Analyzing Large Scale Genotype Datasets With GNOCCHI

Frank Austin Nothaft

Abstract

The development of inexpensive DNA sequencing technologies has enabled projects that sequence large cohorts. While many recent projects have tackled the computationally expensive process of turning raw DNA sequence into genomic variants, cohort analyses still rely on traditional single node techniques. To address this problem, we introduce GNOCCHI, a Spark SQL based toolkit for analyzing genomic variants. GNOCCHI extends the ADAM framework for analyzing genomic data with several variation specific patterns, such as matrix and genotype state views. With GNOCCHI, we are able to parallelize common expensive tasks, such as the training of genome-wide assocation models, or large scale population stratification.

1 Introduction

Since 2001, improvements in DNA sequencing technologies have allowed the cost of sequencing a single human genome to drop from \$1 billion to under \$1,000 [13]. This drastic change in the economics of DNA sequencing has enabled both the use of sequencing in clinical medicine and the genomic study of large cohorts. However, this shift brings new problems: the cost of analyzing genomic data is growing more quickly than the cost of collecting genomic data is falling. This trend is occurring because the decline in the cost of sequencing is outpacing Moore's law.

Several recent projects have attempted to cope with the "DNA data deluge" [19] by implementing application programming interfaces (APIs) that enable computational biologists to write scalable algorithms for analyzing DNA sequence data. These projects include HADOOP-BAM [14], a thin layer that allows HADOOP to read/write genomics file formats; BIOPIG [15] and SEQPIG [20], which present enhancements to the PIG scripting language for querying genomic data; and ADAM [9, 16], a SPARK-based API that provides common genomics query primitives. Additionally, several unpublished projects are ongoing, such as the "HELLBENDER" rewrite of the Genome Analysis Toolkit (GATK, [10]), the Genomics England OPENCB project[1], and the Google Genomics project[2], which builds upon BIGTABLE/DREMEL [11] and SPARK.

While these projects have all enabled the use of a scalable analytics system to analyze genomics data—whether it be APACHE HADOOP [2], APACHE SPARK [27, 28], or some other system—these systems are not optimized for genomic variant data. While genomic variants can be processed with many of the same patterns as other genomic data types, genotype data is distinct in that it is most frequently processed using a sparse matrix-like representation [7, 23], or by running a specific per-site aggregation pattern. In GNOCCHI, we build upon the ADAM project, SPARK SQL [3], and recent work that has added support for common linear algebra operations in APACHE SPARK [26] to support matrix views of genotype data. Additionally, we have done preliminary work to optimize the common per-site aggregation pattern.

In this paper, we describe common kernels that are run across genotype data, and motivate the need for efficient tools for processing this data in a cluster environment. With this motivation,

[1]`https://github.com/opencb/hpg-bigdata`
[2]`https://cloud.google.com/genomics/`

we build a modified genotype representation that is inspired by the ADAM [16] schemas and that can map easily to matrix structures. With this, we demonstrate how multiple common genotype analysis algorithms can be mapped to this datastructure, and explore the parallel performance of these algorithms. Although this work is a first step towards supporting matrix-structured genomics algorithms on top of Spark, our work illustrates several future steps that will allow for great improvements in genotype analysis performance.

2 Background

Although a single run of a DNA sequencer can produce more than 200 GB of raw data from an individual whole genome sequencing (WGS) run, these raw sequences are not typically directly actionable. Typically, raw DNA sequence data is run through a variant calling pipeline. Variant calling pipelines statistically infer the sequence edits between the sample being processed and the reference genome for their species. The reference genome represents the "average" genome sequence for a species, as determined from performing a *de novo* assembly from a pool of individuals. This process estimates both variants and genotypes: variants are the raw sequence edits, while the genotype is the estimate of how many copies of each variant the sample has. Once the sequence data has been transformed into genotype data, it consumes several hundred megabytes (MB) of space for a single human genome when stored in a sparse representation, or tens of gigabytes (GB) when stored in a dense representation [5].

In the case of a single sample, the sparse representation would store genotype data only at sites where a variant was identified in the sample, while the dense representation stores data at every site that was sequenced. This dense representation is used for joint variant calling, which is a process where observations from multiple samples are pooled in order to improve the accuracy of the genotype estimation process [8]. Once samples from a given cohort have been merged, the genotypes are typically stored in a semi-sparse representation where genotypes are stored for all variant where any sample in the cohort has been observed to have the variant. Statistical tests are run upon this representation. This representation yields an n by m matrix where n is the number of variants and m is the number of samples in the cohort. We refer to this representation as semi-sparse as the rows of the matrix are dense, but the matrix is a sparse representation relative to the n by g matrix that would describe the whole genome (g is the genome size; for a 1,000 sample cohort of humans, $10m \sim g, g = 3.2$ Giga basepair, Gbp). This data can be stored in several different ways: the VCF file format stores rows of the matrix [5], while the ADAM [16] and GA4GH[3] schemas store cells of the matrix.

Gnocchi builds these matrix representations on top of the ADAM project's genotype representation. Specifically, Gnocchi adds the simplified `GenotypeState` representation shown in Figure 1. This representation precomputes the genotype state from the allelic representation that the ADAM schema uses, and drops many of the variant calling annotation fields. This representation is a more compact representation of the genotype at the site and is sufficient for most statistical analyses. The additional fields included in the ADAM schema are typically only used for variant quality control analyses, joint variant calling, and variant filtration. All of these steps are conducted prior to any large scale statistical tests that are run on the genotypes.

There are a variety of statistical algorithms that are run on top of genotype data. Most of these algorithms are either clustering or regression algorithms. Clustering algorithms are commonly used to discover population structure in a large genotyped cohort. For example, the genotype data from the 1,000 Genomes project [1] comes from 1,092 individuals who represent 26 different

[3] `https://github.com/ga4gh/schemas`, variants of this schema are used by both the Google Genomics and Genomics England projects.

```
case class GenotypeState(
  contig: String,
  start: Long,
  end: Long,
  ref: String,
  alt: String,
  sampleId: String,
  genotypeState: Int) {
}
```

```
record Genotype {
  // core fields
  Variant variant;
  string sampleId;
  boolean splitFromMultiAllelic;
  array<GenotypeAllele> alleles;

  // quality annotations
  VariantCallingAnnotations
    variantCallingAnnotations;

  // estimated likelihoods
  array<float> genotypeLikelihoods;

  // phasing information
  boolean isPhased;
  int phaseSetId;
  int phaseQuality;
}
```

Figure 1: Gnocchi's `GenotypeState` object differs from the ADAM `Genotype` object by precomputing the genotype state and by not including various variant/genotype annotations. The ADAM `Genotype` representation included here is abbreviated for clarity.

subpopulations. By running principal component analysis (PCA) on the genotype data, we can reconstruct the population labels[4]. While this example is not necessarily useful since the population labels were already known at the start of the study, these sort of clustering/stratification algorithms are useful when studying populations where the relationships between individuals is not known *a priori*. This is the case in most large scale clinical studies.

Regression algorithms are applied to genotype datasets to identify associations between genomic variants and traits of interest. These tests are commonly known as genome wide association studies (GWAS), and they involve the training of many regression models in parallel. For a study with n genotyped sites and p phenotypes[5], a GWAS will typically train np orthogonal regression models. Traditionally, these studies have been run using single machine tools such as PLINK [17]. While these tools are widely used in contemporary genomics, they are quite slow[6]. Recent work [4] has made improvements to the single thread runtime of these tools, but data ingest remains a challenge as moderately sized genotype datasets can easily contain more than one terabyte (TB) of data. For example, the 1,000 Genomes project genotype data is 1.6 TB uncompressed. When stored in more efficient formats like compressed VCF [5] or ADAM/Parquet, it is $\mathcal{O}$(100GB). Since contemporary studies will sequence orders of magnitude more individuals—the Genomics England project will sequence 100,000 individuals [6] while the Million Veteran Program [24] has already genotyped more than 500,000 military veterans—these single node tools will soon be insufficient.

[4]`http://bdgenomics.org/blog/2015/02/02/scalable-genomes-clustering-with-adam-and-spark/`

[5]A phenotype is a trait being studied. Commonly, the phenotype being studied is whether the sample has a disease, or some physical characteristic (e.g., height, eye color). Another common form of association study is an expression quantitative trait locus (eQTL) test. This test looks for links between a single variant (a locus) and the number of copies of a gene that are created (expression).

[6]Recent benchmarking at `https://github.com/jpdna/eQTL-analysis-using-ADAM-and-Spark` shows that running an eQTL on a 77,000 variant subset of chromosome 22 takes over 320 minutes on a single machine.

3 Implementation

Gnocchi is built upon Apache Spark and the ADAM genotype schemas [16]. Gnocchi uses a mix of the RDD [27] and DataFrame/Dataset [3] APIs. In Gnocchi, we have implemented the following algorithms:

- ETL/Import:
    1. Squaring of genotype matrix
    2. Construction of sparse genotype state matrix from genotype data
- Clustering:
    1. Wide/flat PCA
    2. Sample-sample similarity calculation
- Regression:
    1. Case-control GWAS

Both clustering algorithms are implemented on top of the sparse genotype state matrix, while the regression algorithms are currently implemented on top of a squared genotype matrix. This matrix is the semi-sparse/dense row representation described in §2.

3.1 ETL/Import Algorithms

Our ETL/import algorithms focused on merging genotypes from cohorts into the matrix structures that are commonly used for analyzing genotype data. The first algorithm is used to build the semi-sparse/dense row matrix that was described in §2, and that is used for regression (see §3.3). To build this matrix, we first collect a unique list of the sample identifiers from an RDD of ADAM `Genotypes`. We then group all genotypes together by variant. This gives us an RDD element per variant that has called genotypes. We then identify which samples are missing genotypes for this variant. Currently, we just fill these genotypes with a reference call at default ploidy for the sample. In §5, we describe our future plans for implementing a more accurate imputation kernel. We simply used this kernel because the NCI-60 dataset needed a merge function, and since the NCI-60 variant calls were generated from an exome panel, it was reasonable to assume that all genotyped sites were covered for all samples. This is not necessarily accurate, but is a reasonable simplifying assumption.

The sparse matrix used for the remaining calculations was generated using the DataFrame [3] API. Since this matrix was intended to be sparse, we pushed down a predicate to filter out all genotypes that were a reference call before grouping all genotypes together by variant. We used the sample identifiers to map samples to integer indices that we used as the row indices for the sparse matrix. Once genotypes were grouped by position, we sorted the genotypes by the sample indices and created a distributed row matrix [26].

3.2 Clustering

Typically, clustering algorithms that are run on top of genotype datasets on scalable analytics frameworks do not use all of the genotypes for each sample. This happens because typically the clustering algorithms that are supported by systems like Spark are designed to cluster the rows of tall and skinny datasets. For most genomics applications, we are interested in clustering along the rows (the individual samples), but our matrices are wide and flat. The matrix we described in §3.1 is a transpose of this matrix: our columns are our samples, and our rows are our sites. With this representation, our matrix is now tall and skinny, which allows us to take advantage of scalable analytics systems. For the sample similarity algorithm, we are able to natively use the DIMSUM algorithm which is implemented in SPARK [12, 25]. This algorithm efficiently computes the similarity between two columns of a matrix, and is a natural fit for our problem.

For PCA, we cannot use the SPARK implementation, as it would calculate the principal components of our transposed matrix. However, we are able to use MLLIB's singular vector decomposition (SVD) algorithm to compute PCA. For a given matrix A, SVD decomposes the matrix into three matrices:

$$A = U\Sigma V^{\mathrm{T}} \tag{1}$$

Due to the unique structure of the SVD (U and V are unitary and Σ is diagonal), if A is transposed, yields:

$$A^{\mathrm{T}} = V\Sigma U^{\mathrm{T}} \tag{2}$$

Additionally, once we have the SVD, we can compute the principal components of the original matrix via:

$$T_A = U\Sigma \tag{3}$$

With this, we use the transposed genotype matrix described in §3.1, and calculate the SVD. We then multiply the V and Σ matrices together to recover the PCA of the wide and flat genotype matrix.

3.3 Regression

While there are a wide assortment of genotype-phenotype association algorithms, in GNOCCHI, we have currently implemented a χ^2 case-control regression. In this, we take the cross join of all phenotypes and all genotypes and apply an aggregation function on all of the genotypes to calculate the number of observed `(genotype state, phenotype observation)` pairs. Since we are using a χ^2 test, we can directly apply the test for significance to the aggregated value. As discussed in §2, we are working on more regression kernels, and are evaluating different join materialization strategies.

4 Performance

We evaluated GNOCCHI on data from the NCI-60 (National Cancer Institute) collection of cancer cell lines [21, 18] and on data from the 1,000 Genomes project [1]. The NCI-60 dataset was chosen because it is deeply phenotyped, while the 1,000 Genomes project dataset was used because it is the largest open access human genotype dataset. Our evaluation was run on a cluster of 60 machines where each machine was powered by a 16 core INTEL E5-2670 running at 2.6 Gigahertz (GHz), with 256 GB of memory, and four 3 TB hard disk drives (HDD). The cluster was running version 5.4.5 of the CLOUDERA HADOOP DISTRIBUTION, which corresponds to APACHE HADOOP version 2.6.0. We ran a pre-release build of APACHE SPARK 1.6.0, which was built from commit

`3f4efb5c23b029496b112760fa062ff070c20334`. Data was stored in the Hadoop Distributed File System [22], which was configured with a replication factor of two. The nodes in the cluster were connected by a ten gigabit ethernet switch.

First, we evaluated on the NCI-60 dataset. This dataset is composed of exome variant calls for 60 different cancer cell lines. Thus, the data is very sparse (the exome is $\sim 1\%$ of the size of the genome) and fairly small. In total, it is only 51 MB once compressed. However, the size of the dataset increases to 164 MB once the variant matrix is squared off. We need to square off the variant matrix for the NCI-60 dataset because the cell lines were genotyped individually and not joint called. Once we have squared off the variant matrix, we can run the GWAS regression kernels. The other analyses do not require the variants to be semi-sparse/row dense.

The results from the NCI-60 dataset are shown in Figure 2. We ran a scale-out test on our cluster where we scaled the amount of nodes used to run each kernel from eight nodes to 960 nodes. At each step, we doubled the number of nodes used, except for the step where we went from 512 nodes to 960 nodes. We were not able to run on 1,024 nodes because several nodes in our cluster were decommissioned for service. Each node represents a single YARN container executor; we allocated one processor core and 11 GB of memory to each executor, along with 3 GB of memory overhead for Spark off-heap allocation.

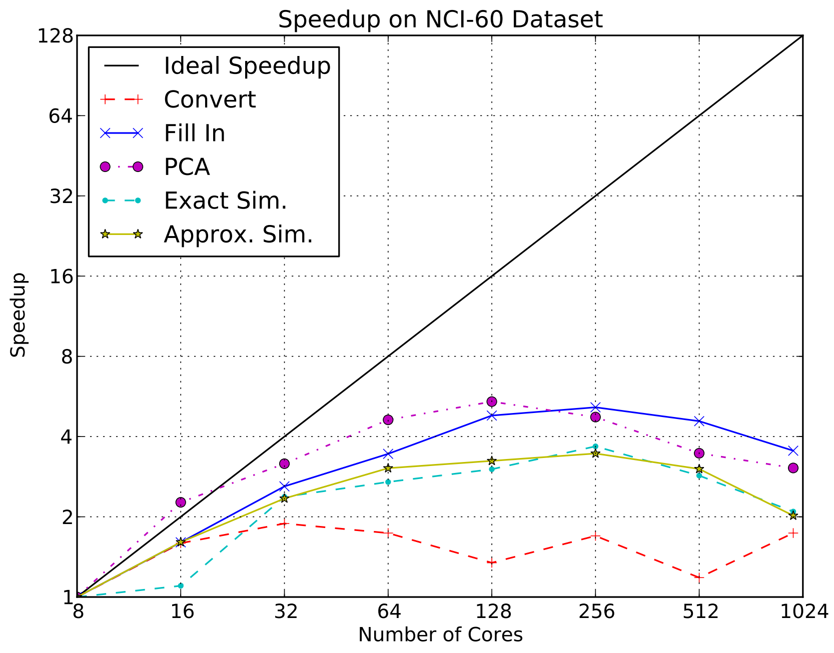

Figure 2: Speedup as the number of cores was increased when processing the NCI-60 dataset.

On the NCI-60 dataset, we ran all of the kernels we implemented, including conversion from the legacy VCF file format to the ADAM/Parquet format. We included the conversion stage because we needed to manually repartition the input dataset so that we could run our scalability experiment. We did this because the typical number of input splits generated when importing the VCF data using Hadoop-BAM [14] was 61 partitions, which would limit our parallelism to 61 executors. This import stage did not parallelize well, as the conversion from VCF to the ADAM schema would run on 61 executors, followed by a stage where the records would be redistributed to more executors via a shuffle. For our experiment, we repartitioned our data so that we had two partitions per executor for load balance. Beyond the conversion step whose performance was approximately constant for all cluster sizes, most of our algorithms would scale through 128–256 nodes. Specifically, the performance of PCA peaked at 128 nodes, while all other algorithms peaked at 256 nodes. Exact runtimes are listed in Table 1. These runtimes excluded YARN job submission and scheduling delay.

Table 1: Performance Summary for NCI-60

Tool	**Single Runtime (sec)**	**Fastest Runtime (sec)**	**Speedup**	**Cores**
Fill In	154.0	29.9	5.15×	256
PCA	117.5	21.7	5.45×	128
Exact Sim.	90.8	24.8	3.66×	256
Approx. Sim.	89.5	25.9	3.45×	256

We ran an additional speedup experiment using the 1,000 Genomes [1] dataset. The 1,000 Genomes genotype dataset is 49 GB when converted to ADAM/Parquet, and is 1.1 TB when stored as uncompressed VCF. We were only able to run the approximate sample similarity kernel on this dataset. We were only able to run this because there is a regression in Spark 1.6 that caused buffer underflows when processing this data with the DataFrame API. Because of this, we fell back on code that we had written originally against the RDD API. Because of this, we ran these experiments with the Spark 1.3.0 stable release that ships with CDH 5.4.5.

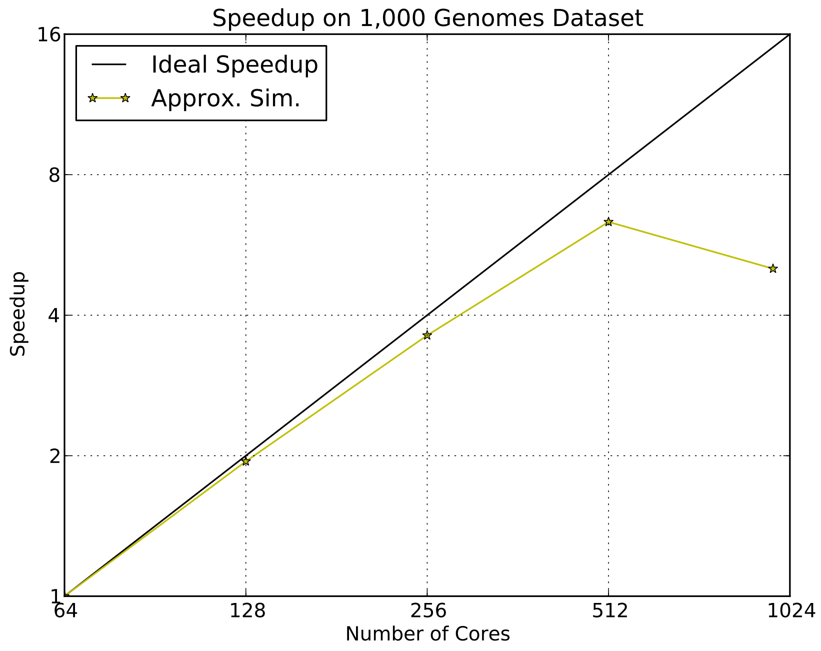

Figure 3: Speedup as the number of cores was increased when processing the 1,000 Genomes dataset.

In this experiment, we forced the input dataset to be repartitioned into 4,096 partitions. This represents four partitions per core when running on a 1,024 executor cluster. Performance peaked at 512 nodes. We did not see a speedup when moving between 512 and 960 nodes for two reasons. First, running with all 960 executors repeatedly caused task failures that did not reproduce on smaller clusters. Although we repeated the experiment multiple times on 960 cores, the errors persisted across all runs. We believe that this may either be due to a hardware or configuration problem in the cluster. Additionally, since the cluster is running as 960 cores instead of the full 1,024 core configuration, there was not an even distribution of tasks across executors. Since SPARK is built around a bulk synchronous parallel [27] scheduling abstraction, we must block the cluster until all tasks in a single stage complete. This is an artifact of our current cluster configuration, as several nodes have been temporarily decommissioned for service.

5 Future Work

This paper presents preliminary work on the GNOCCHI project. We are currently working on an enhanced imputation kernel that makes use of sample similarities to infer genotypes at a site. Currently, our kernel is general, but we plan to use a PCA based approach. Additionally, we are working on more clustering algorithms (specifically, a wide-and-flat k-means implementation), and the implementation of linear and logistic regression for genotype-phenotype association. These two regression kernels will add support for covariates.

One of the more interesting optimizations that we are looking at targets the cross join that is executed as part of a genotype-phenotype regression (see §3.3). In our experiments, we tested on a moderately deeply phenotyped population ($\mathcal{O}(10)$ phenotypes) across a small set of genotypes. As such, the duplication of data created by this cross join was not a big problem. However, for larger analyses—such as large scale eQTL tests with $\mathcal{O}(10,000)$ phenotypes—this cross join is unrealistic. As future work, we plan to evaluate strategies that push the cross join into the aggregation kernel. While this requires repeatedly computing the join, we believe that this join is fairly inexpensive.

6 Conclusion

In this paper, we have demonstrated the GNOCCHI framework for parallel genotype analysis. GNOCCHI is implemented on top of APACHE SPARK's RDD [27, 28] and DATAFRAME/DATASET [3] APIs, as well as the ADAM [9, 16] schemas. With GNOCCHI, we have demonstrated how the analysis of genotype datasets can be scaled to hundreds of cores. GNOCCHI is open source software released under an APACHE 2 license,[7] and can be downloaded from `https://github.com/fnothaft/gnocchi`.

References

[1] 1000 GENOMES PROJECT CONSORTIUM, ET AL. An integrated map of genetic variation from 1,092 human genomes. *Nature 491*, 7422 (2012), 56–65.

[2] APACHE. Hadoop. `http://hadoop.apache.org`.

[3] ARMBRUST, M., XIN, R. S., LIAN, C., HUAI, Y., LIU, D., BRADLEY, J. K., MENG, X., KAFTAN, T., FRANKLIN, M. J., GHODSI, A., AND ZAHARIA, M. Spark SQL: Relational data processing in Spark. In *Proceedings of the SIGMOD International Conference on Management of Data (SIGMOD '15)* (2015).

[4] CHANG, C. C., CHOW, C. C., TELLIER, L. C., VATTIKUTI, S., PURCELL, S. M., AND LEE, J. J. Second-generation PLINK: Rising to the challenge of larger and richer datasets. *GigaScience 4* (2015).

[7] `http://www.apache.org/licenses/LICENSE-2.0`

[5] Danecek, P., Auton, A., Abecasis, G., Albers, C. A., Banks, E., DePristo, M. A., Handsaker, R. E., Lunter, G., Marth, G. T., Sherry, S. T., et al. The variant call format and VCFtools. *Bioinformatics 27*, 15 (2011), 2156–2158.

[6] Genomics England. 100,000 Genomes Project. `https://www.genomicsengland.co.uk/`.

[7] Layer, R. M., Kindlon, N., Karczewski, K. J., Quinlan, A. R., et al. Efficient compression and analysis of large genetic variation datasets. *Nature Methods* (2015).

[8] Li, H. A statistical framework for SNP calling, mutation discovery, association mapping and population genetical parameter estimation from sequencing data. *Bioinformatics 27*, 21 (2011), 2987–2993.

[9] Massie, M., Nothaft, F., Hartl, C., Kozanitis, C., Schumacher, A., Joseph, A. D., and Patterson, D. A. ADAM: Genomics formats and processing patterns for cloud scale computing. Tech. rep., UCB/EECS-2013-207, EECS Department, University of California, Berkeley, 2013.

[10] McKenna, A., Hanna, M., Banks, E., Sivachenko, A., Cibulskis, K., Kernytsky, A., Garimella, K., Altshuler, D., Gabriel, S., Daly, M., et al. The Genome Analysis Toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Research 20*, 9 (2010), 1297–1303.

[11] Melnik, S., Gubarev, A., Long, J. J., Romer, G., Shivakumar, S., Tolton, M., and Vassilakis, T. Dremel: Interactive analysis of web-scale datasets. *Proceedings of the VLDB Endowment 3*, 1-2 (2010), 330–339.

[12] Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S., et al. MLlib: Machine learning in Apache Spark. *arXiv preprint arXiv:1505.06807* (2015).

[13] NHGRI. DNA sequencing costs. `http://www.genome.gov/sequencingcosts/`.

[14] Niemenmaa, M., Kallio, A., Schumacher, A., Klemelä, P., Korpelainen, E., and Heljanko, K. Hadoop-BAM: Directly manipulating next generation sequencing data in the cloud. *Bioinformatics 28*, 6 (2012), 876–877.

[15] Nordberg, H., Bhatia, K., Wang, K., and Wang, Z. BioPig: A Hadoop-based analytic toolkit for large-scale sequence data. *Bioinformatics* (2013), btt528.

[16] Nothaft, F. A., Massie, M., Danford, T., Zhang, Z., Laserson, U., Yeksigian, C., Kottalam, J., Ahuja, A., Hammerbacher, J., Linderman, M., Franklin, M., Joseph, A. D., and Patterson, D. A. Rethinking data-intensive science using scalable analytics systems. In *Proceedings of the International Conference on Management of Data (SIGMOD '15)* (2015), ACM.

[17] Purcell, S., Neale, B., Todd-Brown, K., Thomas, L., Ferreira, M. A., Bender, D., Maller, J., Sklar, P., De Bakker, P. I., Daly, M. J., et al. PLINK: A tool set for whole-genome association and population-based linkage analyses. *The American Journal of Human Genetics 81*, 3 (2007), 559–575.

[18] Reinhold, W. C., Sunshine, M., Liu, H., Varma, S., Kohn, K. W., Morris, J., Doroshow, J., and Pommier, Y. CellMiner: A web-based suite of genomic and pharmacologic tools to explore transcript and drug patterns in the NCI-60 cell line set. *Cancer research 72*, 14 (2012), 3499–3511.

[19] Schatz, M. C., and Langmead, B. The DNA data deluge. *Spectrum, IEEE 50*, 7 (2013), 28–33.

[20] Schumacher, A., Pireddu, L., Niemenmaa, M., Kallio, A., Korpelainen, E., Zanetti, G., and Heljanko, K. SeqPig: Simple and scalable scripting for large sequencing data sets in Hadoop. *Bioinformatics 30*, 1 (2014), 119–120.

[21] Shankavaram, U. T., Varma, S., Kane, D., Sunshine, M., Chary, K. K., Reinhold, W. C., Pommier, Y., and Weinstein, J. N. CellMiner: A relational database and query tool for the NCI-60 cancer cell lines. *BMC genomics 10*, 1 (2009), 277.

[22] Shvachko, K., Kuang, H., Radia, S., and Chansler, R. The Hadoop distributed file system. In *Proceedings of the Symposium on Mass Storage Systems and Technologies (MSST '10)* (2010), IEEE.

[23] Taft, R., Vartak, M., Satish, N. R., Sundaram, N., Madden, S., and Stonebraker, M. Genbase: A complex analytics genomics benchmark. In *Proceedings of the SIGMOD International Conference on Management of Data (SIGMOD '14)* (2014), ACM, pp. 177–188.

[24] U.S. Department of Veterans Affairs. Million Veteran Program (MVP). `http://www.research.va.gov/mvp`.

[25] Zadeh, R. B., and Goel, A. Dimension independent similarity computation. *The Journal of Machine Learning Research 14*, 1 (2013), 1605–1626.

[26] Zadeh, R. B., Meng, X., Yavuz, B., Staple, A., Pu, L., Venkataraman, S., Sparks, E., Ulanov, A., and Zaharia, M. linalg: Matrix computations in Apache Spark. *arXiv preprint arXiv:1509.02256* (2015).

[27] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M., Shenker, S., and Stoica, I. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the Conference on Networked Systems Design and Implementation (NSDI '12)* (2012), USENIX Association, p. 2.

[28] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. Spark: Cluster computing with working sets. In *Proceedings of the Conference on Hot Topics in Cloud Computing (HotCloud '10)* (2010), USENIX Association, p. 10.