Managing Variant Calling Files the Big Data Way

Using HDFS and Apache Parquet

Aikaterini Boufea
Centre for Genomic & Experimental Medicine MRC IGMM
University of Edinburgh
Edinburgh, UK
katerina.boufea@ed.ac.uk

Richard Finkers
Plant Breeding Group
Wageningen University & Research
Wageningen, The Netherlands

Martijn van Kaauwen
Plant Breeding Group
Wageningen University & Research
Wageningen, The Netherlands

Mark Kramer
Information Technology Group
Wageningen University & Research
Wageningen, The Netherlands

Ioannis N. Athanasiadis
Information Technology Group
Wageningen University & Research
Wageningen, The Netherlands
ioannis@athanasiadis.info

ABSTRACT

Big Data has been seen as a remedy for the efficient management of the ever-increasing genomic data. In this paper, we investigate the use of Apache Spark to store and process Variant Calling Files (VCF) on a Hadoop cluster. We demonstrate Tomatula, a software tool for converting VCF files to Apache Parquet storage format, and an application to query variant calling datasets. We evaluate how the wall time (i.e. time until the query answer is returned to the user) scales out on a Hadoop cluster storing VCF files, either in the original flat-file format, or using the Apache Parquet columnar storage format. Apache Parquet can compress the VCF data by around a factor of 10, and supports easier querying of VCF files as it exposes the field structure. We discuss advantages and disadvantages in terms of storage capacity and querying performance with both flat VCF files and Apache Parquet using an open plant breeding dataset. We conclude that Apache Parquet offers benefits for reducing storage size and wall time, and scales out with larger datasets.

CCS CONCEPTS

• **Applied computing → Bioinformatics**; Agriculture; • **Information systems** → Data management systems;

KEYWORDS

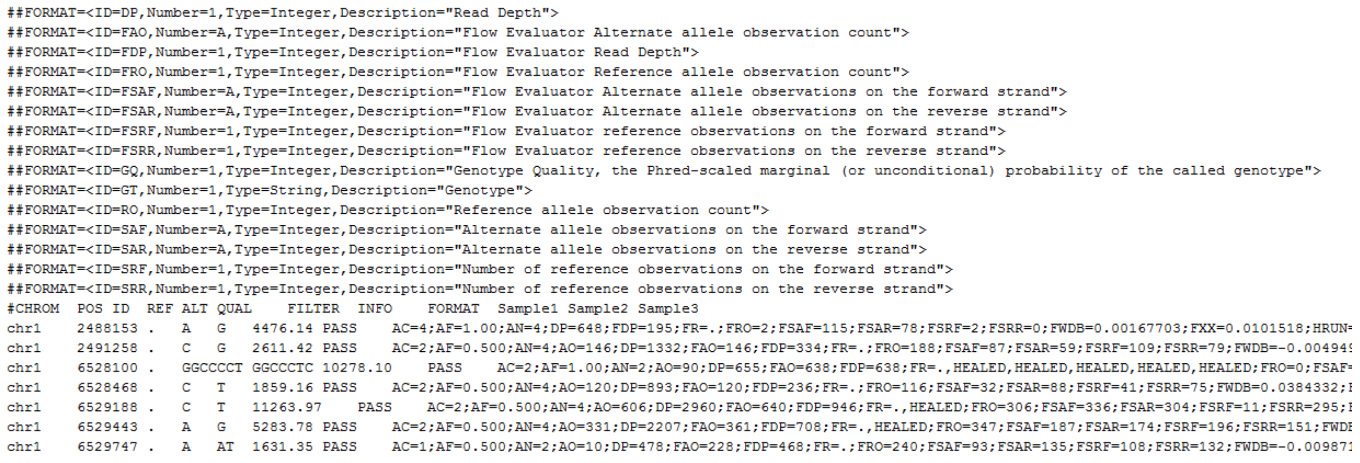
Big Data, bioinformatics, variant calling, Hadoop, HDFS, Apache Spark, Apache Parquet, Tomatula

BDCAT'17, December 5–8, 2017, Austin, TX, USA
© 2017 Copyright held by the owner/author(s). Publication rights licensed to Association for Computing Machinery.
ACM ISBN 978-1-4503-5549-0/17/12…$15.00
https://doi.org/10.1145/3148055.3148060

1 INTRODUCTION

The introduction of Next-Generation Sequencing enabled the fast and cheap sequencing of whole genomes [2, 18]. Today, even small laboratories are able to work on whole-genome sequencing projects and generate vast volumes of genomic data. DNA sequences provide information on transcription, translation, subcellular localization, phenotypic and genotypic variation, replication, recombination, protein - DNA interactions and many more. They may even reveal novel regions of the genome that express previously unknown genes. This new information can have a great impact in the knowledge scientists have about specific organisms, and explain some of their observable and hidden traits, such as susceptibility to a disease or other cell functions. Thus, research is focused on exploring the genome variation among individuals of the same or different species. Because of the large data volume, but also the variety of sources of origin and the velocity new data are generated, we can characterize genomic data as Big Data. However, only a few genomic frameworks are scalable [14, 16].

Big Data platforms such as Hadoop [3] and Apache Spark [5, 24] have been seen as a remedy, enabling efficient storage, fast retrieval and processing of genomic data. Big data methods have been used for the redesign of bioinformatics algorithms and tools. In CloudBurst [20] and Crossbow [13] projects, Hadoop was used to parallelize the procedure of mapping next-generation sequencing data to a reference genome for genotyping and single-nucleotide polymorphisms (SNP) discovering. CloudBurst is a parallel version of RMAP a seed-and-extend read-mapping algorithm that employs Hadoop, and demonstrated to execute RMAP up to 30 times faster than on a single computing machine [20]. Crossbow deployed a similar-purpose tool using cloud computing services [13]. Hadoop was also used in Myrna, a cloud-computing pipeline for calculating differential gene expression from large RNA-seq datasets [12]. SparkSeq is a general-purpose library for genomic cloud computing, that provides methods and tools to build genomic analysis pipelines and query data interactively [23]. VariantSpark makes use of Spark's machine learning library, MLib, to develop a tool for clustering variant calling data [19]. A tool for processing genomic reads and converting them to variant callings is ADAM [17].



```
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Read Depth">
##FORMAT=<ID=FAO,Number=A,Type=Integer,Description="Flow Evaluator Alternate allele observation count">
##FORMAT=<ID=FDP,Number=1,Type=Integer,Description="Flow Evaluator Read Depth">
##FORMAT=<ID=FRO,Number=1,Type=Integer,Description="Flow Evaluator Reference allele observation count">
##FORMAT=<ID=FSAF,Number=A,Type=Integer,Description="Flow Evaluator Alternate allele observations on the forward strand">
##FORMAT=<ID=FSAR,Number=A,Type=Integer,Description="Flow Evaluator Alternate allele observations on the reverse strand">
##FORMAT=<ID=FSRF,Number=1,Type=Integer,Description="Flow Evaluator reference observations on the forward strand">
##FORMAT=<ID=FSRR,Number=1,Type=Integer,Description="Flow Evaluator reference observations on the reverse strand">
##FORMAT=<ID=GQ,Number=1,Type=Integer,Description="Genotype Quality, the Phred-scaled marginal (or unconditional) probability of the called genotype">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##FORMAT=<ID=RO,Number=1,Type=Integer,Description="Reference allele observation count">
##FORMAT=<ID=SAF,Number=A,Type=Integer,Description="Alternate allele observations on the forward strand">
##FORMAT=<ID=SAR,Number=A,Type=Integer,Description="Alternate allele observations on the reverse strand">
##FORMAT=<ID=SRF,Number=1,Type=Integer,Description="Number of reference observations on the forward strand">
##FORMAT=<ID=SRR,Number=1,Type=Integer,Description="Number of reference observations on the reverse strand">
#CHROM  POS ID  REF ALT QUAL    FILTER  INFO    FORMAT  Sample1 Sample2 Sample3
chr1    2488153 .   A   G   4476.14 PASS    AC=4;AF=1.00;AN=4;DP=648;FDP=195;FR=.;FRO=2;FSAF=115;FSAR=78;FSRF=2;FSRR=0;FWDB=0.00167703;FXX=0.0101518;HRUN=
chr1    2491258 .   C   G   2611.42 PASS    AC=2;AF=0.500;AN=4;AO=146;DP=1332;FAO=146;FDP=334;FR=.;FRO=188;FSAF=87;FSAR=59;FSRF=109;FSRR=79;FWDB=-0.00494
chr1    6528100 .   GGCCCCT GGCCCTC 10278.10    PASS    AC=2;AF=1.00;AN=2;AO=90;DP=655;FAO=638;FDP=638;FR=.,HEALED,HEALED,HEALED,HEALED,HEALED;FRO=0;FSAF=
chr1    6528468 .   C   T   1859.16 PASS    AC=2;AF=0.500;AN=4;AO=120;DP=893;FAO=120;FDP=236;FR=.;FRO=116;FSAF=32;FSAR=88;FSRF=41;FSRR=75;FWDB=0.0384332;
chr1    6529188 .   C   T   11263.97    PASS    AC=2;AF=0.500;AN=4;AO=606;DP=2960;FAO=640;FDP=946;FR=.,HEALED;FRO=306;FSAF=336;FSAR=304;FSRF=11;FSRR=295;
chr1    6529443 .   A   G   5283.78 PASS    AC=2;AF=0.500;AN=4;AO=331;DP=2207;FAO=361;FDP=708;FR=.,HEALED;FRO=347;FSAF=187;FSAR=174;FSRF=196;FSRR=151;FWD
chr1    6529747 .   A   AT  1631.35 PASS    AC=1;AF=0.500;AN=2;AO=10;DP=478;FAO=228;FDP=468;FR=.;FRO=240;FSAF=93;FSAR=135;FSRF=108;FSRR=132;FWDB=-0.00987
```

Figure 1: Example of a VCF format file. Header lines start with the '#' character and contain the file metadata. Variant Call records follow, with each line corresponding to a variant location on the reference genome. They include both general information columns and one column for each sequenced sample.

ADAM makes use of Apache Parquet to develop a new data columnar storage and Spark to parallelize the processing pipeline of transforming genomic reads to variant-calling ready reads, accelerating the variant calling procedure. Similarly, Halvade is a framework for executing sequencing pipelines in parallel, and has implemented a DNA sequencing analysis pipeline for variant calling using traditional algorithms such as the Burrow-Wheeler Aligner (BWA) and the Genome Analysis Toolkit (GATK) [9]. One of the latest Hadoop libraries is FASTdoop [10] designed to distribute efficiently collections of long sequences (FASTA files) instead of short reads (SAM/BAM) without losing the information of each piece's origin.

Previous work has also investigated the potential benefits from adopting a more generalized storage format than the standard flat-text files, that enables structured querying. VCF-miner, converts a VCF file to a JSON dictionary as a pre-processing step [11], whereas BioPig [18], SeqPig [21], SBtab [15] and GMQL [16] suggest the tabular storage format that can be queried with an SQL-like language.

The above efforts focus on applying Big Data techniques for handling sequence data files like FASTA and BAM and parallelizing processing pipelines, such as variant calling. Sequence files are usually big and require heavy processing. However, once these files are processed, there is no need to access their content. Studies are then based on the results of processing, which include VCF files that contain combined information of genome variability across several samples. Large cohort studies can have thousands of samples. Enabling fast access and retrieval of information from VCF files can give the opportunity to researchers to ask questions from the data without being limited by time cost. This can be additionally supported by the interactive environment of Spark. Thus, in this work, we focus on using Big Data tools for storing and querying variant calling data. Specifically, we stored VCF datasets on a Hadoop Distributed File System (HDFS) and investigated the effects of storage parameters, such as the replication factor, on the performance of querying applications. We used Apache Spark's Python API for the applications and tested the ability of the system to scale-out by making available more computing nodes. Finally, we investigated potential gains from storing variant calling data in columnar format using Apache Parquet. For this purpose, we developed Tomatula[1], an open-source software tool for converting VCF files to the Apache Parquet format.

To test the different parameters of the distributed filesystem and evaluate the Apache Parquet columnar storage format, we developed a demonstrating application that returns the allele frequencies of variant sites within a requested region on the chromosome. The software tool, the demonstration scripts and an example dataset (intentionally very small for educational purposes) are available as open software repository on GitHub, and as supplementary material.

2 MATERIALS AND METHODS

2.1 Big data tools

Apache Spark is a fast and general-purpose cluster computing system for in-memory computations, overcoming the Hadoop MapReduce bottleneck of communicating to hard disks [24]. By running in-memory computations, Spark can perform up to 100x faster than Hadoop MapReduce operation, allowing also for interactive querying of the data [19]. The master/slave architecture of Apache Spark consists of three layers:

(a) the *Cluster resource manager* for managing the filesystem, i.e. Hadoop YARN, Apache Mesos or Standalone Scheduler;
(b) the *Driver program* responsible for coordinating the application tasks and running the main method of the script; and
(c) the *Application layer* providing APIs that create and manage the Resilient Distributed Datasets (RDDs) – the main programming abstraction of Apache Spark.

[1]https://www.github.com/bigdatawur/tomatula/

Spark applications run as multiple independent RDD operations that can be parallelized on several worker nodes. The driver program running on the master node is responsible for collecting and combining the results of the tasks and return the output of the query.

Apache Parquet offers a columnar storage format, where data are stored in column chunks, which are further split into row groups and spread over the worker nodes. Each column chunk is composed of pages written back to back, sharing the same header. Apache Parquet system supports also two types of metadata for file and column metadata. File metadata point on the locations of all the column metadata start locations. Column metadata store location information for all the column chunks. Readers access first the file metadata to locate all the column chunks they are interested in, and subsequently use column metadata to skip over non-relevant pages [4]. A Parquet table can be easily distributed over many nodes and the main advantage is that using metadata accessing applications can directly jump to the appropriate fields of the record. In the case of vcf data, metadata will include the positions of all fields (columns) as well as embedded information (pages) of each column, as for example the Allele Frequency (AF) part of the INFO field. Moreover, the Parquet storage system offers high compression and reduces network traffic by filtering out early irrelevant column data.

2.2 VCF files and tools

The **Variant Call Format** (VCF) is a standard tab-delimited file format used to represent single-nucleotide polymorphisms (SNP), insertions and deletions, and structural variation calls, storing only the variations, along with a reference genome. The VCF format is a flat-file format without a strict predefined schema and metadata. The file is composed of two main parts; the header and the variant call records (Figure 1). The header contains information about the dataset and relevant reference sources, such as the organism or the genome build version. It can also contain definitions of the annotations used to qualify and quantify the properties of the variant calls contained in the VCF file. The last header line contains the field names of the data lines. Each data line represents one variant with its properties. The fields of a data line include the chromosomal location, the reference and alternative sequence, and the frequency of the alternative sequence among the samples. Additionally, there is one field per sample with information on the observed sequence and statistical information, such as the quality of the sample [8].

The VCF format provides a compact way to to summarize genomic information of a sample and compare it in a systematic way to other samples. It is an important source of information in studies aiming to link phenotypic traits to genetic mutations and statistically test such relationships. Data can be combined and analyzed in several different ways depending on the research questions. In the era of personalized medicine, variant calling data can give clinitians insight on the disease progression and the appropriate treatment for an individual based on the genomic data and information from other samples.

VCFtools[2], the most widely used program package for working with VCF files, uses indexes based on the chromosomal location of each variant site. This is an efficient way to find quickly information about a specific region or site but has no functionality when searching for specific columns of the file, for example a sub-set of the individuals listed in columns. Especially when a study includes hundreds or thousands of individuals, such queries can have very limited performance.

[2]https://vcftools.github.io

```
|-- CHROM: long
|-- POS: long
|-- ID: string
|-- REF: string
|-- ALT: string
|-- QUAL: string
|-- FILTER: string
|-- INFO: struct
|    |-- AB: string
|    |-- ABP: string
...
|    |-- PAIRED: string
|    |-- PAIREDR: string
|-- LYC2740: struct
|    |-- AO: string
|    |-- DP: string
|    |-- GL: string
|    |-- GT: string
|    |-- QA: string
|    |-- QR: string
|    |-- RO: string
...
```

Figure 2: A part of the generated schema with embedded dictionaries for the fields `INFO` and individual `LYC2740`.

2.3 The conversion tool

Tomatula is a generic software tool we developed for converting flat VCF files into Apache Parquet columnar storage format. Tomatula operates in two steps: First, the VCF file is converted to JSON format, using the last header line to identify column names. Next, the JSON file is parsed and saved as a set of Parquet files, a procedure that is automatically done by Apache Spark[5]. VCF fields that contain more detailed information, such as the `INFO` column and the individuals fields, are further split into embedded dictionaries in a nested schema. An example schema of the generated structure is illustrated in Fig.2.

Tomatula converter was tested with VCF files from open datasets available at EMBL-EBI, including those of the 1,000 Genomes Project [6]; and the 100 Tomato Genome Sequencing Project [1].

2.4 Experimental setup on a cluster with Apache Hadoop and Apache Spark

The central idea of a Hadoop cluster architecture is a set of virtual machines forming a local network. We attempted two different approaches to deploy our system on such a network. Our first approach was to build our own Apache Spark cluster using the standalone cluster mode. We created virtual machines on a remote server and installed Apache Spark in each one of the machines.

One of the machines was selected to be the master of the cluster, and the rest the workers. Master and workers were communicating through a secure shell (SSH) connection. Network security was established by private IPs. The cluster consisted of one master node and three worker nodes. The system could scale-out further by adding more worker nodes when needed. A secondary master node could be also added to the design, to prevent loss of functionality in case the primary master node is down. In standalone mode, the data has to be stored in every node, which is not efficient for big data applications[3]. We used this standalone cluster to develop our application but all tests were run on the HDFS system presented next.

Apache Spark can be also deployed on top of an HDFS and run applications using more sophisticated cluster managers, such as YARN. In this setting, Spark RDDs load data using the URI of the underlying HDFS. The infrastructure of the Dutch national e-infrastructure offers such an environment, which we used with the support of SURF Cooperative [22]. SURF Hadoop cluster consists of 170 computing/data nodes and 1370 CPU-cores in total, that is 8 cores per node. The total memory is 10 TB and the available capacity of HDFS is 2.3 PB. The cluster runs Spark 1.6.1 on top of Hadoop HDFS 2.7.1 using YARN manager. In this environment it was possible to first store a (genomic) dataset on HDFS, and subsequently use Apache Spark for running applications. SURF Hadoop cluster nodes run Linux 2.6.32 with Java 1.7.0_79 and Scala 2.10.5. The driver and executor nodes have 6 GB of memory [22].

2.5 Experimental Design

We examined four main factors that can affect the performance of an Apache Spark cluster, using the SURF Hadoop cluster:

(a) the number of computing nodes of the cluster,
(b) the replication factor of HDFS,
(c) the storage format, and
(d) the size of the input files.

We executed our experiments for different cluster sizes, varying between 2 and 150 executor nodes to test the effect of the cluster size on the wall time, time until the query answer is returned to the user, and to what extend accessing VCF information can scale out.

Another parameter of HDFS that may affect the performance of the querying applications is the replication factor, referring to the number of copies of the file in the distributed filesystem. We tested the performance of our applications when increasing the copies of the dataset from the default value of 3 to 5, 7 and 9.

We compared two storage formats, the VCF flat-file and the Parquet columnar storage format. The original VCF file from the study of Aflitos et al. [1], contains variant calling information of whole-genome sequencing of 104 individuals. In order to test the performance of the two file formats and the designed system for bigger input files, we copied the individuals' information 10 times, resulting in a VCF file with 1144 individuals (columns). All other settings of the HDFS and the cluster, such as block size, were kept at the default values.

[3]See http://spark.apache.org/docs/latest/programming-guide.html#external-datasets

All experiments were executed five times and the detailed results are provided as supplementary material [7]. The metrics reported were provided by Spark Web UI and are based on the Coda Hale Metrics Library.

2.6 Querying applications

To investigate querying efficiency of storing VCF files on HDFS, either as flat-files or using Apache Parquet, we developed an application that retrieves the allele frequencies within a certain region of a VCF file. The allele frequency information is reported under the `INFO` field showing what is the frequency of presence of each alternative sequence on a genomic location among all samples. For example, a sample record line of VCF file from location 14370 of chromosome 20 is illustrated in Figure 3. The reference sequence has a guanine base (G), while an adenine (A) was observed in some of the samples with allele frequency 0.5 (encoded in the sub-field `AF` of the `INFO` field).

```
#CHROM POS   ID        REF ALT QUAL FILTER INFO              FORMAT      Sample1
20     14370 rs6054257 G   A   29   PASS   NS=3;DP=14;AF=0.5 GT:GQ:DP:HQ 0|0:48:1:51,51
```

Figure 3: An example of a VCF format showing the last header line with the field headers and a data line with all generic fields and a sample.

We executed a series of experiments, querying for a region of 2 000 bases in the file of chromosome 6 from the VCF files of the 100 Tomato Genome Sequencing Project [1], that corresponds to the approximate length of a gene. When querying the original VCF flat-file format, the application splits each line into a list, one for each field, and the `INFO` field is further split into the constituent parts to find the `AF` part and associate it to the `REF` and `ALT` columns.

In the contrary, using the Apache Parquet format, field names are available via the schema and accessible via a simple SQL-like query, which in Apache Spark syntax looks like:

```
vcf.select("CHROM","POS","REF","ALT","INFO.AF")\
   .filter("POS > " + lower + " AND POS < " + upper)
```

where `vcf` is a Resilient Distributed Dataset to which the Parquet file has been loaded. Note that schema fields are accessed with their names, and the same holds for nested fields (as in `INFO.AF` which corresponds to the allele frequency).

3 RESULTS

3.1 Cluster size

To test the effect of the cluster size we run all the applications for different cluster sizes between 2 (default) and 150 executor nodes. Although the performance was satisfactory already from a cluster size of around 50 nodes, we continued increasing the size as the duration was decreasing. We observed that a minimum duration is reached when around 100 nodes are used. After this point, the wall time slightly increases again mainly due to scheduling delays that are introduced. Figure 4 shows the performance of the PySpark version of the Allele Frequency application for the original VCF file with 104 individuals and replication factor of 5. A cluster with 10 worker nodes can achieve a 4-fold reduction of the wall time compared to the 2-worker cluster.

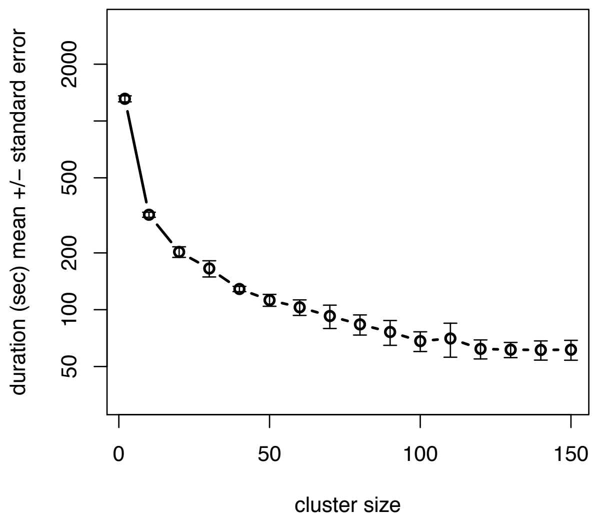


Figure 4: Wall time results for different cluster sizes. Lines show mean over 5 repetitions. Error bars indicate corresponding standard error. Illustration for Allele frequency application on the original VCF file (104 individuals) and replication factor 5

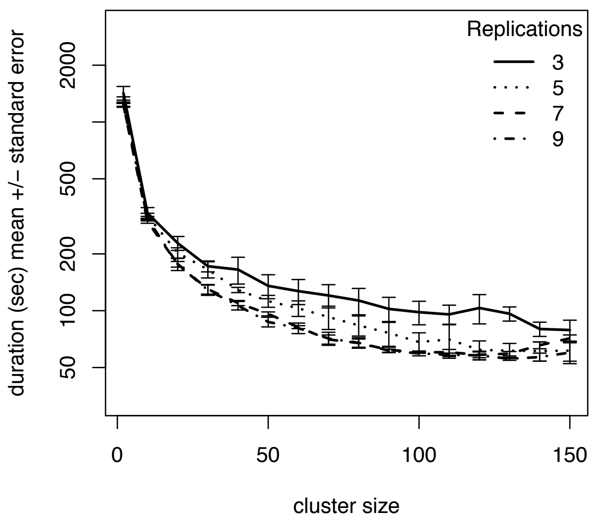


Figure 5: Wall time results for different replication factors and cluster sizes. Lines show mean over 5 repetitions. Error bars indicate corresponding standard error. Illustration for Allele frequency application on the original VCF file (104 individuals)

3.2 Replication factor

The replication factor is an important parameter of the distributed file system that can affect the wall time. There are two main advantages for having copies/replicates of the files stored in the HDFS. First, data is protected against loss due to DataNode failures. Second, since data is stored in multiple locations, it becomes more quickly accessible, avoiding an application being hold on the queue of a busy node. However, more copies of the file require more storage capacity. Thus, aspects such as storage cost and total size of the files have to be considered as well.

Figure 5 shows that, as expected, higher replication factor leads to faster query response. However, with replication factor of 7 a minimum is reached with no further improvement when increasing the copies of the data. Additionally, in this application, there are no significant gains in terms of wall time for increasing the replication from 5 to 7.

3.3 Converting files with Tomatula

We used Tomatula converter for transcribing variant calling data of wild species of the tomato clade, from the study of Aflitos et al. [1] into Apache Parquet. Due to its columnar storage, Apache Parquet offers very high compression, resulting to significant gains in terms of disk storage. The original VCF file contains the variants of 104 sequenced individuals and was approximately 606 GB. This file was split into separate VCF files per chromosome, sized between 10.3 GB and 75.4 GB. The size of chromosome 6 VCF file, which will be used in our experiments for querying allele frequencies, was 37.6 GB and the respective Parquet file was 8.8 GB. To explore future scenarios where more individuals were included, we copied the individuals' fields 10 times, ending up with an extended dataset of 1144 individuals. The resulting VCF file for chromosome 6 was 369.7 GB and in the Parquet format115.1 GB. Table 1 summarizes the file size of storing VCF files on HDFS as plain text and using Apache Parquet. The results demonstrate the capacity of the Apache Parquet format in terms of potential for saving storage capacity. This can support the storage and use of genomic information not only in research but also in clinical practice.

Table 1: File sizes of storing variant calling data of the tomato genome ([1]) on HDFS in VCF format and in Apache Parquet.

	VCF		Apache Parquet	
	Chrom 6	All Chrom.	Chrom 6	All Chrom.
104 Individuals	37.6 GB	471.3 GB	8.8 GB	112 GB
1144 Individuals	369.7 GB	4.53 TB	115.1 GB	1.43 TB

3.4 Accessing allele frequencies with Tomatula

Besides smaller file sizes, converting the VCF flat-file to a columnar storage format, as Apache Parquet) offers more advantages. First, it can save much time from the pre-processing steps needed when parsing the tab-delimited VCF format. Although this procedure is time consuming, it has to be done only once and the output can be re-used for all the queries. The table has to be updated only when new samples are added in the study.

Second, the application on the Apache Parquet format scales-out better, reaching an acceptable wall time with smaller number of executor nodes. Especially for the original dataset of 104 individuals, Apache Parquet format reaches the minimum from the default 2-executors cluster. Additionally, there is no need to increase the replication factor. Figure 6 illustrates the wall time when using Apache Parquet for storing the 104-individuals file, with cluster

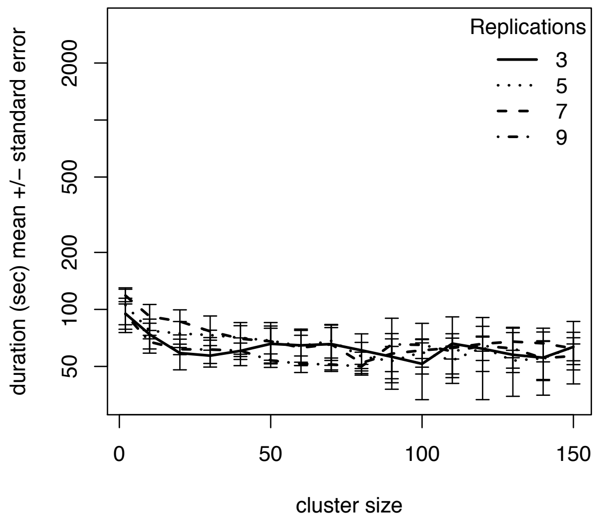


Figure 6: Wall time results for different replication factors and cluster sizes using the Apache Parquet format. Lines show mean over 5 repetitions. Error bars indicate corresponding standard error. Illustration for Allele frequency application on the original Parquet file (104 individuals)

sizes varying between 2 and 150 nodes, and replication factors of 3, 5, 7 and 9.

Third, all information stored under specific headers, is programmatically exposed, thus SparkSQL can be used, allowing not only for easy querying but also to take advantage of the interactive features of Apache Spark.

3.5 Increased input size

Similar conclusions regarding the effect of cluster size, replication factor and file format were reached using a bigger dataset of 1144 individuals (Figure 7). The HDFS replication factor had minimal influence to the results, and higher values rather contributed to increasing complexity. The wall time for both input formats increased, however, note that adding more nodes still improves the performance. It also worths noticing that we observed an overall lower duration of the method that used the VCF than the Parquet format. However, the difference is not significant and the Parquet format is still more efficient in terms of cluster size, needing 10-20 less worker nodes than VCF to reach the minimum duration (Figure 8).

4 DISCUSSION

Big Data methods can have important advantages for big variant calling projects. Through Tomatula conversion tool and our experimental evaluation, in this work we confirmed the capability to easily scale-out when files become bigger without the need for expensive hardware investment. Tomatula could be very easily executed on private or public cloud services that provide the ability to adjust the computing capacity depending on the demands of the application.

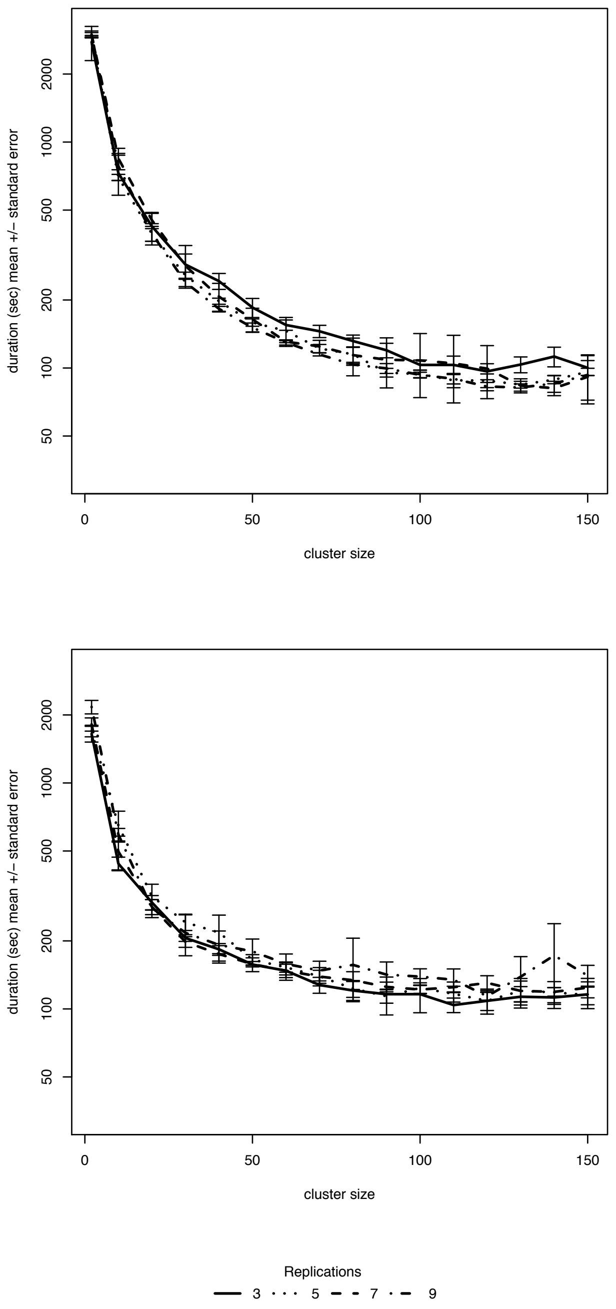


Figure 7: Wall time results of the extended dataset in both VCF and Parquet format, for varying cluster sizes and replication factors. Lines show mean over 5 repetitions. Error bars indicate corresponding standard error.

Apache Parquet offers a more generalized and highly compressed format, compared to the flat VCF format. Fields can be missing or empty without affecting the querying script. The SQL-like flavor allows to easily expand the schema including more data or fields.

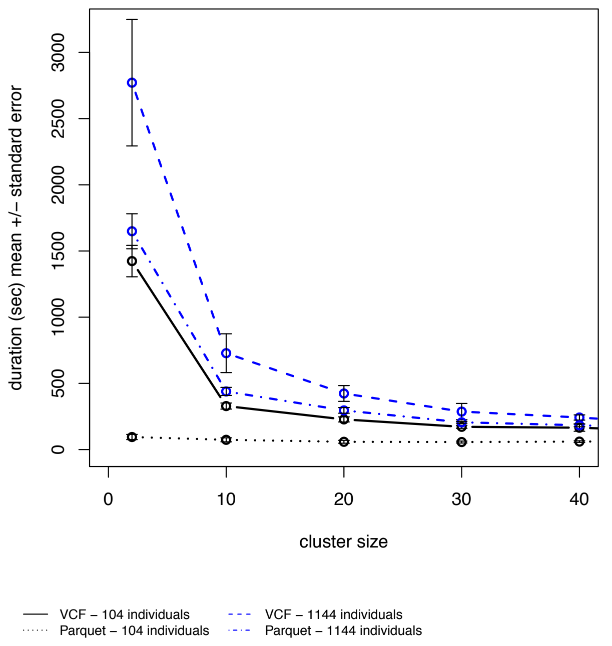


Figure 8: Wall time results for the original and the extended dataset in both VCF and Parquet format. Lines show mean over 5 repetitions. Error bars indicate corresponding standard error. Illustration for Allele frequency application for cluster sizes and between 2 and 40 worker nodes and replication factor 3.

Writing queries for datasets stored in Apache Parquet using Apache Spark is straightforward and allows for interactive querying. We demonstrated the ease of querying Parquet files with a simple application, encouraging scientists to experiment with their own queries using the interactive environment of Spark. The advantage of column metadata can be better demonstrated with queries that request information from multiple columns and embeded fields. A future direction will be to test the schema with more complex queries where data from multiple columns/rows can be combined. Such an interesting query can be the calculation of the alternate allele frequency of each sample. The columnar format of Apache Parquet is expected to be advantageous for distributing columns across worker nodes and combining the information of each sample's rows.

Future work may also be directed towards evaluating other Big Data systems that support better querying by row. Apache Avro provides a schema-enforced format similar to the JSON format but more efficient in terms of storage space and highly supported in the Hadoop environment. For variant calling data, the chromosomal position can be used as the key for fast access to every line of a VCF. Tomatula converter can be expanded to include VCF-to-Avro conversion.

Our experiments showed that the Apache Parquet format outperforms also in terms of the relationship between wall time and cluster size. As shown in our results, Apache Parquet requires less nodes to achieve a similar performance in comparison to the original flat-file format. Note that the querying response performance of flat-files with 104 individuals resembles that of Apache Parquet with 1144 individuals. In our experiments, Apache Parquet was able to handle 10 times more information using the same computing resources compared to a querying application used flat-files. In our estimation, bigger advantages in performance between the two methods will be found when querying for information stored in the fields of individuals.

Finally, we observed significant gains in hard disk storage capacity, which also favor Apache Parquet. The Parquet format can compress the VCF data by a factor of 10 as shown in Table 1. This is a very important advantage in large cohort studies but also in clinical practice, where genomic information of patients can be used for personalized treatment. In such a scenario, the healthcare system will need an efficient and flexible storage format for managing data from thousands of patients.

The cluster size, as expected, highly affects the performance of the wall time. More workers means higher computational power and faster response. However, there is a minimum duration that cannot be further reduced and using too many nodes can lead to the opposite results, with response increasing due to increased time for scheduling the tasks and collecting/reducing the results. In our experiments, we noted that there is not significant gain in wall time for increased replication factor. Yet more copies of the data can improve security against data loss in case a data node is down or damaged. Additionally, accessibility might be improved in more busy Hadoop clusters, in case resources are shared and it is more probable a data node to be busy with another task.

To summarize, we recommend the use of Big Data methods to query variant calling data and provide analytics services, and we encourage the investigation of other bioinformatics applications that can take advantage of parallel computing opportunities offered by the Apache Spark ecosystem. The gains can be better investigated with more complex pipelines, when computations have to be performed on the collected data to provide the results. Since Apache Spark application layer supports APIs for several programming languages, there is the potential of deploying existing bioinformatics tools on a Spark cluster. Still, Hadoop HDFS, Apache Spark and Apache Parquet are relatively new platforms, and are under development. We expect further improvements in Apache Parquet performance, when new features, such as indexing and lineage based recovery will be implemented. The migration to Spark 2.2.0 is also expected to improve the query performance. Spark 2.2.0 implements whole-stage code generation that improves execution performance by combining multiple operators into a single Java function and collapsing queries into a single function reducing memory calls of intermediate results. Finally, Spark 2.2.0 enables the processing of multiple rows together in a columnar format, providing a new Parquet reader.

ACKNOWLEDGMENTS

Three anonymous reviewers provided suggestions that helped improve and clarify this manuscript.

This work was carried out on the Dutch national e-infrastructure with the support of SURF Cooperative.

SUPPLEMENTARY MATERIAL

Software is available on Github online, under General Public License (GPL3): https://github.com/BigDataWUR/tomatula
Experimental results are available on Zenodo online, under CC-BY 2.0 Attribution 2.0 license: doi:10.5281/zenodo.582145

REFERENCES

[1] Saulo Aflitos et al. 2014. Exploring genetic variation in the tomato (Solanum section Lycopersicon) clade by whole-genome sequencing. *Plant Journal* 80, 1 (2014), 136–148. https://doi.org/10.1111/tpj.12616

[2] Abdalla Ahmed. 2016. Analysis of Metagenomics Next Generation Sequence Data for Fungal ITS Barcoding: Do You Need Advance Bioinformatics Experience? *Frontiers in Microbiology* 7, July (2016), 1061. https://doi.org/10.3389/fmicb.2016.01061

[3] Apache Software Foundation. 2011-2017. Apache Hadoop. http://hadoop.apache.org/. (2011-2017). [Online; last accessed 23-Feb-2017].

[4] Apache Software Foundation. 2017. Apache Parquet. (Jan. 2017). http://parquet.apache.org/ [Online; last accessed 23-Feb-2017].

[5] Apache Software Foundation. 2017. Apache Spark. (Jan. 2017). http://spark.apache.org/ [Online; last accessed 23-Feb-2017].

[6] Adam Auton et al. 2015. A global reference for human genetic variation. *Nature* 526 (2015), 68–74. https://doi.org/10.1038/nature15393 arXiv:15334406

[7] Aikaterini Boufea and Ioannis N Athanasiadis. 2017. Experimental results of "Managing variant calling datasets the big data way". (May 2017). https://doi.org/10.5281/zenodo.582145

[8] Petr Danecek, Adam Auton, Goncalo Abecasis, Cornelis A. Albers, Eric Banks, Mark A. DePristo, Robert E. Handsaker, Gerton Lunter, Gabor T. Marth, Stephen T. Sherry, Gilean McVean, and Richard Durbin. 2011. The variant call format and VCFtools. *Bioinformatics* 27, 15 (2011), 2156–2158. https://doi.org/10.1093/bioinformatics/btr330

[9] Dries Decap, Joke Reumers, Charlotte Herzeel, Pascal Costanza, and Jan Fostier. 2015. Halvade: Scalable sequence analysis with MapReduce. *Bioinformatics* 31, 15 (2015), 2482–2488. https://doi.org/10.1093/bioinformatics/btv179

[10] Umberto Ferraro Petrillo, Gianluca Roscigno, Giuseppe Cattaneo, and Raffaele Giancarlo. 2017. FASTdoop: A Versatile and Efficient Library for the Input of FASTA and FASTQ Files for MapReduce Hadoop Bioinformatics Applications. *Bioinformatics* 33, 10 (2017), 1575–1577. https://doi.org/10.1093/bioinformatics/btx010

[11] Steven N Hart, Patrick Duffy, Daniel J Quest, Asif Hossain, Mike A Meiners, and Jean Pierre Kocher. 2016. VCF-Miner: GUI-based application for mining variants and annotations stored in VCF files. *Briefings in Bioinformatics* 17, 2 (2016), 346–351. https://doi.org/10.1093/bib/bbv051

[12] Ben Langmead, Kasper D Hansen, and Jeffrey T Leek. 2010. Cloud-scale RNA-sequencing differential expression analysis with Myrna. *Genome biology* 11, 8 (2010), R83. https://doi.org/10.1186/gb-2010-11-8-r83

[13] Ben Langmead, Michael C Schatz, Jimmy Lin, Mihai Pop, and Steven L Salzberg. 2009. Searching for SNPs with cloud computing. *Genome biology* 10, 11 (2009), R134. https://doi.org/10.1186/gb-2009-10-11-r134

[14] Jeremy Leipzig. 2016. A review of bioinformatic pipeline frameworks. *Briefings in Bioinformatics* 18, 3 (2016), 530–536. https://doi.org/10.1093/bib/bbw020

[15] Timo Lubitz, Jens Hahn, Frank T. Bergmann, Elad Noor, Edda Klipp, and Wolfram Liebermeister. 2016. SBtab: A flexible table format for data exchange in Systems Biology. *Bioinformatics* 32, April (2016), btw179–. https://doi.org/10.1093/bioinformatics/btw179

[16] Marco Masseroli, Pietro Pinoli, Francesco Venco, Abdulrahman Kaitoua, Vahid Jalili, Fernando Palluzzi, Heiko Muller, and Stefano Ceri. 2015. GenoMetric Query Language: A novel approach to large-scale genomic data management. *Bioinformatics* 31, 12 (2015), 1881–1888. https://doi.org/10.1093/bioinformatics/btv048

[17] Matt Massie, Frank Nothaft, Christopher Hartl, Christos Kozanitis, André Schumacher, Anthony D Joseph, David A Patterson, Frank Austin Nothaft, and David Patterson. 2013. *ADAM: Genomics Formats and Processing Patterns for Cloud Scale Computing*. Tech. Rep. UCB/EECS-2013-207. EECS Department, University of California, Berkeley, CA, USA. http://www.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-207.html

[18] Henrik Nordberg, Karan Bhatia, Kai Wang, and Zhong Wang. 2013. BioPig: A Hadoop-based analytic toolkit for large-scale sequence data. *Bioinformatics* 29, 23 (2013), 3014–3019. https://doi.org/10.1093/bioinformatics/btt528

[19] Aidan R. O'Brien, Neil F. W. Saunders, Yi Guo, Fabian A. Buske, Rodney J. Scott, and Denis C. Bauer. 2015. VariantSpark: population scale clustering of genotype information. *BMC Genomics* 16, 1 (2015), 1052. https://doi.org/10.1186/s12864-015-2269-7

[20] Michael C. Schatz. 2009. CloudBurst: Highly sensitive read mapping with MapReduce. *Bioinformatics* 25, 11 (2009), 1363–1369. https://doi.org/10.1093/bioinformatics/btp236

[21] André Schumacher, Luca Pireddu, Matti Niemenmaa, Aleksi Kallio, Eija Korpelainen, Gianluigi Zanetti, and Keijo Heljanko. 2014. SeqPig: Simple and scalable scripting for large sequencing data sets in Hadoop. *Bioinformatics* 30, 1 (2014), 119–120. https://doi.org/10.1093/bioinformatics/btt601 arXiv:arXiv:1307.2331

[22] SURF - Collaborative organization for ICT in Dutch education and research. 2016. SURFsara. https://www.surf.nl/en/about-surf/subsidiaries/surfsara/. (2016). [Online; last accessed 23-Feb-2017].

[23] Marek S. Wiewiorka, Antonio Messina, Alicja Pacholewska, Sergio Maffioletti, Piotr Gawrysiak, and Michal J. Okoniewski. 2014. SparkSeq: Fast, scalable and cloud-ready tool for the interactive genomic data analysis with nucleotide precision. *Bioinformatics* 30, 18 (2014), 2652–2653. https://doi.org/10.1093/bioinformatics/btu343

[24] Matei Zaharia, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, Ion Stoica, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, and Shivaram Venkataraman. 2016. Apache Spark: a unified engine for big data processing. *Commun. ACM* 59, 11 (Oct. 2016), 56–65. https://doi.org/10.1145/2934664