UNIVERSITY OF TARTU

Institute of Computer Science

Software engineering

Kristjan Metsalu

Web Application to Calculate Genetic Risk Scores Based on Imputed Data

Master thesis (30 ECTS)

Supervisors: Prof. Jaak Vilo

Reedik Mägi PhD

Tatjana Iljashenko MSc

Tartu 2016

Web Application to Calculate Genetic Risk Scores Based on Imputed Data

Abstract

The falling cost of genotyping has made feasible to include genetic information to national healthcare system. Estonia is one of the few countries that have great potential of converting this information into a everyday tool for clinicians, who then would be able to make more informed decisions related to their patient's health. Estonian Genome Center, University of Tartu (EGCUT) is one of the institutions that is working on creating new risk prediction models based on genetic information. Researchers in EGCUT have created models to evaluate the risk for polygenic disorders. Current thesis focuses on development of a software that would enable fast application of these risk prediction models to collected genetic data and visualization of the results together with clinical data.

CERCS:

B110 - Bioinformatics, medical informatics, biomathematics, biometrics

Keywords:

Scala, Play framework, Akka, MySQL, Apache Spark, Apache Cassandra, compression, imputed data, polygenic risk prediction models.

Veebipõhine lahendus imputeeritud andmete põhjal geneetiliste riskiskooride arvutamiseks

Lühikokkuvõte

Viimastel aastatel on genotüpiseerimise hinna langus teinud võimalikuks geneetilise informatsiooni lisamise tervishoiusüsteemi. Eesti on üks vähestest riikidest, kellel on võimekus muuta see informatsioon arstide jaoks igapäevaseks tööriistaks, kes saaksid seeläbi teha paremini informeeritud otsuseid oma patsientide kohta. Tartu Ülikooli Eesti Geenivaramu (EGV) on üks asutustest, kes töötavad geneetilise info põhjal uute haigusriskide ennustamise mudelite kallal. Teadustöö EGV-s on loonud erinevaid mudeleid polügeensete haiguste riskide hindamiseks. Selle magistritöö käigus esitame tarkvara, mis võimaldab ennustusmudelite kiiremat väljatöötamist ja arenduse käigus tehtud katsetusi.

CERCS:

B110 - Bioinformaatika, meditsiiniinformaatika, biomatemaatika, biomeetrika

Võtmesõnad:

Scala, Play framework, Akka, MySQL, Apache Spark, Apache Cassandra, pakkimismeetodid, imputeeritud andmed, polügeensed riskiskoorid.

Table of Contents

1 Introduction 6

1.1 Motivation 6

1.2 Layout of the thesis 6

2 Domain description 8

2.1 Biological background 8

2.2 Genomic data 8

2.3 Genetic disorders 10

2.4 Genetic risk evaluation for multifactorial disorders 11

2.5 Double weighted polygenic risk score calculation with Impute2 data 13

3 Development of system architecture 15

3.1 Compression algorithm selection 15

3.2 Integration of parallelized risk score calculation 17

4 Description of the created software 26

4.1 Overview of used software 26

4.2 Components of the created solution 29

4.3 Similar works on processing imputed genomic data 39

4.4 Comparison with similar software 39

Conclusions ..... **Error! Bookmark not defined.**

Bibliography 42

Appendixes 45

Appendix 1. Apache Spark program for lzo file reading. 45

Appendix 2. Quality control and marker effect calculation 47

Appendix 3. Database model 50

Appendix 4. Spark cluster configuration with hadoop-lzo 58

Appendix 5. MySQL configuration 59

Appendix 6. Source code 60

Appendix 7. License..61

1 Introduction

The genomics research over the past decades has yielded a growing number of sequence variants associated with health conditions that may have clinical utility. Despite some unresolved problems, like the large proportion of "missing heritability", attention is turning to strategies that integrate this data into clinical care. Large biobanks with an opportunity to contact their participants, ongoing access to participants health records and permission to intervention can be a a good model to study this clinical utility [1]. Estonian Genome Center, University of Tartu (EGCUT) is a population-based biobank founded in 2001. Since its start in 2001, EGCUT has collected biological samples and electronical questionnaires for more than 52 thousand adult volunteers from all 15 counties in Estonia [2] covering 5% of the entire population in Estonia [3]. Volunteers signed a broad agreement form giving EGCUT permission to gather additional info from different national health registries and permission to re-contact them in the future [4]. There is genotyping microarray data available for 20,000 individuals and in 2015 governmental funding enabled sequencing whole genomes of 2 400 samples with high coverage, to support the use of genetic research in attempt to improve public health.

1.1 Motivation

The population health related research requires not only data availability, but also software, enabling effective use of this data. The EGCUT data bank contains structured clinical data together with risk prediction models developed by EGCUT researchers, but is still missing a software for combining those resources into a one working environment to support collaboration of different interest groups. The aim of current thesis is to create a software solution for combining existing data with scientific results into a tool enabling calculation of disease risk and visualization of results in user-friendly format.

1.2 Layout of the thesis

Thesis is divided into introduction, 3 main chapters followed by summary, bibliography and appendixes.

Chapter 2 focuses on the theoretical background of calculating risk for genetic disorders. We introduce different subgroups of genetic disorders, determine the motivations to use imputed

genomic data in the context of genetic risk calculation. In the end of the chapter more detailed description is given for calculating polygenic risk scores based on imputed genomic data.

In the Chapter 3 we describe the development of architecture for current solution, aimed to improve usability of imputed genomic data for polygenic risk score calculation.

Chapter 4 starts with an overview of frameworks used in the created software then describes the main components of the solution and finishes with an overview of similar solutions in the area of improving usage of imputed genomic data and genetic risk evaluation.

Appendixes contain manual to setup Apache Spark with LZO compression support, example code to use Apache Spark for working with LZO compression, example code how the calculation of a single marker effect is done, created database model and MySQL configuration file.

2 Domain description

This Chapter aims to give background information to better understand genomic data used in this thesis and an overview of different types of genetic disorders. Chapter ends with a more detailed description of implemented genetic risk evaluation model for multifactorial genetic disorders.

2.1 Biological background

Human genome is the genetic material of an organism, that is coded by four nucleotides: cytosine (C), guanine (G), adenine (A), or thymine (T). Human genome is divided into chromosomes that are the double helix molecules, where the chains(strands) of a helix are held together by weak thermodynamic forces. Those forces always connect the A with T and G with C, this is known as a principle of complimentary. The chromosomes in human genome are paired (excluding the X and Y chromosomes), having maternal and paternal copies. The nucleotides observed in a certain genomic position (i.e. locus) of both chromosome copies define the alleles in this position. Both alleles (i.e. pair of alleles) define the genotype value in that position(s).

In this thesis we describe genetic differences between individuals based on single nucleotide polymorphisms (SNPs). SNP is a genetic variation with a minor allele frequency at least 1% in population in which a single nucleotide (A, T, C or G) in the genome differs between members of a biological species or paired chromosomes [5]. Additionally, the current thesis only uses bi-allelic SNPs, thus each SNP has three possible genotype values. For example, if we have allele values A and B, the genotype in that position can be AA, AB, and BB. These genotypes are called as homozygous to allele A, heterozygous and homozygous to allele B.

2.2 Genomic data

Figure 1 gives visual representation of genomic coverage of different genomic data. *Reference genome* is a digital nucleic acid sequence database, assembled by scientists as a representative example of a humans' set of genes [6]. Next generation sequencing technologies often use reference genome in a process called alignment where during sequencing, DNA is split into smaller fragments called reads and later these reads are combined into a *whole sequence* [7]. During comparison of genetic data is important to know which version of reference genome was used during alignment since any errors in the reference genome affect the quality of

aligned data. *Whole sequences* would be the best representation of human DNA to study genomic differences between individuals. *Whole genome sequences* cover approximately 98% of the genome, that is about 3 Billion different base pairs[1] consisting of complimentary nucleotides A-T or C-G. Rest of the sequence is difficult to align with current technologies. Unfortunately, costs for generating full sequences is still at least a degree more expensive than for *genotyped data*. Software solution created for current thesis is using *imputed data*, that is derived analytically based on reference haplotypes and statistical algorithms.

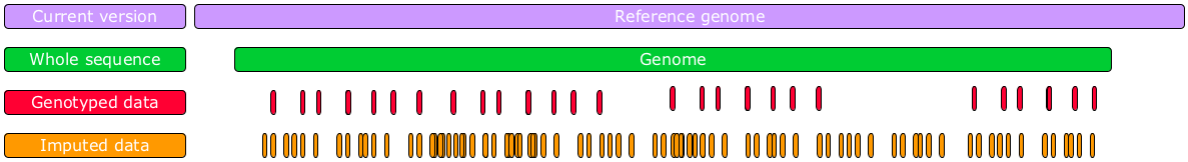


Figure 1 - Types of genetic data.

Genotyped data is a fixed set of positions from the whole genome. These positions are usually selected from previously identified SNPs. The number of genotyped SNPs on a DNA microarray can vary from few hundreds to few millions depending on microarray[2] selected.

2.2.1 Imputed data

Genotype imputation is now an essential tool in the analysis of genome-wide association scans. The technique allows geneticists to accurately evaluate the evidence for association at genetic markers that are not directly genotyped [8]. This can be done due to linkage disequilibrium (LD) between genetic variants, which are located close to each other in a chromosome. Using phased genotyped data and sequenced reference haplotypes covering the same area, we can impute missing positions. As there can be several different reference haplotypes fitting into our genotype data, imputed positions are described as probabilities of genotypes: P(AA), P(AB), P(BB).

EGCUT has used Illumina OMNI microarray for genotyping with 770 thousand positions. By using 1000Genomes[3] reference panel, the number of variants increases to 38 million.

[1] https://tandem.bu.edu/knex/base.pairs.knex.html
[2] http://en.wikipedia.org/wiki/DNA_microarray
[3] http://www.1000genomes.org/data

2.3 Genetic disorders

2.3.1 Monogenic disorders [9]

There are many studies describing the monogenic disorders and methods used for risk prediction. Monogenic diseases are divided into subgroups, such as autosomal, X-linked, mitochondrial inheritance and imprinted genes. Those disorders are resulted by a single defective gene, where the autosomal diseases occur on autosomal[4] chromosomes and X-linked – on the X chromosome. Mitochondrial diseases, also known as mitochondrial or maternal inheritance, occur within the genes in mitochondrial[5] DNA. Diseases caused by imprinting of genes differ from previously described by the fact that their effect depends, which parent this gene is inherited.

There are more than 4000 known diseases caused by single-gene defects [10]. Some examples of the monogenic diseases would be cystic fibrosis, Huntington's disease, hemophilia and red-green color blindness.

Risk evaluation for monogenic diseases involves the detection of know causal mutations in the DNA followed by the evaluation of the effect of those mutations.

2.3.2 Chromosomal aberrations

The disorders that arise due to the absence of a particular chromosome, presence of an additional chromosome or due to structural anomalies (deletions, duplication or translocation) in a chromosome are termed chromosomal disorders [11].

Chromosomal aberrations usually have severe and life-long symptoms. One of the most know structural chromosomal aberration is Down syndrome, caused by trisomy of all or part of human chromosome 21 [12]. Williams syndrome for example, is the result of a deletion of the 7q11.23 region of chromosome 7 containing the elastin gene and is believed to be a contiguous gene syndrome [13].

2.3.3 Multifactorial disorders

Multifactorial disorders, also known as complex or polygenic disorders involve many genetic variants across the genome and are often coupled with environmental factors. Figure 2 shows a multifactorial disease model. Since birth, every person has a risk for a disease based on his

[4] Autosome is a chromosome that is not a sex chromosome [26].
[5] Cell organelle

or her genetic variants. During life, environmental factors, will raise or lower the probability of disease manifestation.

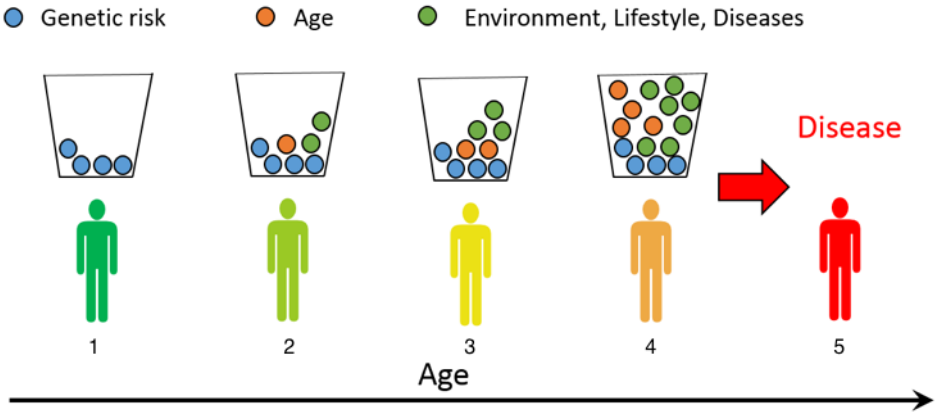


Figure 2 - Polygenic disorder (visualization created by Krista Fischer[6]). Stage 1 – At birth individual has some genetic background, which determines part of the disease risk (genetic predisposition). Stages 1-4 – during life individual is exposed to certain environment and he/she makes different which can increase the disease risk (environmental factors). State 5 – If certain amount of disease causing factors have accumulated a disease manifests.

With multifactorial disorders genetic factors cause disease manifestation in one person and not in another if they both make almost identical lifestyle choices. For this type of diseases knowledge about genetic disorders can motivate individual to correct their lifestyle and reduce the risk of illness.

One of the most studied polygenic diseases is Type 2 Diabetes (T2D). It is estimated that in 2015 there was about 39,7 million people with diabetes living in USA, causing 373.7 billion dollars in medical and indirect societal cost [14]. There are several well-known factors identified for T2D like age, sex, obesity and central obesity and low physical activity that are actively used for prediction [15]. Adding genetic risk to existing prediction models can help with a more targeted prevention and bring down prevention costs [16].

2.4 Genetic risk evaluation for multifactorial disorders

One way to calculate genetic risk for polygenic disorders is using polygenic risk score, where SNPs are combined into a score (GRS_j). There are two commonly used methods to calculate polygenic risk score. Unweighted method expressed as a function,

$$GRS_j = \sum_{i=1}^{N} allele\_dosage_{i,j}$$

[6] https://www.etis.ee/Portal/Persons/Display/7c9ac8d0-9216-422f-80c1-8a8e2740198b?tabId=CV_ENG

and weighted method expressed as a function,

$$GRS_j = \sum_{i=1}^{N} \hat{\beta}_i * allele\_dosage_{i,j}$$

where GRS_j is the polygenetic risk score for given individual j, N – the total number of the markers, i – index of the marker in the model, $\widehat{\beta_i}$- estimated regression parameter obtained from GWAS study for marker i. $Allele\_dosage_{i,j}$ – calculated allele dosage.

For genotyped data we can count the $allele\_dosage_{i,j}$ for bi-allelic SNPs AA, AB and BB as 0, 1, 2 or 2, 1, 0 depending on whether we want the count for A or B. When imputed data $G_{i,j}$ is used then $allele\_dosage_{i,j}$ for B allele is defined as

$$allele\_dosage_{i,j} = 0 * P(AA)_{i,j} + 1 * P(AB)_{i,j} + 2 * P(BB)_{i,j}$$

In a pre-published study researchers showed for T2D [16], that weighted GSR_j can be further improved by reducing bias caused by a "winners curse". A "winners curse" phenomena can be observed, when one systematically selects SNPs with effect overestimated by chance. In this case $\widehat{\beta_i}$ is represented as

$$\widehat{\beta_i} = \widehat{\beta_{i,orig}} * Coef_i$$

Where $\beta_{i,orig}$ is the weight estimation from the original GWAS study and $Coef_i$ probability that the SNP belongs to the set of k SNPs with strongest effect size (k can be selected according to some estimate of the number of SNPs with true effect on the phenotype). An estimate $\hat{\pi}_i(k)$ for this probability is obtained by sampling new estimates for β_i from a simulated normal distribution $N(\beta_i , c\hat{s}_i^2)$ for each i and empirically estimating the proportion of times each SNP belongs to the "top k" SNPs using a Wald type statistic. In created software we implemented algorithm to calculate double weighted polygenic risk scores. Same algorithm can be used to calculate single-weighted method by replacing $Coef_i$ with 1.0 and when unweighted method is needed $\hat{\beta}_i$ with 1.0.

After determining GSR_j for every individual, there is a second issue to be solved, estimation of thresholds that separate individuals into groups with similar susceptibility for the disorder. In the created software we divide individuals into three groups Low, Medium and High risk,

corresponding to scores ranging from lowest to average, average to highest decile and highest decile. There are also methods where these regions are reported as deciles [17].

2.5 Double weighted polygenic risk score calculation with Impute2 data

The process of GRS calculation consists of 4 main operations: 1 - combine markers defined in the model with imputed data, 2- calculate single marker effect for all of the individuals, 3 - sum calculated effects across different markers for each individual and 4 - store the results.

This process has two inputs, a table for the model with structure shown in Figure 3 and imputed genotypes in Impute2 format [18]. IMPUTE2 formatted genotype data is stored using *zlib* compression algorithm to about 60 gigabytes for 8117 individuals.

Name	Effect allele	Ref allele	Chromosome	Position	Beta	Coefficient	EAF
rs40	A	G	1	1000	0.08	1	0.9
rs60	T	C	10	12456789	0.1	0.1	0.1

Figure 3 – Table – Markers used in a model. Name – name of the marker. Effect allele – allele causing the effect. Ref allele – second allele for that marker. Chromosome - chromosome of marker in the initial GWAS. Position – position of the marker in the initial GWAS. Beta – Beta estimation in the initial GWAS. Coefficient – probability in double weighted polygenic risk score. EAF – Effect allele frequency in initial GWAS.

During risk score calculation the effect allele and strand have to be matched between model markers and imputed data before single marker effect can be calculated.

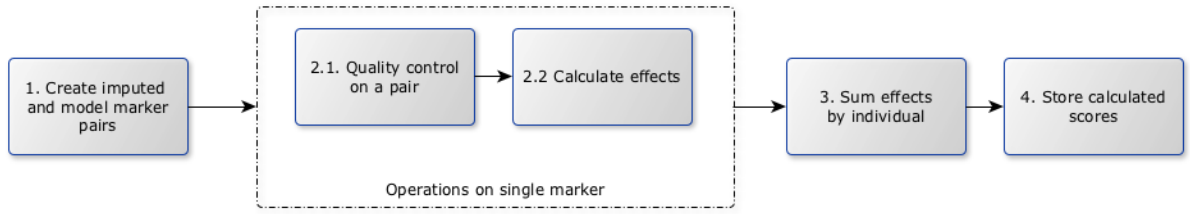


Figure 4 - Risk calculation pipeline. 1 – Combine together markers from the model and imputed data. 2.1 – verify the quality of this pair [Appendix 2. 23-52]. 2.2 – calculate marker effect for each individual [Appendix 2. 88-134]. 3 – Sum individual's marker effects across different markers. 4 – Store results.

In this pipeline we decided to integrate this operation into the calculation pipeline as a quality control step because we did not want to change any of the inputs for the reason we cannot

predict future inputs. Overview of the calculation pipeline is shown in Figure 4. Code example, given in Appendix 2 contains all the details.

3 Development of system architecture

The objectives for this thesis were to find a solution to speed-up polygenic risk calculation and also provide better access to EGCUT data as well as simple interpretation of results for the non-expert user. Objective is to free time spent by researchers on calculations that can be automated and parallelized to interpreting the calculated scores or work on how created predictions should be presented in a clinical context.

The first problem encountered after providing users the means to store new polygenic risk models in the created software, was joining markers in the model with their counterparts in imputed data during calculations. This step requires fast random access to the imputed markers from all over the genome and the current storage method was not optimized for this task.

The second issue was how to manage these calculations in an environment where many of them could be run in parallel. This would cause overuse of system resources and block the work for everyone working on the system.

In this chapter we show the research we did to improve our system in an effort to find ways to optimize different parts of the system with a focus on risk score calculation. Exact implementation of software is detailed in Chapter 4.

3.1 Compression algorithm selection

Filtering imputed markers from a single chromosome file takes about 20min and in total 8hours if done sequentially. The next step would be to parallelize by the number of chromosome files, but 20min was still a lot of time considering the system may contain several models each needing different markers. These operations would then have to run one after another since they otherwise would consume all the system resources since the operation is CPU bounded. Investigation into compression algorithms revealed that *zlib* is not optimized for fast decompression and random access.

Zlib library uses DEFLATE compression [19]. DEFLATE compression is standardized by RFC1951 and it states that DEFLATE combines LZ77 algorithm and Huffman coding. Each block consists of two parts: a pair of Huffman code trees that describe the representation of the compressed data part, and a compressed data part [20]. Usage of Huffman code trees gives

it one of the best compression ratios, but also makes it more complex and increases decompression time.

Snappy[7], *lzo*[8] and *lz4*[9] use LZ77 algorithm with each having their own implementation of storing offsets to previously decompressed data. Out of these formats *snappy* does not have any command line tools and we decided not to spend any time to investigate it further.

Lzo is an older and more stable implementation compared to the others. Command line tests show that *lzo* is remarkably faster than *zlib*, 12 min on one chromosome in the command-line and in total 4.6h if done sequentially. *Lzo* compression ratio is worse than *zlib* and the size of genomic data will increase from 60GB to 137GB.

lz4 has some incompatibilities between different programming languages and tools because of its fast development pace. *lz4* is remarkably faster compared to the others with only 8 min extraction time from one chromosome.

From these results that also supported the results published by Yann Collet a developer of *Lz4* (Figure 5) new compression algorithm selection was feasable.

Compressor	Ratio	Compression	Decompression
memcpy	1.000	4200 MB/s	4200 MB/s
LZ4 fast 17 (r129)	1.607	**690 MB/s**	**2220 MB/s**
LZ4 default (r129)	**2.101**	**385 MB/s**	**1850 MB/s**
LZO 2.06	2.108	350 MB/s	510 MB/s
QuickLZ 1.5.1.b6	2.238	320 MB/s	380 MB/s
Snappy 1.1.0	2.091	250 MB/s	960 MB/s
LZF v3.6	2.073	175 MB/s	500 MB/s
zlib 1.2.8 -1	2.730	59 MB/s	250 MB/s
LZ4 HC (r129)	**2.720**	22 MB/s	**1830 MB/s**
zlib 1.2.8 -6	3.099	18 MB/s	270 MB/s

Figure 5 - Compression and decompression speeds overview[10]. Ratio describes how many times the data is compressed compared to the original.

[7] https://google.github.io/snappy
[8] http://www.oberhumer.com/opensource/lzo
[9] https://cyan4973.github.io/lz4
[10] https://github.com/Cyan4973/lz4

3.2 Integration of parallelized risk score calculation

Apache Spark[11] is fast and general engine for large-scale data processing. It can run some programs as much as 100 times faster compared to a similar MapReduce solution *Apache Hadoop*. Compared to Hadoop HDD intensive operations, Spark runs most of its operations in-memory and this allows writing of more diverse programs. Spark also includes components for stream processing, machine learning and graph processing – all of which are very hard if not impossible to accomplish with Hadoop.

Spark is built in Scala and it provides API's for developing in Java, Scala, Python and later versions R. Since Scala and Python both include REPL, it is possible to experiment on a large-scale data before starting to work on a fully packaged solution. It also supports many of the popular compression algorithms.

3.2.1 Apache Spark with zlib

Testing Spark with *zlib* compression to filter 10K markers resulted an operation time of 40min and the process is CPU bounded. We determined this by comparing the read speed of GPFS filesystem that is about 600MB/s and for Spark 100-200MB/s.

3.2.2 Apache Spark with lzo

Testing Spark with LZO compression we were able to achieve better filtering time of 18 min despite the actual files being more than double the size and achieving 500MB/s in filesystem throughput.

3.2.3 Apache Spark with lz4

We did not pursue any further with Apache Spark and lz4 support since we encountered errors when trying to use command-line tools together with Apache Spark's built-in Lz4 support. Data compressed with lz4 command-line tools was not readable in Spark and other way around.

3.2.4 Apache Spark with lzo indexing

Spark was also tested together with *hadoop-lzo*[12] an open-sourced library from Twitter that enables indexing of compression blocks within *lzo* files. This indexing enables highly parallelized decompression in Hadoop based solutions because decompression can work on

[11] https://spark.apache.org
[12] https://github.com/twitter/hadoop-lzo

different parts of the file simultaneously. Setup of this solution turned out to be difficult because of the lack of information. A setup manual is now included in Appendix 4.

After cluster setup and indexing of *lzo* files, filtering task runs in 5 minutes using a relatively simple Scala[13] program shown in Appendix 1. Because of an initial preference to use Python[14] language, we also tested the same operation in Python language. Python program turned out to be 2-3 times slower and took 13 minutes to complete. The cause could be that Spark is written in Scala and use of Python requires conversions between languages which makes it slower.

3.2.5 Apache Spark as a web service with cache

After some experimentation with Spark, Scala was selected as our main programming language for the reasons explained more in [4.1.1]. Apache Spark was integrated directly into the application code and using Spark cache enabled making risk calculations work in 20 seconds. In regards of achieved speed and programming effort this solution was suitable and on the server memory requirements were not an issue.

Spark is not meant to run this way and during building a part that keeps the client notified about ongoing progress, changes in the source required constant solution restarts. Solution restart would then trigger a read through the imputed data and cost at least 5 minutes.

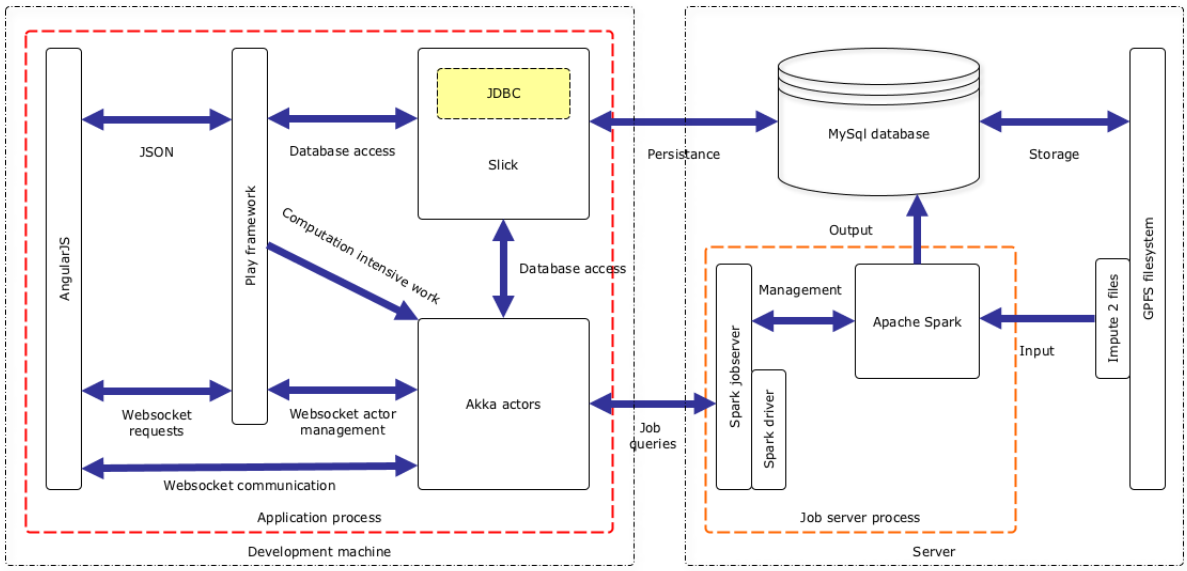


Figure 6 – Apache Spark with spark-jobserver. All of the computation expencive

[13] http://www.scala-lang.org
[14] https://www.python.org

To separate Spark from the web solution *spark-jobserver*[15] an open-source project seemed a good solution. Working with *spark-jobserver* meant that MySQL server and Spark ran on the server and application development could be done locally. Score calculation code was packaged into *jar* format and deployed to job server over HTTP protocol. Managing of calculations was done over HTTP. This architecture is illustrated in Figure 6. After completion, development experience became even worse. Each time part of the algorithm changed in score calculation, recompilation and deployment to *spark-jobserver* was needed. In many cases this resulted a restart of *spark-jobserver,* triggering Spark to read through the imputed data. In our opinion a type of solution where Spark is included into the application server or is managed by a *spark-jobserver* should only be considered on stable processes with experienced supporting team and not in constantly changing development environment.

Our testing server is also used for other purposes and small file operations only few megabytes per second with little CPU usage did not work well with Apache Spark. The throughput of *lzo* indexed file reads dropped from 600MB/s to 100MB/s and even 24MB/s, which causes 25-fold increase in reading data into Spark. This can be explained by Spark using small cache files on the filesystem to save intermediate progress, but attempts to reduce this effect have not yielded any good results. This caused the need to move away from the file system and test databases to store imputed genotypes.

3.2.6 Apache Spark + Apache Cassandra[16] for storage

Apache Cassandra is a column-family type NoSQL database. It supports automatic *lz4* compression, has Spark driver[17], can be embedded into JVM applications and provides fast data access [21]. In the testing environment Cassandra together with Spark was set up as a 3-node cluster and is not 100% comparable with previous setups. Figure 7 illustrates how the testing cluster was constructed. Using Cassandra yielded positive results for risk calculation with 45 seconds to access and calculate scores, using about 128GB of disk space. There was also the benefit of nearly instant random access to the imputed data. Importing data to Cassandra took around 3 hours.

[15] https://github.com/spark-jobserver/spark-jobserver
[16] https://cassandra.apache.org
[17] https://github.com/datastax/spark-cassandra-connector

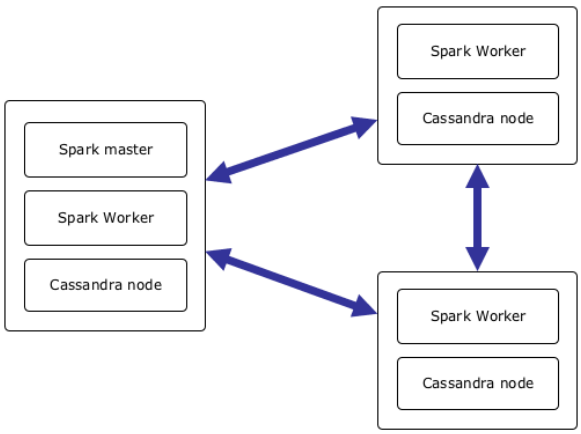


Figure 7 - Cassandra cluster setup with three 2x Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz, 256GB 2133MHz RAM, 600GB SCSI HDD

This solution showed us that it was the correct path to take Impute2 data from the file system and move it into a database letting the database engine control filtering and joins. At this point we had Apache Spark, *spark-jobserver* together with Apache Cassandra and it was the most difficult systems to maintain. This setup was troubled by Apache Cassandra stability issues where nodes from the cluster were shutting down halting the system and issues described in 3.2.5.

3.2.7 MySQL and Akka framework[18] for decompression

Akka is a toolkit and runtime for building highly concurrent, distributed and resilient message driven applications on the JVM. Akka is built on top of Netty[19] that enables asynchronous IO in JVM. Single machine can run millions of Actors (building block in Akka system), that communicate with tens of millions of messages. From these units you can create actor systems with complex hierarchies that are able to span across multiple machines to distribute workload.

Akka is used in Apache Spark and Play framework for internal work distribution and after inspecting Akka documentation[20] we were convicted that this framework could work for our needs. Integration with Play framework made it much easier to use compared to Spark and we could configure it to run computationally intensive work separately from the Play framework threads (that we implemented) or use clustering capabilities. Actor model[21] provided by Akka

[18] http://akka.io
[19] http://netty.io
[20] http://doc.akka.io/docs/akka/2.3.15/scala.html
[21] http://worrydream.com/refs/Hewitt-ActorModel.pdf

in our opinion gives better abstractions over parallelization than using Spark and its abstractions over collections.

MySQL[22] is an open-source relational database management system now owned by the Oracle Corporation. In EGCUT it is already running to store full sequences for Ensembl[23] database. In Ensembl database genomic data is stored as database BLOB objects and Perl[24] pack method[25] is used to handle compression in the application side[26]. Our idea was to use something similar, but with a more conventional compression algorithm.

3.2.8 Lz4 HC compression

To test the usability of MySQL for our case, *Lz4 HC* compression algorithm was selected because of it being one of the fastest decompression algorithms together with good compression ratio. We selected implementation by Adrien Grand in his package *lz4-java*[27]. To perform compression Apache Spark is used, since *lz4 HC* compression is time consuming and and we already had the cluster setup. In the current solution compression is used on the part of the line in gen file, that contains marker probabilities and everything else is stored as structured columns.

Adding data to MySQL takes about 1 hour and the end size for a table with MyISAM engine is about 75GB and for InnoDB 160GB. In MySQL we created a BTREE index on the table based on the RS-numbers to enable faster filtering. InnoDB and MyISAM have different characteristics when access by single marker is needed. In our case InnoDB is faster if access to a single marker is needed, but using multiple markers like in our case there are no obvious effects depending on the MySQL storage engine.

End result was 10K markers from the database within 120 seconds when combined with Akka framework. This method also simplified the whole solution and the end result is shown on Figure 8.

[22] https://www.mysql.com
[23] http://www.ensembl.org/index.html
[24] https://www.perl.org
[25] http://perldoc.perl.org/functions/pack.html
[26] http://feb2014.archive.ensembl.org/info/docs/api/variation/variation_schema.html#compressed_genotype_var
[27] https://github.com/jpountz/lz4-java

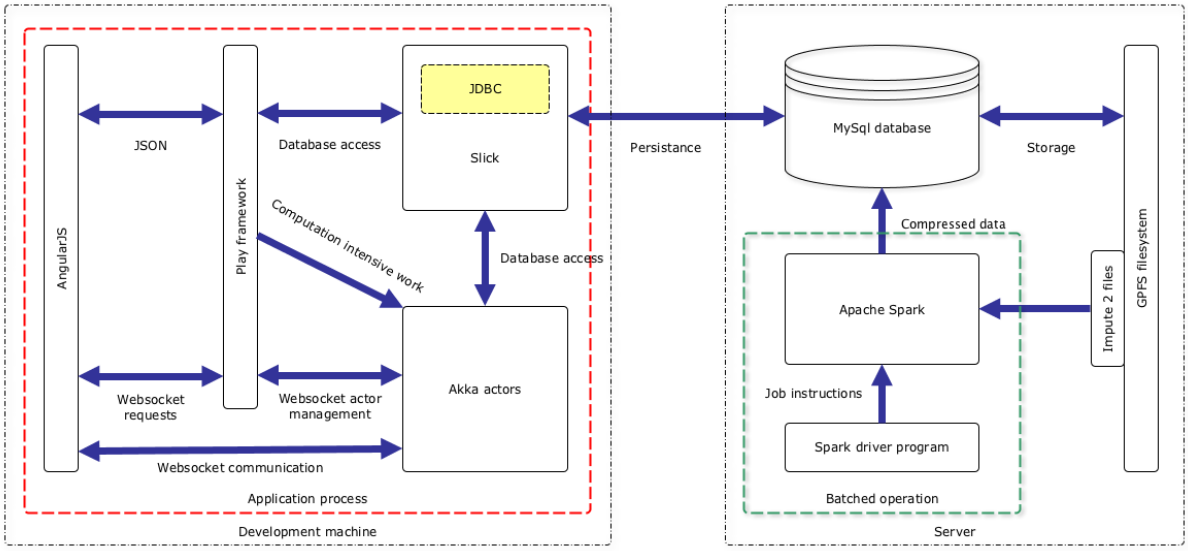


Figure 8 - Final architecture. Application code is separated from Spark and can run in development computer or on production server. Spark is used in a batched mode to insert new impute sets.

3.2.9 MySQL with Lz4 HC and data reduction

Experiments with Spark showed that risk score calculation for 10K markers can be done in 20s and MySQL solution with *Lz4* compression took longer. After a data reduction method illustrated in Figure 9 was applied to the source data, additional throughput was gained.

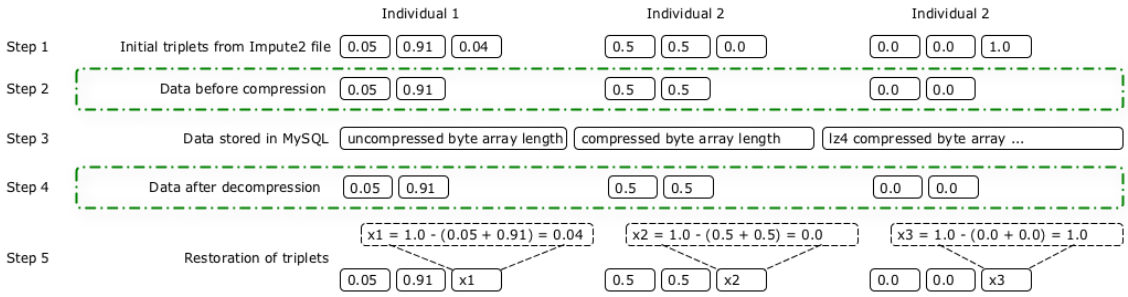


Figure 9 – Compression and decompression of a single line of Impute2 data. Step 1 - is an array of triplets from Impute2 file. Step 2 - every third item from each pair is removed. Step 3 - Data is compressed stored in the database using LARGEBLOB for the binary data. Step 4 – data is decompressed after retrieval from the database. Step 5 – Triplet is restored using the principle that the sum genotype probabilities is 1.0.

Change illustrated in Figure 9 gave additional 10% reduction in database size. The final table size was 67 gigabytes for MyISAM engine and 123GB InnoDB. Total time to calculate to calculate scores dropped to around 40 seconds depending on system usage. The speed gain is the result of smaller amount of data transferred and decompressed. After some modification in MySQL configuration the end result was 10 seconds if the web application and database were

on the same 40 core server and 40 seconds when transferring data from the server to the development machine. Final MySQL configuration is given in Appendix 5

The choice of whether to use InnoDB or MyISAM for this type of storage solution depends on system requirements. MyISAM produces smaller size, but also causes the loss of foreign keys. InnoDB provides transactional integrity and foreign keys, but costs additional storage capacity.

Summary of conducted experiments is given in Table 1 and Table 2

Table 1 - Processing times[28]

Name	Data size	Single marker	10K markers	Complexity (subjective)	Comments
Starting point *gzip*	60GB	7h	7h	Simplest	Nothing needs to be developed
lzo compression	138GB	4.8h	4.8h	Simplest	Nothing needs to be developed
Apache Spark + gzip	60GB	40min	40min	Simple	Spark reads gzip files, no additional setup
Apache Spark + lzo	138GB	18min	18min	Complex	Native libraries needed + native dependencies needed for Spark and custom code
Apache Spark + lzo indexing	138GB	5min	4min	Complex	For the same reasons as 4. Indexing is simple after Spark configuration
Apache Spark cached	138GB	-	20s	Very Complex	It is hard to integrate this into application code and using other m
Apache Spark + Cassandra	128GB	Instantly	60s	Very Complex	Cassandra cluster maintenance requires effort, new modelling methodology, database drivers are not as stable. Different machines
MySQL with lz4 HC	74GB	Instantly	120s	Medium	Medium, custom code for compression and decompression + MySql access.
MySQL with lz4 HC and data reduction	68GB	Instantly	40s	Medium	Medium, custom code for compression and decompression + MySql access. Pairs usage is simple to implement.
Final solution after MySQL optimization	68GB	Instantly	10s	Medium	Configuration in given in Appendix 5

[28] 4x Intel(R) Xeon(R) CPU E7- 8850 @ 2.00GHz, 1024GB 1066MHz RAM, 898GB ServeRAID M5014 local HDD and 831TB GPFS HDD

Table 2 - Resource utilization. Network represents file system usage since we were using networked file system.

Name	CPU	Memory	Network	Comments
Starting point *gzip*	100%	4MB	20MB/s	-
lzo compression	100%	6MB	70MB/s	-
Apache Spark + gzip	2100%	24GB	200MB/s	-
Apache Spark + lzo	2100%	24GB	500MB/s	Tested with entire set since 1 file did not give sufficient results
Apache Spark + Hadoop-lzo indexing	3900%	40GB	24 – 800MB/s	Tested with entire set since 1 file did not give sufficient results
Apache Spark cached	3900%	130GB	-	Data is memory cached
Apache Spark + Cassandra	4800%	-	-	Different machines (Figure 7)
Custom lz4 HC solution using MySql	3900%	6GB	-	Memory needed for the web server for calculations.
Custom lz4 HC solution using MySql + Data reduction	3900%	6GB	-	Memory needed for the web server for calculations.
After MySQL optimization	3900%	6GB + 122GB	-	Memory needed for the web server for calculations. 100GB is memory given to MySQL

4 Description of the created software

In this chapter we describe the software, used in current solution as well as provide the overview of the main components of current version. We also consider the way to perform observational analysis of calculated risk scores, combine results with individual's health information and visualize it in appropriate way. In the end of chapter similar works on processing imputed genomic data and genetic risk evaluation software are also discussed. Appendix 3 contains the database model and Appendix 6 the source code for the application.

4.1 Overview of used software

4.1.1 Programming languages

Scala is relatively mature language running on top of Java Virtual Machine (JVM). Development of Scala started in 2003 by Martin Odersky and his research team in EPFL and now it has gotten one of the most used languages for building Big Data solutions.

Scala is an open-sourced multi-paradigm and type safe programming language that enables both functional and object oriented style of programming. Compared to Java, Scala has less verbose syntax to accomplish the same tasks. Libraries written in Java are also usable in Scala, vice versa is sometimes not possible due to the complexity of the language. When considering the performance of the code Scala is in a disadvantage since it internally wraps some of the Java primitives and this adds some cost. One of Scala's weaknesses is also compilation times compared to Java, but this is only an issue when doing a full compilation. Most Scala are using Simple Build Tool (*sbt*) that provides incremental compilation to speed this together with automatic application reloads on source code changes. *Sbt* also provides support for Scala REPL where project classpath is automatically imported and can be used to tryout small changes.

In our solution we decided to use Scala because of its more concise syntax[29] compared to Java and support for functional programming techniques after we tested it for writing some of the code for Spark. Current solution uses Scala version 2.11.7 and *sbt* version 0.13.11.

CoffeeScript[30] is a language that compiles into JavaScript and has less verbose syntax, which speeds up writing the code for frontend. During the start of the project in 2014 CoffeeScript

[29] http://techblog.realestate.com.au/java-to-scala-cheatsheet

contained many developer productivity increasing functionalities, that JavaScript was missing for example (classes, string interpolation, arrow functions). These features are now making their way into JavaScript version ECMAScript 6. Decision to use CoffeeScript came from a previous experience and the fact that it was supported by Play framework and needed no additional setup.

LESS[31] is a language that compiles into Cascaded Style Sheets (CSS) used for webpage styling. LESS is implemented in a way that all valid CSS code is also valid LESS code. LESS adds support to variables, functions, inheritance, imports from different files and all these functionalities improve productivity without requiring immediate learning of the new language features. Reasons for using LESS were the same as for using CoffeeScript.

4.1.2 Server-side frameworks

Play Framework[32] web framework is one of the leading web frameworks in Scala world. Play is open-sourced, multi-threaded, RESTful full stack web framework. Play uses model–view–controller (MVC) style of development. Play comes with built-in database migrations, JSON support, WebSockets and integration with *sbt* provides a console for trying out code and "hot-reloading".

We used several plugins to speed up the development workflow:

- *webjars-play*[33] – Enables the management of client-side libraries from npm[34] and bower[35];
- *play-auto-refresh*[36] – A plugin that reloads web page in the browser after changes in source code have been compiled in the server side;
- *sbt-uglify*[37] – Uses UglifyJS[38] optimizations for JavaScript for production deployment, resulting in smaller JavaScript file for the browser;

[30] http://coffeescript.org
[31] http://lesscss.org
[32] https://www.playframework.com
[33] https://github.com/webjars/webjars-play
[34] https://www.npmjs.com
[35] http://bower.io
[36] https://github.com/jamesward/play-auto-refresh
[37] https://github.com/sbt/sbt-uglify
[38] http://lisperator.net/uglifyjs

- *sbt-native-packager*[39] – A plugin that enables to build production package from the source code. We configured it to build Docker[40] image from our project for production deployment.

Slick[41] is a database query and access library for Scala that enables to work with database data as you would work with Scala's collection system. It enables to write your database queries in a type safe manner and if needed an option to write queries in SQL and map the results into Scala data structures is also supported. Slick is open sourced and currently supports more than seven database systems [22] and there is a possibility to buy arbitrary drivers for other databases from Typesafe.

Integration between Slick and Play Framework is done by *play-slick*[42] package. In our solution we use Play framework 2.3.9 and Slick 2.1.0.

Play was selected as our web development framework because it provided good documentation, support for push communication from server to the client. Decision to use Slick came from the fact that the built-in Play persistence layer *anorm*[43] was not type safe and used a lot of "magic strings". The fact that it has been replaced with Slick in the future versions was also a consideration.

4.1.3 Frontend frameworks

AngularJS[44] is JavaScript based open-source web application framework maintained by Google and a community of individual developers. Its goal is to simplify development of such applications by providing a framework for client-side (MVC) architecture, along with components commonly used in rich Internet applications.

Restangular[45] is an AngularJS service that simplifies common GET, POST, DELETE, and UPDATE requests with a minimum of client code [23].

Ui-grid[46] is a data gird built for AngularJS. It is good in handling large data sets and uses row virtualization to provide good user experience in the browser.

[39] https://github.com/sbt/sbt-native-packager
[40] https://www.docker.com
[41] http://slick.typesafe.com
[42] https://github.com/playframework/play-slick
[43] https://github.com/playframework/anorm
[44] https://angularjs.org
[45] https://github.com/mgonto/restangular

D3.js[47] is a JavaScript library for manipulating documents based on data. D3 helps you bring data to life using HTML, SVG, and CSS [24]. Initially we tried other visualization libraries like angular-chart[48] and c3.js[49], both of them are actually based on D3.js and added abstractions did not provide some of the needed functionality (custom tooltips on the chart, custom lines on the chart and regions on the chart).

4.2 Components of the created solution

4.2.1 Model entry

For the initial solution we wanted to make creation of new models as simple as possible. Current solution supports parsing CSV format that the user can paste into a specific text field to import new SNPs for the model and also integrates with Ensembl database to provide additional annotation for each SNP. After markers are imported user is able to start the polygenic risk score calculation from the entered data. Software also provides the possibility to link model with ICD10 or ATC codes by using autocompleted text field. ICD10 or ATC codes were priority because EGCUT and Estonian health care system both use ICD10[50] classification for diagnoses and ATC[51] to describe active substances in medications. In the future we want to increase the number of these structured description parameters to automatically link calculated genetic risks with stored health information and provide built-in estimations how calculated genetic risk predicts described medical condition.

4.2.2 Actor system

The core of the application is built on top of Actor model provided by Akka framework. We found that Actor model provides good abstraction for parallelizing risk score calculations. Later tests[52] have shown that in parallel tasks used in our solution Akka is as fast or faster than Scala collections.

Created actor system has three permanent actors, *DB*, *CalculationsManager* and *IndividualsDataProvider*. *CalculationsManager* coordinates access to active calculations and to the results of these calculations. It connects *CalculationProgressSocketHandler* to

[46] http://ui-grid.info
[47] https://d3js.org
[48] https://github.com/GraFiddle/angular-chart
[49] http://c3js.org
[50] http://www.who.int/classifications/icd/en
[51] http://www.whocc.no/atc/structure_and_principles
[52] tests are distributed with source code and located in *tests/SpeedTests.scala*

CalculationCoordinator during risk score calculation and *CalculationResultSocketHandler* to *CalculationDataProvider* when results are viewed. Depending on current system state it either creates new instances or provides access to existing ones. Using *CalculationsManager* we can configure how many concurrent *CalculationCoordinator* or *CalculationDataProvider* actors are running and what to do when any of them crashes.

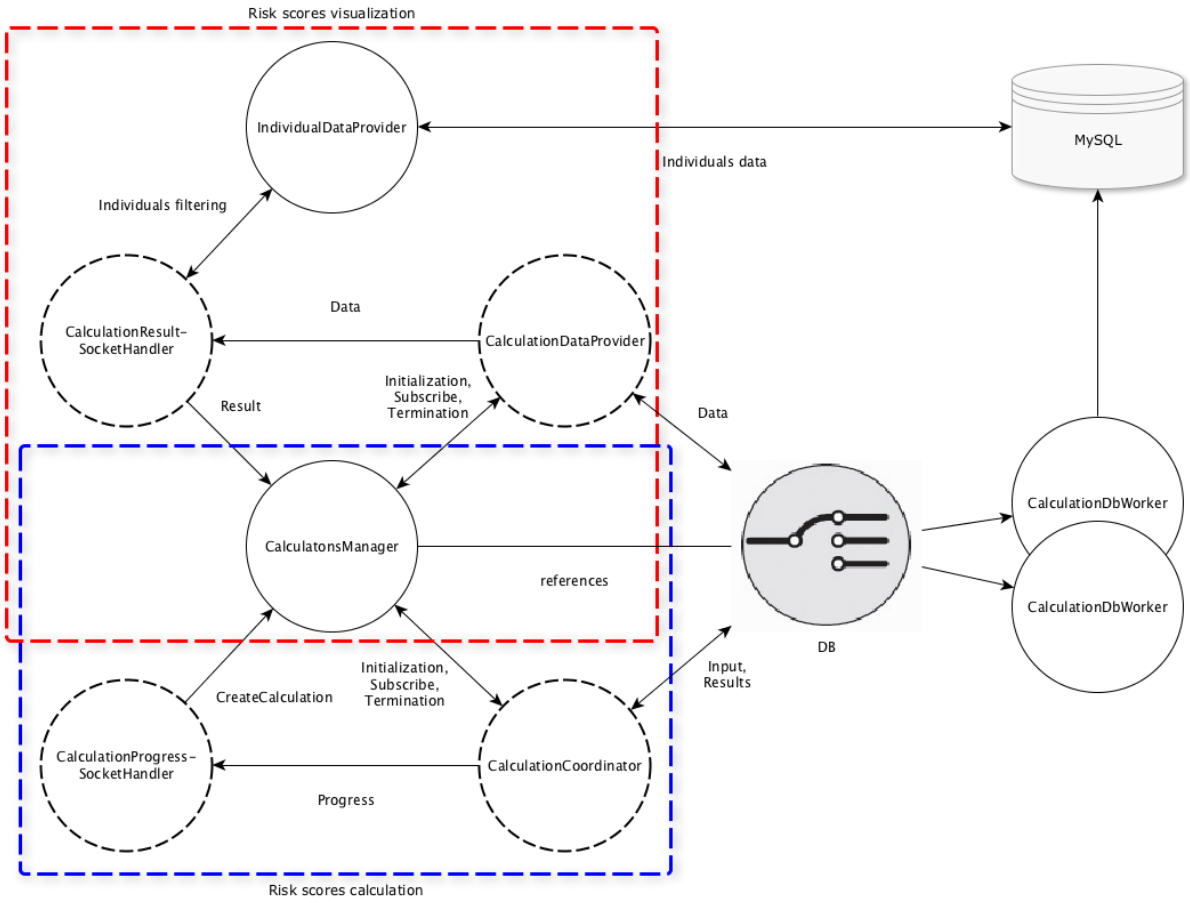


Figure 10 – Actors – Overview of the actor system. Actors with solid lines are permanent and with dashed lines are temporary and constructed in response of user actions.

Created actor system has three permanent actors, *DB*, *CalculationsManager* and *IndividualsDataProvider*. *CalculationsManager* coordinates access to active calculations and to the results of these calculations. It connects *CalculationProgressSocketHandler* to *CalculationCoordinator* during risk score calculation and *CalculationResultSocketHandler* to *CalculationDataProvider* when results are viewed. Depending on current system state it either creates new instances or provides access to existing ones. Using *CalculationsManager* we can configure how many concurrent *CalculationCoordinator* or *CalculationDataProvider* actors are running and what to do when any of them crashes.

DB distributes work between a configurable number of *CalculationDbWorker* actors. Each *CalculationDbWorker* has one connection to MySQL database and depending on received message can either provide or insert data to database. We use this mechanism to provide predictable management of database connections that can be scaled when such need arises.

IndividualsDataProvider provides access to cached individuals, their diagnoses and objective measurements. This information is used by *CalculationResultSocketHandler* to display different subgroups shown in Figure 14.

4.2.3 Genetic risk score calculation within the actor system

Risk scores calculation algorithm described in 2.5, was one of the first operations where we decided to use Akka. With Spark or Scala built-in collections it was very difficult to control how many active calculations are allowed, which resources are used and what is the progress of these calculations. By using Actor model and Akka this became much simpler task because we could connect client WebSocket directly to the actor system to create two-way communication and from there create connection to any actors needed.

Implemented risk score calculation subsystem is shown in Figure 11. Process begins with a browser creating a WebSocket connection to the server. Play framework constructs a new *CalculationProgressSocketHandler* actor with references to *CalculationsManager* and internal *WebSocket* actor responsible for the connection. After the connection is established client browser is expected to send *CreateCalculation* message which is forwarded to the *CalculationsManager* actor. *CalculationsManager* determines by the message if there is a calculation running and subscribes *CalculationProgressSocketHandler* to receive notifications. When no calculation is running, *CalculationsManager* creates a new *CalculationCoordinator* instance with references to the *CalculationsManager* and *DB*. Then *CreateCalculation* message is passed to *CalculationCoordinator* and *CalculationProgressSocketHandler* is subscribed to receive notifications from *CalculationCoordinator*.

On initialization *CalculationCoordinator* creates an internal *CalculationEventBus* that is used to exchange intermediate messages. *AdditiveEffectCombinator*, *InfluencialMarkersSelector, CalculationSummaryCreator* and every *MarkerEffectCalculator* is initialized with the reference to the event bus.

When *CreateCalculation* message is received, *CalculationCoordinator* forwards it to *DB*. One *CalculationDbWorker* picks up the message, deletes previous calculation with same parameters, creates a new *PredictionCalculation* and passes it back to the *CalculationCoordinator*. After receiving *PredictionCalculation, CalculationCoordinator* updates its state and starts the process by sending *EstimateMarkersCount* to the DB. *CalculationCoordinator* receives the response, updates its state and notifies subscribers with *CalculationUpdate* message. In case of subscribing *CalculationProgressSocketHandler* this is sent through websocket and view shown in clients browser is updated from "Waiting data …" to show view illustrated in Figure 12.

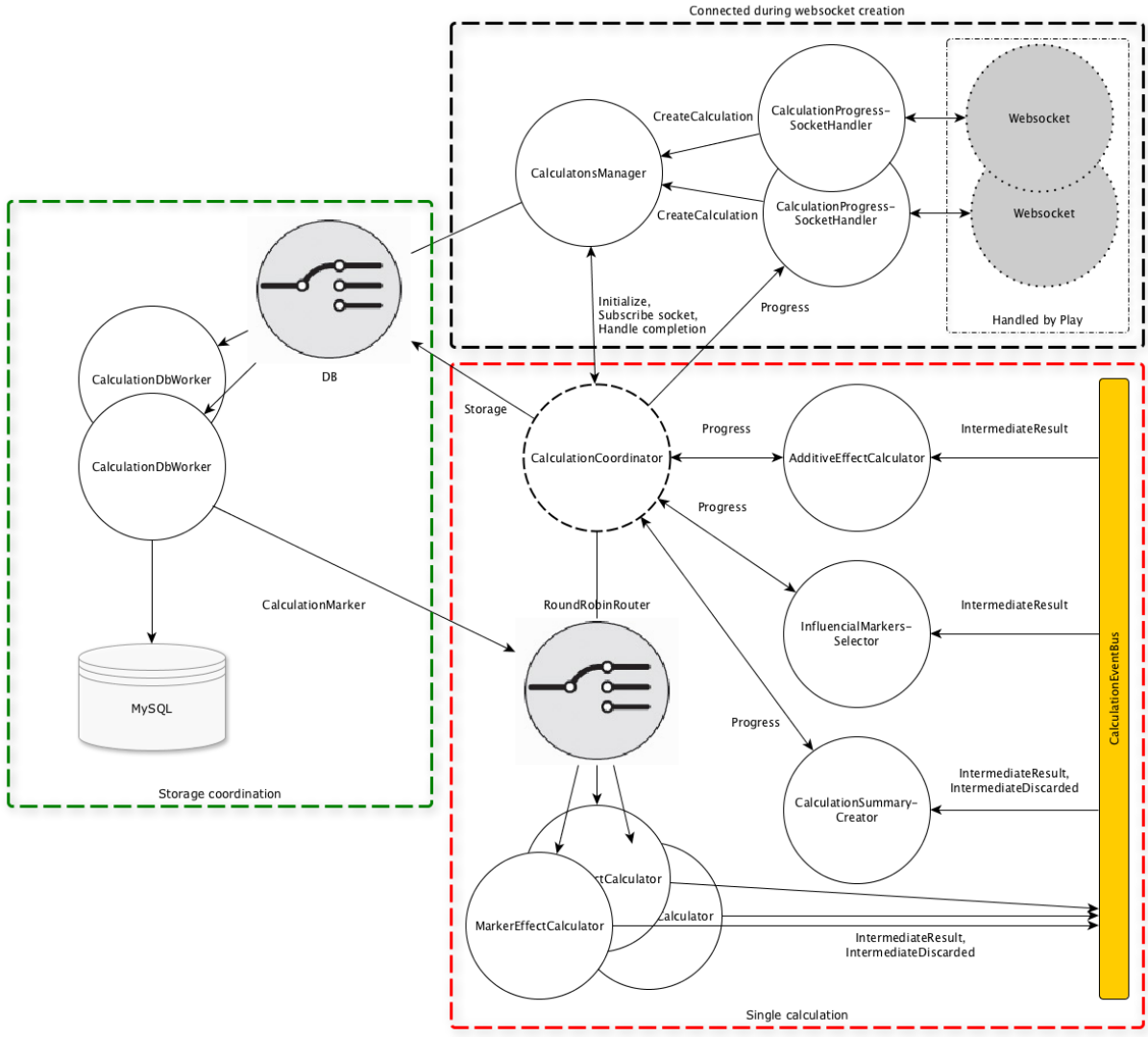
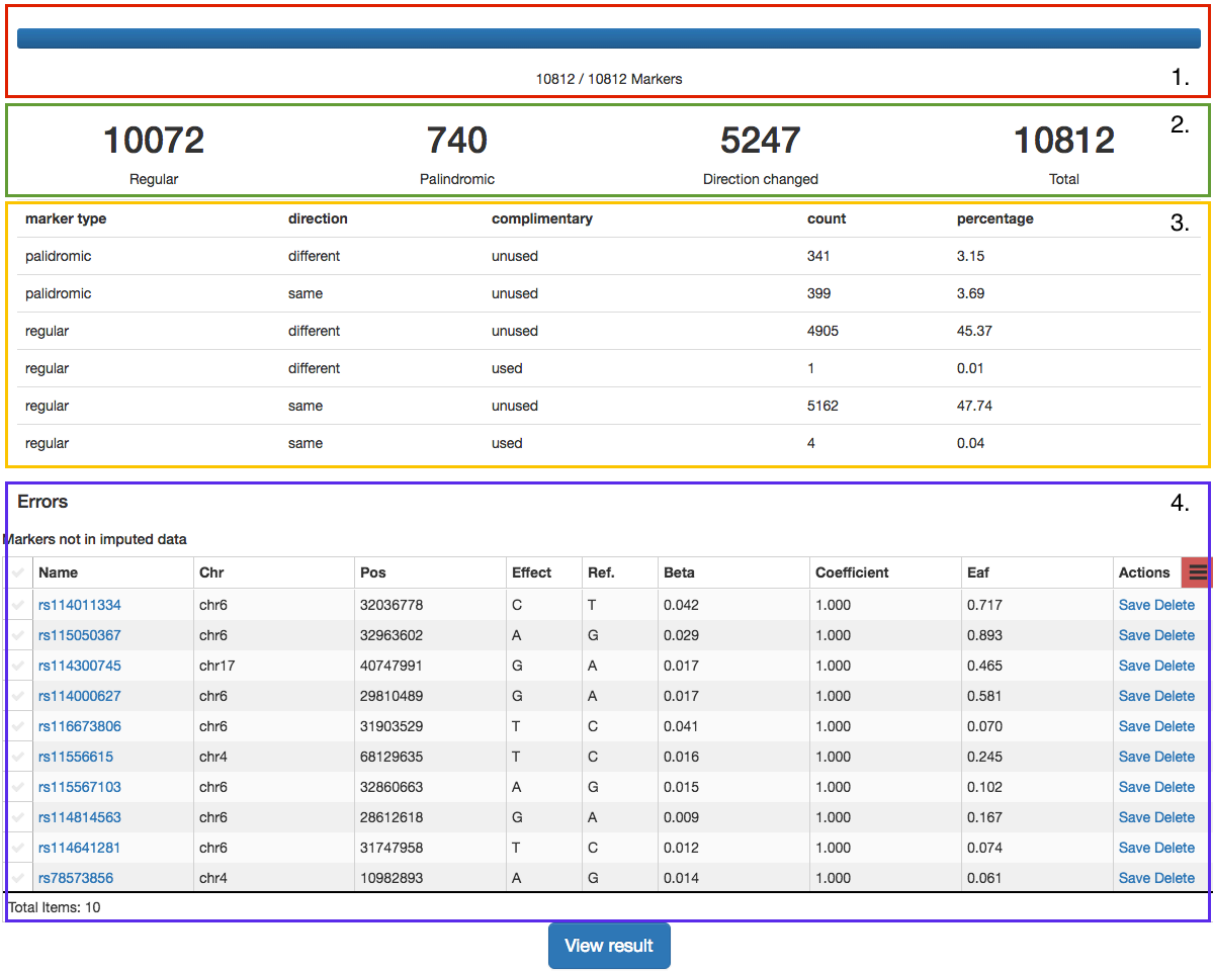


Figure 11 – Actors - Risk scores calculation.

CalculationCoordinator then sends *SendCalculationMarkers* message to the DB with a reference to the *MarkerEffectCalculators* router. *CalculationMarkers* are distributed by the

router between *MarkerEffectCalculators*. Each *MarkerEffectCalculator* passes received *CalculationMarker* through quality control [Appendix 2. 22-52] and calculates the *IntermediateResult* – effect of given marker for each individual in the *ImputeSet* [Appendix 2. 88-146]. Result, either *IntermediateResult* or *IntermediateDiscarded* message containing error details, is passed to the *CalculationEventBus*, where subscribers can access it.



marker type	direction	complimentary	count	percentage
palidromic	different	unused	341	3.15
palidromic	same	unused	399	3.69
regular	different	unused	4905	45.37
regular	different	used	1	0.01
regular	same	unused	5162	47.74
regular	same	used	4	0.04

Name	Chr	Pos	Effect	Ref.	Beta	Coefficient	Eaf	Actions
rs114011334	chr6	32036778	C	T	0.042	1.000	0.717	Save Delete
rs115050367	chr6	32963602	A	G	0.029	1.000	0.893	Save Delete
rs114300745	chr17	40747991	G	A	0.017	1.000	0.465	Save Delete
rs114000627	chr6	29810489	G	A	0.017	1.000	0.581	Save Delete
rs116673806	chr6	31903529	T	C	0.041	1.000	0.070	Save Delete
rs11556615	chr4	68129635	T	C	0.016	1.000	0.245	Save Delete
rs115567103	chr6	32860663	A	G	0.015	1.000	0.102	Save Delete
rs114814563	chr6	28612618	G	A	0.009	1.000	0.167	Save Delete
rs114641281	chr6	31747958	T	C	0.012	1.000	0.074	Save Delete
rs78573856	chr4	10982893	A	G	0.014	1.000	0.061	Save Delete

Total Items: 10

Figure 12 – View – *CalculationProgressPage*. This view is shown to the user during calculation. View result becomes visible after calculation has completed. 1 – we show current progress vs total progress. 2 – Some key facts about operations performed. 3 – We show more detailed statistics of operations performed. Direction – whether the effect allele same or did we need to change it. Complimentary - did we need to use complimentary to correct nucleotides in imputed data. Marker type shows if the marker was palindromic or regular SNP.

While *IntermediateResults* are produced *CalculationCoordinator* periodically sends *SendProgress* messages to *CalculationEventBus*. Each subscriber decides how it responds to this message. To keep track of total progress *CalculationCoordinator* listens for *CalculationMarkersSent* message with info how many *CalculationMarkers* were produced and *IntermediateDiscarded* messages about any failures in processing. When

CalculationEventBus subscriber processes all of the expected *IntermediateResults, CalculationCoordinator* asks its final result. Final results from *AdditiveEffectCombinator* and *InfluencialMarkersSelector* are currently required to finish calculation. After one of these results is saved to the database. *DB* sends back a confirmation and *CalculationCoordinator* updates its state and verifies that all known subscribers have their final result persisted. At some point all required *AdditiveEffectCombinator* and *InfluencialMarkersSelector* both have persisted their final results and *CalculationCoordinator* notifies its subscribers together with *CalculationsManager* about process completion. *CalculationsManager* stops *CalculationCoordinator* and releases all the resources used.

AdditiveEffectCombinator uses *IntermediateResult* messages to sum different marker effects for each individual and after processing ends sends the result to be stored in the database. *InfluencialMarkersSelector* uses *IntermediateResult* to select a configurable number of markers with the largest absolute $\hat{\beta}$ value and after the processing is complete sends them to be stored in database. *CalculationSummaryProvider* uses *IntermediateResult* to extract what operations were performed on each marker and *IndermediateDiscarded* messages to collect processing errors, when *SendProgress* message arrives, *CalculationSummary* is sent to the sender containing the summary of operations currently performed Figure 12 parts 2 and 3 illustrate how it is shown in the client side.

4.2.4 Overview of risk scores visualization

Current system provides means to filter individuals based on age, body mass index and diagnoses. Different subgroups are visualized using *D3.js* and interfaced with AngularJS through the use of custom directives. There is also a possibility to download calculated scores to provide support for other means of analysis. Most complex part is visualization of different subgroups within the calculated set because there is too much data for the browser to handle - in a category of 1 million diagnoses for 8117 individuals. Risk scores visualization is then implemented in a way that filtering is done on the server side and only small part of information that is needed to draw the charts is transferred to the browser.

Communication runs on WebSocket and is implemented asynchronously. When client browser moves to a result viewing page it first creates a WebSocket connection to the application server. Play creates internal *WebSocket* actor to handle message passing between server and the client and creates new *CalculationResultSocketHandler* with references to *CalculationsManager*, *IndividualsDataProvider* and *WebSocket* actor.

After the connection between the client and application server is established, the client is expected to send a *Result* message that is forwarded to the *CalculationsManager*. *CalculationsManager* determines if some *CalculationDataProvider* for this message is running and selects it. When there is no *CalculationDataProvider* then it is created and *CalculationsManager* passes a reference of *CalculationDbWorkers* router (DB) to the *CalculationDataProvider*. After *CalculationDataProvider* is selected *CalculationsManager* and subscribes *CalculationResultSocketHandler* to the *CalculationDataProvider* and messages *CalculationDataProvider* to provide data for *CalculationResultSocketHandler*.

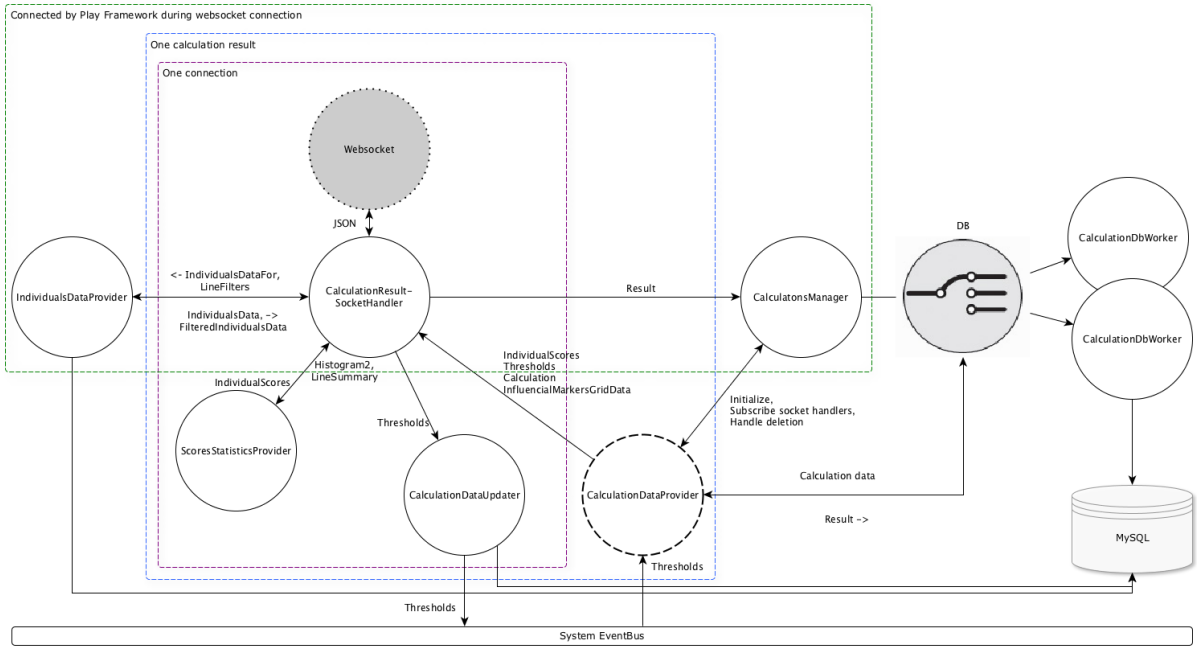


Figure 13 – Actors – Risk scores visualization.

After the connection between the client and application server is established, the client is expected to send a *Result* message that is forwarded to the *CalculationsManager*. *CalculationsManager* determines if some *CalculationDataProvider* for this message is running and selects it. When there is no *CalculationDataProvider* then it is created and *CalculationsManager* passes a reference of *CalculationDbWorkers* router (DB) to the *CalculationDataProvider*. After *CalculationDataProvider* is selected *CalculationsManager* and subscribes *CalculationResultSocketHandler* to the *CalculationDataProvider* and messages *CalculationDataProvider* to provide data for *CalculationResultSocketHandler*.

When *CalculationDataProvider* is created it is in a "*startup"* state and asks data from DB. First available *CalculationDbWorker* runs the needed queries on MySQL database and passes

results back to the *CalculationDataProvider*. If data arrives from *DB, CalculationDataProvider* forwards it to subscribers and updates internal state. If *CalculationDataProvider* has received all the data, it changes its state to "*loaded"* in which case it starts to provide cached data to any actor that sends *ProvideStoredData* message. If all *CalculationResultSocketHanders* are disconnected, *CalculationDataProvider* starts a timer and shuts itself down.

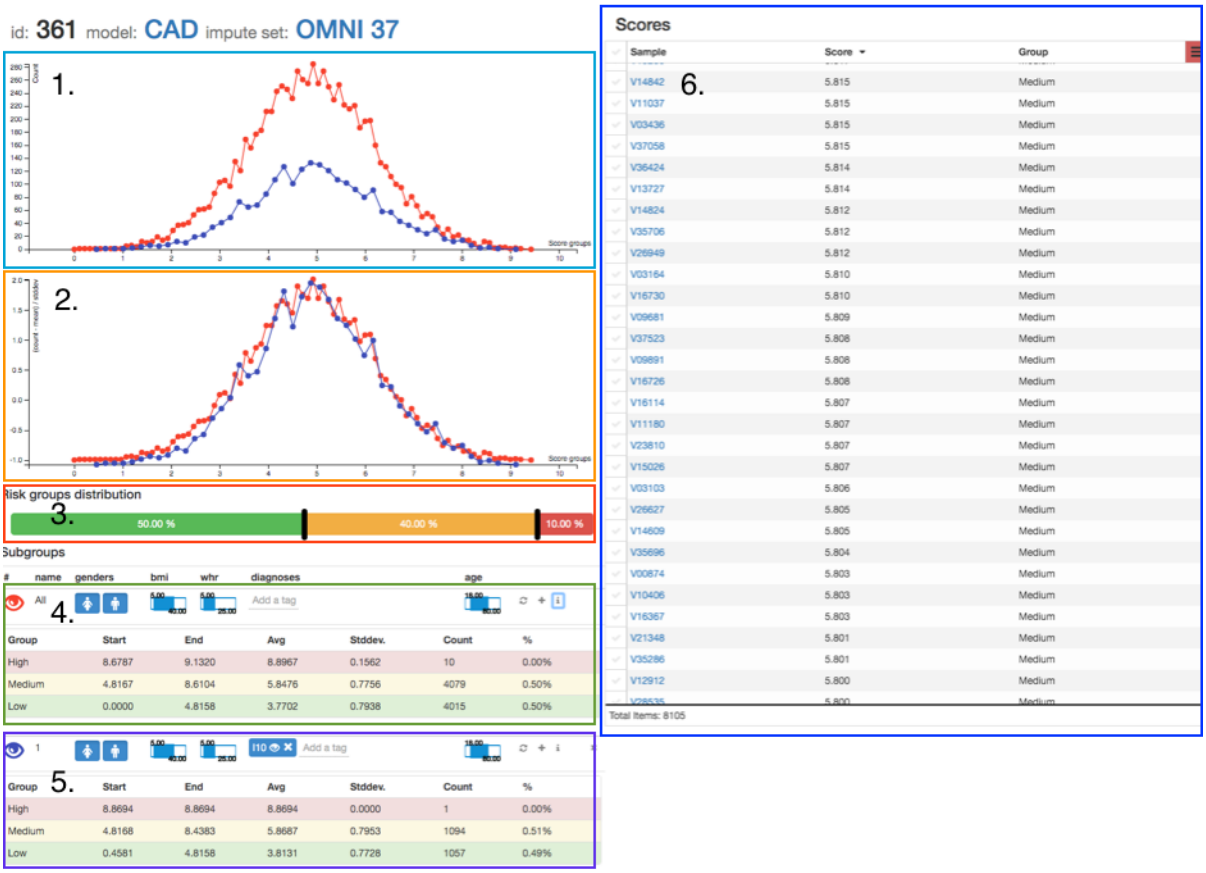


Figure 14 – View - *CalculationResultDetailsPage*. 1 – Histogram showing the distribution of calculated scores based on counts. 2 – Histogram showing calculated scores but normalized to enable comparison of different subgroups. 3 – Risk thresholds selector. 4 – Filters and summary statistics for the entire population. 5 – Filters and summary statistics for defining subgroups within the population. 6 – Calculated scores for all of the individuals within the population by selecting a line it is possible to see the position on the graphics.

During initialization *CalculationResultSocketHander* created *CalculationDataUpdater* actor that has the capability to update the thresholds for risk groups. When thresholds are changed in the browser a *Thresholds* message is sent to *CalculationResultSocketHander*, forwarded to *CalculationDataUpdater* that updates database and notifies the entire system by publishing updated thresholds to the system-wide event bus. All created *CalculationDataProviders* listen for this messages and if the update is related to the *CalculationDataProvider* subscribers are

notified with new data and user interface for every client is updated. This functionality was implemented as a proof of concept to provide simultaneous updates to user views.

When receiving data messages from the *CalculationDataProvider*, *CalculationResultSocketHander* converts them to JSON and forwards them to the outgoing stream. After receiving *IndividualScores* message it is also forwarded to the *ScoresSocketStatisticsProvider* actor, created during initialization. The purpose of *ScoresSocketStatisticsProvider* actor is to calculate *Histogram2* from the scores. *Histogram2* is the data transfer object that includes some summary statistics about the *IndividualScores* data and is used in the client browser to render different subgroups.

This is done on the server side to reduce the load on client browser. *IndividualScores* is also mapped into array of individual id's, that are sent to the *IndividualsDataProvider* using *IndividualsDataFor* message to load phenotype data for filtering operations, when needed.

IndividualDataProvider loads individual's diagnoses and objective measurements and caches them into memory. Since it is used by all the *CalculationResultSocketHandlers* we reduce the overhead of loading multiple sets of data for one individual. Once the data is loaded *CalculationResultSocketHandler* is notified and it forwards this information to the client browser. Client browser then enables the subgroups visualization functionality as seen on Figure 14. To create new subgroups or modify existing we pass *LineFilters* messages from the browser to the application server as detailed on Figure 15.

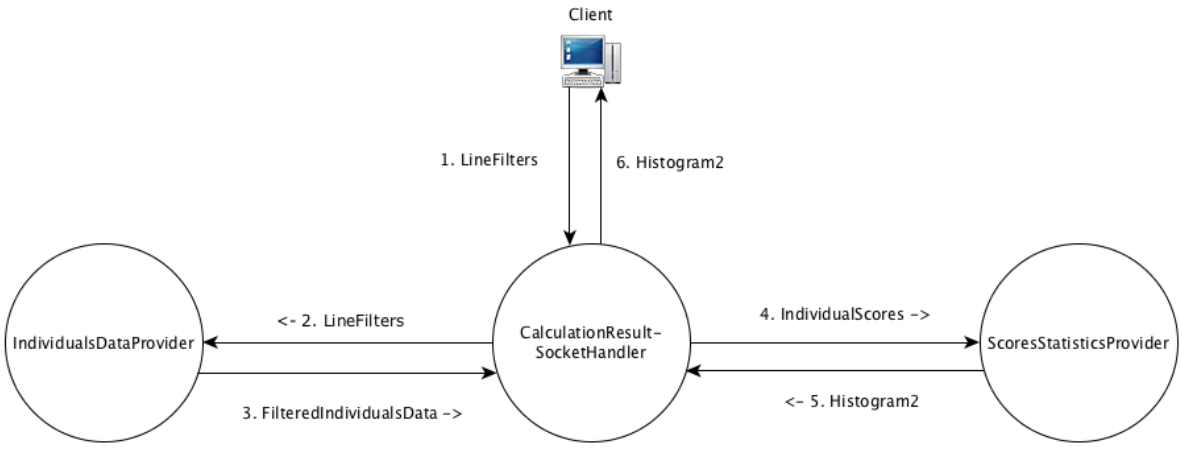


Figure 15 – Actors – Calculation result subgroup filtering.

4.2.5 Individual data overview

One of the requirements for the system was the ability to show genetic risks together with other health information available in EGCUT. EGCUT is setting up a new information system and does not yet provide online access to stored health records. Data is imported into system from CSV format. View given in Figure 16 shows how this data is used. Created timeline is based on *visjs*[53] JavaScript library. Database model used to store this information is detailed in Appendix 3.

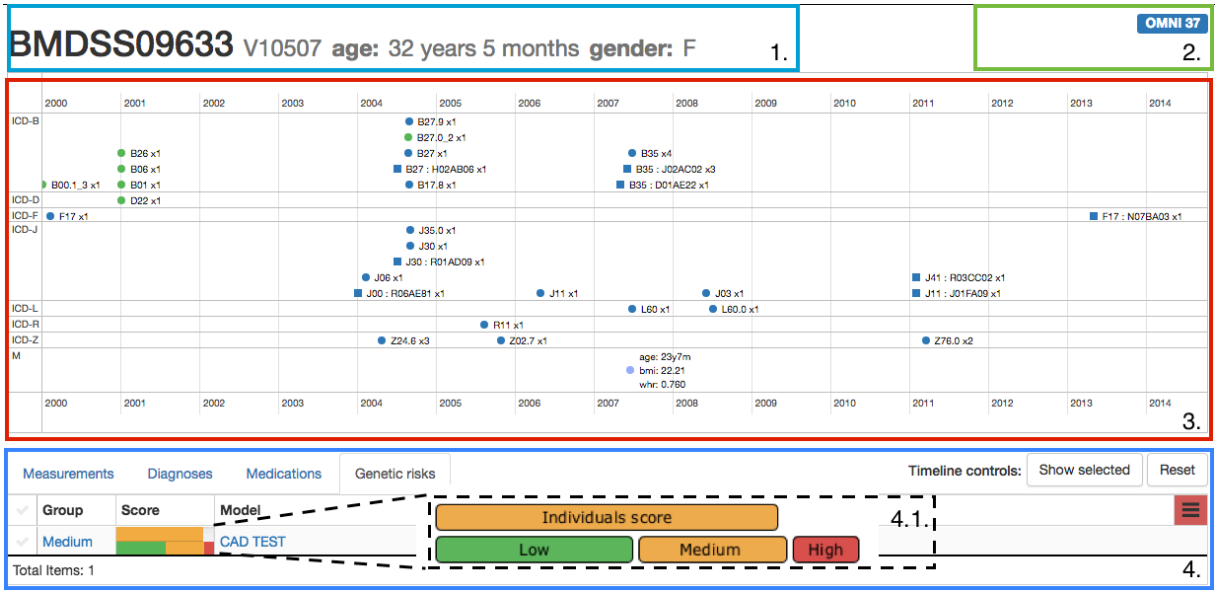


Figure 16 – View - *IndividualDetailsPage*. 1 – Individual code and sample codes. 2 – Is individual included in some impute set. 3 – Individual's timeline containing diagnoses (B27 x1) and medications (B27: H02AB06 x1) it is possible to zoom in and out to more precise information by default diseases are grouped 1 in a year. 4 – it is possible to select which measurements, diagnoses and medications are shown in the timeline. 4.1 – for genetic risks we show individuals risk relative to the other groups in the used model.

There was also the problem of how to present genetic risk size to the user and a custom chart was created shown in Figure 16 part 4.1.

[53] http://visjs.org

4.3 Similar works on processing imputed genomic data

A big part of this thesis was to improve the usability of imputed genomic data. We found that there is not much work done in this area. One format we discovered was BGEN that has gained popularity in recent years. Unfortunately, we were unable to find this format when the initial work begun.

4.3.1 BGEN format[54]

BGEN is a binary format used to store imputed genotypes. This format also compresses each marker probabilities with *zlib*. Compared by the final implementation used in our solution it has the benefit of being supported by many popular tools (QCTOOL[55], PLINK[56], SNPTEST[57]) so it is more general purpose. This format uses a clever technique for storing imputed probabilities file size reduction. Imputed probabilities are stored as 2-byte integers and during usage converted to floating point numbers by dividing them with 32,768. This enables accuracy of 4 places after comma which is enough for this type of data. In our solution we generally use much more than 2 bytes to store each probability as a string and by incorporating this technique additional size reduction can be achieved.

Compared to our solution it uses slower compression algorithm, database gives us better querying mechanism and options to generate indexes to speed up random access. We believe that using a database and Lz4 compression actually makes our solution more portable than BGEN. BGEN homepage does not describe any implementation in any other programming languages than C++. MySQL and Lz4 compression in the other hand have wide platform support.

4.4 Comparison with similar software

4.4.1 Promethease

Promethease is closed source software to create personalized DNA report based on the genome and data found in SNPedia database[58]. At the time of the initial testing (2014 Dec) Promethease had a free desktop application that would generate DNA report and was fixed to run at least 4 hours and a paid solution, that would do it faster. Now (2016 May) Promethease

[54] http://www.well.ox.ac.uk/~gav/bgen_format/bgen_format.html
[55] http://www.well.ox.ac.uk/~gav/qctool/#overview
[56] https://www.cog-genomics.org/plink2
[57] https://mathgen.stats.ox.ac.uk/genetics_software/snptest/snptest.html
[58] http://www.snpedia.com/index.php/SNPedia

is a fully paid web application. Reports generated by Promethease bring out dangerous SNPs in individual's genotype based on the data in SNPedia database[59].

Compared to the created solution Promethease has access to more curated genotype-phenotype through the use of SNPedia collaborative wiki database. This results with data that is difficult to verify and makes Promethease not suitable for clinical use. By current understanding Promethease does not use polygenic risk scores to calculate genetic risks, but there is no way to verify this since it is closed-source.

4.4.2 PRSice: Polygenic Risk Score software

PRSice is a software package written in R, including wrappers for bash data management scripts and PLINK2 to minimize computational time; thus much of its functionality relies entirely on computations written originally by Shaun Purcell in PLINK. PRSice runs as a command-line program with a variety of user-options and is freely available for download below, compatible for Unix/Linux/Mac OS and in dockerized form also Windows.

Compared to the created solution PRSice does not aim to provide environment to store GWAS results or store the results. It focuses more on expert user who works on the command line than to create collaborative environment that is intent for the given software.

[59] http://www.snpedia.com/index.php/SNPedia

Summary

In the scope of this thesis, software to support polygenic risk model development is built. Created solution focuses on combining collected data with scientific results, speeding-up of GRS calculation as well as provides options for descriptional analysis and visualization of obtained results in user-friendly way. Scala with Play Framework and Akka framework was selected as backend development environment. Frontend development is done with AngularJS framework. For user feedback, there is WebSockets based solution between Akka cluster and AngularJS.

Processing imputed data during calculations, was the bottleneck of the system and several architectures were tested to solve this issue.

Three different storage systems GPFS, MySQL and Apache Cassandra were evaluated for imputed data storage. In the end, MySQL storage engine was used, because of stable drivers, existing setup and familiar data-modelling techniques. For data compression, *Lz4 HC* with custom algorithm for reducing imputed marker's probability counts was implemented. Selected solution stores imputed data for 8117 individuals in 69GB table, which is the same as storing it in the file system using *gzip*. Access to specific markers throughout the genome is better and extraction of 10,000 markers covering the entire genome takes on average 30s.

For parallelization, Apache Spark and custom Akka based solution was implemented. Apache Spark had a lot of promise in running batched jobs, but using it as constantly running environment proved hard to maintain. It had many dependencies including compilation of native code, which made development difficult. Actor model based Akka solution was easier to develop and as performant, it processed data on the fly, without first loading full set into memory.

Future plans include building command line tool to enable usage of created storage solution in other pipelines within EGCUT. Improving model description and model risk score analysis options.

Bibliography

[1] O. Gottesman, S. A. Scott, S. B. Ellis and J. Hall, "The CLIPMERGE PGx Program: Clinical Implementation of Personalized Medicine Through Electronic Health Records and Genomics–Pharmacogenomics," 03 04 2013. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1038/clpt.2013.72/full. [Accessed 01 05 2016].

[2] L. Leitsalu, T. Haller, T. Esko, M.-L. Tammesoo, H. Alavere, H. Snieder, M. Perola, P. C. Ng, R. Mägi, L. Milani, K. Fischer and a. A. Metspalu, "Cohort Profile: Estonian Biobank of the Estonian Genome Center, University of Tartu," no. 44, 2015.

[3] "Population pyramid of Estonia," 2011. [Online]. Available: http://www.stat.ee/public/rahvastikupyramiid/. [Accessed 01 05 2015].

[4] "EGCUT agreement form - EST," [Online]. Available: http://www.geenivaramu.ee/et/doonorile/geenidoonoriks-saamise-nousoleku-vorm-naidis. [Accessed 01 05 2016].

[5] "SNP," [Online]. Available: http://en.wikipedia.org/wiki/Single-nucleotide_polymorphism. [Accessed 20 05 2015].

[6] "Reference genome," [Online]. Available: http://en.wikipedia.org/wiki/Reference_genome. [Accessed 19 05 2015].

[7] N. G. Sequencing. [Online]. Available: https://en.wikibooks.org/wiki/Next_Generation_Sequencing_%28NGS%29/Alignment. [Accessed 01 05 2016].

[8] C. W. Yun Li, "Genotype Imputation," 23 08 2010. [Online]. Available: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2925172/. [Accessed 01 05 2016].

[9] University of Leicester, "Monogenic Disorders," [Online]. Available: http://www2.le.ac.uk/departments/genetics/vgec/healthprof/topics/patterns-of-inheritance/patterns-of-inheritance-conditions.

[10] "Genetic disorder," [Online]. Available: http://en.wikipedia.org/wiki/Genetic_disorder. [Accessed 19 05 2015].

[11] J. Pakhare, 17 05 2013. [Online]. Available: http://www.buzzle.com/articles/genetic-diseases-list-disorders.html. [Accessed 01 05 2015].

[12] P. D, "Molecular genetic analysis of Down syndrome.," 09 06 2009. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/19526251. [Accessed 02 05 2016].

[13] Williams-syndrome association, 2014. [Online]. Available: https://williams-syndrome.org/content/diagnosing-williams-syndrome-0. [Accessed 01 05 2016].

[14] "Diabetes Costs and Prevalence Forecasts," Novo Nordisk, 02 05 2016. [Online]. Available: http://www.novonordisk-us.com/whoweare/changing-diabetes-barometer/research-and-data/diabetes-costs-prevalence-forecasts.html. [Accessed 02 05

2016].

[15] P. L1, W. M, L. J, d. B. AT, B. E, E. JG, F. E, I.-P. P, K.-K. SM, W. M, M. JC, U. M and T. J., "Importance of weight loss maintenance and risk prediction in the prevention of type 2 diabetes: analysis of European Diabetes Prevention Study RCT," 25 02 2013. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/23451166. [Accessed 02 05 2016].

[16] K. Lall, R. Magi, A. Morris, A. Metspalu and K. Fischer, "Personalized Risk Prediction for Type 2 Diabetes: the Potential of Genetic Risk Scores," 2016. [Online]. Available: http://dx.doi.org/10.1101/041731. [Accessed 01 05 2016].

[17] S. W. Kong, I.-H. Lee and I. S. Kohane, "Summarizing polygenic risks for complex diseases in a clinical whole genome report," 24 07 2015. [Online]. Available: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC4547452/. [Accessed 01 05 2016].

[18] "Imputed file format," [Online]. Available: http://www.stats.ox.ac.uk/~marchini/software/gwas/file_format.html. [Accessed 25 02 2015].

[19] 05 1996. [Online]. Available: https://tools.ietf.org/html/rfc1950. [Accessed 05 2016].

[20] "DEFLATE Compressed Data Format Specification version 1.3," 05 1996. [Online]. Available: https://tools.ietf.org/html/rfc1951. [Accessed 01 05 2016].

[21] "NoSQL performance benchmarks," [Online]. Available: http://planetcassandra.org/nosql-performance-benchmarks/. [Accessed 25 04 2015].

[22] Typesafe, "Slick - Supported database systems," Typesafe, [Online]. Available: http://slick.typesafe.com/doc/2.1.0/introduction.html#supported-database-systems. [Accessed 25 04 2015].

[23] "Restangular - github.com," [Online]. Available: https://github.com/mgonto/restangular.

[24] "D3.js Website," [Online]. Available: https://d3js.org. [Accessed 01 05 2016].

[25] "Autosome definition," [Online]. Available: http://en.wikipedia.org/wiki/Autosome.

[26] S. A. S. S. B. E. J. H. O Gottesman, "The CLIPMERGE PGx Program: Clinical Implementation of Personalized Medicine Through Electronic Health Records and Genomics–Pharmacogenomics," 03 04 2013. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1038/clpt.2013.72/full. [Accessed 01 05 2016].

[27] T. H. T. E. M.-L. T. H. A. H. S. M. P. P. C. N. R. M. L. M. K. F. a. A. M. Liis Leitsalu, "Cohort Profile: Estonian Biobank of the Estonian Genome Center, University of Tartu," no. 44, 2015.

[28] [Online]. Available: http://www.blc.arizona.edu/molecular_graphics/dna_structure/dna_tutorial.html.

[29] [Online]. Available: https://en.wikibooks.org/wiki/Next_Generation_Sequencing_%28NGS%29/Alignment. [Accessed 01 05 2016].

[30] [Online]. Available: http://en.wikipedia.org/wiki/Genetic_disorder. [Accessed 19 05 2015].

[31] W.-s. association, 2014. [Online]. Available: https://williams-syndrome.org/content/diagnosing-williams-syndrome-0. [Accessed 01 05 2016].

[32] W. M. L. J. d. B. A. B. E. E. J. F. E. I.-P. P. K.-K. S. W. M. M. J. U. M. T. J. Penn L1, "Importance of weight loss maintenance and risk prediction in the prevention of type 2 diabetes: analysis of European Diabetes Prevention Study RCT," 25 02 2013. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/23451166. [Accessed 02 05 2016].

[33] R. M. A. M. A. M. K. F. Kristi Lall, "Personalized Risk Prediction for Type 2 Diabetes: the Potential of Genetic Risk Scores," 2016. [Online]. Available: http://dx.doi.org/10.1101/041731. [Accessed 01 05 2016].

[34] I.-H. L. I. S. K. Sek Won Kong, "Summarizing polygenic risks for complex diseases in a clinical whole genome report," 24 07 2015. [Online]. Available: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC4547452/. [Accessed 01 05 2016].

Appendixes

Appendix 1. Apache Spark program for lzo file reading.

```
001  import java.io.File
002  import java.text.SimpleDateFormat
003  import java.util.Date
004
005  import com.hadoop.compression.lzo.LzoIndexer
006  import org.apache.hadoop.conf.Configuration
007  import org.apache.hadoop.io.Text
008  import org.apache.log4j.Logger
009  import org.apache.spark.{SparkConf, SparkContext}
010  import scala.reflect.io.Path
011
012  object ImputeFilter {
013
014    class ReaderConfig () {
015      var files: List[String] = List[String]()
016      var positions: String = ""
017      var output: String = ""
018    }
019
020    def parseOptions(args: List[String],options: ReaderConfig ): ReaderConfig = {
021      args match{
022        case Nil =>
023          options
024        case "-f"::tail =>
025          val (nextOptions,values) = parseOptionValues(tail)
026          options.files=values
027          parseOptions(nextOptions,options)
028        case "-p"::positions::tail =>
029          options.positions = positions
030          parseOptions(tail,options)
031        case "-o"::output::tail =>
032          options.output = output
033          parseOptions(tail,options)
034        case other::tail =>
035          println("Known options: " + options.files)
036          println("Unknown options: " + other::tail)
037          sys.exit(0)
038      }
039    }
040
041  def parseOptionValues(args: List[String],values:
     List[String]=Nil):(List[String],List[String]) = {
042      args match{
043        case Nil =>
044          (Nil,values)
045        case option::tail if option.startsWith("-") =>
046          (option::tail,values)
047        case value::tail =>
048          parseOptionValues(tail,value::values)
049      }
050    }
051
052    def main(args: Array[String]) {
053      val options = parseOptions(args.toList, new ReaderConfig())
054      val indexer = new LzoIndexer(conf)
055      for (file<-options.files){
056        if (!new File(file+".index").exists()){
057          indexer.index(new org.apache.hadoop.fs.Path(file))
058          logger.info("Indexing file: " + file)
059        }
060        else{
061          logger.info("File " + file + " has index")
062        }
063      }
064
065      logger.info("Adding files to context")
```

```
066       val files = sparkContext.union(
067         options.files.map(
068           f=>sparkContext.newAPIHadoopFile(f,
069             classOf[com.hadoop.mapreduce.LzoTextInputFormat],
070             classOf[org.apache.hadoop.io.LongWritable],
071             classOf[org.apache.hadoop.io.Text])))
072
073       // Read in needed markers
074       val source = scala.io.Source.fromFile(options.positions)
075       val broadcast = sparkContext.broadcast(source.getLines().toIndexedSeq)
076
077       // Filter the results
078       val results = files.filter( line => byRsNumber(broadcast.value,
079         line._2)).repartition(20).map(line=>line._2)
080
081       results.saveAsTextFile(Path(options.output).toAbsolute + "/" +
082         new SimpleDateFormat("yyyyMMdd-HHmmss").format(new Date()))
083     }
084
085     def byRsNumber(list: IndexedSeq[String],line: Text): Boolean ={
086       val beginning = Text.decode(line.getBytes,0,100)
087       val rs = beginning.split(" ")(1).split(":")(0)
088       list.contains(rs)
089     }
090   }
```

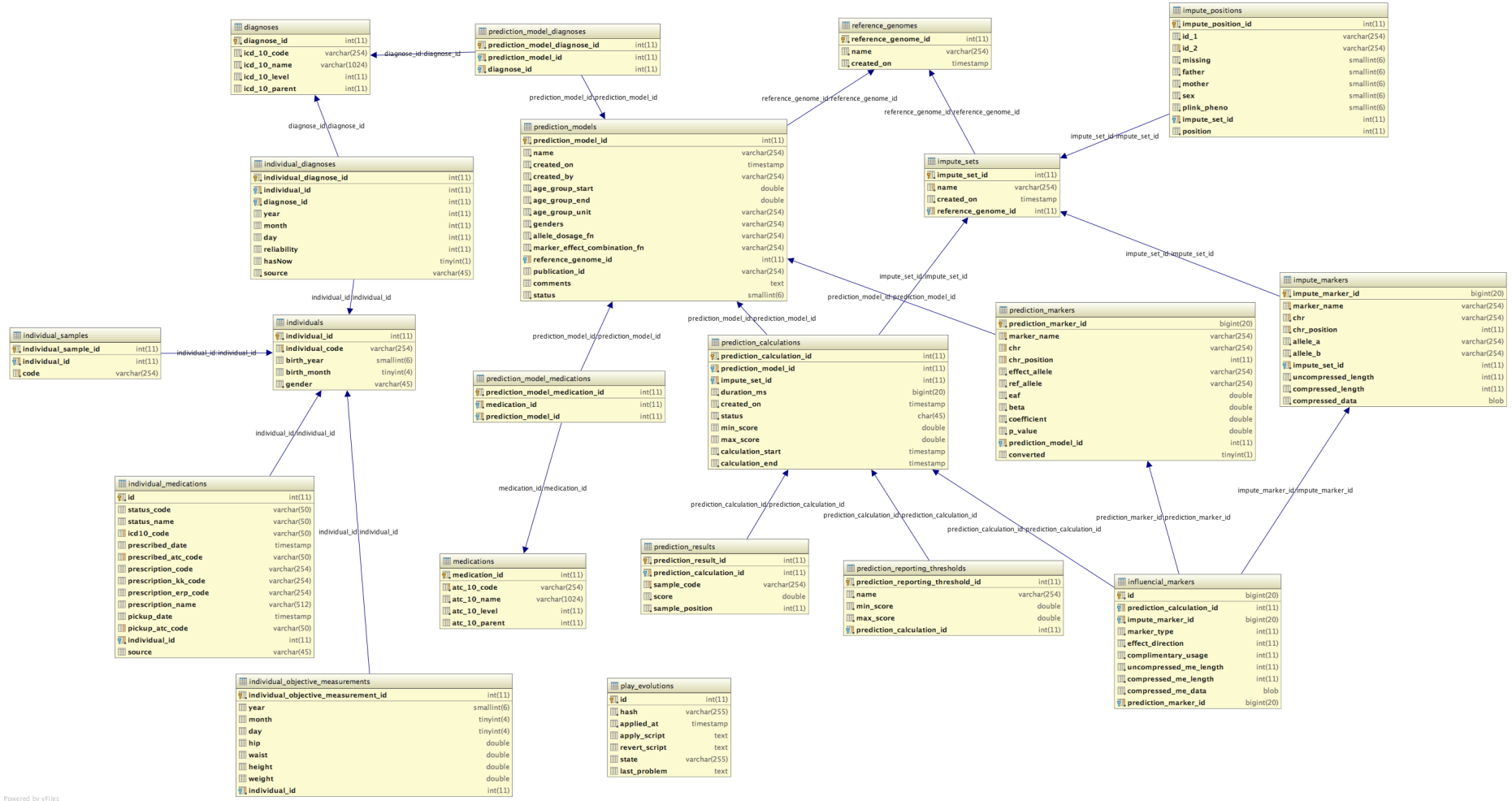
Appendix 2. Quality control and marker effect calculation

```
001 // Either type .left is usually error and .right the result
002 import scalaz.\/
003
004 val complimentaryMap = Map(
005   "A" -> "T", "C" -> "G", "G" -> "C", "T" -> "A",
006   "a" -> "t", "c" -> "g", "g" -> "c", "t" -> "a"
007 )
008
009 val logAdditiveModel = new Model {
010   // Max value for allele dosage
011   val alleleDosageMax = 2.0
012   // each B allele adds to the final effect from [AA, AB, BB]
013   def alleleDosageFn(triplets: Seq[Double]) = triplets(1) + 2.0 * triplets(2)
014   // Allele dosage needs no correction
015   def same(alleleDosage: Double) = alleleDosage
016   // Allele calculated allele dosage is for wrong allele
017   def reverse(alleleDosage: Double) = alleleDosageMax - alleleDosage
018 }
019
020 // Used to correct palindromic marker estimated effect
021 val similarityThreshold = 0.2
022
023 def qualityControl(calculationMarker: CalculationMarker):
024 CalculationMarkerError \/ CalculationMarker =
025   calculationMarker match {
026     // Extract properties from object for simpler access
027     case CalculationMarker(_, (predictionMarker, imputeMarker)) =>
028       // Alleles can only contain [A, T, C, G]
029       if (!complimentaryMap.contains(imputeMarker.alleleA) ||
030         !complimentaryMap.contains(imputeMarker.alleleB)) {
031         return UnknownAlleles(calculationMarker).left
032       }
033       // Alleles should not be the same
034       if (imputeMarker.alleleA == imputeMarker.alleleB) {
035         return ImputationMistake(calculationMarker).left
036       }
037       // Check if effect allele exists on one of the markers
038       if (predictionMarker.effectAllele != imputeMarker.alleleA &&
039         predictionMarker.effectAllele != imputeMarker.alleleB) {
040
041         val complimentary = imputeMarker.complimentary
042         // Check again if effect allele exists in case of complimentary
043         if (predictionMarker.effectAllele != complimentary.alleleA &&
044           predictionMarker.effectAllele != complimentary.alleleB) {
045           return MarkersUncomparable(calculationMarker, complimentary).left
046         }
047
048         return CalculationMarker(calculationMarker.calcId, (predictionMarker,
    complimentary)).right
049       }
050
051       return calculationMarker.right
052   }
053
054 def processImputeMarker(imputeMarker: ImputeMarker, model: Model):
055 (Vector[Double], GenotypeFrequencies, Double) = {
056   // Vector append is eC
057   // @specialized tells the computer to optimize this vector
058   @specialized var alleleDosages: Vector[Double] = Vector()
059   var count = 0.0 // Total individuals count
060   var AASum = 0.0 // AA genotype sum
061   var ABSum = 0.0 // AB genotype sum
062   var BBSum = 0.0 // BB genotype sum
063   var BSum = 0.0 // Used for eaf calculation
064
065   for (triplet <- imputeMarker.triplets) {
066     // Append dosage
067     alleleDosages = alleleDosages :+ model.alleleDosageFn(triplet)
068     AASum += triplet.head
069     ABSum += triplet(1)
```

```
070       BBSum += triplet(2)
071       // Current position max B frequency from AB, BB
072       BSum += triplet(1) + 2.0 * triplet(2)
073       count += 1
074     }
075
076     val imputeMarkerEAF = BSum / (2.0 * count)
077
078     val genotypeFreq = GenotypeFrequencies(
079       imputeMarker.alleleA
080       , imputeMarker.alleleB
081       , AA = AASum / count
082       , AB = ABSum / count
083       , BB = BBSum / count)
084
085     return (alleleDosages, genotypeFreq, imputeMarkerEAF)
086   }
087
088   def calculateIntermediate(calculationMarker: CalculationMarker, model: Model):
089   CalculationMarkerError \/ IntermediateResult = calculationMarker match {
090     case CalculationMarker(calculationId, (predictionMarker, imputeMarker)) => {
091
092       val (alleleDosages, genotypeFrequencies, imputeMarkerEAF) =
093         processImputeMarker(imputeMarker, model)
094
095       val (markerType, effectDirection, correctionFn) = if (imputeMarker.isPalindromic) {
096
097         // Markers are same within similarityThreshold precision
098         // |________|      |<  im.eaf|____|_____________________________|
099         // |            |< pm.EAF       |                             |
100         if (math.abs(predictionMarker.eaf - imputeMarkerEAF) <= similarityThreshold)
101           (Palindromic, Same, model.same _)
102
103         // Markers are Different within similarityThreshold precision
104         // |_____________________________|_________|      |< -im.eaf|_____|
105         // |                             |           |< pm.EAF           |
106         else if (math.abs(predictionMarker.eaf - (1 - imputeMarkerEAF)) <=
    similarityThreshold)
107           (Palindromic, Different, model.reverse _)
108
109         // Others that we cannot use
110         // |_____________________________|_________|      |<  im.eaf|_____|
111         // | |< pm.EAF                   |                               |
112         //
113         // |________|      |<  im.eaf|____|_____________________________|
114         // |                           | |< pm.EAF                       |
115         else
116           return UndetectablePalindromicMarker(calculationMarker, imputeMarkerEAF).left
117       } else {
118
119         // Alleles match both in model and imputed data
120         if (predictionMarker.effectAllele == imputeMarker.alleleB)
121           (Regular, Same, model.same _)
122
123         // Alleles differ in model and imputed data
124         else
125           (Regular, Different, model.reverse _)
126       }
127
128       // Combined Beta estimation
129       val multiplier = predictionMarker.beta * predictionMarker.coefficient
130
131       // Marker effect for all individuals
132       val markerEffects = alleleDosages.map((i) => correctionFn(i) * multiplier)
133
134       return IntermediateResult(calculationId
135         , markerName = predictionMarker.markerName
136         , predictionMarkerId = predictionMarker.id
137         , imputeMarkerId = imputeMarker.id
138         , genotypeFrequencies = genotypeFrequencies
139         , imputeMarkerEAF = imputeMarkerEAF
140         , markerType = markerType
141         , effectDirection = effectDirection
142         , complimentaryUsage = imputeMarker.complimentaryUsage
```

```
143         , multiplier = multiplier
144         , markerEffects = markerEffects
145       ).right
146     }
147   }
```

Appendix 3. Database model



play_evolutions

Stores database migration scripts.

Name	Database type	Comments
id	INT(11)	Primary key
hash	VARCHAR(255)	
applied_at	TIMESTAMP	
apply_script	TEXT	SQL to apply this database migration
revert_script	TEXT	SQL to revert this database migration
state	VARCHAR(255)	State of current migration
last_problem	TEXT	Description of last problem with this migration

prediction_models

Stores metadata for prediction models.

Name	Database type	Comments
prediction_model_id	INT(11)	Primary key
name	VARCHAR(254)	Human readable name for the model
modified	TIMESTAMP	Last modification timestamp
age_group_start	DOUBLE	Age group start – NOT ACTIVE
age_group_end	DOUBLE	Age group end – NOT ACTIVE
age_group_unit	VARCHAR(254)	Age group units – NOT ACTIVE
genders	VARCHAR(254)	Genders, which the model applies. Possible values “Men”, “Women”, “Both” – NOT ACTIVE
allele_dosage_fn	VARCHAR(254)	Identifier for a function to calculate allele dosages. Currently possible value “0 1 2” – NOT ACTIVE
marker_effect_combination_fn	VARCHAR(254)	Identifier for a function to combine allele dosages. Currently possible values “Sum” – NOT ACTIVE
reference_genome_id	INT(11)	Nullable. Foreign key to referece_genomes – NOT ACTIVE
status	SMALLINT(6)	“Deleted”, “Unknown”, “Usable” - NOT ACTIVE
created_on	TIMESTAMP	Timestamp of model creation
created_by	VARCHAR(254)	Creator identificator. Currently USER or ENSEMBL
comments	TEXT	User entered comments.

prediction_markers

Prediction markers for one prediction model.

Name	Database type	Comments
prediction_marker_id	INT(11)	Primary key
marker_name	VARCHAR(254)	Human readable name for the marker
chr	VARCHAR(254)	Chromosome
chr_position	INT(11)	Position in chromosome

effect_allele	VARCHAR(254)	Effect allele
ref_allele	VARCHAR(254)	Other allele
eaf	DOUBLE	Effect allele frequency in source study
beta	DOUBLE	Beta of effect_allele in source study
coefficient	DOUBLE	Additional coefficient for double weighted
p_value	DOUBLE	p-value in source study
prediction_model_id	INT(11)	Foreign key to prediction_models
Converted	TINYINT(1)	0 – Original beta, 1- Beta converted to positive

prediction_model_diagnoses

Link prediction model to diagnoses

Name	Database type	Comments
prediction_model_diagnose_id	INT(11)	Primary key
prediction_model_id	INT(11)	Foreign key to prediction_models
diagnose_id	INT(11)	Foreign key to diagnoses

prediction_model_medications

Link prediction model to medications.

Name	Database type	Comments
prediction_model_medication_id	INT(11)	Primary key
prediction_model_id	INT(11)	Foreign key to prediction_models
medications_id	INT(11)	Foreign key to diagnoses

prediction_calculations

Store meta info about prediction calculation

Name	Database type	Comments
prediction_calculation_id	INT(11)	Primary key
prediction_model_id	INT(11)	Foreign key to prediction_models
impute_set_id	INT(11)	Foreign key to impute_sets
duration_ms	BIGINT(20)	Duration of calculation in milliseconds
created_on	TIMESTAMP	Creation timestamp of calculation
status	CHAR(45)	Status of calculation. Possible values are “Running, Completed”
min_score	DOUBLE	Minimum calculated score
max_score	DOUBLE	Maximum calculated score
calculation_start	TIMESTAMP	Calculation start time – NOT ACTIVE
calculation_end	TIMESTAMP	Calculation end time – NOT ACTIVE

prediction_results

Stores risk score for each position in impute_positions

Name	Database type	Comments
prediction_result_id	INT(11)	Primary key
prediction_calculation_id	INT(11)	Foreign key to prediction_calculations
sample_code	VARCHAR(255)	Code for sample in the impute_set_positions
score	DOUBLE	Calculated total score for given sample
sample_position	INT(11)	Sample position in impute_marker genotype for this calculation – NOT ACTIVE replaced by sample_code

prediction_reporting_thresholds

Stores information for each risk group for the calculation.

Name	Database type	Comments
prediction_reporting_threshold_id	INT(11)	Primary key
prediction_calculation_id	INT(11)	Foreign key to prediction_calculations
name	VARCHAR(255)	Name of threshold currently “Low, Medium, High”
min_score	DOUBLE	Minimum in threshold range
max_score	DOUBLE	Maximum in threshold range

influencial_markers

Stores marker effects for selected markers for each calculation.

Name	Database type	Comments
id	INT(11)	Primary key
prediction_calculation_id	INT(11)	Foreign key to prediction_calculations
impute_marker_id	BIGINT(20)	Foreign key to impute_markers
prediction_marker_id	BIGINT(20)	Foreign key to prediction_markers
marker_type	INT(11)	0 – Complimentary, 1-Palindromic, 2-Regular
effect_direction	INT(11)	1 – Same, -1- Different
complimentary_usage	INT(11)	1 – Used, -1 – Unused
uncompressed_me_length	INT(11)	Array[Byte] length of marker effects before compression
compressed_me_length	INT(11)	Array[Byte] length of marker effects after compression
compressed_me_data	BLOB	Array[Byte] of Lz4 compressed marker effects separated by “|”

impute_sets

Group of one impute2 dataset.

Name	Database type	Comments
impute_set_id	INT(11)	Primary key
name	VARCHAR(254)	Human readable name for the impute set
created_on	TIMESTAMP	Creation date
reference_genome_id	INT(11)	Nullable. Links impute dataset to specific reference genome

impute_markers

All the marker information in the impute2 dataset.

Name	Database type	Comments
impute_marker_id	BIGINT(20)	Primary key
impute_set_id	INT(11)	Foreign key to impute_sets
marker_name	VARCHAR(254)	Human readable name for the marker
allele_a	VARCHAR(254)	First allele for the marker
allele_b	VARCHAR(254)	Second allele for the marker
chr	VARCHAR(254)	Chromosome
chr_position	INT(11)	Position in the chromosome
uncompressed_length	INT(11)	Array[Byte] length of reduced Impute2 genotypes before compression
compressed_length	INT(11)	Array[Byte] length of reduced Impute2 genotypes after compression
compressed_data	BLOB	Array[Byte] of reduced Impute2 genotypes in Lz4 compression

impute_positions

Matches sample file in impute2 dataset

Name	Database type	Comments
impute_position_id	INT(11)	Primary key
impute_set_id	INT(11)	Foreign key to impute_sets
id_1	VARCHAR(254)	Saved to preserve sample file structure
id_2	VARCHAR(254)	Used to match code in individual_samples to determine individual
missing	VARCHAR(254)	Saved to preserve sample file structure
father	SMALLINT(6)	Saved to preserve sample file structure
mother	SMALLINT(6)	Saved to preserve sample file structure
sex	SMALLINT(6)	Saved to preserve sample file structure
plink_pheno	SMALLINT(6)	Saved to preserve sample file structure
position	INT(11)	Position of current sample in the impute_markers compressed_data

reference_genomes

Table to keep track over used reference genomes.

Name	Database type	Comments
reference_genome_id	INT(11)	Primary key
name	VARCHAR(254)	Human readable name for reference genome
created_on	TIMESTAMP	Creation date

individuals

Table to identify individual.

Name	Database type	Comments
individual_id	INT(11)	Primary key
individual_code	VARCHAR(254)	Code to identify individual
birth_year	SMALLINT(6)	Individual birth year
birth_month	TINYINT(4)	Individual birth month
gender	VARCHAR(45)	Gender: “Unknown”, “M”, “F”

individual_samples

Table to store individual’s samples. Used to link with imputed data or prediction results.

Name	Database type	Comments
individual_id	INT(11)	Foreign key to individuals
individual_sample_id	INT(11)	Primary key
code	VARCHAR(254)	Code to identify sample

individual_diagnoses

Table to link individual with their diagnoses.

Name	Database type	Comments
individual_id	INT(11)	Foreign key to individuals
individual_diagnose_id	INT(11)	Primary key
diagnose_id	INT(11)	Foreign key to diagnoses
year	INT(11)	Year of diagnose
month	INT(11)	Month of diagnose
day	INT(11)	Day of diagnose
reliability	INT(11)	Reliability of diagnose. Possible values ()
hasNow	TINYINT(1)	Weather the individual had the diagnose when joining

source	VARCHAR(45)	Source of individuals diagnose: “EGCUT”, “EHK”, “PERH”, “TYK”, “SPR”

individual_medications

Diagnoses for individual

Name	Database type	Comments
id	INT(11)	Primary key
status_code	VARCHAR(50)	Status code for the diagnose given by Estonian Health Foundation (EHF)
individual_id	INT(11)	Foreign key to individuals
status_name	VARCHAR(50)	Status name given by EHF
icd10_code	VARCHAR(50)	ICD10 code related to the medication
prescribed_date	TIMESTAMP	Time medication was prescribed
prescribed_atc_code	VARCHAR(50)	ATC code for the medication
prescription_code	VARCHAR(254)	EHF code for prescription
prescription_kk_code	VARCHAR(254)	EHF old code for prescription
prescription_erp_code	VARCHAR(254)	EHF middle generation code for prescription
prescription_name	VARCHAR(512)	EHF prescription name
pickup_date	TIMESTAP	Time the prescription was puchased
pickup_atc_code	VARCHAR(50)	ATC code for the pickup prescription
source	VARCHAR(45)	Source of the medication info EHF by default

individual_objective_measurements

Different hip, waist, weight and height measurements.

Name	Database type	Comments
id	INT(11)	Primary key
year	SMALLINT(11)	Year of measurement
month	TINYINT(11)	Month of measurement
day	TINYINT(11)	Day of measurement
hip	DOUBLE	Hip measurement
waist	DOUBLE	Waist measurement
height	DOUBLE	Height measurement
weight	DOUBLE	Weight measurement
individual_id	INT(11)	Foreign key to individuals

diagnoses

Stores diagnoses found in ICD-10 classification.

Name	Database type	Comments

diagnose_id	INT(11)	Primary key
icd_10_code	VARCHAR(254)	ICD-10 code
icd_10_name	VARCHAR(1024)	ICD-10 name in Estonian
icd_10_level	INT(11)	ICD-10 is hierarchical. Levels start from 0 and increases for each group of children
icd_10_parent	INT(11)	diagnose_id of parent element or NULL if there is none.

medications

Stores medications found in ATC-10 classification.

Name	Database type	Comments
medication_id	INT(11)	Primary key
atc_10_code	VARCHAR(254)	ATC-10 code
atc_10_name	VARCHAR(1024)	ATC-10 name in Estonian
atc_10_level	INT(11)	ATC-10 is hierarchical. Levels start from 0 and increases for each group of children
atc_10_parent	INT(11)	diagnose_id of parent element or NULL if there is none.

Appendix 4. Spark cluster configuration with hadoop-lzo

On OSX, installation of Homebrew with *lzop* and *lzo* packages is required. In Linux distributions *lzo* development libraries.

```
# Navigate to user folder
cd /home/user

# Clone git repo with some premade configurations
git clone git@github.com:kmetsalu/spark-with-lzo.git spark-with-lzo
cd /home/user/spark-with-lzo

# Export BASE_DIR variable for easier use.
export $BASE_DIR=`pwd`

# Download Spark 1.2.0
wget http://www.apache.org/dyn/closer.cgi/spark/spark-1.2.0/spark-1.2.0-bin-hadoop2.4.tgz
tar -xzvf spark-1.2.0-bin-hadoop2.4.tgz
mv spark-1.2.0-bin-hadoop2.4 spark

# Download hadoop-lzo
git clone https://github.com/twitter/hadoop-lzo.git hadoop-lzo
cd $BASE_DIR/hadoop-lzo
git checkout release-0.4.19

# Build native libraries according to hadoop-lzo instructions
# After build there should be target directory in hadoop-lzo folder
# Make new directory to store current compilation
mkdir $BASE_DIR/hadoop-lzo/current

# Copy required native libraries to created directory for Linux
# folder name within native directory differs, but everything else
# is the same
cp -R $BASE_DIR/hadoop-lzo/target/native/Mac_OS_X-x86_64-64/lib/ $BASE_DIR/hadoop-
lzo/current/native

# Copy created jar
cp $BASE_DIR/hadoop-lzo/target/hadoop-lzo-0.4.19.jar $BASE_DIR/hadoop-lzo/current

# Copy configuration files to spark
cp $BASE_DIR/spark-conf/conf/* $BASE_DIR/spark/conf/
mv $BASE_DIR/spark/sbin/spark-config.sh{,.backup}
cp $BASE_DIR/spark-cond/sbin/* $BASE_DIR/spark/sbin/

# Spark now runs with lzo support and on Linux with Lz4 and Snappy support
$BASE_DIR/spark/bin/spark-shell --master local[*]
```

Appendix 5. MySQL configuration

```
[mysqld]
datadir=/local/mysql
socket=/var/lib/mysql/mysql.sock
user=mysql
key_buffer_size = 10000M
max_allowed_packet = 256M
table_cache = 512
sort_buffer_size = 64M
read_buffer_size = 256M
read_rnd_buffer_size = 128M
myisam_sort_buffer_size = 256M
thread_cache_size = 128
query_cache_size= 64M
thread_concurrency = 8
innodb_buffer_pool_size = 100000M

# Disabling symbolic-links is recommended to prevent assorted security risks
symbolic-links=0
bulk_insert_buffer_size = 128M
delayed_insert_limit = 2000
innodb_log_file_size = 512M
innodb_flush_method = O_DIRECT
innodb_thread_concurrency = 16
innodb_log_buffer_size = 32M
innodb_file_per_table=1

[mysqld_safe]
log-error=/var/log/mysqld.log
pid-file=/var/run/mysqld/mysqld.pid
```

Appendix 6. Source code

Source code for this application is available at:

https://cloud.biobank.ee/index.php/s/YCwiYYkYJi5B4HW

There is a README.md file included with the source code with further instructions.

Appendix 7. License

Non-exclusive license to reproduce thesis and make thesis public

I, Kristjan Metsalu (date of birth: 19.12.1986),

1. herewith grant the University of Tartu a free permit (non-exclusive license) to:

1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright, of my thesis

Web application to calculate genetic risk scores based on imputed data.

supervised by Prof. Jaak Vilo, Reedik Mägi PhD and Tatjana Iljashenko MSc

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive license does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, 21.05.2015