7 ENSE00000652502    207     383     173     198     83      129     188     176     169
8 ENSE00000652503    215     384     182     265     93      136     177     161     135
9 ENSE00000652506    112     196     87      185     73      61      120     101     103

```

A Data sets used in tests

A.1 RNA-sequencing experiment

Reference genome: Equus_caballus.EquCab2

Sequencing performed: Illumina HiSeq 2000, RNA whole transcript sequencing, unpaired, unstranded, polyA protocol, alignment: Tophat 2.0.4

BAM filename	Number of reads	Mean reads length	Size in [MB]
./Fam1/Case/Sample_25_sort.bam	8622606	101	627
./Fam1/Case/Sample_38_sort.bam	19357579	101	1411
./Fam1/Case/Sample_39_sort.bam	14087644	101	913
./Fam1/Case/Sample_42_sort.bam	18824924	101	1318
./Fam1/Case/Sample_44_sort.bam	9651030	101	697
./Fam1/Case/Sample_45_sort.bam	22731556	101	1538
./Fam1/Case/Sample_47_sort.bam	15975604	101	1164
./Fam1/Case/Sample_53_sort.bam	17681528	101	1267
./Fam1/Control/Sample_26_sort.bam	16323269	101	1129
./Fam1/Control/Sample_56_sort.bam	18408612	101	1293
./Fam1/Control/Sample_74_sort.bam	15934054	101	1147
./Fam1/Control/Sample_76_sort.bam	22329258	101	1447
./Fam1/Control/Sample_77_sort.bam	14788631	101	1051
./Fam1/Control/Sample_83_sort.bam	14346120	101	1020
./Fam1/Control/Sample_94_sort.bam	16693869	101	1194
./Fam2/Case/Sample_100_sort.bam	23159345	101	1569
./Fam2/Case/Sample_111_sort.bam	13720969	101	904
./Fam2/Case/Sample_29_sort.bam	9854977	101	775
./Fam2/Case/Sample_36_sort.bam	27592119	101	1849
./Fam2/Case/Sample_52_sort.bam	13604670	101	941
./Fam2/Case/Sample_55_sort.bam	12516180	101	889
./Fam2/Case/Sample_64_sort.bam	18952789	101	1332
./Fam2/Case/Sample_69_sort.bam	13213424	101	914
./Fam2/Control/Sample_110_sort.bam	10132735	101	754
./Fam2/Control/Sample_30_sort.bam	12921437	101	934
./Fam2/Control/Sample_31_sort.bam	15063095	101	1067
./Fam2/Control/Sample_51_sort.bam	16567508	101	1184
./Fam2/Control/Sample_54_sort.bam	26612475	101	1740
./Fam2/Control/Sample_58_sort.bam	26030133	101	1737
./Fam2/Control/Sample_63_sort.bam	22403075	101	1604
./Fam2/Control/Sample_91_sort.bam	12977072	101	908
./Fam2/Control/Sample_99_sort.bam	8676735	101	649

A.2 Multi-amplicon experiment

Reference genome: Hsapiens.UCSC.hg19

Sequencing performed: Ion Torrent, DNA, AmpliSeq multi amplicon sequencing, Comprehensive Cancer Panel (design, 4477685_CCP), TorrentServer Alignment plugin (v3.6.56201)

BAM filename	Number of reads	Mean reads length	Size in [MB]
Sample_012.bam	73397856	110	29952
Sample_012_scaled.bam	27524196	110	11232

B Additional literature

The list of papers that have influenced the work on SparkSeq, but could not have been mentioned in the main manuscript due to the limited space:

- Hong, D. et al. (2012). FX: an RNA-Seq analysis tool on the cloud. *Bioinformatics*, 28(5), 721–723.
- Langmead, B. et al. (2010). Cloud-scale RNA-sequencing differential expression analysis with Myrna. *Genome biology*, 11(8), R83.
- Li, H. and Homer, N. (2010). A survey of sequence alignment algorithms for next-generation sequencing. *Briefings in Bioinformatics*, 11(5), 473–483.
- Okoniewski, M. J. et al. (2013). Precise breakpoint localization of large genomic deletions using PacBio and Illumina next-generation sequencers. *BioTechniques*, 54(2), 98–100.
- Nordberg, H. et al. (2013). BioPig: a Hadoop-based analytic toolkit for large-scale sequence data. *Bioinformatics*, 29(23), 3014–3019.
- O'Connor, B. D. et al. (2010). SeqWare Query Engine: storing and searching sequence data in the cloud. *BMC bioinformatics*, 11(Suppl 12), S2.
- Roy, A. et al. (2012). Massive genomic data processing and deep analysis. *Proceedings of the VLDB Endowment*, 5(12), 1906–1909.
- Schumacher, A. et al. (2013). Scripting for large-scale sequencing based on hadoop. *EMBnet.journal*, 19(A).
- Taylor, R. C. (2010). An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics. *BMC bioinformatics*, 11(Suppl 12), S1.
- Zaharia, M. et al. (2012). Fast and interactive analytics over Hadoop data with Spark. *USENIX*, vol 37, no 4
- Zou, Q. et al. (2013). Survey of mapreduce frame operation in bioinformatics. *Briefings in Bioinformatics*.

It is also important now to mention the technical report of Massie et al., as a set of standards complementary and useful in the future for SparkSeq. However it was placed on the internet in parallel with the initial submission of SparkSeq paper and thus was not influencing the work on its design and implementation:

Massie et al (2013) , ADAM: Genomics Formats and Processing Patterns for Cloud Scale Computing, Technical Report No. UCB/EECS-2013-207, University of California, Berkeley



Original article

Benchmarking distributed data warehouse solutions for storing genomic variant information

Marek S. Wiewiórka¹, Dawid P. Wysakowicz¹, Michał J. Okoniewski²
and Tomasz Gamin^{1,3,*}

¹Institute of Computer Science, Warsaw University of Technology, Nowowiejska 15/19, Warsaw 00-665, Poland, ²Scientific IT Services, ETH Zurich, Weinbergstrasse 11, Zurich 8092, Switzerland and ³Department of Medical Genetics, Institute of Mother and Child, Kasprzaka 17a, Warsaw 01-211, Poland

*Corresponding author: Tel.: +48693175804; Fax: +48222346091; Email: tgamin@ii.pw.edu.pl

Citation details: Wiewiórka, M.S., Wysakowicz, D.P., Okoniewski, M.J. et al. Benchmarking distributed data warehouse solutions for storing genomic variant information. *Database* (2017) Vol. 2017: article ID bax049; doi:10.1093/database/bax049

Received 15 September 2016; Revised 4 April 2017; Accepted 29 May 2017

Downloaded from https://academic.oup.com/database/article/doi/10.1093/database/bax049/3953981 by guest on 01 May 2021

Abstract

Genomic-based personalized medicine encompasses storing, analysing and interpreting genomic variants as its central issues. At a time when thousands of patientss sequenced exomes and genomes are becoming available, there is a growing need for efficient database storage and querying. The answer could be the application of modern distributed storage systems and query engines. However, the application of large genomic variant databases to this problem has not been sufficiently far explored so far in the literature. To investigate the effectiveness of modern columnar storage [column-oriented Database Management System (DBMS)] and query engines, we have developed a prototypic genomic variant data warehouse, populated with large generated content of genomic variants and phenotypic data. Next, we have benchmarked performance of a number of combinations of distributed storages and query engines on a set of SQL queries that address biological questions essential for both research and medical applications. In addition, a non-distributed, analytical database (MonetDB) has been used as a baseline. Comparison of query execution times confirms that distributed data warehousing solutions outperform classic relational DBMSs. Moreover, pre-aggregation and further denormalization of data, which reduce the number of distributed join operations, significantly improve query performance by several orders of magnitude. Most of distributed back-ends offer a good performance for complex analytical queries, while the Optimized Row Columnar (ORC) format paired with Presto and Parquet with Spark 2 query engines provide, on average, the lowest execution times. Apache Kudu on the other hand, is the only solution that guarantees a sub-second performance for simple genome range queries returning a small subset of data, where low-latency response is expected, while still offering decent performance for running analytical queries. In summary, research

and clinical applications that require the storage and analysis of variants from thousands of samples can benefit from the scalability and performance of distributed data warehouse solutions.

Database URL: <https://github.com/ZSI-Bio/variantsdwh>

Introduction

Variant information in genomic-based personalized medicine and biomedical research

In the current era of high-throughput sequencing, the reliable and comprehensive analysis of genomic variant data has become a central task in many clinical and research applications related to precision medicine. Joint analysis of such data from thousands of samples collected under large scale sequencing projects, such as Exome Sequencing Project (ESP, <http://evs.gs.washington.edu/EVS/>), The Atherosclerosis Risk in Communities Study (1), Centers for Mendelian Genomics (2), UK10K (3), The Cancer Genome Atlas (4) provides a detailed and medically useful insight into molecular basis of many genetic conditions.

Although a plethora of statistical methods (e.g. for variant prioritization (5–9) or variant association (10–14) was developed, there is a lack of tools that allow researchers to perform *ad hoc*, unrestricted queries on large data sets. Such tools should be powerful enough to deal with population-scale sequencing data that will be generated by such large-scale projects as Genomic England's '100 000 Genomes Project' (<http://www.genomicsengland.co.uk/the-100000-genomes-project/>) or Precision Medicine Initiative (<http://www.nih.gov/precision-medicine-initiative-cohort-program>) announced by the US administration, which aim in sequencing of at least 1 million Americans.

The early attempts of applying big data technologies to interactive analyses of sequencing datasets were focused on providing the end user with an API (application programming interface) in Pig (15) or Scala (16) languages and integration with the existing bioinformatics file formats using middleware libraries like Hadoop-Binary Alignment Map (BAM) (17). Those approaches while very flexible, but impose imperative programming paradigm which assume that the end user explicitly declares query execution plans. This is not suitable for many researchers and data scientists who are not experts in distributed computing programming at the same time.

Recently, several emerging technologies such as big data query engines offering SQL (Structured Query Language) interfaces like Apache Impala (<http://impala.io/>), Apache SparkSQL (<https://spark.apache.org/>), Presto (<https://prestodb.io/>), Apache Drill (<https://drill.apache.org/>) made

it possible to adapt declarative paradigms of programming to analysing very large datasets. Finally, big data multi-dimensional analytics cube solutions such as Apache Kylin (<http://kylin.apache.org/>) can significantly speed up SQL queries by storing already pre-aggregated results in a fast noSQL database (e.g. Apache HBase).

Efficient *ad hoc* data analysis has been already for years a main goal of OLAP (Online Analytical Processing) solutions design based upon data warehouses. In the case of genomic OLAP systems, the end-users are clinicians and medical researchers. Both groups have different needs and expectations towards data analysis, still those are not mutually exclusive. On the clinical level it is important to find knowledge about variants in well-known genes and to compare the variant information of newly sequenced patients against a larger knowledge base. Here the *ad hoc* question may have the form e.g. 'tell me what diseases and phenotypes may have a patient having a specific set of variants'. On the research side, the queries are focused on unsupervised searches that combine sets of variants, sets of genomic intervals and phenotypic information. This is often about getting the estimates of specific measures of aggregates, e.g. 'tell me which phenotype has its related genes with an over-represented amount of variants'.

Current solutions for analysing sequencing data using SQL and storing genomic variant information can be divided into two categories. First group of tools tries to take advantage of classic single-node RDBMS (Relational Database Management System) like MySQL (18, 19) or newer analytical with column-oriented store like MonetDB (20). Those solutions are able to provide flexibility and very good ANSI SQL conformance and reasonable performance [(21) while working on datasets that are already pre-aggregated (e.g. using Apache Pig in case of (19)] or limited in size [500 MB reported by (20)]. On the other hand, they do not offer horizontal scalability, efficiently compressed store and high availability features out of the box.

The other group of prototypes focuses on providing distributed and scalable bioinformatics file formats equivalent to well-known ones such as BAM or Variant Calling Format (VCF). The major example of such an approach is ADAM (Avro Data Alignment Map) formats family (22) using Avro (<https://avro.apache.org/>) data serialization and

Parquet (<https://parquet.apache.org/>) columnar data representation. These files can be processed using popular query engines like Apache Impala, Apache Spark or Presto. Such a modular approach where storage is not tightly connected to a specific computing framework opens up possibility of choosing freely both a query engine and file format that provides the best performance.

The goal of this prototyping study is to provide hints on combining both approaches in order to create scalable and flexible data warehouse infrastructures for genomic population-scale studies at single sample and variant granularity. For this purposes, the benchmarking suite consisting of data generator, data model and set of queries has been proposed. The benchmarking results of this project are intended to point out the future directions of work on genomic variant data warehouse biobanks. Such database study may be needed in all the areas of application of genomic systems: research, medical and commercial.

Biomedical issues that require *ad hoc* variant data analysis

Accurate detection and interpretation of genomic variation obtained from next generation sequencing data became the central issues in the precision medicine and human genetics research studies. To enable a comprehensive and efficient variant analysis the storage and query engines should allow to run a wide range of *ad hoc* queries providing the answer to the most relevant biomedical questions.

Variant prioritization. Although many methods have been proposed to evaluate variant pathogenicity (23), the allele frequency measured in control populations remains as one of the best variant effect predictors (24). Most of the disease-causing variants are absent or rarely observed in general population but their allele frequency may vary among ethnic groups. Therefore, it is important to distinguish population-specific polymorphisms (likely benign) from real pathogenic variants that are rare enough in every sub-population represented in the control data.

Publicly available databases (1000 genomes, ESP, ExAC) report variant frequencies for a small set of pre-defined ethnic groups (e.g. Europeans, Africans, Asians). Still, no information about variant frequencies in smaller sub-populations (e.g. on the country/state/county level) can be found in them. There are many reasons of not reporting this potentially useful data. First, detailed information about origin of samples included in these studies was not always collected or was not available. Second, the number of individuals in other sub-populations was too small so reporting allele frequency was not justified. Finally, storing allele frequency in the VCF file for every possible level of granularity of population structure would become impractical.

It can be expected that first two issues will be resolved in the near future when more good quality data sets will become available. Still, the pre-computing, updating and storing of such high dimensional allele frequency information will remain a challenge. Although testing the prototype, the main aspect taken into account was the performance of our variant data warehouse in calculating variants' allele frequencies for various subsets of samples. In particular, allele frequencies have been computed for each of four major ethnic groups and for each of 181 countries represented in our simulated data set.

Masking genomic regions with excess of rare variants. Disease-causing variants are not uniformly distributed across the genome (5). They are often clustered in certain parts of the gene, such as selected exons or protein domains, as those functional elements of the genome are more likely to be protein-coding. These regions are usually characterized by a depletion of rare variants in control databases (25, 26). Analogously, an excess of rare variants indicates tolerant, less critical regions in which one should not expect disease-causing mutations. Filtering variants located within these commonly mutated regions reduces the number candidates and therefore can improve the final interpretation (27, 28).

The focus of the benchmarking was the ability of the data warehouse to compute the cumulative frequencies of rare, predicted deleterious variants in different types of genomic regions such as exons, genes and cytogenetic bands. This information can be further utilized to determine genomic intervals of higher than expected mutational rate (29). It is worth noting that the same calculations can be further repeated for any subset of samples, e.g. individuals from selected ethnic groups or countries.

Association tests. Multiple statistical methods have been developed to support novel disease gene discoveries in large case-control analysis of sequencing data (10–14). Classical GWAS approaches aim at identifying single variants, for which allele frequency differ significantly between cases and controls. The main drawback of these methods is a lack of power while dealing with very rare variants (30). To overcome this issue, various aggregation tests have been proposed. They analyse the cumulative effects of multiple variants in a gene or a genomic region, either by collapsing information from multiple variants into single score [Burden test (10–12) or evaluating aggregated score test statistics of individual variants, such as C-alpha (13) or SKAT (14)].

An important issue for aggregated tests is selecting an appropriate subset of variants to be tested for association (30). The allele frequency cutoff is usually determined using the information on disease prevalence and expected inheritance model. In order to refine further the subset of

variants one can use prediction scores to select the most damaging mutations. However, despite many existing prediction algorithms and a variety of filtering strategies there is no single solution that fits to all studies.

Some types of queries have been run to assess the efficiency of our variant warehouse in performing customized region-based association tests. In particular, the search was done for genes or exons enriched for rare deleterious variants in selected disease populations.

Investigating the depth of coverage. Coverage statistics obtained from a large population of samples can be used in many ways, including prediction of copy number variants (CNVs) (31, 32) and detection of regions of poor or variable coverage. CNVs are of great importance in clinical investigation because they often allow to explain patient's phenotype. On the other hand, an information about poorly covered regions can be used to improve the results of association studies (33).

Interactive variant browsing. To achieve the clinical applicability of genomic variant knowledge and for real bench-to-bedside impact of personalized medicine, it is necessary to provide clinicians and medical researchers with user-friendly tools for flexible querying and real-time browsing variant information for a currently investigated patient. Tasks such as phenotypic or ontology-based searches, comparing large knowledge bases with local sequenced biobanks of patients or finding associations between drug response and variants must certainly have the basis in the efficient *ad hoc* variant database queries. Having efficient querying for large variant databases with convenient data management interfaces may convince clinicians to use more of the accumulated genomic knowledge in their daily practice and will have in consequence beneficial influence on patients and therapies.

State-of-the-art techniques for distributed data processing

To construct the benchmarks for genomic variant database it is necessary to systematize the current technologies, formats and software tools used in this area. Those relevant aspects are listed with short descriptions below.

File formats and storage engines

Apache ORC (<https://orc.apache.org>) and Apache Parquet are the most advanced type-aware columnar file formats used in the Hadoop Ecosystem stored using Hadoop File System (HDFS). Both exhibit many design similarities such as pluggable data compression mechanism (e.g. Snappy, gzip/Zlib or LZO), data type specific encoders and support for nested data structures. Apache ORC and Parquet are also widely adopted and many distributed query engines

provide support for both, including Apache Spark, Apache Hive and Presto. ORC introduces also a lightweight indexing that enables skipping of complete blocks of rows that do not satisfy the query predicates. However, there has been no consensus yet on whether one of them is superior in terms of performance. Recent studies (34) suggest that ORC might be significantly faster (ca. 1.3–5.8×). Apache Kudu (35) is a novel open source storage engine for structured data which supports low-latency random access together with efficient analytical access patterns. It can be run in parallel with the existing HDFS installation.

Query engines

Among the modern query engines the differentiating factors are e.g. query performance, memory requirements, compatibility or APIs. In order to choose a subset of engines to be tested with the benchmark data the list of requirements has been prepared:

- ANSI SQL or its dialect as a querying language,
- ODBC/JDBC availability provides interoperability with analytical tools like R, visualization and reporting solutions,
- I/O operations with HDFS file system,
- support for both popular columnar storage: Apache ORC and Parquet,
- support for Apache YARN (Yet Another Resource Negotiator) to provide easy and efficient resource sharing among different jobs running on a cluster the same time,
- support for Hive metastore to provide an abstraction layer over physical storage model details,
- at least basic data access authorization.

Big data query engines can be divided into four main categories. Historically, the first group introduced an implementation of MapReduce paradigm for executing SQL queries and despite the fact that offers in many cases poor performance it is still widely used because of its maturity and stability. Directed Acyclic Graph (DAG)-based category that is a natural successor of the legacy MapReduce approach is currently under active development and seems to become the most popular nowadays. MPP-like engines that has its origin in dedicated analytical appliances (e.g. Netezza, Greenplum, Teradata). The last category are OLAP cubes solutions that store pre-computed results (aggregates) using distributed storage systems. The query engines that have been initially evaluated as candidates for including in the benchmark are as follows:

Hive is data warehouse software that enables querying and managing large datasets in a distributed storage. It provides a metastore which can keep the information on

data specific parameters such as tables, schema, file format or location of the files. Hive can be operated with HiveQL, highly similar to ANSI SQL. There are various execution engines where HiveQL queries can be run, such as MapReduce, Tez or Spark. Among those, MapReduce is the only execution engine that is supported by all Hadoop distributions (i.e. Hortonworks and Cloudera) and therefore only this engine was included in our benchmark.

Hive on MapReduce Hive initially had used MapReduce as the execution engine. MR introduced the paradigm (36) of writing distributed algorithms using two phases: map and reduce. Hive transforms each query into multiple stages consisting of both phases. In case of MapReduce each stage is run independently with sub-results persisted which may lead to IO overhead.

SparkSQL Apache Spark was designed to solve similar problems as Apache Tez and also utilities the concept of DAG's. It is based on the concept of Resilient Distributed Datasets (37). Spark apart from running directly Hive queries have its own optimizer for HiveQL called Catalyst. It also puts great emphasis on memory usage with project Tungsten that uses off-heap memory. Recent major performance improvement introduced in the Spark 2.x branch called whole stage code generation was a reason for including two Spark releases (branch 1.6.x—still widely used and 2.1.x the most recent one as of writing) in the benchmarking procedure.

Apache Presto is a project that was not aimed to replace MapReduce as such but to improve interactive analytical queries on large datasets. It allows querying data sources of different sizes from traditional RDBMS and distributed storages like HDFS. Presto also aims to be ANSI SQL compliant, thus it does not support HiveQL. It is a columnar execution engine initially developed at Facebook and supported by Teradata. It can connect to Hive metastore with a connector.

Apache Impala similar to Presto puts a lot more effort on interactive analytics but it has much more limited support for file formats and data sources. Most notably it lacks support for Apache ORC file format. According to benchmark (34) it is however slower than Presto with Apache ORC as storage in terms of wall time.

Apache Kylin is a distributed OLAP cube solution developed upon Hive and HBase software. It provides a web user interface for both logical (dimensional modelling) and physical (noSQL database table storage) design. Cuboids are computed using map-reduce jobs and loaded into key-value store for fast retrieval. Queries that cannot be answered using OLAP cube can be rerouted to Hive for runtime processing.

MonetDB is a parallel, analytical RDBMS with a columnar-oriented data store. Over the years (project was initiated in the 1990s) it has introduced a great number of unique features e.g. CPU-tuned query execution architecture, usage of CPU-caches, run-time query optimizations

just to name a few. An optional SAM/BAM module for processing of sequence alignment data has been also released recently (20).

Major limitations and challenges in the modern distributed database systems

Although distributed computing research area has been developing rapidly for the last 2–3 years, still there are many challenges and limitations that designers and developers of the system should to be aware and which need to be addressed in the future shapes of the software:

- i. cost-based query optimization is still in its infancy when compared with classic RDBMS—there is still very often a need for manual query tuning like table joins reordering or queries reformulation,
- ii. ANSI SQL conformance is often not yet fully satisfied, which leads to situations where one query needs to be customized for each execution engine,
- iii. many analytical/window functions are missing or named differently,
- iv. distributed queries launch overheads—there is still a lot of effort put into providing more interactive user experience as known from classic RDBMS,
- v. engines self-tuning features are also not yet implemented which very often results in manual, time-consuming triaging and tuning on the level of engines as well as single queries,
- vi. in most of the cases the underlying storage layer is either optimized for fast sequential reads or random access patterns (Apache Kudu is an exception) and thus sometimes data need to be duplicated.

Materials and methods

Base data model

In the area of data warehouses many design patterns have been proposed (38, 39) that can be applied in the prototype with some adjustments, specific to the requirements of distributed computing model and query engines. The main issue to be solved is slow joins, which should be preferably replaced with filtering or map-joins. One of the solutions it is to apply the star schema which can lead to executing map-joins when the dimension table can fit into the memory. Figure 1 depicts the star schema of the prototype. Dimension tables are also designed to enable implementation of hierarchies for flexible adjusting of an area of interest, e.g. for the geography dimension one can query over region → subregion → country or for genomic position gene → transcript → exon → chromosome → position.

The ‘fact’ table contains information retrieved from VCF files, such as chromosome, position, reference and alternative

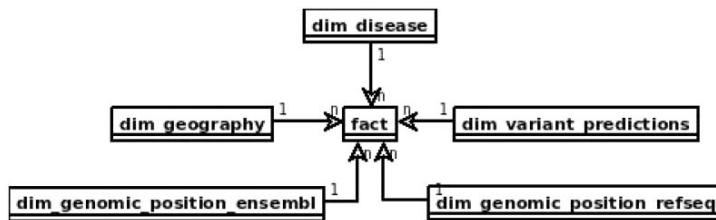


Figure 1. Proposed star schema of the genomic variant data warehouse, with central fact table and tables modelling patients' genotypes and phenotypes and genomic variant annotation to RefSeq and Ensembl.

alleles, depth of coverage, genotype and genotype likelihoods. It also includes references to all the dimension tables.

The table *dim_geography* represents world region of the patient divided into hierarchical areas. That type of patient information may be relevant in case of population genomics studies. From clinical perspective it may help to identify population specific polymorphisms and trace the origin of causative variants, e.g. in epidemiology.

Both *dim_genomic_position_ensembl* and *dim_genomic_position_refseq* cover referential genes and transcript annotations as available in RefSeq and Ensembl databases, respectively. Each record corresponds to a single exon and contains information on its genomic location and associated transcript's data. Exons included in canonical transcripts are indicated by 'iscanonical' flag. The transcripts with 'ismerged' flag, are results of overlapping all transcripts that map to the same HGNC gene symbol.

Table *dim_disease* represents a set of Online Mendelian Inheritance in Man (OMIM) diseases. This table models all the phenotypic and disease information about the patient and sample that can be possibly stored in a genomic data warehouse. This may be extended into a set of data tables, including phenotype ontology and all the clinical parameters relevant for the diseases of patients whose samples are stored. This is the most obvious direction of development of the warehouse structure for practical applications.

The *dim_variant_predictions* contains variant information that is available in dbNSFP database (40) with selected results from some of the major predictors available.

The full definitions of all the tables in the database schema can be found in the project results repository (<https://github.com/ZSI-Bio/variantsdwh>).

Data model optimizations

Base data model organized as a classic star schema is mainly suitable for running queries that require the highest granularity of data but using only a small subset of rows from the fact table. Queries that perform full-table scans over fact table in order to calculate aggregated measures over e.g. geographical items or genomic regions can benefit

from being rewritten to be run over aggregation tables. It can be further optimized by introducing aggregation tables that have been pre-joined with some of the most often used dimensions. This can be particularly beneficial in the case of high or ultra-high cardinality dimensions like *dim_genomic_position* or *dim_variant_prediction*. Last but not least, the base data can be transformed into OLAP cubes storing all aggregates along predefined query patterns for running fast *slice and dice* operations. To address all the needs above, four levels of data storage have been introduced:

- i. raw data of genotype calls (raw)—is a raw, not aggregated fact table with the highest granularity and references to all the dimension tables,
- ii. aggregation tables level (aggr)—storing all variant counts aggregated by countries, exons, diseases, and keeping references to all dimension tables,
- iii. aggregation and full denormalization level (aggr+denorm)—the aggregated fact table with pre-joined dimension tables stored as one table,
- iv. OLAP cube with all aggregates pre-computed and stored in noSQL database (Kylin).

Construction of benchmarks

Cluster infrastructure overview

Hardware. All the test have been run using a 6 node cluster (5 data/processing nodes and 1 used as master and name node). Machines were equipped with 2xE5-2650 CPU resulting in 16 cores/32 threads and 256GB of RAM. Each node had local 6 hard drives in RAID0 (400 GB of disk space) mode with peak throughput around 1.3 GB/s of sequential read speed. Network interconnects allowed stable transfer at around 200MB/s.

Software. Cloudera (CDH) 5.8.2 distribution were installed with Hadoop 2.6.0, Hive 1.1.0, HBase 1.1.1, Kudu 1.0.1 and Zookeeper 3.4.5 as main software components. Versions of other components are summarized in **Table 1**. Please, note that in the text Spark in version 1.6.3 is referred to as Spark 1 and Spark in version 2.1.0 as Spark 2.

Table 1. Query engines comparison

Query engine	Version	ORC	Parquet	Kudu	JDBC/ODBC	YARN	Security
Apache Hive	1.1.0	+	+	-	+	+	+
Apache Spark	1.6.3/2.1.0	+	+	+/-	+	+	+
Presto	0.169	+	+	-	+	+/-	+/-
Apache Impala	2.8.0	-	+	+	+	+	+
Apache Kylin	1.2	+	+	-	+	+	+
MonetDB	11.21.11	N/A	N/A	N/A	+	N/A	+

Query engines

The benchmark measures the performance of four distributed query engines described above, i.e. Apache Hive (MapReduce), Apache Spark (versions 1.x and 2.x), Presto, and Apache Impala. First three engines were tested using two different file formats: ORC and Parquet. Apache Impala, which does not support ORC, was tested in configuration with Parquet and Apache Kudu.

In addition, in the case of queries against aggregation tables, MonetDB database was used as a baseline to compare performance of distributed query engines versus one of the fastest parallel, columnar but still single-node relational databases. Distributed cube OLAP solution—Apache Kylin has been also reviewed to indicate a possibility of reducing query times even more, with the aim of execution time below a single second.

Data warehouse physical data model details

In all the tests queried tables were stored in either ORC or Parquet format with gzip compression and registered as Hive tables in Hive Metastore.

Table 2 presents the details of the physical tables stored in Hive and MonetDB.

In the case of MonetDB only $1 * 10^9$ rows (approximately one-fifth of the dataset) has been loaded to the fact table, while the rest of the tables (aggregation and dimension) were the exact copies of those stored in Hive. It was because of the disk space constraints, as MonetDB does not provide any data compression mechanism. It has been estimated that the total size of fact and dimension tables would exceed 400 GB that was attached as local storage.

The first level (*fact*) table enables running most general queries even for single samples. The next two levels (2 and 3) (*fact_agg_counts* and *fact_agg_counts_dims*) are pre-aggregated by all dimension's foreign keys. Levels 1–3 can be queried using different computing engines (Hive on MapReduce, SparkSQL, Presto) and the data is stored as one of columnar storage such as Apache ORC or Parquet, whereas the fourth level is implemented using Apache Kylin which with HBase as a storage.

The size of the Kylin OLAP cube was 47.6 GB and it took ~6.5 h to build it. In case of Apache Kudu there was

a need to add an artificial primary key in case of aggregated tables (levels 2 and 3).

Formulation of queries and testing

The performance of the variant, data warehouse has been tested using 12 types of queries that correspond to biomedical issues discussed in the previous section. The detailed description of queries and engine-specific versions of SQLs are available in the results repository.

- i. Q1: Allele frequencies—breakdown by geographical region. For every variant the set of population specific allele frequencies corresponding to four continents (subquery A) or 181 countries (subquery B) is calculated. This type of data can be used to identify and flag common polymorphisms observed only in selected populations.
- ii. Q2: Cumulative frequencies per genomic interval. The number of distinct rare (allele frequency <1 %), deleterious [predicted as damaging by Functional Analysis through Hidden Markov Models (FATHMM)] variants and their cumulative allele frequencies are computed for every single transcript (subquery A) or exon (subquery B) in the human genome. This information may help to detect the commonly mutated regions that could be masked in the clinical investigation of patients variant data.
- iii. Q3: Enrichment of variants in disease population. The aggregated variant counts are computed for selected subpopulation of disease patients for every transcript (subquery A) or exon (subquery B) in the human genome. These queries provide substrates for aggregation tests and can be easily used to identify genes or exons with excess of damaging variants in disease population.
- iv. Q4: Distribution of variant in disease population across intervals. Minimum, 25th percentile, median, 75th percentile and maximum depth of coverage across all samples is computed for every transcript (subquery A) or exon (subquery B) in the human genome. These queries, allow to locate poorly or variably covered regions that may be a cause of inflation of false positive values in association studies.

Table 2. Physical data model properties

Table	Rows	Columns	ORC-Zlib size	PQ-Zlib size	Kudu-Zlib size	MonetDB size
fact	4827005513	14	24.9	21.7	76.3	69.6
fact_agg_counts	230380448	20	3.1	3.2	8.7	25.7
fact_agg_counts_dims	230380448	67	10.9	10.1	34.8	72.1
dim_gen_pos_ens	1519361	12	0.0097	0.0122	0.366	0.064
dim_gen_pos_rs	725811	12	0.0064	0.007		0.041
dim_geography	249	7	0.000004	0.000004	0.000012	0.007
dim_disease	7569	3	0.000104	0.000113	0.000348	0.003
dim_variant_predict	391271826	28	10.8	11.8	19.1	54.2

- v. Q5–Q12: Set of queries on the fact table. In addition to the eight complex queries (four query families) described above, the benchmark includes a set of eight queries that act on the fact table only, i.e. without joins with dimension tables. In particular, ‘Q5’ returns the list of all variants from the single sample and the given genomic region. This simulates a typical scenario in which clinician/analyst explore patientan variants in the gene/region of interest. It is important that such a simple genome range queries are processed efficiently, possibly providing an answer in less than a few seconds. Remaining queries can be useful in other types of exploratory analysis or in a process of quality control. These queries return:
- a. Q6: the number of variantso occurrences corresponding to the same substitution type (e.g. C > G) in all samples,
 - b. Q7: the number of distinct variants per chromosome observed in all samples,
 - c. Q8: the number of variants for which a ratio of variant to total reads exceeds 90% in the given sample,
 - d. Q9: the ratio of heterozygous to homozygous variants on X chromosome in the given sample,
 - e. Q10: the number of variants per sample for a given chromosome,
 - f. Q11: the number of different genes containing variants per sample,
 - g. Q12: the number of variants per chromosome for a given sample.

Test set properties and its generation

The largest publicly available variant datasets (such as ExAC) contain data from >50 000 of samples. Unfortunately, ExAC does not provide sample level genotype data. Downloadable VCFs contain variant allele frequencies across different populations; however, no genotype data for individual samples are reported. To generate dataset that has similar properties to the ExAC one, a

data simulator has been implemented that uses real population-specific allele frequencies extracted from ExAC.

For the purpose of testing, an artificial SNV data set has been generated, simulating 50 000 whole exome sequences. To ensure the actual distribution of genomic variants in geographical populations, the ethnic-specific allele frequencies available in ExAC database have been used. This simulation procedure consists of three steps:

First, every sample is assigned to one of four ethnic groups, i.e. Europeans, Americans, Asians or Africans. Then, samples within an ethnic group are associated with countries, which are randomly selected with respect to relative population sizes.

Subsequently, variants are simulated based on information present in the dbNSFP (40) including chromosomal position, reference/alternative alleles and allele frequencies from ExAC. For every variant in dbNSFP the genotype has been selected with a probability $p(af)$ and corresponding to the proper ethnic group allele frequency:

$$p(af) = \begin{cases} 1 - 2 \cdot af + af^2 & \text{for genotype 0/0} \\ 2 \cdot af \cdot (1 - af) & \text{for genotype 0/1} \\ af^2 & \text{for genotype 1/1} \end{cases}$$

For every genotype generated in previous step, the total depth and allelic depth of coverage have been simulated. For a given position of the variant, the average value of the total number of reads has been fixed, using information reported in ExAC about a mean depth of coverage at this location.

Test procedure automation

SQL-dialect specific version of all the queries have been run using all query engines and in case of aggregated (storage level 2), aggregated and denormalized (storage level 3) also using MonetDB. Queries on storage level 3 have been also tested using the Kylin cube.

To execute queries using selected engines and storage formats there has been prepared a parametrized YAML-based configuration file for each query (see <https://github.com/ZSI-Bio/variantsdwh> for details). This enabled us to automate the process of testing different combinations of query engines and storage formats. Proposed automation framework consists of Scala utility that reads parametrized SQL query in a YAML file (together with additional metadata information) and executes it using a selected execution engine (using a JDBC interface) against tables stored in a desired file format a specified number of times. In order to even further automate the benchmarking process additional shell scripts were developed that execute end to end scenario, e.g. start Spark 1 Thrift server, run queries, stop it, and repeat the same steps with Spark 2. Framework is also shipped with tools that can generate the physical model, populate dimension tables and generate fact table of a given size. Data analyses and visualization step are implemented as a set of R scripts that process CSV output of the benchmark procedures.

In the case of Hive tables stored as ORC or Parquet format gzip/Zlib compression has been used in both cases. Each query from the test set has been run several times and average value has been calculated. Disk buffers at operating system level were purged before each query launch.

Results and discussion

After performing the queries with the selection of database and query engines, a number of conclusions and recommendations can be formulated. The numeric results of the benchmarking tests can be found in the table

File formats and storage engines

Testing both ORC and Parquet file formats revealed that there exist serious differences in their implementations between the computing engines. The same query run using a different file format can slow down even by factor of $\sim 1.5\text{--}2\times$ (see [Table 3](#)). It can be observed that Apache Hive performs better using ORC than Parquet. Apache Spark always favours Parquet format, so does Presto with ORC but here the difference seems to be less obvious (with the exception of queries without join operations where the difference can be significant). The difference in size of the compressed files (using gzip/Zlib algorithm) were comparable, varied depending on a table but in the case of the fact table reached the maximum value of ca. 20%.

In summary, Parquet-based file formats for storing genomic information (as e.g. in ADAM) are a good choice for running complex analytical queries (e.g. Q1–Q4) whereas are not really suitable for fast random access patterns, e.g.

interactive variants browsing for a given sample identifier and genomic position ranges (Q5). When choosing this file format Apache Spark 2 seems to best query engine that in most of the cases outperforms both Apache Spark 1 and Apache Impala. The other option that is currently worth considering is combination of ORC file format with Presto query engine. In most of the cases it is slower in case of the queries on raw data requiring joins with dimension tables but on the other hand is more performant in single table scan operations and data browsing. Neither of these combinations of query engine/file format can compete with Apache Impala/Kudu configuration in terms of fast data browsing that can offer sub-second responses in most of the cases.

Query engines

None of the evaluated computing engines was an obvious winner in all the queries and storage levels (see [Figures 2](#) and [3](#) and [Table 3](#)). The results can be summarized as follows:

- i. Presto is a perfect choice for simpler queries (with fewer or no joins) or smaller tables (e.g. aggregated and denormalized), on the other hand it is also suitable for complex queries with many joins but is slower than Apache Spark 2.
- ii. Apache Hive (MapReduce) is particularly good at complex queries with joins run on large fact tables but still slower than other DAG-based and MPP-like solutions. It does not excel at interactive, simpler queries when it always was slower and the difference even more visible.
- iii. Apache Spark 2 seems to be most general purpose tool in the study. It is suitable for both heavy processing and comparable to Presto when interactivity is of importance. A clear performance boost when compared to Apache Spark 1 was observed - in almost all the test cases it was faster and in some the difference was even $5\text{--}6\times$.
- iv. Apache Kylin whereas is not as flexible as fully fledged query engines with properly designed OLAP cube can be truly interactive tool with sub-second response times.
- v. all of the query engines show superior performance over MonetDB in case of running star-queries over the aggregation tables with an average speedup $\sim 3\text{--}7\times$, in case of queries against aggregated and denormalized tables execution times converged, in a few queries MonetDB proved to be equally fast as Kylin. MonetDB seems to exceptionally well deal with queries run against a single table.
- vi. Apache Impala was the only query engine that was used to access data stored in Apache Kudu. When used

Table 3. Queries average execution times for Apache Hive, Apache Spark, Presto, Impala, MonetDB and Apache Kylin

Query	Level	Hive on MR [s]		Presto[s]		Spark 1[s]		Spark 2[s]		Impala [s]		MonetDB [s]	Kylin [s]
	Format	ORC	Parquet	ORC	Parquet	ORC	Parquet	ORC	Parquet	Parquet	Kudu	Custom	HFile
Q1 _A	raw	648	1548	283	314	572	434	330	280	814	1523		
	aggr	216	212	25	28	55	35	27	23	85	54	207	
	aggr+denorm	123	315	11	19	24	21	15	11	35	83	182	0.32
Q1 _B	raw	710	1603	270	316	230	145	335	219	685	1580		
	aggr	193	195	24	28	39	27	25	22	84	56	44	
	aggr+denorm	141	367	13	19	21	20	17	14	49	90	143	0.85
Q2 _A	raw	386	543	103	172	613	478	143	81	157	976		
	aggr	285	300	34	29	138	79	33	28	51	82	263	
	aggr+denorm	53	71	1.81	6.74	16	14	12	7.16	5.82	8.2	18	1.7
Q2 _B	raw	failed	failed	271	320	721	492	144	79	180	1357		
	aggr	542	680	31	33	144	89	39	27	65	80	321	
	aggr+denorm	51	67	2.24	7.49	18	8.59	12	6.37	6.91	8.64	17	2.5
Q3 _A	raw	423	577	118	196	649	552	139	76	221	1082		
	aggr	294	300	35	36	154	94	44	32	85	82	549	
	aggr+denorm	45	60	1.01	7.25	20	9.62	9.67	5.94	5.43	18	0.5	13
Q3 _B	raw	422	573	113	193	655	427	136	75	233	1074		
	aggr	292	299	39	36	141	82	48	39	89	82	549	
	aggr+denorm	44	62	1.18	8.63	21	11	11	5.47	6.3	11	0.3	14
Q4 _A	raw	171	207	33	66	78	34	47	21	59	516		
	aggr	174	170	12	13	26	17	21	15	25	20	146	
	aggr+denorm	123	142	3.66	6.36	16	14	9.51	9.72	12	18	91	0.26
Q4 _B	raw	160	200	29	67	79	32	42	15	20	524		
	aggr	155	164	11	14	23	16	12	12	3.81	20	157	
	aggr+denorm	56	70	0.71	4.58	12	7.19	5.1	2.6	4.35	24	3	0.91
Q5	raw	64	95	1.01	37	111	26	42	8.49	16	0.33		
Q6	raw	23	49	0.45	24	67	13	31	3.05	11	18		
Q7	raw	73	87	25	55	98	37	49	15	71	308		
Q8	raw	28	76	5	39	66	20	38	3.55	16	0.36		
Q9	raw	75	101	0.69	49	107	22	48	4.61	6.26	0.46		
Q10	raw	58	84	3.86	41	85	19	47	8.82	21	58		
Q11	raw	60	87	3.58	30	59	17	37	8.13	18	27		
Q12	raw	77	100	4.47	25	29	14	28	3	8.78	0.27		

together with Apache Kudu, it was the best combination capable of answering genome range queries (Q5) in a truly interactive way (<1s). It also offered comparable execution times to the Apache Spark 2 and Presto in case of single table queries in many cases.

Data compression

The impact of data compression on table size and query execution times has been tested for two best performing configuration, i.e. Presto with ORC and Spark 2 with Parquet.

Using columnar storage together with gzip/Zlib or Snappy compression can significantly reduce storage requirements (Figure 4A). In comparison to MonetDB the storage space required can be even 5–7× smaller (Table 2). In general data encoding and compression in Parquet is 15% better than ORC in case of Snappy and gzip/Zlib

compression methods. Besides non-compressed data encoded with ORC format can be even three times bigger than non-compressed data encoded with Parquet (Figure 4A).

Comparison of execution times revealed that there are no visible overhead of Snappy data compression on the performance. On the other hand, gzip/Zlib compression may have either positive or negative impact on query execution times, depending on the query (Figure 4B). In case of queries that require full table scans negative impact of gzip/Zlib compression can be observed, e.g. compare execution times for Q7.

Data model optimizations

The use of aggregation tables can result in huge speedup when running queries that require full-table scan to

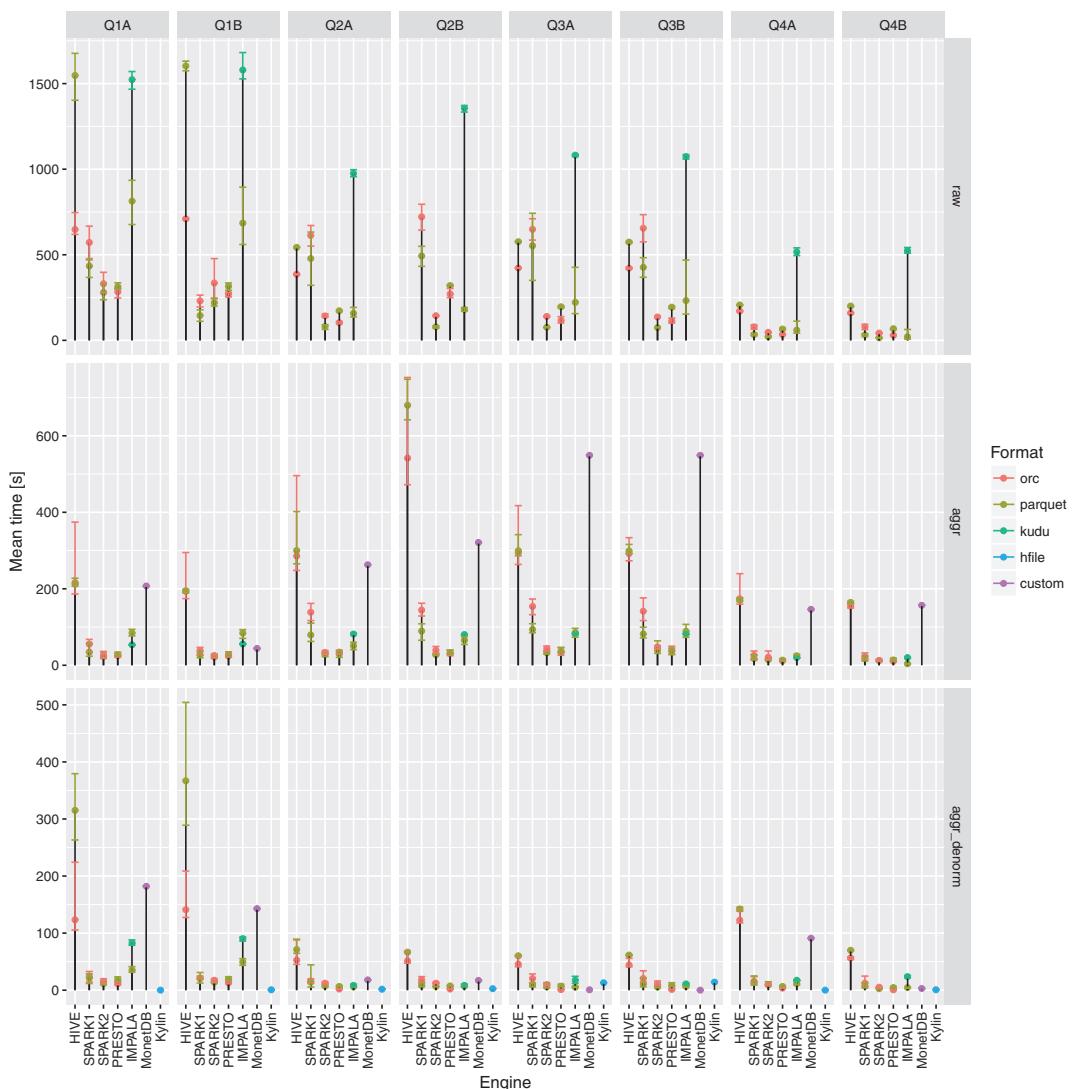


Figure 2. Execution times for all the query engines and file formats for the queries Q1–Q4 over raw, aggregation and denormalized tables with MonetDB as a baseline. For a given configuration (query engine and file format) each query was executed three to five times. Different colors were used to show the average execution times for different file formats. In addition, lower and upper bounds of error bars indicate the minimum and maximum query execution time, respectively.

compute aggregated measures (Figure 5). In the conducted tests it ranged from $6\times$ in case of Hive up to $100\times$ in case of Presto with ORC file format.

Application of OLAP cube

Apache Kylin can offer unbeatable performance at a cost of flexibility when running queries following predefined patterns (i.e. hierarchies, groupings, measures). In most of the cases it proved to be $\sim 5\text{--}20\times$ faster than Presto or

SparkSQL. It can be seen as a ‘coprocessor’ boost component that can offload SparkSQL or Presto by handling predefined but parametrized queries.

Recommendations for genomic data warehouse designers

Designing a scalable performant analytical system that can handle rapidly growing amount of genomic data is by no

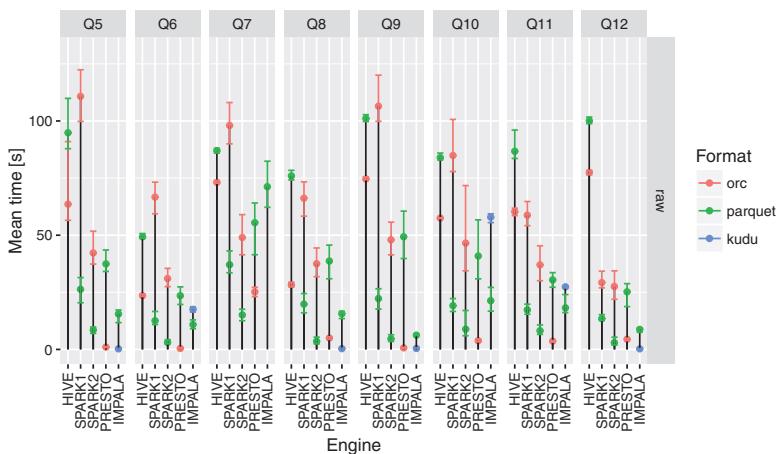


Figure 3. Execution times for all the query engines and file formats for queries Q5–Q12. For a given configuration (query engine and file format) each query was executed between three and five times. Different colors were used to show the average execution times for different file formats. In addition, lower and upper bounds of error bars indicate the minimum and maximum query execution time, respectively.

Downloaded from <https://academic.oup.com/database/article/doi/10.1093/database/bax049/3953981> by guest on 01 May 2021

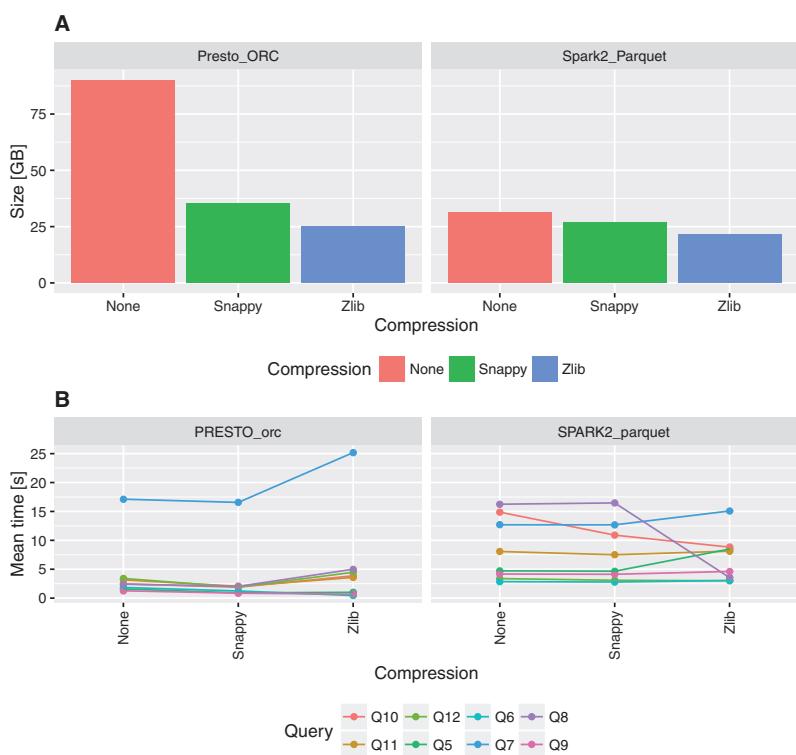


Figure 4. Impact of compression on fact table size and execution times for queries Q5–Q12.

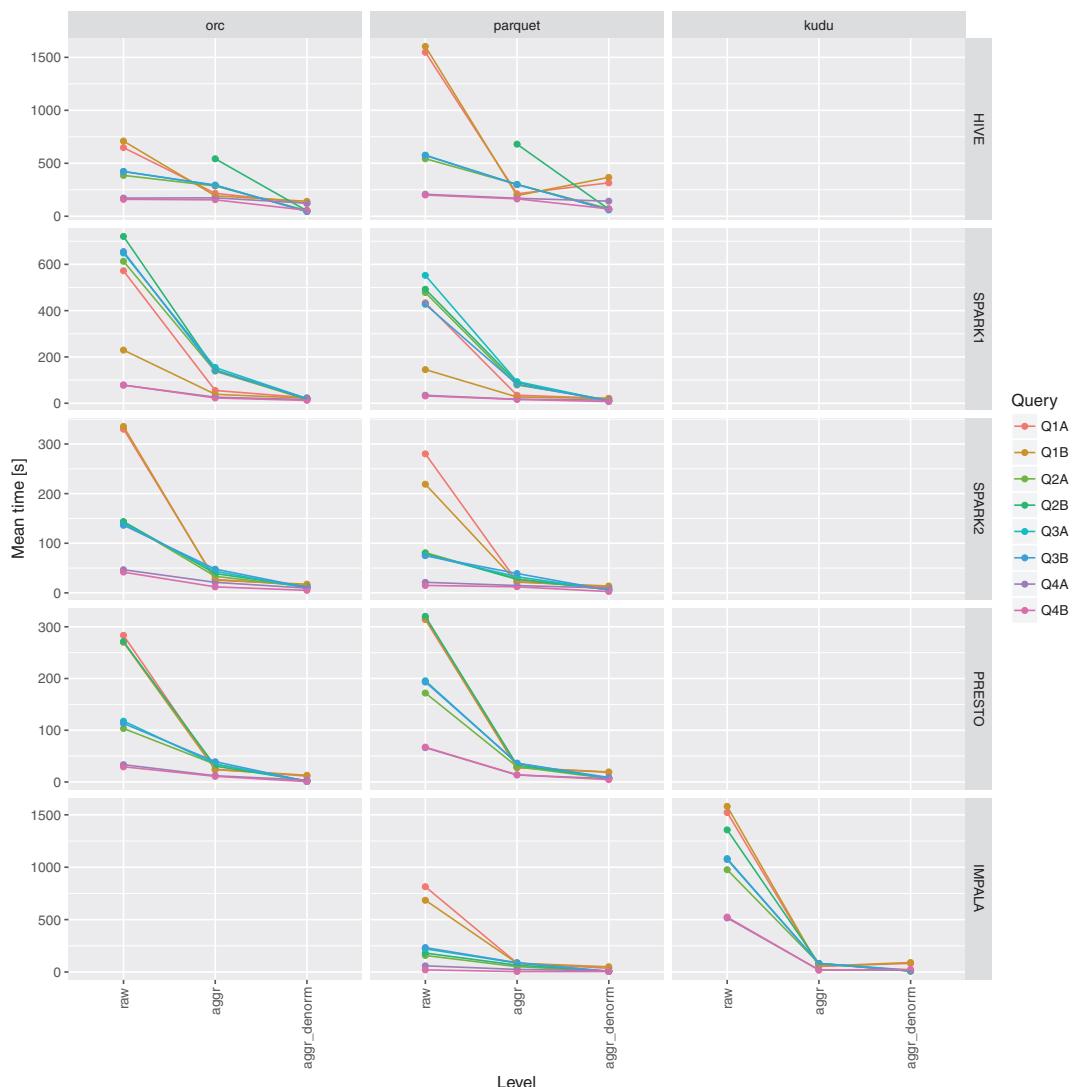


Figure 5. Impact of aggregation and denormalization on query performance for queries Q1–Q4.

means an easy task. In this manuscript a few crucial findings that may be treated as design guidelines have been highlighted:

- There are many competing big data ready file formats, query engines and their combinations can substantially differ in performance characteristics. Moreover, serious performance differences can be observed after upgrade from one version of a tool to another. Furthermore, since all of the discussed solutions are truly distributed, also infrastructure characteristics such as network interfaces and storage systems

throughput can impact the system performance. This is why it is advisable to use benchmarking frameworks prior taking a final decision on the architecture of the genomic data warehouse.

- Another clear finding is that there is no superior combination of query engine and storage format. Besides, in case of data warehouses solution there is a need for at least three kinds of processing: (i) ETL/ELT (extract, transform, load) processes for loading/refreshing tables, materialized views (aggregation tables), (ii) running large scale analytics and finally (iii) random

- records browsing. They all differ in query latency requirements. In case of the first two scenarios, it is acceptable that processing may take longer than a few seconds (e.g. up to few minutes), whereas range queries that are used to populate views of end user interfaces are expected to execute in a fraction of second. Taking into consideration presented results we recommend to use ORC file format together Presto as a tool for running interactive queries and Apache Spark 2 for implementing ETL processes and background queries for answering population scale research questions.
- iii. Distributed machine learning libraries that integrate seamlessly with Apache Spark such as MLlib (<http://spark.apache.org/mllib/>) or Sparkling Water (<https://www.h2o.ai/download/sparkling-water/>) can be recommended for running more sophisticated exploratory analyses.
 - iv. Distributed OLAP cubes solution together with denormalized aggregation tables can serve as acceleration layer suitable for plugging into performant end user interfaces.
 - v. In most of the cases usage of data compression (such as Snappy) is advisable as it substantially reduces storage requirements and do not negatively impact the performance. Since the data are stored in a columnar fashion sorting data by low cardinality columns such as chromosome, reference allele, alternative allele together with a sample identifier can improve the compression ratios even further.
 - vi. All the tested query engines support JDBC/ODBC standards and with some additional effort can execute the same SQL queries. This provides a possibility of relatively easy way of switching between execution engines.

Summary and future directions

This study is intended to point out directions for database designers and bioinformaticians wishing to work on genomic big data currently being produced by sequencing. The computational experiment presented in the paper is an initial proof of the utility of modern columnar databases and query engines to genomic variant data warehouses. At the same time, it has been pointed out in the experiments results that for specific purposes the data structures and queries can be optimized and various query engines are complementary. The development of new distributed systems is an ongoing process, so benchmarks as presented in the paper should be run in the future also for the novel solutions that almost certainly will be developed in the software ecosystems. Such a benchmark can be easily updated,

since the source code for our automated benchmarking framework along with data simulator, complete testing data set, SQL queries and raw results described above are publicaly available at <https://github.com/ZSI-Bio/variantsdwh>.

It needs to be also clearly stated that there is still a room for improvement in terms of the performance of genomic data warehouse solutions. Big data technologies such as Apache Kudu or more recent one—Apache CarbonData (<http://carbondata.incubator.apache.org>) indicates that it might be possible soon to have one storage format that supports OLAP style queries, sequential scans and random access efficiently. Moreover, both technologies allow performant random updates and inserts of data which would be desirable in many cases. Further improvements in vectorized query processing together with better support for Single Instruction Multiple Data extension available in modern CPUs would result in better performance of query engines. Integration of computation engines with hardware accelerators, such as graphic cards (General-purpose computing on graphics processing units), could be beneficial, especially in case of machine learning analyses.

Since the genomic variants datasets are indeed large, the execution time optimization can play significant positive role in personalized medicine research and near future applications of large genomic biobanks. This is not a trivial task, so will require more research and close collaboration between the medical domain experts and creators of modern distributed data processing applications. In particular, knowing the results of this paper, it is highly recommended that a definition of query types and templates is created in advance by the working together of database designers with the experts of clinical genomics and that its performance with particular storage and execution engines is tested with similar benchmarks.

Supplementary data

Supplementary data are available at *Database* Online.

Funding

This work has been supported by the Polish National Science Center grants (Opus 2014/13/B/NZ2/01248 and Preludium 2014/13/N/ST6/01843).

Conflict of interest. None declared.

References

1. The ARIC Investigators. (1989) The Atherosclerosis Risk in Communities (ARIC) Study: design and objectives. *Am. J. Epidemiol.*, 129, 687–702.

2. Chong,J., Buckingham,K.J., Jhangiani,S.N. *et al.* (2015) The genetic basis of mendelian phenotypes: discoveries, challenges, and opportunities. *Am. J. Hum. Genet.*, 97, 199–215.
3. Kaye,J., Hurles,M., Griffin,H. *et al.* (2014) Managing clinically significant findings in research: the UK10K example. *Eur. J. Hum. Genet.*, 22, 1100–1104.
4. Cancer Genome Atlas Research Network. *et al.* (2013) The Cancer Genome Atlas Pan-Cancer analysis project. *Nat. Genet.*, 45, 1113–1120.
5. Davydov,E.V., Goode,D.L., Sirota,M. *et al.* (2010) Identifying a high fraction of the human genome to be under selective constraint using GERP++. *PLoS Comput. Biol.*, 6, e1001025.
6. Adzhubei,I., Jordan,D.M., and Sunyaev,S.R. (2013) Predicting functional effect of human missense mutations using PolyPhen-2. *Curr. Protoc. Hum. Genet.*, 2013, 7–20.
7. Shihab,H.A., Gough,J., Cooper,D.N. *et al.* (2013) Predicting the functional, molecular, and phenotypic consequences of amino acid substitutions using hidden Markov models. *Hum. Mut.*, 34, 57–65.
8. Schwarz,J.M., Cooper,D.N., Schuelke,M. and Seelow,D. (2014) MutationTaster2: mutation prediction for the deep-sequencing age. *Nat. Methods*, 11, 361–362.
9. Vaser,R., Adusumalli,S., Leng,S.N. *et al.* (2016) SIFT missense predictions for genomes. *Nat. Protoc.*, 11, 1–9.
10. Morgensthaler,S. and Thilly,W.G. (2007) A strategy to discover genes that carry multi-allelic or mono-allelic risk for common diseases: a cohort allelic sums test (CAST). *Mut. Res.*, 615, 28–56.
11. Li,B. and Leal,S.M. (2008) Methods for detecting associations with rare variants for common diseases: application to analysis of sequence data. *Am. J. Hum. Genet.*, 83, 311–321.
12. Madsen,B.E. and Browning,S.R. (2009) A groupwise association test for rare mutations using a weighted sum statistic. *PLoS Genet.*, 5, e1000384.
13. Neale,B.M., Rivas,M.A., Voight,B.F. *et al.* (2011) Testing for an unusual distribution of rare variants. *PLoS Genet.*, 7, e1001322.
14. Wu,M., Lee,S., Cai,T. *et al.* (2011) Rare-variant association testing for sequencing data with the sequence Kernel association test. *Am. J. Hum. Genet.*, 89, 82–93.
15. Schumacher,A., Pireddu,L., Niemenmaa,M. *et al.* (2014) SeqPig: simple and scalable scripting for large sequencing data sets in Hadoop. *Bioinformatics*, 30, 119–120.
16. Wiewiórka,M.S., Messina,A., Pacholewska,A. *et al.* (2014) SparkSeq: fast, scalable, cloud-ready tool for the interactive genomic data analysis with nucleotide precision. *Bioinformatics*, 30, 2652–2653.
17. Niemenmaa,M., Kallio,A., Schumacher,A. *et al.* (2012) Hadoop-BAM: directly manipulating next generation sequencing data in the cloud. *Bioinformatics*, 28, 876–877.
18. Ameur,A., Bunikis,I., Enroth,S. and Gyllensten,U. (2014) CanvasDB: a local database infrastructure for analysis of targeted-and whole genome re-sequencing projects. *Database*, 2014, bau098.
19. Cheng,W.Y., Hakkenberg,J., Li,S.D. and Chen,R. (2016) DIVAS: a centralized genetic variant repository representing 150 000 individuals from multiple disease cohorts. *Bioinformatics*, 32, 151–153.
20. Cijvat,R., Manegold,S., Kersten,M. *et al.* (2015) Genome sequence analysis with MonetDB: a case study on Ebola virus diversity. *Datenbanksyst. Business Technol. Web.*, 242, 143–149.
21. Dorok,S. (2015) The relational way to dam the flood of genome data. In: *Proceedings of the 2015 ACM SIGMOD on PhD Symposium*. ACM, Melbourne Australia, pp. 9–13.
22. Massie,M., Nothaft,F., Hartl,C. *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.
23. Dong,C., Wei,P., Jian,X. *et al.* (2015) Comparison and integration of deleteriousness prediction methods for nonsynonymous SNVs in whole exome sequencing studies. *Hum. Mol. Genet.*, 24, 2125–2137.
24. Lupski,J.R., Belmont,J.W., Boerwinkle,E. *et al.* (2011) Clan genomics and the complex architecture of human disease. *Cell*, 147, 32–43.
25. MacArthur,D.G., Balasubramanian, S., Frankish, A. *et al.* (2012) A systematic survey of loss-of-function variants in human protein-coding genes. *Science*, 335, 823–828.
26. Lek, Monkol. *et al.* (2016) Analysis of protein-coding genetic variation in 60,706 humans. *Nature*, 536, 285–291.
27. Fajardo,K.V.F., Adams,D., Mason,C.E. *et al.* (2012) Detecting false positive signals in exome sequencing. *Hum. Mut.*, 33, 609–613.
28. Shyr,C., Tarailo-Graovac,M., Gotlieb, M. *et al.* (2014) FLAGS, frequently mutated genes in public exomes. *BMC Med. Genomics*, 7, 64.
29. Brownstein,C.A., Beggs,A.H., Homer,N. *et al.* (2014) An international effort towards developing standards for best practices in analysis, interpretation and reporting of clinical genome sequencing results in the CLARITY Challenge. *Genome Biol.*, 15, R53.
30. Lee,S., Abecasis,G.R., Boehnke,M. and Lin,X. (2014) Rare-variant association analysis: study designs and statistical tests. *Am. J. Hum. Genet.*, 95, 5–23.
31. Fromer,M. and Purcell,S.M. (2014) Using XHMM software to detect copy number variation in whole-exome sequencing data. *Curr. Protoc. Hum. Genet.*, 81, 7.23.1–7.23.21.
32. Krumm,N., Sudmant,P.H., Ko,A. *et al.* (2012) Copy number variation detection and genotyping from exome sequence data. *Genome Res.*, 22, 1525–1532.
33. Do,R., Kathiresan,S. and Abecasis,G.R. (2012) Exome sequencing and complex disease: practical aspects of rare variant association studies. *Hum. Mol. Genet.*, 21, R1–R9.
34. Sundstrom,D. (2015). Even faster: data at the speed of Presto ORC. <https://code.facebook.com/posts/370832626374903/> even-faster-data-at-the-speed-of-presto-orc/.
35. Lipcon,T., Alves,D., Burkert,D. *et al.* (2015). Kudu: storage for fast analytics on fast data. <https://kudu.apache.org/kudu.pdf>.
36. Dean,J. and Ghemawat,S. (2004) MapReduce: simplified data processing on large clusters. To Appear in OSDI, 2008, 107–113.
37. Zaharia,M., Chowdhury,M., Das,T. *et al.* (2012) Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, San Jose, CA USA, pp. 2–2.

-
38. Chaudhuri,S. and Dayal,U. (1997) An overview of data warehousing and OLAP technology. *ACM Sigmod Rec.*, 26, 65–74.
39. Cornell,M., Paton,N.W., Wu,S. *et al.* (2001) GIMS-a data warehouse for storage and analysis of genome sequence and functional data. In *Bioinformatics and Bioengineering Conference, 2001. Proceedings of the IEEE 2nd International Symposium on*. Bethesda, MD, USA. IEEE, pp. 15–22.
40. Liu,X., Wu,C., Li,C. and Boerwinkle,E. (2015) dbNSFP v3.0: a one-stop database of functional predictions and annotations for human non-synonymous and splice site SNVs. *Hum. Mut.*, 37, 235–241.

Scalable Framework for the Analysis of Population Structure Using the Next Generation Sequencing Data

Anastasiia Hryhorzhevska¹, Marek Wiewiórka¹, Michał Okoniewski², and Tomasz Gambin¹

¹ Institute of Computer Science, Warsaw University of Technology, Warsaw, 00-665, Poland

² Scientific IT Services, ETH Zurich, Zurich, 8092, Switzerland.

Abstract. Genomic variant data obtained from the next generation sequencing can be used to study the population structure of the genotyped individuals. Typical approaches to ethnicity classification/clustering consist of several time consuming pre-processing steps, such as variant filtering, LD-pruning and dimensionality reduction of genotype matrix. We have developed a framework using R programming language to analyze the influence of various pre-processing methods and their parameters on the final results of the classification/clustering algorithms. The results indicated how to fine-tune the pre-processing steps in order to maximize the supervised and unsupervised classification performance. In addition, to enable efficient processing of large data sets, we have developed another framework using Apache Spark. Tests performed on 1000 Genomes data set confirmed the efficiency and scalability of the presented approach. Finally, the dockerized version of the implemented frameworks (freely available at: <https://github.com/ZSI-Bio/popgen>) can be easily applied to any other variant data set, including data from large scale sequencing projects or custom data sets from clinical laboratories.

1 Introduction

Understanding of the genomic basis of diseases has become a central part of human and molecular genetics. It motivates novel biological hypothesis, teaches the things about epidemiology, causal risk factors and relations between different parts of biological system and enables to develop new tools that can be used in diagnosis and treatment. Identification of novel disease genes is a challenging task that requires large scale case-control studies, which often involve individuals from various human populations.

Genome-wide association studies (GWAS) and rare variant association studies (RVAS) give the ability to exam markers across whole genomes simultaneously and test hundreds of thousands allele variants to find associations between genotype in the markers and likelihood of disease. GWAS have focused on the analysis of common variants, however most of genetic heritability remained unexplained [13]. Next generation sequencing enabled to sequence whole genomes and explore

the entire spectrum of allele frequencies, including rare variants. Currently, most studies focus on variants with low minor-allele frequency ($0.5\% < \text{MAF} < 5\%$) or rare variants ($\text{MAF} < 0.5\%$) [16, 20].

Population structure retrieving and inferences are critical in association studies, in which population stratification (i.e. the presence of genotypic differences among different groups of individuals) can lead to inferential errors. Genotype-based clustering of individuals is an important way of summarizing the genetic similarities and differences between individuals of different ancestry. A number of methods have been proposed to deal with the problem such as principal component analysis (PCA) [22], multidimensional scaling (MDS) [18], linkage disequilibrium-based approach [15]. However, these algorithms need to be well-tuned to complete the learning process in the best possible way and considerably low computation time.

At the same time, the whole genome sequencing data generated in large-scale sequencing projects, increase the number of sequenced individuals and the feature space (the number of unique allele variants) by orders of magnitude. This sharp increase in both sample numbers and features per sample requires a massively parallel approach for data processing [25]. Traditional parallelization strategies implemented e.g. in PLINK [23] cannot scale with variable data sizes at runtime [21].

To address this issue, we developed two frameworks. The first one was developed in R programming language and was built on the top of a SNPRelate toolset [10]. It was designed for fine-tuning the quality-control (QC) and the classification model parameters. It provides also the graphical representation of the results. The other one is a distributed computing framework for scalable stratification analysis in Apache Spark [3] using ADAM [6] in combination with machine learning (ML) libraries Spark MLlib [9], H2O[8] and Apache System ML [4]. It is more effective in processing of datasets containing large number of observations (i.e. number of samples $> 10,000$) comparing to the first framework.

2 Methods

2.1 Dataset

To test our approach we have used the 1000 Genomes Project genotype data [1] released in the Variant Call Format (VCF) [11]. The dataset contains variants obtained from sequencing of 2,504 individuals (observations) across more than 30,000,000 alleles (features). The individuals are distributed across five super populations, i.e.: African (AFR), Mixed American (AMR), East Asian (EAS), South Asian (SAS) and European (EUR) and 26 sub-populations [1].

Our aim was to test the performance of ML algorithms to reconstruct both super- and sub-populations. To do this, the classification/clustering models were built either separately, i.e. for either super or sub-population groups or hierarchically, starting from super-populations and drilling down to the sub-populations.

2.2 Frameworks for Inferring the Population Structure

Due to the limitation of the current implementation of PCA in Apache Spark, which does not allow to process matrices with more than 65,535 columns [2], we implemented two separate frameworks. The first framework implemented in R provides an efficient way for testing and fine-tuning QC, PCA and machine learning parameters on the relatively small number of observations (up to several thousands). The second framework implemented in Apache Spark, besides its limitation of the number of features deals gracefully with a large number of observations (i.e. hundred of thousands of individuals).

Since the original data are stored in VCFs, the first task was to transform the data into a more efficient format. In case of the first framework, we used *gdsfmt* R package that provides an interface to CoreArray Genomic Data Structure (GDS) files designated for storing SNP genotypes. In this format each byte encodes up to four SNPs genotypes reducing the file size and access time [7]. In the second framework, we used ADAM format [6] to store and process genomic variant data in Hadoop Distributed File System (HDFS). The original 1000 Genomes data was transformed from VCF files to the respective ADAM format. Then, using ADAM Parquet file, we read the genotypes into Spark data frame and extracted the information that is required for further processing.

Next, we perform three pre-processing steps, including: (i) missing values treatment, (ii) feature selection, and (iii) dimensionality reduction. These steps are done to simplify the data analysis, reduce the noise in the data and increase the accuracy of ML algorithms, and are particularly important in case of high dimensional datasets.

Finally, we apply classification and clustering methods to the reduced dataset and present algorithms quality and efficient performance for different sets of parameters and subsets of the data. Figure 1 indicates the alternative steps in the analysis process.

2.3 Missing Values Handling

In the high dimensional genomic data it is likely that some genotypes are missing, e.g. due to low coverage of sequencing reads in selected samples. The default strategy used in the population stratification is to drop features containing missing values. However, sometimes (e.g. in case of limited genotype datasets from small gene panels) it could be desirable to test on "predictive missingness", that is, what dependency the response may have upon missing values. Therefore, in our framework we implemented both approaches. In the first one we drop all variants (features) with any missing data. In the second approach we filter out variants with the proportion of missing values exceeding the given threshold, defined as P_1 .

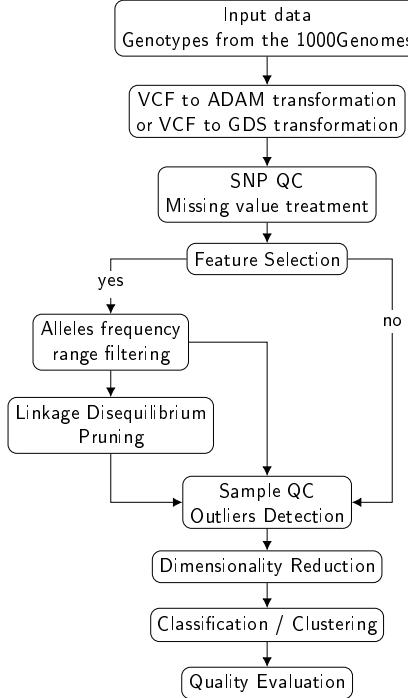


Fig. 1. The workflow for both supervised and unsupervised classification of genomes variant data

2.4 Feature Selection

Alleles Frequency Range Filtering. To reduce the number of variants in the data, we first compute the frequencies of alternate alleles across all individuals in the dataset. For further analysis we select variants with allele frequencies within a certain range, defined as P_2 .

Linkage Disequilibrium Pruning. Linkage disequilibrium (LD) is a non-random association of alleles at different genomic positions (loci) [24]. LD pruning on the variants is suggested before running dimensionality reduction methods because the LD blocks can decrease the ability of the algorithms to separate populations [27]. Such blocks should be removed from the analysis.

In our framework we implemented three methods for calculation of LD values. The first algorithm calculates so called composite coefficient that is estimated

from di-locus counts and sample allele frequencies [26]. The second LD calculation algorithm uses a common standardization method [17] that is a relative measure of disequilibrium compared to its maximum D' [14]. Third method involves so called R coefficient that is computed using expectation maximization algorithm under assumption of Hardy-Weinberg Equilibrium (HWE).

The LD pruning algorithm recursively removes SNPs that are greater than a LD threshold (defined as parameter P_3) within a sliding window based on the pairwise genotypic correlation.

2.5 Dimensionality Reduction

To correct the population stratification and detect true population structure we implemented PCA [19]. The PCA algorithm is carried out on a set of possibly collinear features and performs a transformation to produce a new set of uncorrelated features. Although the Spark MLlib [9] library provides methods for principal components (PCs) computation, these implementations could not handle 1000 Genomes variant dataset and therefore this step was performed using parallelized algorithm implemented in SNPRelate R package.

The number of PCs that are further used for classification/clustering is selected according to the percentage of cumulative variance of the first n PCs. We tested the optimal number of PCs, defined as an input parameter P_4 .

2.6 Unsupervised Learning Algorithms

To group individuals and recover the population structure three algorithms from the Spark MLlib library [9] (K-means, Bisecting K-means and Gaussian Mixture), and three implemented in R libraries (Hierarchical, K-means and Expectation-Maximization) are used. The quality of clustering is assessed using both external (Purity, Adjusted Rand Index [ARI]) and internal (Dunn Index [DI], Calinski-Harabasz Index [CH]) evaluation criteria.

2.7 Supervised Learning Algorithms

In our framework we test three algorithms for supervised classification from Spark MLlib and Spark ML, Apache System ML and R, including: Support Vector Machine (SVM), Random Forests, and Decision Trees.

To evaluate the quality of classification with respect to every class in the dataset, we build confusion matrix and compute common classification metrics such as accuracy, precision, recall, and the F1-Score (i.e. harmonic mean of precision and recall). Identification of ethnic group for an individual is a multi-class classification problem. Therefore in addition to per-class measures we average them over all the classes resulting in macro-averaged precision, recall, F1-Score. In addition, we compute the Kappa statistic, which is a measure of agreement between the predictions and the actual labels. It is interpreted as a comparison of the overall accuracy to the expected random chance accuracy.

3 Results and Discussion

To compare the quality of classifiers implemented in our frameworks, we performed all experiments on chromosome 22 from 1000 Genomes dataset.

Results of unsupervised classification. First, we applied QC procedures to exclude problematic SNPs from the analysis. Since there was no missing values, the P_1 was set to zero in all of the experiments. Next, allele frequency filtering, LD pruning, dimensionality reduction and clustering/classification algorithms were performed. We repeated experiments for different input parameters (P_2, P_3, P_4) and investigated the clustering quality for the three methods by comparing the annotated super-population label (AMR, EUR, AFR, EAS, SAS) for each individual in the dataset to the label assigned. Table 1 presents the final selection of the fine-tuned QC parameters for each unsupervised method and the quality performance of these methods.

Table 1. Summary of quality control parameters and performance of the unsupervised classifiers

Method	MAF	LD	nPCs	Purity	ARI	CH	DI
Hierarchical	(0.005; 0.05)	0.2	5	97.96	95.75	1472.83	0.02
K-means	(0.005; 0.05)	0.2	4	92.21	86.89	7911.91	0.03
E-M	(0.005; 0.1)	0.2	4	96.21	91.76	3232.82	0.01

The best result was obtained using hierarchical method (95.75 % of ARI). However, in case of K-means the CH index is much higher, i.e. the clusters are more compact and the distance between groups are longer than in case of other models. Visualization of the results of hierarchical algorithm for the fine-tuned parameters is presented in Figure 2 and corresponding confusion matrix is shown in Table 2. As expected, AMR and EUR are tended to be clustered into one group, which indicates the similarities between these two populations.

Table 2. Confusion matrix of the hierarchical clustering, $P_2 \in (0.005; 0.05)$, $P_3 = 0.2$, $P_4 = 5$

	1	2	3	4	5
AFR	0	0	4	656	1
AMR	33	1	302	11	0
EAS	0	504	0	0	0
EUR	502	0	1	0	0
SAS	0	0	0	0	489

Our frameworks can be applied to cluster sub-populations. The performed experiments indicate that the analysis of sub-populations is more challenging

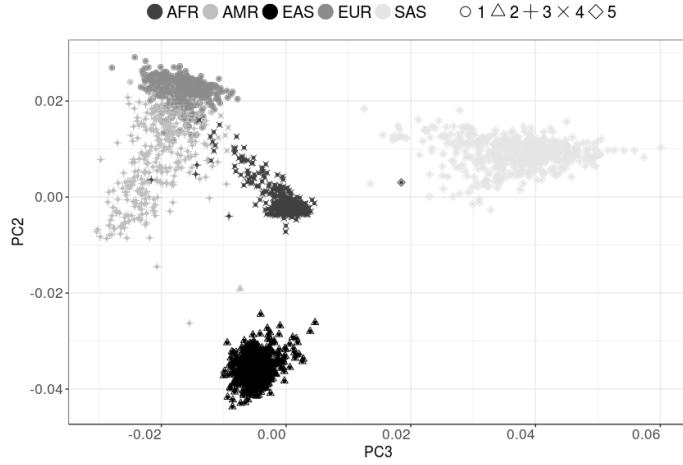


Fig. 2. Results of the hierarchical clustering. Colors indicate true populations, whereas groups discovered by clustering are marked using different shapes. Hierarchical clustering was performed on dataset pre-processed using the following thresholds: $P_2 \in (0.005; 0.05)$, $P_2 = 0.2$, $P_4 = 5$.

and requires to include more variation to obtain satisfactory results, i.e. use wider MAF interval and increased number of PCs.

Results of supervised classification. The process of building supervised classifiers consisted of two steps. First, we fixed classification model parameters and tuned QC parameters. Second, we estimated classifier parameters for the QC parameters obtained in the first step. Final fine-tuned QC and classification model parameters are summarized in Table 3. The results indicate that the QC parameters should be tuned for each supervised method individually. Comparing with unsupervised learning, much larger number of PCs is required to build the best-performing model. Corresponding classification models' prediction performances are presented in Table 4. Comparison of five classifiers revealed that SVM with radial kernel ($F1 = 98.91\%$) and Random Forest ($F1 = 99.31\%$) outperformed Decision Trees and SVM with either linear or quadratic kernel.

Tests of distributed framework. Our distributed framework enables to process the large amount data in parallel on multiple nodes, which not only allow to overcome the problem of limited memory on a single machine but also significantly reduce the overall computation time.

The tests were done for the data on which all pre-processing steps were performed. The dataset consisted of 2,504 human individuals across 5 super-

Table 3. Summary of quality control and classifiers parameters

Method	<i>MAF</i>	<i>LD</i>	<i>nPCs</i>	<i>C</i>	γ	<i>split</i>	<i>cp</i>	<i>ntree</i>
SVM linear	(0.005; 0.05)	0.2	60	50	-	-	-	-
SVM quadratic	(0.005; 0.05)	0.2	50	60	0.1	-	-	-
SVM radial	(0.005; 0.05)	0.2	70	20	10	-	-	-
Decision Trees	(0.005; 0.05)	0.2	60	-	-	<i>gain</i>	0.01	-
Random Forest	(0.01; 0.05)	0.2	60	-	-	-	-	109

Table 4. Prediction models performance of classification of super population groups

Method	Precision	Recall	F1-Score	Accuracy	Kappa
SVM linear	96.43	93.63	94.75	96.25	0.95
SVM quadratic	96.72	94.40	95.38	96.61	0.96
SVM radial	98.57	98.78	98.67	98.91	0.99
Decision Trees	98.14	95.53	96.60	97.70	0.97
Random Forest	99.05	99.33	99.17	99.31	0.99

populations assayed for 5,000 SNPs. We recorded time execution of both frameworks performing PCA on the datasets generated by duplicating the original 1000 Genomes data several times as required [12]. For datasets with the number of samples greater than 20,000 non-distributed framework could not complete computation in a reasonable amount of time, and therefore for larger dataset we report the results of distributed (MLlib-based) implementation only.

Table 5 shows that non-distributed framework deals better with a small number of observations (and thus is more suitable for fine-tuning performed on small datasets), whereas distributed implementation is much faster for a large number of observations, i.e. $N > 10,000$. In particular, Spark MLlib was able to process a dataset of 500,000 samples in less than 23 minutes, while SNPRelate did not complete for a dataset of 50,000 samples after 18 hours of computing.

Table 5. Time execution performance for distributed versus non-distributed framework on increasing datasets derived from 1000 Genomes data, using 5,000 SNPs (total number of executors is 128, executor memory is 12 GB)

Number of observations	Distributed [hh:mm:ss]	Non-distributed [hh:mm:ss]
2,504	00:13:06	00:00:12
5,008	00:13:10	00:01:12
10,016	00:13:37	00:12:18
15,024	00:13:52	00:41:06
20,048	00:14:11	01:37:06
50,080	00:15:40	>18h
100,160	00:16:29	-
250,400	00:19:03	-
500,800	00:22:43	-

4 Conclusions

Our framework for the analysis of the population structure implemented in R programming language provides automated estimation of the optimal QC's and machine learning parameters, and ensures the high-quality performance of both clustering ($ARI = 95.75\%$) and classification ($F1 = 99.31\%$) algorithms for analysis of super-populations. Importantly, it allows for efficient comparison of different machine learning solutions and test for a wide range of input parameters and pre-processing strategies in order to fine-tune clustering/classification methods. Furthermore, the obtained results show that our framework gives better quality performance of clustering of super-population groups ($ARI = 95.75\%$) than other approaches, e.g. VariantSpark ($ARI = 84\%$) [5].

Our second framework has been developed to deal with dataset containing large number of samples (e.g. $> 10,000$ individuals). Both dimensionality reduction and machine learning methods have been implemented using Apache Spark. The results of performed experiments on 1000 Genomes data set demonstrate that the tool can handle the population genetic analysis on the data from large sample size whole-genome sequencing cohorts and process it 100x faster than single node solution.

Acknowledgments. This work has been supported by the Polish National Science Center grants: Opus 2014/13/B/NZ2/01248 and Preludium 2014/13/N/ST6/01843.

References

1. The 1000 genomes project, <http://www.internationalgenome.org/>
2. Apache Spark, RowMatrix, <https://github.com/apache/spark>
3. Apache Spark™, <http://spark.apache.org/>
4. Apache SystemML - Declarative Large-Scale Machine Learning, <https://systemml.apache.org/>
5. BauerLab/VariantSpark, <https://github.com/BauerLab/VariantSpark>
6. Big Data Genomics, <http://bdgenomics.org/>
7. Bioconductor - gdsfmt, <http://bioconductor.org/packages/gdsfmt>
8. H2o.ai, <http://www.h2o.ai/download/sparkling-water/>
9. MLlib | Apache Spark, <http://spark.apache.org/mllib/>
10. SNPRelate, <http://bioconductor.org/packages/SNPRelate/>
11. The variant call format specification, <https://github.com/samtools/hts-specs>
12. Abraham, G., Inouye, M.: Fast Principal Component Analysis of Large-Scale Genome-Wide Data. PLoS ONE 9(4) (Apr 2014)
13. Auer, P.L., Lettre, G.: Rare variant association studies: considerations, challenges and opportunities. Genome Medicine 7(1) (Feb 2015)
14. Hamilton, D.C., Cole, D.E.C.: Standardizing a Composite Measure of Linkage Disequilibrium. Annals of Human Genetics 68(3), 234–239 (May 2004)
15. Hinrichs, A.L., Larkin, E.K., Suarez, B.K.: Population Stratification and Patterns of Linkage Disequilibrium. Genetic epidemiology 33(Suppl 1), S88–S92 (2009)

10 Scalable framework for the analysis of population structure using the NGS data

16. Lee, S., Abecasis, G., Boehnke, M., Lin, X.: Rare-Variant Association Analysis: Study Designs and Statistical Tests. *American Journal of Human Genetics* 95(1), 5–23 (Jul 2014)
17. Lewontin, R.C.: The Interaction of Selection and Linkage. I. General Considerations; Heterotic Models. *Genetics* 49(1), 49–67 (Jan 1964)
18. Li, Q., Yu, K.: Improved correction for population stratification in genome-wide association studies by identifying hidden population structures. *Genetic Epidemiology* 32(3), 215–226 (Apr 2008)
19. Liu, L., Zhang, D., Liu, H., Arendt, C.: Robust methods for population stratification in genome wide association studies. *BMC Bioinformatics* 14, 132 (2013)
20. Manolio, T.A., Collins, F.S., Cox, N.J., Goldstein, D.B., Hindorff, L.A., Hunter, D.J., McCarthy, M.I., Ramos, E.M., Cardon, L.R., Chakravarti, A., Cho, J.H., Guttmacher, A.E., Kong, A., Kruglyak, L., Mardis, E., Rotimi, C.N., Slatkin, M., Valle, D., Whittemore, A.S., Boehnke, M., Clark, A.G., Eichler, E.E., Gibson, G., Haines, J.L., Mackay, T.F.C., McCarroll, S.A., Visscher, P.M.: Finding the missing heritability of complex diseases. *Nature* 461(7265), 747–753 (Oct 2009)
21. O'Brien, A.R., Saunders, N.F.W., Guo, Y., Buske, F.A., Scott, R.J., Bauer, D.C.: VariantSpark: population scale clustering of genotype information. *BMC Genomics* 16, 1052 (2015)
22. Price, A.L., Patterson, N.J., Plenge, R.M., Weinblatt, M.E., Shadick, N.A., Reich, D.: Principal components analysis corrects for stratification in genome-wide association studies. *Nature Genetics* 38(8), 904–909 (Aug 2006)
23. Purcell, S., Neale, B., Todd-Brown, K., Thomas, L., Ferreira, M., Bender, D., Maller, J., Sklar, P., de Bakker, P., Daly, M., Sham, P.: PLINK: A Tool Set for Whole-Genome Association and Population-Based Linkage Analyses. *American Journal of Human Genetics* 81(3), 559–575 (Sep 2007)
24. Slatkin, M.: Linkage disequilibrium — understanding the evolutionary past and mapping the medical future. *Nature Reviews Genetics* 9(6), 477–485 (Jun 2008)
25. Stein, L.D.: The case for cloud computing in genome informatics. *Genome Biology* 11(5), 207 (2010)
26. Weir, B.S.: *Genetic Data Analysis*. Sunderland, Massachusetts:Sinauer Associates, Inc (1996)
27. Zou, F., Lee, S., Knowles, M.R., Wright, F.A.: Quantification of Population Structure Using Correlated SNPs by Shrinkage Principal Components. *Human Heredity* 70(1), 9–22 (Jun 2010)



Genome analysis

SeQuiLa: an elastic, fast and scalable SQL-oriented solution for processing and querying genomic intervals

Marek Wiewiórka^{1,†}, Anna Leśniewska^{2,†}, Agnieszka Szmurto¹,
Kacper Stępień², Mateusz Borowiak², Michał Okoniewski³
and Tomasz Gaminb ^{1,*}

¹Institute of Computer Science, Warsaw University of Technology, Warsaw 00-665, Poland, ²Department of Computer Science, Poznań University of Technology, Poznań 60-965, Poland and ³Scientific IT Services, ETH Zurich, Zürich 8092, Switzerland

*To whom correspondence should be addressed.

†The authors wish it to be known that, in their opinion, the first two authors should be regarded as Joint First Authors.
Associate Editor: John Hancock

Received on May 7, 2018; revised on July 23, 2018; editorial decision on November 11, 2018; accepted on November 13, 2018

Abstract

Summary: Efficient processing of large-scale genomic datasets has recently become possible due to the application of ‘big data’ technologies in bioinformatics pipelines. We present SeQuiLa—a distributed, ANSI SQL-compliant solution for speedy querying and processing of genomic intervals that is available as an Apache Spark package. Proposed range join strategy is significantly ($\sim 22\times$) faster than the default Apache Spark implementation and outperforms other state-of-the-art tools for genomic intervals processing.

Availability and implementation: The project is available at <http://biodatageeks.org/sequila/>.

Contact: tgamin@ii.pw.edu.pl

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Analyses requiring intersection of genomic intervals as defined in Layer (2013) are supported by several reported software tools, including featureCounts (Liao, 2014), samtools (Li, 2009) and GenomicRanges (Lawrence, 2013). Despite their popularity and ease-of-use, they suffer from similar performance limitations, thereby making genome-wide analyses infeasible. On the other hand, recently, there has been an outburst of scalable solutions for genomics, including sparkhit (Huang, 2018), ADAM (Massie *et al.*, 2013) and the latest GATK version leveraging Apache Spark execution engine. Moreover, adapting relational algebra principles in a form of a declarative Structured Query Language (SQL) interface for querying genomic datasets is a novel approach (Kozanitis, 2014; Masseroli, 2015). A proof-of-concept solution that combines big data techniques and SQL interface for handling large-scale interval queries was proposed in GenAp (Kozanitis and Patterson, 2016).

This approach, however, requires modifications in the Apache Spark source code, making this tool hard to maintain and extend; it also discards low-level optimizations resulting in a suboptimal performance. Furthermore, as a consequence of introducing the new keywords, it is effectively non-compliant with ANSI SQL standards, what may cause integration difficulties. To address the aforementioned issues, we have developed a SeQuiLa Apache Spark package which is a distributed, SQL-compliant solution, implementing fast range join computations between two tables, representing genomic intervals.

2 Materials and methods

2.1 Algorithm and implementation

Consider datasets $s1$ and $s2$, storing genomic intervals such as $|s1| < |s2|$. The main idea of the algorithm is to transform $s1$ into a

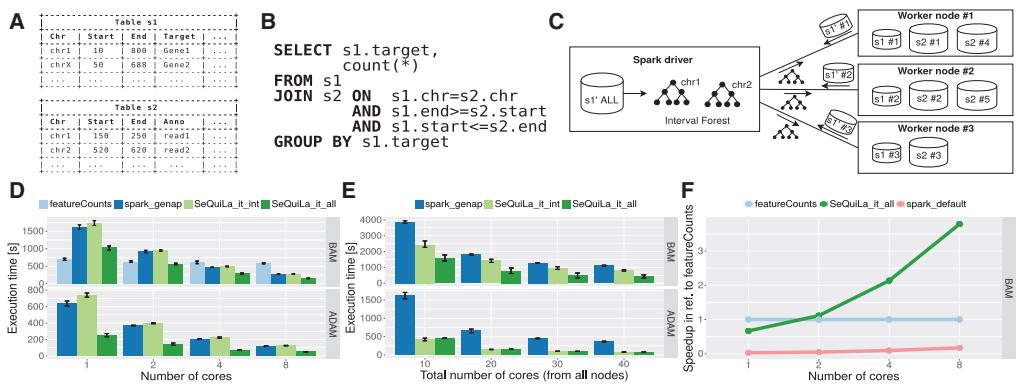


Fig. 1. Example of datasets' structure (s_1, s_2), including required columns, i.e. chr, start, end (A) and a sample SQL query (B). Broadcasting interval forest to worker nodes (C). Performance comparison of featureCounts (state-of-the-art single node solution) against GenAp (the only available distributed solution) and SeQuiLa on single node (D), and on a cluster (E). Speedup characteristics of SeQuiLa and default join implementation in Spark with featureCounts as a baseline (F)

broadcastable structure of an interval forest [a hash map of interval trees (Cormen *et al.*, 2009), each representing one chromosome]. The intervals from s_2 can be efficiently intersected with the interval forest (Fig. 1A–C).

SeQuiLa package introduces a rule-based optimizer that chooses the most efficient join strategy based on input data statistics computed in runtime. First, the dataset with smaller row count (s_1) is designated for constructing an interval forest. Then it estimates the size of dataset s_1' defined as projection of s_1 on the set of columns referenced by SQL query (Fig. 1B). If it fits into dedicated Spark Driver's memory (controlled by *maxBroadcastSize* parameter) the interval forest is augmented with all columns from s_1' (*SeQuiLa_it_all* strategy) completing map-side join procedure in one stage. Otherwise an interval tree is used as an index for additional lookup step before the equi-shuffle-join operation between s_1 and s_2 (*SeQuiLa_it_int* strategy).

SeQuiLa has been developed in Scala using the Apache Spark 2.2 environment. In runtime it extends SparkSQL Catalyst optimizer with custom execution strategies. It implements distributed map joins using interval forest for inner range join operations. Useful genomic transformations have been added as User Defined Functions/Aggregates and exposed to the SQL interface. Furthermore, SeQuiLa data sources for both BAM and ADAM file formats have been implemented. It can also be integrated with third-party tools using SparkSQL JDBC driver and with R using sparklyr package. SeQuiLa is also available as a Docker container, and can be run locally or on a Hadoop cluster using Yet Another Resource Negotiator (see Supplementary Material for implementation details).

2.2 Performance evaluation

Testing infrastructure consisted of a six-node Hadoop cluster (Cloudera Hadoop distribution version 5.12 with Apache Spark upgraded to version 2.2.1), including four data nodes, a master node and an edge node with 24 CPUs and 64 GB RAM each. To prove the vertical and horizontal scalability of our solution and to compare its performance against existing tools, two tests scenarios have been executed, i.e. on a single node (edge node) and on a cluster, using a whole-exome and whole-genome alignment datasets from NA12878 sample, respectively. In each test, the number of sequencing reads overlapping each one of the pre-defined genomic

regions (i.e. either list of exons or genes specified in BED files) has been computed. This type of data processing is widely used in both DNA and RNA sequencing pipelines (see use case examples in Supplementary Material). We have used featureCounts software as a baseline for performance and accuracy comparisons. Finally, we have converted original BAM files into columnar storage format (ADAM) and performed tests on both file formats to observe its impact on the performance (see Supplementary Material for details).

3 Results

SeQuiLa outperforms featureCounts, GenAp and default Spark join implementation in terms of speed on a single node (1.7–22.1×) and a cluster (3.2–4.7×) (Fig. 1D–F). *SeQuiLa_it_all* strategy has proven to perform best in most of the scenarios (no network shuffling required), whereas *SeQuiLa_it_int* performs comparable to, or better than, GenAp. All algorithms favor columnar to row oriented file format due to column pruning and disk reduction of I/O operations.

4 Conclusions

When run in parallel mode, SeQuiLa is the fastest tool in our benchmark, achieving significant performance gain in genomic interval queries. Further, SeQuiLa has a potential to unlock the doors to build additional scalable genomic data warehouse solutions, as well as to implement other higher-level applications for various types of bioinformatics analyses.

Funding

This work was supported by the National Science Center grants [OPUS 2014/13/B/N/Z2/01248, PRELUDIUM 2014/13/N/ST6/01843, SONATA 2015/17/D/ST6/04063]; and by the Polish budget funds for science in years 2016–2019 [Iuventus Plus grant IP2015019874].

Conflict of Interest: none declared.

References

- Cormen,T. H. et al. (2009) Data structures. In: *Introduction to Algorithms*. MIT Press, Cambridge, Massachusetts, pp. 348–354.
- Huang,L. et al. (2018) Analyzing large scale genomic data on the cloud with Sparkhit. *Bioinformatics*, **34**, 1457–1465.
- Kozanitis,C. and Patterson,D.A. (2016) GenAp: a distributed SQL interface for genomic data. *BMC Bioinformatics*, **17**, 63.
- Kozanitis,C. et al. (2014) Using Genome Query Language to uncover genetic variation. *Bioinformatics*, **30**, 1–8.
- Lawrence,M. et al. (2013) Software for computing and annotating genomic ranges. *PLoS Comput. Biol.*, **9**, e1003118.
- Layer,R.M. et al. (2013) Binary Interval Search: a scalable algorithm for counting interval intersections. *Bioinformatics*, **29**, 1–7.
- Li,H. et al. (2009) The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, **25**, 2078–2079.
- Liao,Y. et al. (2014) featureCounts: an efficient general purpose program for assigning sequence reads to genomic features. *Bioinformatics*, **30**, 923–930.
- Masseroli,M. et al. (2015) GenoMetric Query Language: a novel approach to large-scale genomic data management. *Bioinformatics*, **31**, 1881–1888.
- Massie,M. et al. (2013) Adam: genomics formats and processing patterns for cloud scale computing. *Technical Report*, No. UCB/EECS-2013-207. University of California, Berkeley.



TECHNICAL NOTE

SeQuiLa-cov: A fast and scalable library for depth of coverage calculations

Marek Wiewiórka[†], Agnieszka Szmurło[†], Wiktor Kuśmirek and Tomasz Gambin *

Institute of Computer Science, Warsaw University of Technology, ul. Nowowiejska 15/19, 00-665 Warsaw, Poland

*Correspondence address. Tomasz Gambin, Institute of Computer Science, Warsaw University of Technology, ul. Nowowiejska 15/19, 00-665 Warsaw, Poland. E-mail: tgambin@gmail.com  <http://orcid.org/0000-0002-0941-4571>

[†]Contributed equally.

Abstract

Background: Depth of coverage calculation is an important and computationally intensive preprocessing step in a variety of next-generation sequencing pipelines, including the analysis of RNA-sequencing data, detection of copy number variants, or quality control procedures. **Results:** Building upon big data technologies, we have developed SeQuiLa-cov, an extension to the recently released SeQuiLa platform, which provides efficient depth of coverage calculations, reaching >100× speedup over the state-of-the-art tools. The performance and scalability of our solution allow for exome and genome-wide calculations running locally or on a cluster while hiding the complexity of the distributed computing with Structured Query Language Application Programming Interface. **Conclusions:** SeQuiLa-cov provides significant performance gain in depth of coverage calculations streamlining the widely used bioinformatic processing pipelines.

Keywords: NGS data analysis; depth of coverage; big data; distributed computing; SQL; CNV-calling; RNA-seq; quality control for sequencing data

Findings

Introduction

Given a set of sequencing reads and a genomic contig, depth of coverage for a given position is defined as the total number of reads overlapping the locus.

The coverage calculation is a frequently performed but time-consuming step in the analysis of next-generation sequencing (NGS) data. In particular, copy number variant detection pipelines require obtaining sufficient read depth of the analyzed samples [1–3]. In other applications, the coverage is computed to assess the quality of the sequencing data (e.g., to calculate the percentage of genome with $\geq 30\times$ read depth) or to identify genomic regions overlapped by an insufficient number of reads for reliable variant calling [4]. Finally, depth of coverage is one of

the most computationally intensive parts of differential expression analysis using RNA-sequencing data at single-base resolution [5–7].

A number of tools supporting this operation have been developed, with 22 of them specified in the Omictools catalog [8]. Well-known, state-of-the-art solutions include samtools depth [9], bedtools genomecov [10], GATK DepthOfCoverage [11], sambamba [12], and mosdepth [13] (see comparison presented in Table 1).

Traditionally, these methods calculate the depth of coverage using a pileup-based approach (introduced in samtools [9] and used in GATK [11]), which is inefficient because it iterates through each nucleotide position at every read in a BAM file. An optimized, event-based [10] and mosdepth [13]. These algorithms use only specific "events," i.e., start and end of the alignment

Received: 13 December 2018; Revised: 24 May 2019; Accepted: 10 July 2019

© The Author(s) 2019. Published by Oxford University Press. This is an Open Access article distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted reuse, distribution, and reproduction in any medium, provided the original work is properly cited.

Table 1: Comparison of leading coverage calculation software tools

Tool	Approach	Functionality		Windows	Language	Implementation		Interface
		Bases	Blocks			Intel GKL	Parallelism type	
samtools	Pileup	Yes	No	No	C	No	None	Command line
bedtools	Events	Yes	Yes	No	C++	No	None	Command line
GATK ¹	Pileup	Yes	No	No	Java	Yes	Distributed	Command line
sambamba	Pileup	No	Yes	Yes	D	No	Multithreaded	Command line
mosdepth	Events	No	Yes	Yes	Nim	No	Multithreaded ²	Command line
SeQuiLa-cov	Events	Yes	Yes	Yes	Scala	Yes	Distributed	Scala, SQL

¹GATK DepthOfCoverage has not yet been ported to the latest version, i.e., GATK 4.x.

²Only for BAM decompression.

blocks within each read (Fig. 1A) instead of analyzing every base of each read, which substantially reduces the overall computational complexity.

Samtools and bedtools depth of coverage modules do not provide any support for a multi-core environment. Mosdepth implements parallel BAM decompression, but its main algorithm remains sequential. Sambamba, on the other hand, promotes itself as a highly parallel tool, implementing depth of coverage calculations in a map-reduce fashion using multiple threads on a single node. Regardless of parallelization degree, all of the aforementioned tools share a common bottleneck caused by using a single thread for returning results. Finally, GATK was the first genomic framework to provide support for distributed computations; however, the DepthOfCoverage method has not yet been ported to the current software release of the toolkit.

We present the first fully scalable, distributed, SQL-oriented solution designated for depth of coverage calculations. SeQuiLa-cov, an extension to the recently released SeQuiLa [14] platform, runs a redesigned event-based algorithm for the distributed environment and provides a convenient, SQL-compliant interface.

Algorithm and implementation

Algorithm

Consider an input data set, `read.set`, of aligned sequencing reads sorted by genomic position from a BAM file partitioned into n data slices (`read.set1`, `read.set2`, `read.setn`) (Fig. 1B).

In the most general case, the algorithm can be used in a distributed environment where each cluster node computes the coverage for the subset of data slices using the event-based method. Specifically, for the i th partition containing the set of reads (`read.seti`), the set of $\text{events}_{i,\text{chr}}$ vectors (where chr is an index of genomic contig represented in `read.set`) is allocated and updated, based on the items from `read.seti`. For all reads, the algorithm parses the concise idiosyncratic gapped alignment report (CIGAR) string, and for each continuous alignment block characterized by start position and length len it increments by 1 the $\text{events}_{i,\text{chr}}(\text{start})$ and decrements by 1 the value of $\text{events}_{i,\text{chr}}(\text{start} + \text{len})$. To compute the partial coverage vector for partition i and contig chr , a vector value at the index j is calculated as follows:

$$\text{partial_coverage}_{i,\text{chr}}(j) = \sum_{m=1}^j \text{events}_{i,\text{chr}}(m).$$

The result of this stage is a set of $\text{partial_coverage}_{i,\text{chr}}$ vectors distributed among the computation nodes. To calculate the final coverage for the whole `read.set`, an additional step of cor-

rection for overlaps between the partitions is required. An overlap $\text{overlap}_{i,\text{chr}}$ of length l between vectors $\text{partial_coverage}_{i,\text{chr}}$ and $\text{partial_coverage}_{i+1,\text{chr}}$ may occur on the partition boundaries where l trailing genomic positions of $\text{partial_coverage}_{i,\text{chr}}$ are the same as l leading genomic positions of $\text{partial_coverage}_{i+1,\text{chr}}$ (see Fig. 1C).

If an overlap is identified, then the coverage values from the $\text{partial_coverage}_{i,\text{chr}}$'s l -length tail are added into the $\text{partial_coverage}_{i+1,\text{chr}}$'s head and subsequently the last l elements of $\text{partial_coverage}_{i,\text{chr}}$ are removed. Once this correction step is completed, non-overlapping $\text{coverage}_{i,\text{chr}}$ vectors are collected and yield the final coverage values for the whole input `read.set`.

The main characteristic of the described algorithm is its ability to distribute data and calculations (such as BAM decompression and main coverage procedure) among the available computation nodes. Moreover, instead of simply performing the full data reduction stage of the partial coverage vectors, our solution minimizes required data shuffling among cluster nodes by limiting it to the overlapping part of coverage vectors. Importantly, the SeQuiLa-cov computation model supports fine-grained parallelism at a user-defined partition size in contrast to the traditional, coarse-grained parallelization strategies that involve splitting input data at a contig level.

Implementation

We have implemented SeQuiLa-cov in Scala programming language using the Apache Spark framework. To efficiently access the data from a BAM file we have prepared a custom data source using Data Source API exposed by SparkSQL. Performance of the read operation benefits from the Intel Genomics Kernel Library (GKL) [15] used for decompressing the BAM file chunks and from a predicate push-down mechanism that filters out data at the earliest stage.

The implementation of the core coverage calculation algorithm aimed to minimize the memory footprint whenever possible by using parsimonious data types, e.g., "Short" type instead of "Integer," and to implement an efficient memory allocation strategy for large data structures, e.g., favoring static Arrays over dynamic size ArrayBuffer. Additionally, to reduce the overhead of data shuffling between the worker nodes in the correction for overlap stage, we used Spark's shared variables [16] "accumulators" and "broadcast variables" (Fig. 1C). Accumulator is used to gather information about the worker nodes' coverage vector ranges and coverage vector tail values, which are subsequently read and processed by the driver. This information is then used to construct a broadcast variable distributed to the worker nodes in order to perform adequate trimming and summing operations on partial coverage vectors.

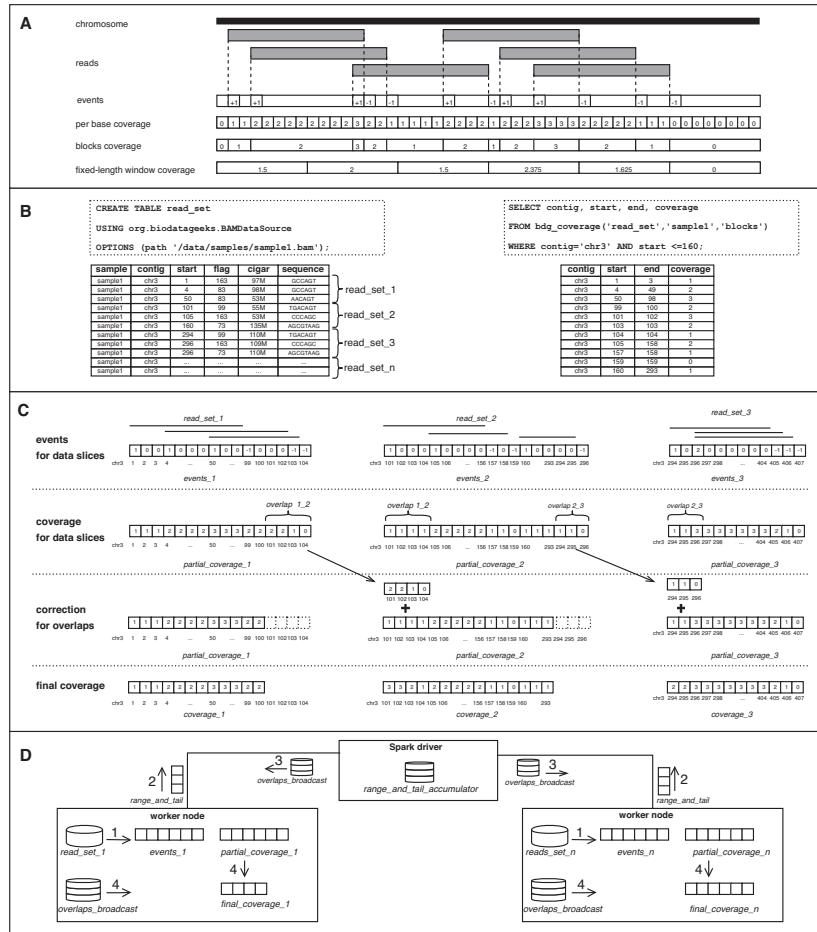


Figure 1: SeQuiLa-cov: functionality, algorithm, and implementation. (A) General concept of events-based algorithm for depth of coverage calculation. Given a genomic chromosome and a set of aligned sequencing reads, the algorithm allocates “events” vector. Subsequently, it iterates the list of reads and increments/decrements by 1 the values of the events vector at the indexes corresponding to start/end positions of each read. The depth of coverage for a genomic locus is calculated using the cumulative sum of all elements in the events vector preceding the specified position. The algorithm may produce 3 typically used coverage types: (i) per-base coverage, which includes the coverage value for each genomic position separately, (ii) blocks, which lists adjacent positions with equal coverage values merged into a single interval, and (iii) fixed-length windows coverage, which generates a set of equal-size, non-overlapping and tiling genomic ranges and outputs the arithmetic mean of base coverage values for each region. (B) Provided SQL API to interact with NGS data. The first statement creates a relational table `read_set` over compressed BAM files using the provided custom Data Source, whereas the second statement demonstrates the use of the `bdg.coverage` function to calculate depth of coverage for a specified sample. The presented call for coverage method takes sample identifier (`sample1`) and result type (`blocks`) as input parameters. `bdg.coverage` is implemented as a table-valued function. Therefore, it outputs a table as a result, allowing for customizing a query using Data Manipulation Language, e.g., in the `SELECT` or `WHERE` clause. For the purpose of this example, we assume that the BAM file for `sample1` contains only reads from `chr3`. (C) Concept of distributed version of events-based algorithm. Assuming that we run our calculations in a distributed environment, the computation nodes do not work on the whole input data set (table `read.set`) but on n smaller data partitions ($\text{slice}_1, \text{slice}_2, \dots, \text{slice}_n$), each containing a subset of input aligned reads. The algorithm first calculates the partial events vector for available data slices and subsequently produces a corresponding partial `partial.coverage` vector. Because of the possibility of overlapping of ranges between 2 consecutive data slices, an additional correction step needs to be performed. When an overlap is identified, the corresponding coverage values from the preceding vector’s tail are cut and added to the head values of the subsequent vector. On the figure, 2 overlaps are shown, one of them situated between `partial.coverage_1` and `partial.coverage_2` (`overlap_{1,2}` of length 4) encompassing positions `chr3:101-104`. The coverage values from `partial.coverage_1` for `overlap_{1,2}` are removed from `partial.coverage_1` and added to the head of `partial.coverage_2`. As a result, a set of non-overlapping coverage vectors are calculated, which is further integrated into the depth of coverage for the whole input data set. (D) Implementation details of SeQuiLa-cov. We have used the Apache Spark environment, where a single driver node runs the high-level driver program, which schedules tasks for multiple worker nodes. On each worker node, a set of data partitions are accessed and manipulated in order to generate events and `partial.coverage` vectors. To gather data about `partial.coverage` vectors’ ranges along with tailing coverage values, and to distribute data needed for rearranging coverage vector values and ranges, we have used Spark’s shared variables “`accumulator`” and “`broadcast`,” respectively.

Functionality

Supported coverage result types

SeQuiLa-cov features 3 distinct result types: "per-base," "blocks," and "fixed-length windows" coverage (Fig. 1A). For per-base, the depth of coverage is calculated and returned for each genomic position, making it the most verbose output option. The method producing block-level coverage (blocks) involves merging adjacent genomic positions with equal coverage values into genomic intervals. As a consequence, fewer records than in the case of per-base output type are generated, with no information loss. For the fixed-length windows the algorithm generates set of fixed-length, tiling, non-overlapping genomic intervals and returns the arithmetic mean of coverage values over positions within each window.

ANSI SQL compliance

The SeQuiLa-cov solution promotes SQL as a data query and manipulation language in genomic analysis. Data flows are performed in SQL-like manner through the custom data source, supporting the convenient Create Table as Select and Insert as Select methods. SeQuiLa-cov provides a table abstraction over existing alignment files, with no need of data conversion, which can be further queried and manipulated in a declarative way. The coverage calculation function `b6g.coverage`, as described in the Algorithm subsection, has been implemented as a table-valued function (Fig. 1D).

Execution and integration options

SeQuiLa-cov can be used as an extension to Apache Spark in the form of an external JAR dependency or can be executed from the command line as a Docker container. Both options can be run locally (on a single node) or on a Hadoop cluster using YARN (see project documentation for sample commands). The tool accepts BAM/CRAM files as input and supports processing of short and long reads. The tabular output of the coverage computations can be stored in various file formats, e.g., binary (ORC, Parquet), as well as text (CSV, TSV). The tool can be integrated with state-of-the-art applications through text files or can be used directly as an additional library in bioinformatics pipelines implemented in Scala, R, or Python.

Benchmarking

We have benchmarked SeQuiLa-cov solutions with leading software for depth of coverage calculations, specifically samtools depth, bedtools genomeCov, sambamba depth, and mosdepth (results of DepthOfCoverage from outdated GATK version are available at <http://biodatageeks.org/sequila/benchmarking/benchmarking.html#depth-of-coverage>). The tests were performed on the aligned whole-exome sequencing (WES) and whole-genome sequencing (WGS) reads from the NA12878 sample (see Methods for details) and aimed at calculating blocks and window coverage. To compare the performance and scalability of each solution, we executed calculations for 1, 5, and 10 cores on a single computation node (see Table 2).

Samtools depth and bedtools genomeCov are both natively non-scalable and were run on a single thread only. Exome-wide calculations exceeded 10 minutes and genome-wide analyses took >2 hours in the case of samtools, while bedtools'

performance was substantially worse, i.e., ~1.9× for WES and ~4.7× for WGS. Sambamba depth claims that it can take advantage of fully parallelized data processing with the use of multithreading. However, our results revealed that even when additional threads were used, the total execution time of coverage calculations remained nearly constant and greater than samtools' result. Mosdepth shows substantial speedup (~1.3×) against samtools when using a single thread. This performance gain increases to ~3.7× when using 5 decompression threads; however, it does not benefit from adding additional CPU power. In the case of fixed-length window coverage mosdepth achieves more than ~1.3 speedup against sambamba.

SeQuiLa-cov achieves performance similar to mosdepth when run using a single core. However, SeQuiLa-cov is ~1.3× and ~2.5× as fast as mosdepth when using 5 and 10 CPU cores, respectively, demonstrating its better scalability. Similar performance is observed for both block and fixed-length window methods.

To fully assess the scalability profile of our solution, we performed additional tests in a cluster environment (see Methods for details). Our results show that when utilizing additional resources (i.e., >10 CPU cores), SeQuiLa-cov is able to reduce the total computation time to 15 seconds for WES and <1 minute for WGS data (Fig. 2). The scalability limit is achieved for 200 and ~500 CPU cores for WES and WGS data, respectively.

To evaluate the impact of the Intel GKL library on the deflate operation (BAM bzgf block decompression), we performed block coverage calculations on WES data on 50 CPU cores. The results showed on average ~1.18× speedup when running with the Intel GKL deflate implementation.

Finally, our comprehensive functional unit testing showed that the results calculated by SeQuiLa-cov and samtools depth are identical.

Conclusions

Recent advances in big data technologies and distributed computing can contribute to speeding up both genomic data processing and management. Analysis of large genomic data sets requires efficient, accurate, and scalable algorithms to perform calculations using the computing power of multiple cluster nodes. In this work, we show that with a sufficiently large cluster, genome-wide coverage calculations may last <1 minute and at the same time be >100× faster than the best single-threaded solution.

Although the tool can be integrated with non-distributed software, our primary aim is to support large-scale processing pipelines, and the full advantage of SeQuiLa-cov's scalability and performance will be available once it is deployed and executed in a distributed environment. We expect that there will be a growing number of scalable solutions (Big Data Genomics project [17] with tools DECA and Cannoli as well as GATK4 [18]) that can take advantage of reading input data directly from distributed storage systems.

SeQuiLa-cov is one of the building blocks of the SeQuiLa [14] ecosystem, which initiated the move towards efficient, distributed processing of genomic data and providing SQL-oriented API for convenient and elastic querying. We foresee that following this direction will enable the evolution of genomic data analysis from file-oriented to table-oriented processing.

Table 2: Benchmarking leading solutions against SeQuiLa-cov on WES/WGS data in execution time of block and window calculations

Data	Operation type	Cores	samtools	bedtools	sambamba	mosdepth	SeQuiLa-cov
WGS	Blocks	1	2h 14m 58s ¹	10h 41m 27s	2h 44m 0s	1h 46m 27s	1h 47m 5s
		5			2h 47m 53s	36m 13s	26m 59s
		10			2h 50m 47s	34m 34s	13m 54s
	Fixed-length windows	1			1h 46m 50s	1h 22m 49s	1h 24m 8s
		5			1h 41m 23s	20m 3s	18m 43s
		10			1h 50m 35s	17m 49s	9m 14s
	WES	1	12m 26s ¹	23m 25s	25m 42s	6m 43s	6m 54s
		5			25m 46s	2m 25s	1m 47s
		10			25m 49s	2m 20s	1m 4s
	Fixed-length windows	1			14m 36s	6m 11s	6m 29s
		5			14m 54s	2m 8s	1m 42s
		10			14m 40s	2m 14s	1m 1s

Both samtools and bedtools calculate coverage using only a single thread; however, their results differ significantly, with samtools being approximately twice as fast. Sambamba positions itself as a multithreaded solution, although our tests revealed that its execution time is nearly constant, regardless of the number of CPU cores used, and even twice as slow as samtools. Mosdepth achieved speedup against samtools in blocks coverage and against sambamba in windows coverage calculations; however, its scalability reaches its limit at 5 CPU cores. Finally, SeQuiLa-cov achieves performance nearly identical to that of mosdepth for the single core, but the execution time decreases substantially for greater number of available computing resources, which makes this solution the fastest when run on multiple cores and nodes.

¹Per-base results are treated as block output. Samtools lacks the functionality of block coverage calculations; however, we included this tool in our benchmark for completeness, treating its per-base results as block outcome assuming that both result types require nearly the same resources.

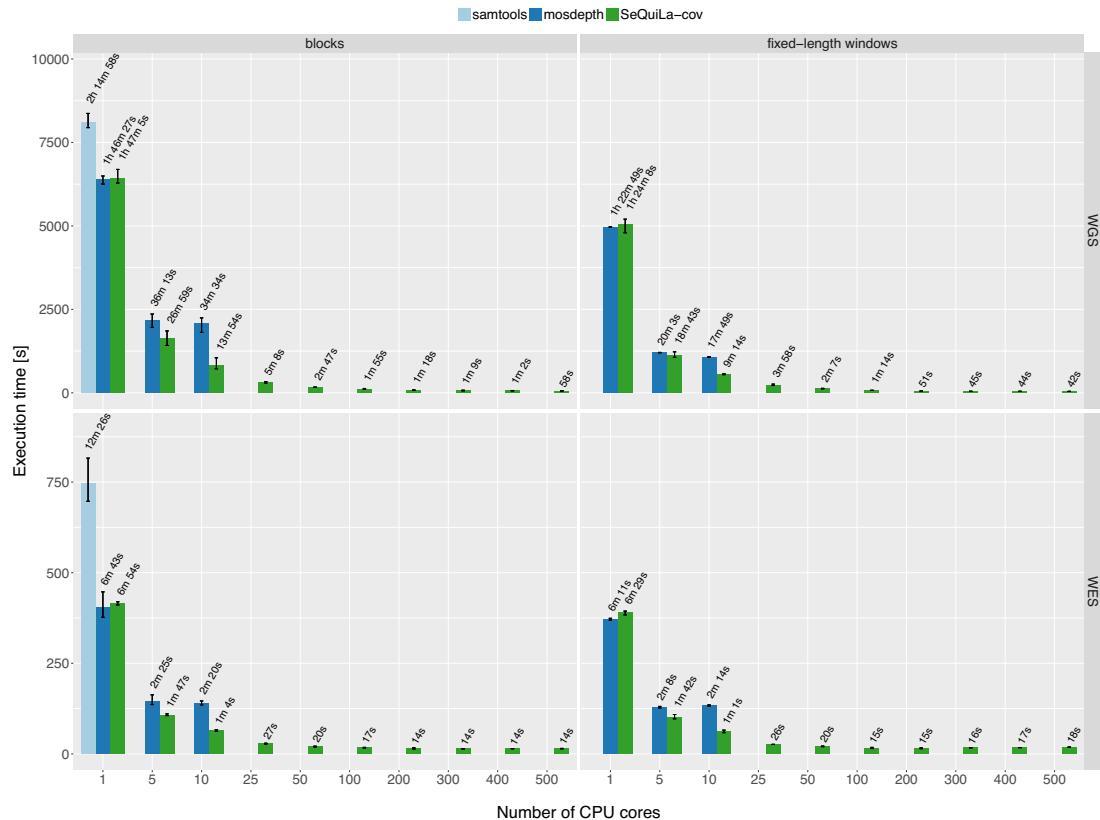


Figure 2: Performance and scalability comparison of samtools, mosdepth, and SeQuiLa-cov. Each experiment setting was repeated several times. Bar height and error bars indicate mean and range of execution time, respectively. The best pileup-based solution is definitely slower (2 times for WGS calculations) than both event-based solutions, which clearly shows the superiority of the latter one. Mosdepth execution time scales up to 5 cores; afterwards it shows no further gain in performance. SeQuiLa-cov has nearly the same execution time results as mosdepth for both block and window calculations for a single core, but scales out desirably using all 500 CPU cores on cluster nodes and at the same time performing WGS calculations in <1 minute.

Methods

Test data

We tested our solution using reads from the NA12878 sample, which were aligned to the hg18 genome. The WES data contained >161 million reads (17 GB of disk space) and WGS data included >2.6 billion reads (272 GB of disk space). Both BAM files were compressed at the default BAM compression level (5).

Testing environment

To perform comprehensive performance evaluation, we set up a test cluster consisting of 28 Hadoop nodes (1 edge node, 3 master nodes, and 24 data nodes) with Hortonworks Data Platform 3.0.1 installed. Each data node has 28 cores (56 with hyper-threading) and 512 GB of RAM; YARN resource pool has been configured with 2,640 virtual cores and 9,671 GB RAM.

Investigated solutions

In our benchmark we used the most recent versions of the investigated tools, i.e., samtools version 1.9, bedtools 2.27.0, sambamba 0.6.8, mosdepth version 0.2.3, and SeQuiLa-cov version 0.5.1.

Availability of source code and requirements

- Project name: SeQuiLa-cov
- Project home page: <http://biodatageeks.org/sequila/>
- Source code repository: <https://github.com/ZSI-Bio/bdg-sequila>
- Operating system: Platform independent
- Programming language: Scala
- Other requirements: Docker
- License: Apache License 2.0
- RRID: SCR.017220

Availability of supporting data and materials

The Docker image is available at <https://hub.docker.com/r/biodatageeks/>. Supplementary information on benchmarking procedure as well as test data are publicly accessible at the project documentation site <http://biodatageeks.org/sequila/benchmarking/benchmarking.html#depth-of-coverage>. An archival copy of the code and supporting data is also available via the GigaScience database GigaDB [19].

Abbreviations

API: Application Programming Interface; BAM: Binary Alignment Map; CPU: central processing unit; CSV: comma-separated values; GKL: Genomics Kernel Library; NGS: next-generation sequencing; ORC: optimized row columnar; RAM: random access memory; SQL: Structured Query Language; TSV: tab-separated values; YARN: Yet Another Resource Negotiator; WES: whole-exome sequencing; WGS: whole-genome sequencing.

Competing interests

The authors declare that they have no competing interests.

Funding

This work has been supported by the Polish budget funds for science in years 2016–2019 (Iuventus Plus grant IP2015 019874), as well as Polish National Science Center grant Preludium 2014/13/N/ST6/01843.

Authors' contributions

M.W.: conceptualization, formal analysis, investigation, software, and writing. A.S.: data curation, formal analysis, investigation, software, visualization, and writing. W.K.: formal analysis, investigation, writing. T.G.: formal analysis, supervision, investigation, visualization, and writing. All authors approved the final manuscript.

References

1. Fromer M, Purcell SM. Using XHMM software to detect copy number variation in whole-exome sequencing data. *Curr Protoc Hum Genet* 2014;81:1–21.
2. Jiang Y, Oldridge DA, Diskin SJ, et al. CODEX: a normalization and copy number variation detection method for whole exome sequencing. *Nucleic Acids Res* 2015;43(6):e39.
3. Gambin T, Akdemir ZC, Yuan B, et al. Homozygous and hemizygous CNV detection from exome sequencing data in a Mendelian disease cohort. *Nucleic Acids Res* 2017;45(4):1633–48.
4. Okonechnikov K, Conesa A, García-Alcalde F. Qualimap 2: advanced multi-sample quality control for high-throughput sequencing data. *Bioinformatics* 2015;32(2):btv566.
5. Fraze AC, Sabuncyan S, Hansen KD, et al. Differential expression analysis of RNA-seq data at single-base resolution. *Biostatistics* 2014;15(3):413–26.
6. Nellore A, Collado-Torres L, Jaffe AE, et al. Rail-RNA: scalable analysis of RNA-seq splicing and coverage. *Bioinformatics* 2016;33(24):btw575.
7. Collado-Torres L, Nellore A, Fraze AC, et al. Flexible expressed region analysis for RNA-seq with derfinder. *Nucleic Acids Res* 2017;45(2):e9–e9.
8. Coverage/Depth analysis bioinformatics tools | Next-generation sequencing analysis - OMICtools. <https://omictools.com/depth-of-coverage-category>. Accessed 24 May 2019.
9. Li H, Handsaker B, Wysoker A, et al. The Sequence Alignment/Map format and SAMtools. *Bioinformatics* 2009;25(16):2078–9.
10. Quinlan AR, Hall IM. BEDTools: a flexible suite of utilities for comparing genomic features. *Bioinformatics* 2010;26(6):841–2.
11. McKenna N, Hanna M, Banks E, et al. The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Res* 2010;20(9):1297–303.
12. Tarasov A, Vilella AJ, Cuppen E, et al. Sambamba: fast processing of NGS alignment formats. *Bioinformatics* 2015;31(12):2032–4.
13. Pedersen BS, Quinlan AR. Mosdepth: quick coverage calculation for genomes and exomes. *Bioinformatics* 2018;34(5):867–8.
14. Wiewiórka M, Leśniewska A, Szmurlo A, et al. SeQuiLa: an elastic, fast and scalable SQL-oriented solution for processing and querying genomic intervals. *Bioinformatics* 2019 25, 12, 2156–8.

15. Guilford J, Powley G, Tucker G, et al. Accelerating the compression and decompression of genomics data using GKL provided by Intel. 2017. <https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/accelerating-genomics-data-gkl-white-paper.pdf>. Accessed 24 May 2019.
16. Zaharia M, Chowdhury MJ, Franklin M, et al. Spark: Cluster computing with working sets. In: HotCloud'10 Proceedings of the 2nd USENIX conference on Hot Topics in Cloud Computing, Boston, MA, 2010. Berkeley, CA: USENIX Association; 2010:10.
17. Massie M, Nothaft F, Hartl C, et al.. Adam: Genomics Formats and Processing Patterns for Cloud Scale Computing. University of California, Berkeley, Technical Report No. UCB/EECS-2013-207; 2013. <https://www2.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-207.html>. Accessed 23 July 2019.
18. GATK. <https://software.broadinstitute.org/gatk/gatk4>. Accessed 24 May 2019.
19. Wiewiórka M, Szmurlo A, Kuśmirek W, et al.. Supporting data for “SeQuiLa-cov: a fast and scalable library for depth of coverage calculations.” GigaScience Database 2019.<http://dx.doi.org/10.5524/100617>. Accessed 23 July 2019.



Genome analysis

Cloud-native distributed genomic pileup operations

Marek Wiewiórka^{1,†}, Agnieszka Szmurło^{1,†}, Paweł Stankiewicz² and Tomasz Gamin ^{1,*}

¹Institute of Computer Science, Warsaw University of Technology, Warsaw, Warsaw 00-661, Poland and ²Department of Molecular and Human Genetics, Baylor College of Medicine, Houston, TX 77030, USA

*To whom correspondence should be addressed.

†The authors wish it to be known that, in their opinion, the first two authors should be regarded as Joint First Authors.

Associate Editor: Peter Robinson

Received on August 27, 2022; revised on November 16, 2022; editorial decision on December 11, 2022; accepted on December 13, 2022

Abstract

Motivation: Pileup analysis is a building block of many bioinformatics pipelines, including variant calling and genotyping. This step tends to become a bottleneck of the entire assay since the straightforward pileup implementations involve processing of all base calls from all alignments sequentially. On the other hand, a distributed version of the algorithm faces the intrinsic challenge of splitting reads-oriented file formats into self-contained partitions to avoid costly data exchange between computational nodes.

Results: Here, we present a scalable, distributed and efficient implementation of a pileup algorithm that is suitable for deploying in cloud computing environments. In particular, we implemented: (i) our custom data-partitioning algorithm optimized to work with the alignment reads, (ii) a novel and unique approach to process alignment events from sequencing reads using the MD tags, (iii) the source code micro-optimizations for recurrent operations, and (iv) a modular structure of the algorithm. We have proven that our novel approach consistently and significantly outperforms other state-of-the-art distributed tools in terms of execution time (up to 6.5× faster) and memory usage (up to 2× less), resulting in a substantial cloud cost reduction. SeQuila is a cloud-native solution that can be easily deployed using any managed Kubernetes and Hadoop services available in public clouds, like Microsoft Azure Cloud, Google Cloud Platform, or Amazon Web Services. Together with the already implemented distributed range join and coverage calculations, our package provides end-users with a unified SQL interface for convenient analyses of population-scale genomic data in an interactive way.

Availability and implementation: <https://biodatageeks.github.io/sequila/>

Contact: tomasz.gamin@pw.edu.pl

1 Introduction

1.1 State-of-the-art

The sorted collection of the aligned sequencing reads can be transformed into a set of pileup records, also known as a coverage position summary. This format summarizes information about the base calls in all genomic positions from the reads aligned to a reference sequence, including total depth of coverage, non-reference (alternative) bases, and base qualities (see detailed definition in <http://www.htslib.org/doc/samtools-mpileup.html>). Pileup format was designed to provide the evidence of the single-nucleotide variants or the short insertions/deletions at given genomic positions. It is commonly used as an entry point to the well-established variant calling pipelines (Li, 2011) as well as to novel approaches to variant detection frameworks based on the neural networks (Luo *et al.*, 2020) or other methods, e.g. the binomial model, partial-order alignment, and de

Brujin graph local assembly (Liu *et al.*, 2021) in fast variant calling. Coverage position summary is also used for identification of somatic mutations and copy number variation (Koboldt *et al.*, 2012).

Samtools suite (Li *et al.*, 2009) includes the mpileup tool, a gold standard for both data format and correctness of pileup calculations; however, it is a single-threaded program that does not provide the scalability feature (Sater *et al.*, 2020).

Research developments in the bioinformatics field emphasize a common need to use a technology that allows distributing long-lasting big data tasks into the multiple computing nodes or in the cloud computing infrastructure (Yuan and Wildish, 2020). In the recent Genome Analysis Toolkit (GATK, McKenna *et al.*, 2010) version, several programs (including pileup calculations) have been implemented in a distributed manner ready to be run on the Apache Spark cluster (Zaharia *et al.*, 2010). Other research studies confirm that big data programming paradigms can be successfully applied to

many genomic analyses (Capuccini *et al.*, 2020; Guo *et al.*, 2018; Wiewiórka *et al.*, 2017, 2018) including variant calling (Ahmad *et al.*, 2021). The analysis of the ever-increasing genomic datasets involves significant financial investments and administrative efforts to maintain secure and fault-tolerant storage solutions as well as fast and scalable processing units. To minimize those efforts, medical clinics and research centers consider migrating bioinformatics pipelines and custom analyses to private or public cloud infrastructure. The evolution towards cloud architecture is embraced by the widely used bioinformatics products both open-source (e.g. GATK) and commercial (e.g. DNA Nexus, Terra) (Koppad *et al.*, 2021).

Although significant progress has been made, there are still areas in bioinformatics analyses that are not easily transferable to the distributed and cloud environments with traditionally used sets of tools.

1.2 Contribution

In the previous works, we proved that it was possible to implement very efficient and highly scalable tools, facilitating time-consuming common bioinformatics operations: interval joins [SeQuiLa-int algorithm (Wiewiórka *et al.*, 2018), available in SeQuiLa package since version 0.3.0] and coverage calculations [SeQuiLa-cov algorithm (Wiewiórka *et al.*, 2019), available in the SeQuiLa package since version 0.4.0]. In the next essential release of SeQuiLa (0.6.11) in 2021, we updated the code base to run on Apache Spark 3.1.2. We have also fixed the reported issues and improved the existing features; however, no major new functionalities have been introduced since version 0.4.0 (see detailed release history on <https://github.com/biodatageeks/sequila/tags>).

In the described current version (1.0.0), we have significantly extended the previously implemented functionality of the SeQuiLa package by introducing a novel distributed algorithm for summarizing reads using the Compact Idiosyncratic Gapped Alignment Report (CIGAR) strings and MD tags in a pileup format (SeQuiLa-pileup, **Algorithm 2**) (see **Table 1**). We also propose a custom data partitioning mechanism optimized to work with the alignment reads (**Algorithm 1**) as it significantly influences the performance of all subsequent steps by eliminating the need for data-exchange among partitions. At the same time, it enables single-pass over input data and lowers memory requirements since no intermediate data caching is required.

We have developed the algorithm in a modular way, enabling additional reduction of the execution time in two specific scenarios (when compared to regular SeQuiLa-pileup method), i.e. (i) pileup summary without information on basequalities (denoted as SeQuiLa-pileup-cov-only) and (ii) depth of coverage information only (denoted as SeQuiLa-pileup-cov-only). Since the functionality of SeQuiLa-pileup-cov-only is the same as the one provided by the previously published SeQuiLa-cov tool (Wiewiórka *et al.*, 2019) and the performance of the new algorithm (SeQuiLa-pileup-cov-only) is superior (see Section 3), we now recommend usage of SeQuiLa-pileup-cov-only for coverage calculations instead of SeQuiLa-cov while working with the current version of the SeQuiLa package (1.0.0).

Table 1. SeQuiLa package release history

Version (year)	Publication	Interval joins	Coverage	Pileup	Other features
0.3.0 (2018)	Wiewiórka <i>et al.</i> (2018)	<i>int</i>	—	—	—
0.4.0 (2019)	Wiewiórka <i>et al.</i> (2019)	<i>int</i>	<i>cov</i>	—	—
0.6.11 ^b (2021)	—	<i>int</i>	<i>cov</i>	—	—
1.0.0 (2022)	—	<i>int</i>	<i>pileup-cov-only</i> ^a	<i>pileup</i> ^a	<i>reads-aware partitioning</i> ^a , <i>cloud recipes</i> ^a

Note: *int*, interval joins (SeQuiLa-int); *cov*, coverage calculations (SeQuiLa-cov); *pileup-cov-only*, coverage calculations using simplified pileup algorithm (SeQuiLa-pileup-cov-only); *pileup*, pileup calculations (SeQuiLa-pileup).

^aNovel tools and features described in this manuscript.

^bTechnical update (Apache Spark update and fixed reported issues).

Besides the custom partitioner and pileup algorithm, the new version of the SeQuiLa package provides Terraform modules, Docker images, and code examples that facilitate straightforward deployment in the public clouds infrastructure.

2 Materials and methods

2.1 Rationale

The foundations for the distributed pileup algorithm are based on three key observations.

Firstly, the majority of bases in the aligned sequencing reads are concordant with the reference sequence. Therefore, we designed our algorithm to use both CIGAR strings, representing spliced alignment operations and MD tags, encoding mismatched and deleted reference bases, as defined in <https://samtools.github.io/hts-specs/SAMv1.pdf> and <https://samtools.github.io/hts-specs/SAMtags.pdf> accordingly. The use of the above mentioned strings in conjunction allows to handle deletions, insertions, and substitutions without decoding and parsing the entire read sequence and base qualities. To the best of our knowledge, it is the only algorithm that takes advantage of this information for pileup construction.

Secondly, our pileup computation is divided into four units of work: (i) coverage computation, (ii) identification of non-reference base calls, (iii) collection of base qualities and (iv) output projection. This decomposition allows us to reduce computational complexity by skipping certain steps that are not required.

Finally, the most limiting factor of performance and scalability for any distributed processing is the data exchange among the worker nodes that always requires costly data serialization as well as network transfer. Therefore, we propose a new data partitions coalescing mechanism, which guarantees proper handling of reads overlapping more than one partition without the need of data shuffling. In addition, we use BAM indexes for efficient partition boundaries adjustment and thus significantly reducing input/output (IO) operations.

2.2 Algorithm

2.2.1 Defining partitions

Consider an input sorted collection of the aligned sequencing reads R divided into n partitions by the underlying file system (**Fig. 1A**). The set of all partitions constitutes an immutable collection of data, i.e. resilient distributed dataset (RDD) which is the main logical unit of data in the Apache Spark framework.

For each partition, we calculate two values: lower and upper bounds that create self-contained virtual read partitions ($V_1 = lb_1 - ub_1$, $V_2 = lb_2 - ub_2$, etc, from **Fig. 1C, Algorithm 1**). For clarity, in pseudo-code we assume that all reads are aligned to a single chromosome.

This information is further required for changing the default Apache Spark partitioning schema (P_1, P_2, \dots, PN —see **Fig. 1A**), creating new coalesced partitions (C_1, C_2, \dots, CN —see **Fig. 1D**). Within each coalesced partition, the algorithm processes only the reads overlapping with the corresponding virtual partition (see

Algorithm 1 Reads-aware partitioning: Calculating lower(lb) and upper(ub) bounds for self-contained read partitions

Require: P , RDD (Resilient Distributed Dataset) containing all reads' partitions

```

1. for  $i \in (0, length(P) - 1)$  do
2.    $r(i) \leftarrow p(i)(0)$  get the first read in  $i$  partition
3.    $lb(i) \leftarrow r(i)_{start}$  get the lower bound of  $i$  partition
4. end for
5. for  $j \in (0, length(P) - 1)$  do construct interval tree of all reads overlapping any of  $lb(j)_{start}$ 
6.    $it \leftarrow IntervalTree(p(i).getReadsOverlapping(lb(j)_{start}))$ 
7. end for
8. for  $i \in (0, length(P) - 1)$  do
9.   if  $i \neq (length(P) - 1)$  then
10.     $ub(i)_{pos} \leftarrow \max(it.overlapers(lb(i + 1)_{pos}))$ 
11.   else
12.     $ub(i)_{pos} \leftarrow Int.MaxValue$ 
13.   end if
14.   if  $i > 0$  then
15.     $lb(i) \leftarrow ub(i - 1)$ 
16.   end if
17. end for
18. return ( $lb, ub$ )

```

Algorithm 2 SeQuiLa-pileup

Require: ref : Reference sequence
 $conf$: Configuration
 $PartitionSet$, RDD (Resilient Distributed Dataset) containing all reads' partitions

```

procedure CALCULATEPILEUP( $ref, conf$ )
2:   for  $p \in PartitionSet$  do
3.     aggregates := assembleAggregates( $p, conf$ )
4:
5.   generatePileupRecords(aggregates,  $ref, conf, p.lb, p.ub$ )
6.   end for
7: end procedure
procedure ASSEMBLEAGGREGATES( $partition, conf$ )
8:   agg := initAggregateForContig
9.   for  $read \in partition$  do
10.    agg.events := calculateCoverageEvents( $read$ )
11.    if  $conf.includeAlts$  then
12.      agg.alts += calculateAlts(agg,  $read, MDTag$ )
13.      agg.treeCache += createReadSummary(agg,  $read$ )
14.    end if
15.  end for
16: end procedure
procedure GENERATERECORDS( $aggregates, ref, conf, lb, ub$ )
18:   for  $a \in aggregates$  do
19.     for  $pos \in .ub$  do
20.       sum := cumulativeSum( $a.events, pos$ )
21.       if  $pos \geq lb$  and coverageChange or hasAlt then
22.         if  $conf.includeQuals$  then
23.           qual := calculateQuals( $a.treeCache, pos$ )
24.         end if
25.         createRow(sum,  $ref, a.alts, qual$ )
26.       end if
27.     end for
28.   end for
end procedure

```

Fig. 1E. Note that our approach is very lightweight—it dynamically calculates boundaries for virtual partitions, which can be considered as view upon original Spark partitions. Unlike GATK which divides input data into fixed length multikilobase-size pieces called *shards*, we do not introduce any other level of data splitting and push all computations down to the partition level.

2.2.2 Calculating coverage and alternative alleles

For each virtual partition, the program generates an aggregate object holding: (i) an array of alignment events (i.e. start and end of alignment) which is gathered for the event-based coverage calculations, (ii) a map of alternative bases count calculated using read MD tag and (iii) an interval tree structure of succinct read representation—*ReadSummary* (i.e. start, end, CIGAR derived configuration) used for further base qualities calculations. The program calculates base qualities only for the positions where at least one alternative base is present (Algorithm 2).

2.2.3 Merging and rendering the results

Once all reads in each virtual partition are analyzed, the program calculates the set of final pileup records. Depending on the configuration, our modular pileup algorithm can generate different outputs: full pileup (SeQuiLa-pileup), pileup without base qualities (SeQuiLa-pileup-no-qual) or depth of coverage only (SeQuiLa-pileup-cov-only).

2.3 Technical design

Our algorithm is implemented as a plugin to Apache Spark Catalyst optimizer (Ambrust *et al.*, 2015). We used its three extension points: (i) SQL Analyzer—to register new table-valued functions, (ii) Planner—to add our optimized execution strategies for pileup calculations and (iii) Logical Optimizer—to detect *CreateDataSourceTableAsSelectCommand* and *InsertIntoHadoopFsRelationCommand* actions and apply optimizations for direct vectorized writes into the Optimized Row Columnar (ORC) files (Fig. 2).

We designed the relational model to represent alignments and pileup function results as proposed in Sun *et al.* (2018) and Smith *et al.* (2021). Our package provides both SQL (Structured Query Language) and Dataframe programming interfaces for the Scala and Python (<https://github.com/biodatageeks/pysequila>) languages.

For reading Binary Alignment Map (BAM) and Compressed Reference-oriented Alignment Map (CRAM) files our solution can use Hadoop-BAM (Niemennma *et al.*, 2012) or disq libraries as configured by the end-user. For better support of the CRAM files that have been recently added to the HTSJDK library, we extended the Hadoop-BAM project (<https://github.com/biodatageeks/Hadoop-BAM>). Also, minor changes required for serialization of genomic intervals parameters were added to the disq (<https://github.com/mwiewior/disq>) library. For saving output we support not only ORC but also Parquet file format (Ivanov and Pergolesi, 2020). In our code, we re-used partition coalescing mechanism as implemented in the GATK.

2.4 Essential optimizations

Our main goal was to deliver a fast, distributed and scalable implementation of the pileup algorithm. In addition to the already presented novel algorithm, we highlight other essential implementation decisions that improve overall software performance. They can be grouped into three main categories: (i) optimization of distributed

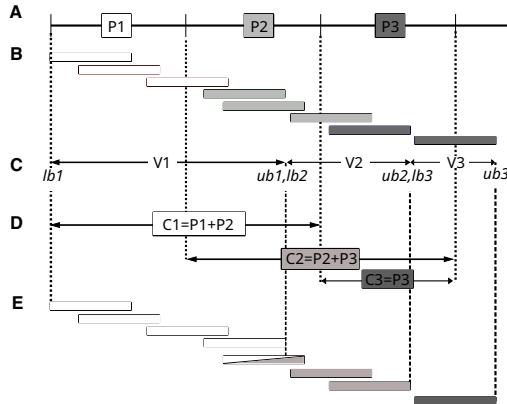


Fig. 1. Reads-aware partitioning algorithm: original distributed partitions (A); read assignment (color coded) to original partitions according to alignment starting position (B); virtual partitions and their boundaries calculated by [Algorithm 1](#) (C); coalesced partitions (D); and read assignment (color coded) to coalesced partitions and corresponding virtual partitions (E). Note that some of the reads will be processed in more than one coalesced partition. This approach produces on average equally sized virtual partitions (no data skewness) except for the first one and last one that are a bit larger and smaller than the rest, respectively

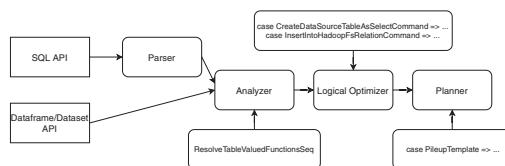


Fig. 2. SeQuiLa extensions to Apache Spark Catalyst optimizer

processing, (ii) Scala source code micro-optimizations, and (iii) output vectorization and fine-tuning.

2.4.1 Optimization of distributed processing

In the straightforward approach where the default partitioning is used, pileup implementation in Apache Spark would have to be split into two stages with a data shuffle step in between. This would require either explicit caching of the intermediate results from the first one or at least partial recomputing of the evicted partitions to get the final results. Implementation of a custom partitioning mechanism ([Algorithm 1](#)) to appropriately split data and determine the boundaries for each split was essential to achieve a single-pass solution without any extra data exchange between the executors or caching intermediary results.

2.4.2 Source code micro-optimizations

We have observed an apparent speedup when using an interval tree to store a short representation of reads (*ReadSummary*) since data retrieval from this structure is performed frequently with interval conditions. After analyzing the profiling results in a form of flame graphs obtained with *async-profiler* ([Nisbet et al., 2019](#)), we identified the most time-consuming and frequently invoked methods, such as calculation of the relative position in a read for a given genomic coordinate, and re-implemented them in the state-aware manner, thus eliminating traversing collection on each call. Similarly, we substituted computationally expensive CIGAR parsing and interpretation with fast lookups to lazily evaluated custom objects with derived cigar configuration with quick checks for existence of clip (and its length) or deletion, as well as deletion and insertion positions.

2.4.3 Output and auxiliary optimizations

The default output generation mechanism, which accounted for around 30% of the total processing time of our algorithm turned out to be another bottleneck. Therefore, we have implemented two novel approaches for optimizing output rendering.

Firstly, we have developed a custom direct pileup record projection to Apache Spark's internal binary row representation applying several micro-optimizations, e.g. casting reference bases and contig names to bytes and caching them in map to avoid repeating this task for each record. This mechanism can be used for further processing pileup rows within Spark-SQL engine as well as for persisting the results in any supported file format.

The second optimization is intended for improving performance of saving the results in ORC file format only. Inspired by the idea of *direct-path load* introduced in the relational database management systems, in particular in Oracle database ([Heller, 2019](#)), we implemented a mechanism that enables bypassing Spark's internal data representation and provides the support for vectorized row batches (as proposed in [Shen et al., 2021](#)) that are used for producing ORC output.

Other auxiliary optimizations including external dependencies configuration and environment setup were evaluated, and their impact on the overall performance is described in Section 3.

2.5 Cloud readiness

The increasing availability of cloud computing services for research is gradually changing the way scientific applications are developed, deployed and run ([Vaillancourt et al., 2020](#)). To ensure portability and reproducibility of SeQuiLa-based data processing, we followed the Infrastructure as Code ([Guerrero et al., 2019](#)) and DevOps principles for setting up the computing resources that can be used for both private and public clouds deployments. Hence, we have used technologies like Terraform (for cloud infrastructure provisioning, [Modi, 2021](#)), Helm (for deploying applications on Kubernetes clusters, [Shah and Dubaria, 2019](#)), and Docker (for application code packaging and shipment, [Boettiger, 2015](#)). SeQuiLa has been successfully deployed to both popular managed Hadoop services like Google Dataproc (utilized also in [Kriasaane et al., 2020](#)) and managed Kubernetes services like Google Kubernetes Engine (GKE), Azure Kubernetes Service, or Amazon Elastic Kubernetes Service. [Figure 3](#) presents an exemplary setup on GKE using the spark-on-k8s-operator and SeQuiLa application defined as a Kubernetes Custom Resource Definition. This architecture was suggested in [Castro et al. \(2019\)](#) as a preferred Apache Spark deployment scenario for scaling data analytics workloads and enabling efficient, on-demand utilization of resources in the cloud infrastructure. More detailed information on setup and corresponding Terraform modules can be found in the dedicated GitHub repository (<https://github.com/biodatageeks/sequila-cloud-recipes>).

2.6 Features

[Table 2](#) summarizes the features of the SeQuiLa package and compares them with state-of-the-art software, including samtools and GATK.

SeQuiLa-pileup operates on sorted aligned sequencing reads both in BAM and CRAM format. The fast pileup algorithm requires reads to have MD Tag attribute which can be determined during the alignment process or calculated and added to BAM files independently after alignment is completed. MD tag is described in <https://samtools.github.io/hts-specs/SAMtags.pdf> as a string encoding the mismatched and deleted reference bases, used in conjunction with the CIGAR and SEQ fields to reconstruct the bases of the reference sequence interval to which the alignment has been mapped. This can enable variant calling without requiring access to the entire original reference. Input files can be read either from the local file system, distributed file system or object storage using a custom data source which allows representing input reads as relational data. The dataset used for calculating the pileup can be restricted according to the

user-provided parameters including reads bit flag and mapping quality.

Samtools and GATK produce verbose output for every coordinate. On the contrary, our software implements lossless block compression of adjacent genomic positions which results in output an order of magnitude smaller. SeQuiLa-pileup result includes the genomic coordinates, reference bases, depth of coverage, the ratio of reference to non-reference bases, alternative bases (strand-aware) with occurrences counts and optionally the base qualities for the positions where at least one non-reference base is present. The output is stored in the popular big data-ready file formats such as ORC or Parquet making it easy to run further analyses, e.g. in Apache Spark or tools like Trino (Sethi *et al.*, 2019).

The SeQuiLa package is also distributed as a Python module (<https://github.com/biodatageeks/pysequila>) and can be used on local resources or cloud infrastructure. It can be easily integrated with widespread open-source notebook-based environments for data analysis including Google Colab and Jupyter.

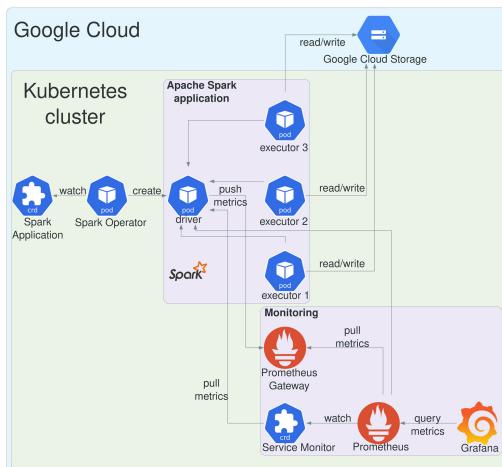


Fig. 3. SeQuiLa deployment on GKE with spark-on-k8s-operator with Kubernetes Custom Resource Definition, Prometheus for runtime metrics collection and Grafana as observability platform

3 Results

3.1 Datasets

We have used publicly available Exome Sequencing (ES) and Whole Genome Sequencing (WGS) datasets. We performed quality assurance tests on both short reads (sample NA12878) and long reads (guppy), represented in BAM and CRAM formats that were aligned to human reference genome GRCh38 with MD tags included.

3.2 Investigated solutions

Table 2 summarizes the functionalities of tools included in our comparison. Among the solutions included in the benchmark, only Spark-based GATK and SeQuiLa offer both multi-threaded and distributed versions of the depth coverage and pileup algorithms. For ADAM and SeQuiLa we used Apache Spark 3.1.2 runtime, in the case of GATK that does not provide support for Spark 3.x, we used Apache Spark 2.4.3. Megadepth, mosdepth and samtools 1.14 are multi-threaded applications but only the parts of their algorithms responsible for the IO operations (BZGF block compression/decompression) are parallelized—the remaining stages of their algorithm are sequential.

We have also included the previous version of SeQuiLa software (0.6.11) to assess the improvement of our single-pass and cache-less SeQuiLa-pileup-cov-only algorithm over the previously published SeQuiLa-cov. Several tools require additional input, i.e. genomic intervals in case of GATK's coverage and PaCBAM's (Valentini *et al.*, 2019) pileup or the list of genomic positions in case of aseq's (Romanel *et al.*, 2015) pileup that restricts the processed data and affects algorithm's computational complexity therefore the aforementioned solutions were not included in the final benchmark.

3.3 Testing environment

3.3.1 Single machine

Table 3 presents key information regarding the hardware and operating system configuration of the machine used for benchmark purposes. No hardware or software virtualization was used.

3.3.2 Hadoop cluster

Hadoop cluster (HDP 3.1.4) consists of 6 master and 34 worker nodes, 680 (1360 logical) cores, 700 TB of Hadoop File System (HDFS) disks, 6.8 TB of RAM for Yet Another Resource Negotiator (YARN) node pool and a 100 Gbits interconnect network. Master node specification was the same as in the single-node benchmark, in the case of workers the only difference was in disks configuration—

Table 2. Investigated solutions

Tool	Coverage	Pileup	MQF	SA	MT	DIST
samtools 1.9 (Li <i>et al.</i> , 2009)	+	+	+	+	-	-
samtools 1.14 (Danecik <i>et al.</i> , 2021)	+	+	+	+	+ (I/O for coverage)	-
GATK 4.2.3.0 (McKenna <i>et al.</i> , 2010)	+ (intervals)	+	-	-	-	-
GATK-Spark 4.2.3.0	+ (intervals)	+	-	-	+	+
ADAM 0.36.0 (Massie <i>et al.</i> , 2013)	+	-	-	-	+	+
megadepth 1.1.1 (Wilks <i>et al.</i> , 2021)	+	-	-	-	+ (I/O)	-
mosdepth 0.3.2 (Pedersen and Quinlan, 2018)	+	-	-	-	+ (I/O)	-
sambamba 0.8.1 (Tarasov <i>et al.</i> , 2015)	+	-	-	-	-	-
SeQuiLa 0.6.11 (Wiewiórka <i>et al.</i> , 2019)	cov	-	-	-	+	+
SeQuiLa 1.0.0	pileup-cov-only	pileup	+	+	+	+

MQF, Mapping Quality Filter; SA, strand-awareness; MT, multi-threaded; DIST, distributed.

Table 3. Technical specification—single node.

Processor	Base freq (GHz)	CPUs	Total cores (logical)	Memory (GB)	Operating system	Disk
Intel(R) Xeon(R) E5-2618L v4	2.20	2	20 (40)	256	RHEL 7.8 (Maipo)	3TB (RAID1)

each node has additional 12 disks in Just a Bunch of Disks setup for HDFS storage.

3.4 Performance testing scenarios and configuration

We have arranged four testing scenarios: (i) the pileup function performance on the local machine and (ii) its scalability characteristics on the Hadoop Cluster, (iii) the depth of coverage function performance on the local machine and (iv) its scalability on the Hadoop Cluster. All tools from Table 2 were included in the presented benchmarks. ES and WGS alignment datasets in the BAM format have been used as inputs. In the case of tools (ADAM, GATK and SeQuiLa) running on top of Java Virtual Machine (JVM) we used three distributions of Java Development Kit (JDK)—for a single node, we used GraalVM CE JDK8 for GATK (it does not support JDK11 yet) and GraalVM CE JDK11 for ADAM and SeQuiLa for running tests on the Hadoop cluster OpenJDK8 was used for all solutions. For tests using disq library, an additional BAM index was created.

3.5 Results pileup

In the pileup benchmarks (Fig. 4), SeQuiLa-pileup proved to be the fastest tool outperforming samtools in the single thread scenarios by $1.25x - 1.4x$ and GATK (both Spark, and non-Spark based) by $\approx 3.9 - 6.5x$ GATK (both Spark, and non-Spark based). In the case of Hadoop cluster benchmarks SeQuiLa-pileup again proved to be faster by $\approx 2.8 - 5.3x$ than GATK that also required twice as much memory (8 instead of 4GB) per Spark executor to be able to complete the computations. It is worth noting that we were unable to run GATK with 10 or fewer Spark executors (10 cores) facing errors related to too many opened files (even after increasing Linux

nofile limit to more than 1 million that is more than the recommended value for Hadoop clusters). We have verified that the algorithm's modularity is gainful when the user does not need to obtain the full pileup summary statistics. In particular, if reporting of base qualities is not required, SeQuiLa-pileup-no-qual that improves the performance by $\approx 35\%$ (compare SeQuiLa-pileup and SeQuiLa-pileup-no-qual in Fig. 4A) can be used. The correctness of the algorithm output was ensured by its rigorous comparison to the results of Samtools mpileup (v 1.14).

3.5.1 Java virtual machine optimizations

For development and local testing purposes, we have chosen GraalVM which uses an optimized compiler, generating high-performance code, and therefore noticeably accelerates the execution of the JVM-based applications (Sipek et al., 2020). Additionally, on the source code level, we have applied inlining annotations for frequently called concise methods which are further handled by the Scala compiler, thus avoiding the overhead of method invocation. In our diagnostic tests, we have confirmed that GraalVM choice results in 15% speedup while in-lining improved the timing by another 3% (Fig. 5).

3.5.2 Input–output optimizations

When performing direct vectorized writes into ORC files we have saved 18% of the computing time. We also take advantage of Intel's Genomics Kernel Library (GKL) providing high-performance operations of decompressing BAM file records. Our benchmarking confirms that the proper use of GKL's methods results in a 12% decrease of compute time (Fig. 5).

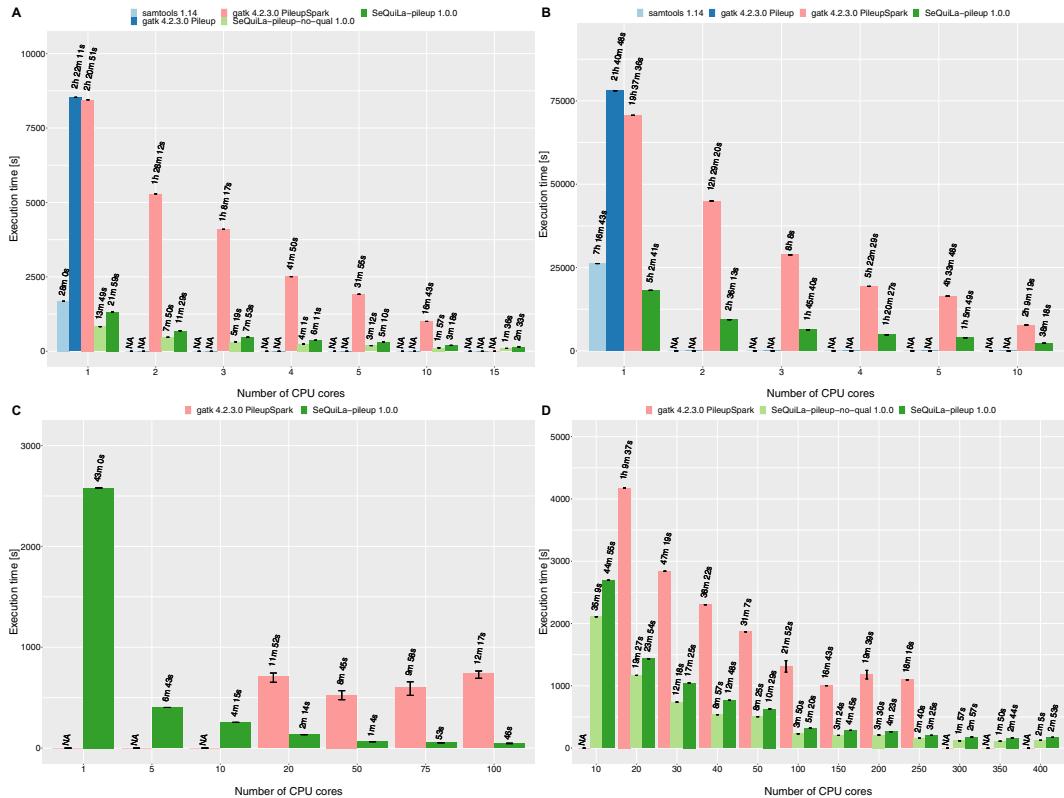


Fig. 4. Pileup summary function comparison. Tests were performed on a single node for ES (A), WGS (B), and on the Hadoop cluster for ES (C), WGS (D)

3.6 Results depth of coverage

In the case of both ES and WGS datasets (Fig. 6), megadepth proved to be the fastest tool in a single-machine single thread setup outperforming the second one, SeQuiLa-pileup-cov-only by 1.3–1.6×. The gap between them decreases steadily with an increasing

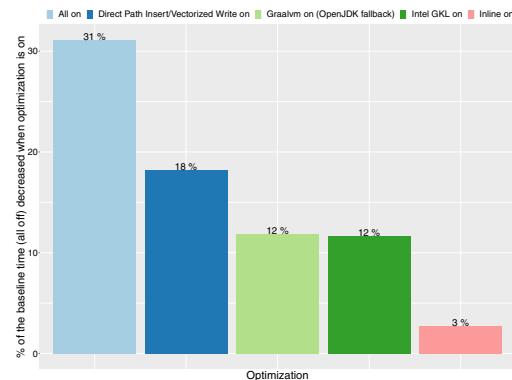


Fig. 5. Impact of various optimizations techniques on overall performance (percentage of time reduction) as compared with baseline (all optimizations off) for pileup computation. First bar shows the performance gain when all optimizations are on

number of threads. While processing ES and WGS data SeQuiLa-pileup-cov-only becomes the fastest tool when 5 and 10 threads are used, respectively. It is worth emphasizing that for the following tools: megadepth, mosdepth, and samtools, we observed very similar performance characteristics—in contrast to SeQuiLa-pileup-cov-only they do not scale up beyond 5–10 threads at all. These results confirm the fact that these tools only implement the parallel read and blocks decompression operations and the main part of their algorithms does not take advantage of multiple cores. For ADAM, we only measured the single-threaded performance that proved to be substantially worse than the remainder of the best performing tools ($\approx 40x$). Last but not least, we confirmed that the current version of the coverage calculations algorithm implemented in SeQuiLa 1.0.0 package (SeQuiLa-pileup-cov-only) is approximately 2× faster than our previous version of the coverage algorithm (SeQuiLa-cov) described in (Wiewiórka *et al.*, 2019) [compare SeQuiLa-pileup-cov-only (1.0.0) and SeQuiLa-cov (0.6.11) on Fig. 6A].

In the case of WGS on the Hadoop cluster (Fig. 6), we benchmarked tools allocating from 10 to 200 cores. SeQuiLa-pileup-cov-only outperformed ADAM on average by more than an order of magnitude (11–16x). We used the same memory (8GB—driver, 4GB executor) and Central Processing Unit configurations (1 core) for both Spark processes to ensure comparability of the results between solutions. Also Spark dynamic allocation mechanism has been explicitly disabled. In the case of ADAM tests, we observed random fails of Spark tasks (or even the whole stages) due to network timeouts.

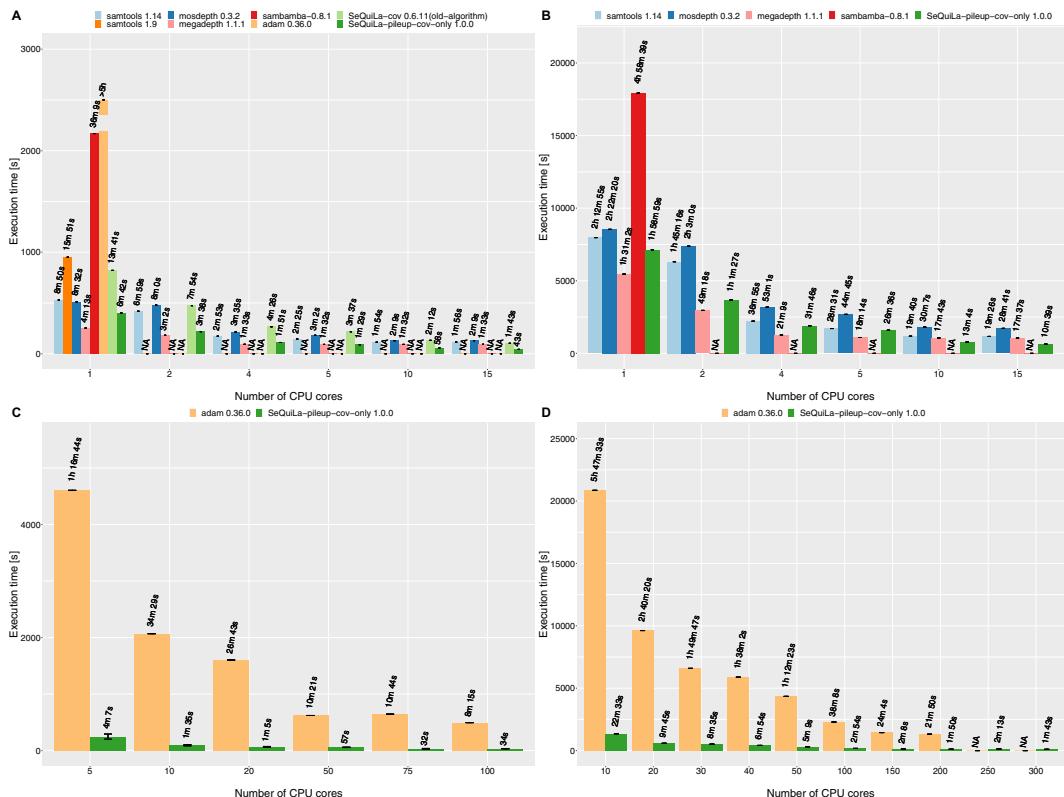


Fig. 6. Depth of coverage function comparison. Tests were performed on a single node for ES (A), WGS (B), and on the Hadoop cluster for ES (C), WGS (D). SeQuiLa-pileup designates the execution time of the full pileup calculations; SeQuiLa-pileup-no-qual indicates the execution time of the simplified pileup calculations in which base qualities are not computed

Discussion

Our pileup method is designed for compatibility with modern distributed computing systems originating from the Hadoop ecosystem mostly implemented using JVM languages such as Java or Scala. This approach incurs additional overheads and causes inefficiencies in the single-node deployments that justify why it does not significantly outperform samtools (a tool written in C language compiling to the native code) in a single thread comparisons. Our pileup algorithm performs especially well on the alignment files with the high-quality short reads when MD tags contain a relatively low number of mismatch/deleted bases. Calculating the complete pileup summaries from the long reads with a large number of mismatches is more challenging for our approach and requires additional modifications that we plan to introduce in its future versions. This limitation does not apply to calculations of the depth of coverage. It also favors BAM over CRAM alignment file formats (data not shown). This is because both the alignment file index scans (random access) as well as the sequential reads are much slower ($\approx 3 - 4x$) in the case of CRAM when compared to BAM file format (our results confirm the findings presented in Supplementary Material) (Bonfield et al., 2019). Finally, saving results in the distributed processing can be substantially reduced with adding support for direct, vectorized writes (currently available in the local mode) that is on our project roadmap as well.

Since the complexity of the cloud-native distributed computing systems have been acknowledged in many studies, including Vaillancourt et al. (2020), we have also prepared ready-to-use cloud deployment examples that can help users to start using SeQuiLa in public cloud environments.

4 Conclusions

We present a new module that extends and optimizes our SeQuiLa Apache Spark library. This component introduces a new algorithm for fast, scalable, and fully distributed computation of pileup summary from the alignment files (BAM, CRAM). Our solution combines a distributed computing engine based on the extended Apache Spark Catalyst query optimizer with the SQL interface for handling large-scale processing and analyzing next-generation sequencing datasets in a consistent tabular form. This approach will help to facilitate the adoption of scalable solutions among users that are neither proficient in distributed computing nor in cloud infrastructures as envisioned in Lawlor and Sleator (2020).

Funding

This work was supported by the (POB Cybersecurity and data analysis) of Warsaw University of Technology within the Excellence Initiative: Research University (IDUB) program. Grant's Principal Investigator: A.S.. The funding body did not play any role in the design of the study, the collection, analysis and interpretation of data or in writing of the article. Publication costs are funded by Warsaw University of Technology.

Conflict of Interest: none declared.

Data availability

The datasets supporting the results of this article are available upon request in Google Cloud Storage bucket: <gs://biodatageeks/sequila/data/> (for downloading using Google Cloud Storage compatible tool like gsutil) or <https://www.googleapis.com/storage/v1/b/biodatageeks/o?prefix=sequila/data/>. Project source code is publicly available (Apache License) at the GitHub platform at <https://github.com/biodatageeks/sequila>. Cloud deployments documentation and recipes are publicly available at the GitHub platform at <https://github.com/biodatageeks/sequila-cloud-recipes>. Detailed documentation is available on the project site at <https://biodatageeks.github.io/sequila/>.

References

- Ahmad,T. et al. (2021) VC@scale: scalable and high-performance variant calling on cluster environments. *GigaScience*, 10.
- Armbrust,M. et al. (2015) Spark SQL. In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, New York, NY, USA, pp. 1383–1394.
- Boettiger,C. (2015) An introduction to Docker for reproducible research. *SIGOPS Oper. Syst. Rev.*, 49, 71–79.
- Bonfield,J.K. et al. (2019) Crumble: reference free lossy compression of sequence quality values. *Bioinformatics*, 35, 337–339.
- Capuccini,M. et al. (2020) MaRe: processing big data with application containers on apache spark. *GigaScience*, 9.
- Castro,D. et al. (2019) Apache spark usage and deployment models for scientific computing. *EPJ Web Conf.*, 214, 07020.
- Danecek,P. et al. (2021) Twelve years of SAMtools and BCFtools. *GigaScience*, 10.
- Guerrero,M. et al. (2019) Adoption, support, and challenges of infrastructure-as-code: insights from industry. In: *2019 IEEE International Conference on Software Maintenance and Evolution (ICSM)*, Cleveland, OH, USA. IEEE, pp. 580–589.
- Guo,R. et al. (2018) Bioinformatics applications on apache spark. *GigaScience*, 7, 1–10.
- Heller,J. (2019) Modify data with advanced DML. In: *Pro Oracle SQL Development*. Apress, Berkeley, CA, pp. 191–218.
- Ivanov,T. and Pergolesi,M. (2020) The impact of columnar file formats on SQL-on-hadoop engine performance: a study on ORC and parquet. *Concurr. Comput. Pract. Exper.*, 32.
- Koboldt,D.C. et al. (2012) VarScan 2: somatic mutation and copy number alteration discovery in cancer by exome sequencing. *Genome Res.*, 22, 568–576.
- Koppad,S. et al. (2021) Cloud computing enabled big multi-omics data analytics. *Bioinform. Biol. Insights*, 15, 1177932211035921.
- Krissaane,I. et al. (2020) Scalability and cost-effectiveness analysis of whole genome-wide association studies on Google Cloud platform and Amazon Web Services. *J. Am. Med. Inform. Assoc.*, 27, 1425–1430.
- Lawlor,B. and Sleator,R.D. (2020) The democratization of bioinformatics: a software engineering perspective. *GigaScience*, 9, 1–3.
- Li,H. (2011) A statistical framework for SNP calling, mutation discovery, association mapping and population genetic parameter estimation from sequencing data. *Bioinformatics*, 27, 2987–2993.
- Li,H. et al.; 1000 Genome Project Data Processing Subgroup. (2009) The sequence alignment/map format and SAMtools. *Bioinformatics*, 25, 2078–2079.
- Liu,Y. et al. (2021) Psi-Caller: a lightweight short read-based variant caller with high speed and accuracy. *Front. Cell Dev. Biol.*, 9, 731424.
- Luo,R. et al. (2020) Exploring the limit of using a deep neural network on pileup data for germline variant calling. *Nat. Mach. Intell.*, 2, 220–227.
- Massie,M. et al. (2013). Adam: genomics formats and processing patterns for cloud scale computing. *Technical Report*, EECS Department, University of California, Berkeley.
- McKenna,A. et al. (2010) The genome analysis toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Res.*, 20, 1297–1303.
- Modi,R. (2021) Deep-dive into terraform. In: *Deep-Dive Terraform on Azure*. Apress, Berkeley, CA, pp. 77–113.
- Nieminenmaa,M. et al. (2012) Hadoop-BAM: directly manipulating next generation sequencing data in the cloud. *Bioinformatics*, 28, 876–877.
- Nisbet,A. et al. (2019). Profiling and tracing support for Java applications. In: *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*. ACM, New York, NY, USA, pp. 119–126.
- Pedersen,B.S. and Quinlan,A.R. (2018) Mosdepth: quick coverage calculation for genomes and exomes. *Bioinformatics*, 34, 867–868.
- Romanel,A. et al. (2015) ASEQ: fast allele-specific studies from next-generation sequencing data. *BMC Med. Genomics*, 8, 9.
- Sater,V. et al. (2020) UMI-gen: A UMI-based read simulator for variant calling evaluation in paired-end sequencing NGS libraries. *Comput. Struct. Biotechnol. J.*, 18, 2270–2280.
- Sethi,R. et al. (2019) Presto: SQL on everything. In: *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, Macao, China. IEEE, pp. 1802–1813.
- Shah,J. and Dubaria,D. (2019) Building modern clouds: using Docker, Kubernetes & Google Cloud Platform. In: *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, Las Vegas, NV, USA. IEEE, pp. 0184–0189.
- Shen,Y. et al. (2021) Using vectorized execution to improve SQL query performance on spark. In: *50th International Conference on Parallel Processing*. ACM, New York, NY, USA, pp. 1–11.

- Sipek,M. *et al.* (2020) Enhancing performance of cloud-based software applications with GraalVM and Quarkus. In: 2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO), Opatija, Croatia. IEEE, pp. 1746–1751.
- Smith,J. *et al.* (2021) Scalable analysis of multi-modal biomedical data. *GigaScience*, 10.
- Sun,X. *et al.*; CAAPA consortium. (2018) Optimized distributed systems achieve significant performance improvement on sorted merging of massive VCF files. *GigaScience*, 7.
- Tarasov,A. *et al.* (2015) Sambamba: fast processing of NGS alignment formats. *Bioinformatics*, 31, 2032–2034.
- Vaillancourt,P. *et al.* (2020). Reproducible and portable workflows for scientific computing and HPC in the cloud. In: *Practice and Experience in Advanced Research Computing*. ACM, New York, NY, USA, pp. 311–320.
- Valentini,S. *et al.* (2019) PaCBAM: fast and scalable processing of whole exome and targeted sequencing data. *BMC Genomics*, 20, 1–5.
- Wiewiórka,M. *et al.* (2018) SeQuiLa: an elastic, fast and scalable SQL-oriented solution for processing and querying genomic intervals. *Bioinformatics*, 35, 2156–2158.
- Wiewiórka,M. *et al.* (2019) SeQuiLa-cov: a fast and scalable library for depth of coverage calculations. *GigaScience*, 8.
- Wiewiórka,M.S. *et al.* (2017) Benchmarking distributed data warehouse solutions for storing genomic variant information. *Database*, 2017.
- Wilks,C. *et al.* (2021) Megadepth: efficient coverage quantification for BigWigs and BAMs. *Bioinformatics*, 37, 3014–3016.
- Yuan,D.Y. and Wildish,T. (2020). Bioinformatics application with Kubeflow for Batch Processing in Clouds. In: *Bioinformatics Application with Kubeflow for Batch Processing in Clouds*. Springer, Cham, pp. 355–367.
- Zaharia,M. *et al.* (2010) Spark: cluster computing with working sets. In: *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, Boston, MA, USA, Vol. 10, p10.

CHAPTER 7

Co-Authors' statements

- [P1] M.S. Wiewiórka, A. Messina, A. Pacholewska, S. Maffioletti, P. Gawrysiak, and M.J. Okoniewski. SparkSeq: Fast, scalable and cloud-ready tool for the interactive genomic data analysis with nucleotide precision. *Bioinformatics*, 30(18), 2014, **Page: 157**
- [P2] M.S. Wiewiórka, D.P. Wysakowicz, M.J. Okoniewski, and T. Gambin. Benchmarking distributed data warehouse solutions for storing genomic variant information. *Database : the journal of biological databases and curation*, 2017, **Page: 161**
- [P3] Anastasiia Hryhorzhevska, Marek Wiewiórka, Michał Okoniewski, and Tomasz Gambin. Scalable Framework for the Analysis of Population Structure Using the Next Generation Sequencing Data. In *Lecture Notes in Computer Science (including sub-series Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 10352 LNAI, pages 471–480. 2017, **Page: 164**
- [P4] Marek Wiewiórka, Anna Leśniewska, Agnieszka Szmurło, Kacper Stępień, Mateusz Borowiak, Michał Okoniewski, and Tomasz Gambin. SeQuiLa: an elastic, fast and scalable SQL-oriented solution for processing and querying genomic intervals. *Bioinformatics*, 35(12):2156–2158, June 2019, **Page: 167**
- [P5] Marek Wiewiórka, Agnieszka Szmurło, Wiktor Kuśmirek, and Tomasz Gambin.

SeQuiLa-cov: A fast and scalable library for depth of coverage calculations. *Giga-Science*, 8(8), August 2019, **Page:** [171](#)

- [P6] Marek Wiewiórka, Agnieszka Szmurło, Paweł Stankiewicz, and Tomasz Gambin. Cloud-native distributed genomic pileup operations. *Bioinformatics*, December 2022, **Page:** [174](#)

Warszawa, 15.11.2021

dr hab. inż. Michał Okoniewski
ID Scientific IT Services
ETH Zürich

OŚWIADCZENIE

Niniejsze oświadczenie dotyczy mojego wkładu w publikacji:

1. Wiewiórka, M. S., Messina, A., Pacholewska, A., Maffioletti, S., Gawrysiak, P., & Okoniewski, M. J. (2014). SparkSeq: fast, scalable and cloud-ready tool for the interactive genomic data analysis with nucleotide precision. *Bioinformatics*, 30(18), 2652-2653.

Mój wkład polegał na stworzeniu wraz z Markiem Wiewiórką koncepcji systemu, pomocy w uruchomieniu prototypu oprogramowania i napisaniu części artykułu.



Michał Okoniewski

Zürich, December 18th 2022

To Whom It May Concern:

This letter is to certify my contribution to the following publication of which I was co-author:

- 1) Wiewiórka, M. S., Messina, A., Pacholewska, A., Maffioletti, S., Gawrysiak, P., & Okoniewski, M. J. (2014). SparkSeq: fast, scalable and cloud-ready tool for the interactive genomic data analysis with nucleotide precision. *Bioinformatics*, 30(18), 2652-2653.

My input included help in formulating the infrastructure requirements on the on-premise ScienceCloud infrastructure, supervise the provisioning and the tuning of the underlying virtualized resources.

Sincerely,

Dr. Sergio Maffioletti
ID Scientific IT Services
OCT G 35
Binzmühlestrasse 130
8092 Zürich

A handwritten signature in black ink, appearing to read "Sergio Maffioletti".

Zurich, December 18th 2022

To Whom It May Concern:

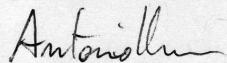
This letter is to certify my contribution to the following publication of which I was co-author:

- 1) Wiewiórka, M. S., Messina, A., Pacholewska, A., Maffioletti, S., Gawrysiak, P., & Okoniewski, M. J. (2014). SparkSeq: fast, scalable and cloud-ready tool for the interactive genomic data analysis with nucleotide precision. *Bioinformatics*, 30(18), 2652-2653.

My input included help in automation of provisioning as well as tuning of virtual resources running on top of the OpenStack cluster required for performance evaluation of SparkSeq tool.

Sincerely,

Antonio Messina



Warszawa, 18.12.2022

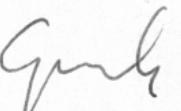
dr hab. inż. Piotr Gawrysiak, profesor uczelni,
Instytut Informatyki,
Wydział Elektroniki i Technik Informacyjnych
Politechnika Warszawska

OŚWIADCZENIE

Niniejsze oświadczenie dotyczy mojego wkładu w publikacji:

1. Wiewiórka, M. S., Messina, A., Pacholewska, A., Maffioletti, S., Gawrysiak, P., & Okoniewski, M. J. (2014). SparkSeq: fast, scalable and cloud-ready tool for the interactive genomic data analysis with nucleotide precision. *Bioinformatics*, 30(18), 2652-2653.

Mój wkład polegał na współudziale w zarządzaniu pracami badawczymi, krytycznej analizie wyników oraz redakcji tekstu publikacji.



Piotr Gawrysiak

Warszawa, 18.12.2022

Dawid Wysakowicz

OŚWIADCZENIE

Niniejsze oświadczenie dotyczy mojego wkładu w publikacji:

1. Wiewiórka, M. S., Wysakowicz, D. P., Okoniewski, M. J., & Gambin, T. (2017). Benchmarking distributed data warehouse solutions for storing genomic variant information. *Database*, Volume 2017, 2017, bax049.

Mój wkład polegał na:

- zaimplementowaniu i opisaniu generatora danych SNV symulującego całe sekwencje eksonów
- zaimplementowaniu, przetestowaniu i opisaniu wyników zapytań w silnikach Apache Spark, Presto i Apache Hive
- przetestowaniu i opisaniu wpływu formatów danych Apache ORC i Parquet na zapytania
- współzaprojektowaniu i opisaniu modelu danych oraz wypełnieniu tabel danymi.

Dawid Wysakowicz
Wysakowicz

Warszawa, 15.11.2021

dr hab. inż. Michał Okoniewski
ID Scientific IT Services
ETH Zürich

OŚWIADCZENIE

Niniejsze oświadczenie dotyczy mojego wkładu w publikacji:

1. Wiewiórka, M. S., Wysakowicz, D. P., Okoniewski, M. J., & Gambin, T. (2017). Benchmarking distributed data warehouse solutions for storing genomic variant information. *Database*, Volume 2017, 2017, bax049.

Mój wkład polegał na pomocy w stworzeniu koncepcji oprogramowania i pomocy w pisaniu artykułu.



Michał Okoniewski



Warszawa, 03.03.2023

dr hab. inż. Tomasz Gambin, prof. uczelni
Instytut Informatyki,
Politechnika Warszawska

OŚWIADCZENIE

Niniejsze oświadczenie dotyczy mojego wkładu w publikacji:

1. Wiewiórka, M. S., Wysakowicz, D. P., Okoniewski, M. J., & Gambin, T. (2017). Benchmarking distributed data warehouse solutions for storing genomic variant information. *Database*, Volume 2017, 2017, bax049.

Mój wkład polegał na sformułowaniu problemu, nadzorowaniu projektu oraz współudziale w przygotowaniu manuskryptu.



Tomasz Gambin

Leuven, December 24th 2022

To Whom It May Concern:

This letter is to certify my contribution to the following publication of which I was a co-author:

- 1) Hryhorzhevska, A., Wiewiórka, M., Okoniewski, M., & Gambin, T. (2017, June). Scalable framework for the analysis of population structure using the next generation sequencing data. In *International Symposium on Methodologies for Intelligent Systems* (pp. 471-480). Springer, Cham.

My input included

1. The development of the bioinformatics pipelines for the analysis of various pre-processing methods and their parameters on the final results of the classification and clustering algorithms for the population structure studies;
2. Interpretation of data and results presented in the publication;
3. Manuscript drafting and preparation for publication.

Sincerely,



Anastasiia Hryhorzhevska
Max Planck Institute of Psychiatry
Kraepelinstr. 2-10
80804 München

Warszawa, 15.11.2021

dr hab. inż. Michał Okoniewski
ID Scientific IT Services
ETH Zürich

OŚWIADCZENIE

Niniejsze oświadczenie dotyczy mojego wkładu w publikacji:

1. Hryhorzewska, A., Wiewiórka, M., Okoniewski, M., & Gabin, T. (2017, June). Scalable framework for the analysis of population structure using the next generation sequencing data. In *International Symposium on Methodologies for Intelligent Systems* (pp. 471-480). Springer, Cham.

Mój wkład polegał na pomocy w napisaniu części artykułu.



Michał Okoniewski

Warszawa, 03.03.2023

dr hab. inż. Tomasz Gambin, prof. uczelnii
Instytut Informatyki,
Politechnika Warszawska

OŚWIADCZENIE

Niniejsze oświadczenie dotyczy mojego wkładu w publikacji:

1. Hryhorzhevska, A., Wiewiórka, M., Okoniewski, M., & Gambin, T. (2017, June). Scalable framework for the analysis of population structure using the next generation sequencing data. In *International Symposium on Methodologies for Intelligent Systems* (pp. 471-480). Springer, Cham.

Moim wkładem było sformułowanie funkcjonalności skalowalnego narzędzia do obliczeń z zakresu genetyki populacyjnej umożliwiającego automatyczną kalibrację parametrów. Nadzorowałem implementację i ewaluację zaproponowanego rozwiązania. Uczestniczyłem w pisaniu manuskryptu oraz ostatecznej wersji artykułu.



Tomasz Gambin

Warszawa, 03.03.2023

mgr inż. Agnieszka Szmurło
Instytut Informatyki,
Politechnika Warszawska

OŚWIADCZENIE

Niniejsze oświadczenie dotyczy mojego wkładu w publikacji:

1. Wiewiórka, M., Leśniewska, A., Szmurło, A., Stępień, K., Borowiak, M., Okoniewski, M., & Gambin, T. (2019). SeQuiLa: an elastic, fast and scalable SQL-oriented solution for processing and querying genomic intervals. *Bioinformatics*, 35(12), 2156-2158.

Mój wkład polegał na częściowej implementacji algorytmu, realizacji testów jakościowych, opracowaniu figur oraz przygotowaniu tekstu manuskryptu.

Agnieszka Szmurło



Warszawa, 03.03.2023

dr hab. inż. Tomasz Gambin
Instytut Informatyki,
Politechnika Warszawska

OŚWIADCZENIE

Niniejsze oświadczenie dotyczy mojego wkładu w publikacji:

1. Wiewiórka, M., Leśniewska, A., Szmurło, A., Stępień, K., Borowiak, M., Okoniewski, M., & Gambin, T. (2019). SeQuiLa: an elastic, fast and scalable SQL-oriented solution for processing and querying genomic intervals. *Bioinformatics*, 35(12), 2156-2158.

Moim wkładem był współudział w sformułowaniu funkcjonalności narzędzia oraz wskazanie potencjalnych zastosowań. Uczestniczyłem również w przygotowaniu i interpretacji wyników testów oraz przygotowaniu części manuskryptu.



Tomasz Gambin

Warszawa, 18.12.2022

dr inż. Anna Leśniewska
Instytut Informatyki,
Politechnika Poznańska

OŚWIADCZENIE

Niniejsze oświadczenie dotyczy mojego wkładu w publikacji:

1. Wiewiórka, M., Leśniewska, A., Szmurło, A., Stępień, K., Borowiak, M., Okoniewski, M., & Gambin, T. (2019). SeQuiLa: an elastic, fast and scalable SQL-oriented solution for processing and querying genomic intervals. *Bioinformatics*, 35(12), 2156-2158.

Mój wkład polegał na kierowaniu projektem naukowym obejmującym badania opisane w tej pracy, w szczególności implementacji indeksu wpływającego na proces optymalizacji zapytań przedziałowych, wykonaniu testów oraz przygotowaniu fragmentu tej publikacji.

Anna Leśniewska



Warszawa, 15.11.2021

dr hab. inż. Michał Okoniewski
ID Scientific IT Services
ETH Zürich

OŚWIADCZENIE

Niniejsze oświadczenie dotyczy mojego wkładu w publikacji:

1. Wiewiórka, M., Leśniewska, A., Szmurło, A., Stępień, K., Borowiak, M., Okoniewski, M., & Gambin, T. (2019). SeQuiLa: an elastic, fast and scalable SQL-oriented solution for processing and querying genomic intervals. *Bioinformatics*, 35(12), 2156-2158.

Mój wkład polegał na pomocy w stworzeniu koncepcji oprogramowania, udziale w testowaniu i napisaniu części artykułu.



Michał Okoniewski

Warszawa, 18.12.2022

dr inż. Wiktor Kuśmirek
Instytut Informatyki,
Politechnika Warszawska

OŚWIADCZENIE

Niniejsze oświadczenie dotyczy mojego wkładu w publikacji:

1. Wiewiórka, M., Szmurło, A., Kuśmirek, W., & Gambin, T. (2019). SeQuiLa-cov: a fast and scalable library for depth of coverage calculations. *Gigascience*, 8(8), giz094.

Mój wkład polegał na częściowym przygotowaniu manuskryptu.



Wiktor Kuśmirek

Warszawa, 03.03.2023

mgr inż. Agnieszka Szmurło
Instytut Informatyki,
Politechnika Warszawska

OŚWIADCZENIE

Niniejsze oświadczenie dotyczy mojego wkładu w publikacji:

1. Wiewiórka, M., Szmurło, A., Kuśmirek, W., & Gambin, T. (2019). SeQuiLa-cov: a fast and scalable library for depth of coverage calculations. *Gigascience*, 8(8), giz094.

Mój wkład polegał na częściowej implementacji algorytmu, współludziale w przygotowaniu koncepcji optymalizacji i analize formalnej rozwiązania oraz realizacji testów jakościowych, opracowaniu figur i przygotowaniu tekstu manuskryptu.

Agnieszka Szmurło



Warszawa, 03.03.2023

dr hab. inż. Tomasz Gambin, prof. uczelni
Instytut Informatyki,
Politechnika Warszawska

OŚWIADCZENIE

Niniejsze oświadczenie dotyczy mojego wkładu w publikacji:

1. Wiewiórka, M., Szmurło, A., Kuśmirek, W., & Gambin, T. (2019). SeQuiLa-cov: a fast and scalable library for depth of coverage calculations. *Gigascience*, 8(8), giz094.

Moim wkładem był współudział w sformułowaniu funkcjonalności narzędzia oraz wskazanie potencjalnych zastosowań. Uczestniczyłem również w przygotowaniu i interpretacji wyników testów oraz przygotowaniu części manuskryptu.



Tomasz Gambin

Warszawa, 03.03.2023

mgr inż. Agnieszka Szmurło
Instytut Informatyki,
Politechnika Warszawska

OŚWIADCZENIE

Niniejsze oświadczenie dotyczy mojego wkładu w publikacji:

1. Wiewiórka, M., Szmurło, A., Stankiewicz, P., & Gambin, T. (2022). Cloud-native distributed genomic pileup operations. *Bioinformatics (Oxford, England)*, btac804. Advance online publication. <https://doi.org/10.1093/bioinformatics/btac804>

Mój wkład polegał na częściowej implementacji algorytmu, współludziale w przygotowaniu koncepcji optymalizacji i analize formalnej rozwiązania oraz realizacji testów jakościowych, opracowaniu figur i przygotowaniu tekstu manuskryptu.

Agnieszka Szmurło



Houston, 12.01.2023

prof. dr hab. n. med. Paweł Stankiewicz
Department of Molecular and Human Genetics,
Baylor College of Medicine,
Houston, TX 77030, USA

OŚWIADCZENIE

Niniejsze oświadczenie dotyczy mojego wkładu w publikacji:

1. Wiewiórka, M., Szmurło, A., Stankiewicz, P., & Gambin, T. (2022). Cloud-native distributed genomic pileup operations. *Bioinformatics (Oxford, England)*, btac804. Advance online publication. <https://doi.org/10.1093/bioinformatics/btac804>

Mój wkład polegał na współpracy w przygotowaniu tekstu manuskryptu

Z poważaniem,

Paweł Stankiewicz



Warszawa, 03.03.2023

dr hab. inż. Tomasz Gambin, prof. uczelnii
Instytut Informatyki,
Politechnika Warszawska

OŚWIADCZENIE

Niniejsze oświadczenie dotyczy mojego wkładu w publikacji:

1. Wiewiórka, M., Szmurło, A., Stankiewicz, P., & Gambin, T. (2022). Cloud-native distributed genomic pileup operations. *Bioinformatics (Oxford, England)*, btac804. Advance online publication. <https://doi.org/10.1093/bioinformatics/btac804>

Moim wkładem był współludział w sformułowaniu funkcjonalności narzędzia oraz wskazanie potencjalnych zastosowań. Uczestniczyłem również w przygotowaniu i interpretacji wyników testów oraz przygotowaniu części manuskryptu.



Tomasz Gambin