SNP Genotype Calling with MapReduce

Simone Leo [†‡]
simone.leo@crs4.it

Luca Pireddu [†‡]
luca.pireddu@crs4.it

Gianluigi Zanetti [†]
gianluigi.zanetti@crs4.it

[†] CRS4, Pula, CA, Italy
[‡] University of Cagliari, CA, Italy

ABSTRACT

Genotype measurement is a key step in genome-wide association studies – those studies that aim to uncover the underlying genetic causes of physical traits, including disease. The leading technology for measuring genotypes is the SNP microarray, where hundreds of thousands of genetic variants are interrogated simultaneously. For some of the most commonly used high-throughput genotyping technologies, the conversion from raw measured data to genotype calls (i.e., identifying the specific genomic variants) requires the concurrent analysis of many samples, with the quality of the results crucially depending on the size of the batch. However, current software for microarray analysis is characterized by poor scalability with respect to input batch sizes. In large-scale studies, this limits the ability to harness the large number of samples available to improve the accuracy of genotype calling. Here, we present a scalable MapReduce application that offers both greater scalability and flexibility than the current state-of-the-art. The software can process datasets as large as 7000 samples in a day, it is more than one order of magnitude faster than previous solutions, and it is currently used in production.

Categories and Subject Descriptors

D.1.3 [**Programming Techniques**]: Concurrent Programming—*Distributed Programming*

General Terms

Algorithms, Performance, Reliability

Keywords

Genotyping, MapReduce

1. INTRODUCTION

Genetic variations strongly influence phenotypic traits like height, longevity and susceptibility to diseases. However, since each individual variation only makes a small contribution to the overall effect, a large number of variants must be studied simultaneously in order to uncover possible risk factors for disease. Genome-wide association studies (GWAS) [11] have been made possible by the availability of high-density genotyping microarrays [16], where hundreds of thousands of variations are measured simultaneously across the whole genome.

At CRS4 we are currently involved in the computational aspects of a large-scale study [25, 29] that collects high-throughput data on thousands of individuals coming from an ethnically homogeneous population – including genotyping with different high-resolution technologies (Affymetrix and Illumina) and next-generation sequencing. Data from Affymetrix 6.0 arrays [23] alone consists of more than 7700 samples, each pertaining to an individual enrolled in the study, for a total data size of about 500 GB.

The sequence of computations needed to estimate genetic variants from the raw data coming from the genotyping technology is called *genotype calling* (GC). In the case of the Affymetrix arrays, GC is based on the concurrent analysis of many samples, with results that depend on the overall statistics, the quality of individual samples, and the number of samples in the batch. The current gold standard implementations of this procedure – `apt-probeset-genotype` from the Affymetrix power tools (APT) [4] and Birdseed [15] – are designed as scalar applications optimized to run on a single workstation. Although `apt-probeset-genotype` allows a certain degree of load distribution – being able to run on a subset of the whole probeset collection at a time – this approach is not efficient, because the initial normalization procedure on which it depends must be re-executed for every subset of probesets. For this reason, the application is typically run on relatively small batches of samples. However, processing data in small batches results in well-known adverse effects [19, 27]. A strategy to counteract these effects is to consider large batch sizes, which, in the case of an ethnically homogeneous population, can be as large as the total number of available samples. Unfortunately, `apt-probeset-genotype` presents serious scalability issues when the number of samples to process reaches the thousands, such that completing the GC analysis would take months even on a very fast computational node. This makes ordinary GC barely feasible, and systematic studies on how the results depend on, for instance, the statistics of the samples in the dataset, practically impossible.

To enable GC analysis on large sample sizes, we designed and implemented a distributed MapReduce [8] implementa-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
MapReduce'12, June 18–19, 2012, Delft, The Netherlands.
Copyright 2012 ACM 978-1-4503-1343-8/12/06 ...$10.00.

tion of the algorithms contained in `apt-probeset-genotype` that easily scales to thousands of samples. The software is scalable both in the number of nodes it can effectively use and in the number of input samples, and is capable of performing GC for about 7000 samples in less than one day on a 30-node Hadoop cluster – a task that we had previously completed in 15 days on 18 machines, at the cost of laboriously manually partitioning the input data and reassembling the results. This new MapReduce application is currently being used in production at CRS4. To the best of our knowledge, ours is the first distributed implementation of the Affymetrix 6.0 analysis pipeline.

The remainder of this article is structured as follows: section 2 provides background information regarding genotyping and genotype calling; section 3 describes the MapReduce workflow that implements the GC procedure; section 4 deals with the evaluation of our solution; section 5 delineates the related work in this area.

2. SNP GENOTYPE CALLING

Single nucleotide polymorphisms (SNPs) are variations at a single position in a DNA sequence that appear in at least 1% of the population. In most cases, a SNP consists of only two variants, or *alleles*, customarily denoted by the letters A and B. Thus, for diploid organisms such as humans, there are three possible genotype configurations at each SNP site: AA, AB and BB. Measuring genotypes is a fundamental step in genome-wide association studies (GWAS) [11], where a large number of SNPs distributed across the whole genome are tested for correlation with qualitative (e.g., a disease) or quantitative (e.g., BMI) phenotypic traits.

Several techniques have been developed for measuring genotypes: here, we concentrate on high-throughput genotyping with SNP microarrays [16] such as the Affymetrix Genome-Wide Human SNP Array 6.0 [23], where hundreds of thousands of SNPs are interrogated simultaneously. These arrays are based on the biochemical binding property of complementary nucleotides (A to T and C to G). The microarray chip is covered with probes, each designed to be complementary to a DNA sub-sequence containing a particular SNP. Probes are included for each variant of the SNP (A or B) and are grouped in redundant collections called *probesets*. When an experiment is run, DNA is fragmented, amplified via polymerase chain reaction (PCR), and labeled with a fluorescent dye; the prepared DNA is then hybridized to the probes in the array; finally, the microarray machine detects the intensity of the fluorescence at each probeset location and transfers these measurements into an Affymetrix CEL file for the sample. The relative intensity of the fluorescence of a SNP's probes contains the information necessary to understand which variants exist in the sample.

The path from raw probe intensities to discrete genotypes consists of a series of computations culminating in the *genotype calling* (GC) step. The gold standard for this calculation is set by Birdseed, developed by the Broad Institute of MIT and Harvard and integrated into the Affymetrix power tools (APT) [4]. Birdseed performs a genotype call by analyzing the SNP's intensities from many samples together. To this data it fits a two-dimensional Gaussian mixture model (GMM) – where the two dimensions represent the summarized intensities (see below) for allele A and B – by expectation-maximization initialized with prior expected means, variances and proportions of genotype classes calcu-

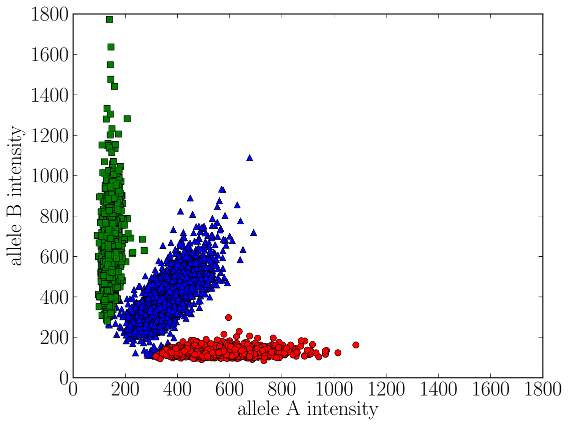

Figure 1: Scatter plot of allele A versus allele B intensities for one of the probesets in a production run. Red circles, blue triangles and green squares represent, respectively, the AA, AB and BB clusters as determined by Birdseed.

lated from HapMap [31] data. The model reveals the clusters of AA, AB and BB variants (figure 1), and each sample's genotype is given by the cluster to which it belongs, while the distance from the cluster centroid gives a measure of confidence associated with the call. A limit may be placed on the acceptable smallest distance from any cluster centroid, beyond which the SNP is labeled as *no call* given the excessive risk of error.

Yet, Birdseed is not a complete tool for GC since it does not perform the preliminary steps required prior to its analysis. On the other hand, the APT version (`apt-probeset-genotype`) implements gender calling, quantile normalization and summarization (i.e., computation of a single per-probeset intensity value for each allele) of probe intensities, allowing it to take its input from CEL files directly. The information on gender is used to decide the relevant GMM parameters that should be used, e.g., to handle SNPs on the X and Y chromosomes differently depending on the gender of the sample. The normalization step is a key factor that influences scalability, since data from all samples must be combined to build the reference distribution that is then assigned to each intensity vector. Let m be the number of probes, n the number of samples, and A the $m \times n$ matrix whose columns represent intensity vectors. So, if $i(p_j, s_k)$ is the intensity for probe j and sample k we have the matrix

$$A = \begin{bmatrix} i(p_1,s_1) & i(p_1,s_2) & i(p_1,s_3) & \dots & i(p_1,s_n) \\ i(p_2,s_1) & i(p_2,s_2) & i(p_2,s_3) & \dots & i(p_2,s_n) \\ i(p_3,s_1) & i(p_3,s_2) & \multicolumn{2}{c}{\dots\dots\dots\dots} & i(p_3,s_n) \\ \dots & & & & \\ i(p_m,s_1) & i(p_m,s_2) & \multicolumn{2}{c}{\dots\dots\dots\dots} & i(p_m,s_n) \end{bmatrix}$$

The quantile normalization procedure acts as follows [6]:

1. each column in A is sorted;
2. in each row, all elements are replaced by the row mean;
3. each column is rearranged to have the same ordering as the original A.

Since the APT version of Birdseed is optimized for a single CPU (the only option for load distribution being partitioning by probeset subset, as discussed above), analyzing thousands of samples at the same time can be prohibitive in terms of both processing time and disk space usage (for APT temporary files). The most straightforward method for reducing the computational burden is to split the input data set into batches and perform GC per batch. However, this technique introduces batch effects that are an important source of errors [19, 27]. To obtain the best possible GC, it is therefore necessary to process all available samples together. However, for large-scale genome-wide studies such as the one mentioned in the introduction, conventional single-core techniques fail to perform this kind of computation efficiently, since data structures (e.g., the normalization matrix) become too large to fit into memory. In this case, the APT implementation swaps data to disk, leading to a severe loss of performance (see section 4). This reason is what motivated us to develop a distributed implementation of the algorithm.

3. MAPREDUCE WORKFLOW

The distributed genotype calling algorithm is implemented as a sequence of Pydoop [20] MapReduce jobs that leverage NumPy [33] for numerical computation. Python is our programming language of choice for most applications, primarily due to its rapid development cycle and ease of maintenance. Pydoop is, in several ways, preferable to other solutions for Python Hadoop development: with respect to Streaming, it provides access to a broader range of MapReduce components (e.g., record reader/writer); being written in CPython, it has none of the shortcomings of Jython; its HDFS API allows for a flexible implementation of components that require interaction with the distributed file system. Each job (figure 2) produces intermediate results that are serialized with protocol buffers [10] (protobuf) and saved to HDFS. The algorithm's overall strategy is to transpose the sample-probeset matrix according to the direction of record flow needed by each job. Each stage is driven by a wrapping script that performs the necessary setup steps (e.g., uploading input and auxiliary files), runs the MapReduce job and post-processes results where necessary.

While most of the code has been written from scratch, we integrated numerical functions and Affymetrix-specific I/O routines from the APT as Boost.Python [2] extension modules. Specifically, the `algo` extension wraps gender and genotype calling routines, while the `io` extension integrates tools for reading CEL files and other Affymetrix file formats (e.g., chip layout files).

The individual workflow steps can be summarized as follows.

Get Probe Arrays. Probe intensities are extracted from CEL files and stored into probe array (PA) objects. This is a map-only stage where a file containing a serialized PA is written for each input CEL file. The MapReduce job takes as input a list of CEL file paths. A custom record reader reads each file in the list, extracts a vector of probe intensities from it using APT's `FusionCELData` API (made available by the `io` extension), serializes it and sends it to the mapper. The latter, in turn, writes the serialized probe intensities to a file in a user-specified directory.

Call Gender. Gender calling is performed on each PA. This is a map-only stage that executes one or both of the two supported gender calling methods on each probe intensity vector; the wrapping script collects all results in a table that will be used by the genotype calling step. One of the gender calling methods is based on computing the intensity ratio between the sum of the signals on groups of probes that are, respectively, on the Y and X chromosome; the other one is based on an expectation-maximization algorithm. This stage takes as input a list of PA file paths. A custom record reader reads each file in the list and passes its contents to the mapper, which deserializes the probe intensity vector and performs gender calling on it using one or both of the above methods, made available by the `algo` extension. Results (the male/female call plus method-specific parameters) are emitted in tabular format and collected into a single file by the wrapping script.

Find Reference Distribution. The mean quantile distribution of probe intensity vectors (see section 2) is computed. The idea is to compute the row-averages of the matrix where each column is a sorted intensity vector. To perform this operation, the mapper, for each PA, emits a value consisting of a protobuf-serialized structure containing the sorted intensity vector and a sample counter set to 1, and the host name as the key. The reducer then calculates the element-wise sum of all the sorted intensity vectors and counters it receives and emits data with the same structure. Since the reduce operation is idempotent, we also use it as a combiner to do most of the work directly on the mapper nodes and reduce the amount of data to shuffle. By using the host name as the reduce key, we effectively partition the work by host. The motivation behind this approach is as follows. A first, intuitive algorithm for this problem might be to have the mappers sort the PA and then emit $(rownumber, value)$ tuples. The combiners and reducers would sum all the elements for each row, and these could then be fetched and assembled into a vector by the wrapping script, which would also divide by the number of samples to compute the means. However, this approach would force us to perform the numeric additions in bare Python, two numbers at a time. Furthermore, it would generate a large number of intermediate values, which in this case are even more time-consuming to process than in Java-based Hadoop programs because of the overhead of the pipes framework. On the other hand, our approach allows the algorithm to perform the additions in bulk, thus taking advantage of the speed of NumPy vector operations and reducing the number of intermediate values to shuffle. Yet, with large matrices the column reduction operation can be quite computationally and data intensive. Therefore, a straightforward implementation using a single reducer that would see all the sorted PAs and calculate the row-averages would introduce an obvious bottleneck. For this reason we opted for a hybrid approach where we use multiple reducers, and rely on the wrapper script to compute the final sum and the averages. The final result is serialized with protobuf and written to HDFS.

Normalize. This is a map-only stage that maps each PA to a normalized probe array (NPA), using the distribution computed in the previous step as reference. As prescribed by the quantile normalization procedure, each value in the original PA is replaced with the corresponding probe value

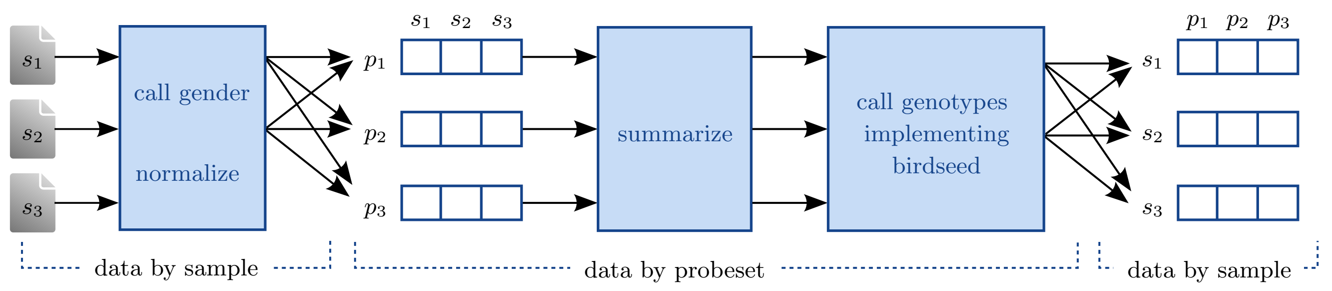

Figure 2: A schematic representation of the MapReduce workflow for GC. All steps up to and including normalization process data by sample, while summarization and GC proper act on a per-probeset basis. The final reduce phase rearranges results by sample.

from the reference distribution. The applications reads PAs from an input list as in the previous stages, and the reference distribution from the corresponding HDFS file, whose path is passed to the mapper as a configuration parameter. For each PA, normalization with respect to the reference distribution is performed with a NumPy-based routine. Finally, NPAs are serialized with protobuf and written to HDFS.

Summarize. This step operates on a collection of NPAs, generating per-probeset intensity values. Essentially, it implements a transposition from a sample $\rightarrow$ collection-of-probesets map to a probeset $\rightarrow$ collection-of-samples map, allowing summarization of normalized probe intensities to per-probeset values. The mapper receives NPAs as input values from a custom record reader, gathers probe intensities by probeset and emits the probeset ID as the key and the corresponding protobuf-serialized intensity array as the value. The reducer summarizes all per-probe intensities corresponding to the same probeset to a per-probeset value using the PLIER method from the APT SDK, available through the `algo` extension. Finally, summarized intensities are serialized with protobuf and written to HDFS.

Call Genotypes. The final step performs actual SNP calling on a collection of summarized probeset intensities. The mapper receives intensity vectors from the record reader, applies APT's Birdseed method – again, through the `algo` extension – to call genotypes, and emits the sample IDs as the keys and the protobuf-serialized call results as the values for each probeset. With the above key choice, the reducer automatically receives calls grouped by sample, which it writes to HDFS – one file per sample – in a specialized protobuf-based binary format.

4. EVALUATION

We evaluate the performance of the MapReduce workflow as compared to that of `apt-probeset-genotype`. In addition, we measure the scalability of the MapReduce implementation with respect to the number of input samples.

All tests have been run on a homogeneous set of machines where each node runs CentOS-5.2 and is equipped with: two Intel Xeon CPUs @ 2.83 GHz with 4 cores each; 16 GB of RAM; two 250 GB SATA hard disks, one of which has been used for HDFS storage (the other one is reserved for the OS); Gigabit Ethernet NIC. Hadoop clusters have been configured with each slave running a task tracker and a data

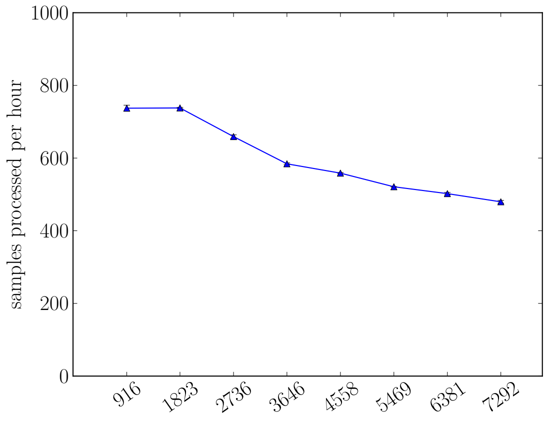

Figure 3: MapReduce throughput on a 30-node cluster for dataset sizes ranging from 916 to 7292 samples. Each measurement has been repeated three times. Deviations from the ideal flat line are mostly due to shuffle and sort and to the single-node data collection performed by wrapper scripts. The line has been drawn only to guide the eye.

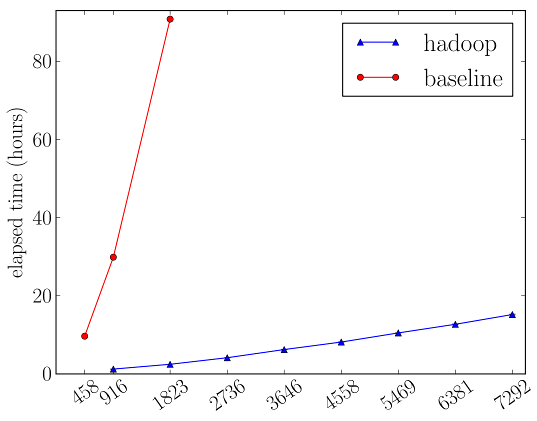

Figure 4: Total running times, in hours, for different dataset sizes (number of input CEL files). Each measurement has been repeated three times. The line has been drawn only to guide the eye.

node, plus two dedicated machines for the task tracker and the name node. We used Hadoop-0.20.2 to run the MapReduce workflow: however, with the latest version of Pydoop, 0.5, the software runs on the latest Hadoop version (1.0.0 at the time of writing). For our performance measurements, we were given access to a dataset of 7292 CEL files from the aforementioned study on autoimmunity diseases [29].

To evaluate the accuracy of our implementation, we used it to re-compute genotypes for 6863 samples from a previous study [32] (see below for details). The average *no call* rate (see section 2) was slightly lower (5.017×10^{-2} vs 5.206×10^{-2}, with the default 0.1 confidence threshold), which resulted in calling about 12 million additional genotypes over all samples. The average allelic discordance rate (i.e., the fraction of single allele mismatches between the two runs) was 3.6×10^{-4}, with a standard deviation of 10^{-4}. Note that the Affymetrix microarray's own error rate [3], measured as discordance with the HapMap genotypes, is 3×10^{-3}.

Figure 3 shows MapReduce throughput for a 30-node cluster, expressed in samples processed per hour, for dataset sizes ranging from 916 samples to the full set of 7292 samples. Deviations from the ideal flat line are mostly due to shuffle and sort in those workflow steps that include a reduce phase (shuffle and sort are reduce sub-phases) and to the single-node data collection performed by wrapper programs.

Figure 4 displays the total running time in hours for different dataset sizes, separately for the MapReduce version on the aforementioned 30-node Hadoop cluster and the APT implementation. Since the latter is single-CPU, running GC with it for increasing numbers of samples quickly becomes a matter of days. As discussed above, a certain degree of parallelization can be achieved by partitioning the probeset space and running a separate `apt-probeset-genotype` instance for each partition. However, since the normalization matrix must be recomputed each time for all samples, this leaves the disk space requirements unmodified and results in poor scalability. In a previous work by Valentini et al. [32], this strategy was employed to perform GC on 6863 samples in 15 days on 18 nodes.

To compare the efficiencies of the various strategies, let us consider the throughput per node. Let N be the number of nodes, S that of samples and t the run time in hours; the throughput per node is then:

$$T_n = \frac{S}{Nt}.$$

The distributed probeset solution achieves a T_n of 1.1. On the other hand, the T_n of our MapReduce implementation ranges from 16.0 (for 7292 samples) to 25.4 (for 916 samples). Again in [32], the running time was further reduced to two days on 7 CPUs ($T_n = 20.4$) by manually partitioning the dataset into 7 overlapping (for quality control) batches of about 1000 individuals. However, this is an ad-hoc solution that requires substantial user intervention: to minimize the batch effects mentioned in section 2, partitions must be carefully balanced with respect to case/control ratio and plate/laboratory representation; calls on the overlapping regions must be quality checked; individual jobs must be manually monitored and rerun in case of failure. Furthermore, even with careful balancing, the average allelic discordance rate with respect to the by-probeset run was 3.4×10^{-3}.

Table 1: scalability for varying cluster size

N	T_n	
	mean	std. error
5	26.9	0.1
10	23.1	0.2
15	23.6	0.5
20	21.8	0.3
25	25.9	0.3
30	24.4	0.1

Finally, to evaluate scalability with respect to the number of nodes, we performed a series of runs on a 1823 samples dataset, with cluster size varying from 5 to 30 nodes, in steps of 5. With three repeated instances of the workflow run for each cluster size, we measured an average T_n of 24.3, with a standard deviation of 1.87. Details are shown in table 1.

5. RELATED WORK

Since Affymetrix started producing SNP arrays, new algorithms for genotype calling have been developed in response to subsequent improvements of the technology [16]. For the 10K (i.e., containing approximately 10000 SNPs) array, Affymetrix adopted a modified version of the partitioning around medoids (PAM) algorithm [21]. Call rates on the 10K array were later improved with the development of SNiPer [13]. With the introduction of the 100K array, the manufacturer abandoned the PAM method in favor of the dynamic model (DM) approach [1], in turn replaced by a modified version of the RLMM algorithm [28] with the advent of the 500K array. For the more recent Genome-Wide SNP 6.0 arrays, such as the ones that produced the data used for our tests, Affymetrix has integrated Birdseed [15] – the current state-of-the-art in SNP GC – in the Affymetrix power tools (APT) [4] software suite, which includes the `apt-probeset-genotype` GC tool.

The MapReduce model has been increasingly adopted in data-intensive bioinformatics since late 2008, with works on the parallelization of BLAST and other popular tools [9,22]. Its adoption was stimulated by the rapid increase in data rates resulting from the advancement in high-throughput sequencing technologies. As such, many of the MapReduce applications in bioinformatics relate to sequencing data analysis problems such as short read mapping [18,26,30], quality control [14] and RNA-sequencing [17]. To the best of our knowledge, no other work has been published on applying MapReduce or other distributed computing approaches to the SNP GC problem.

6. CONCLUSIONS AND FUTURE WORK

We have presented a MapReduce workflow for genotype calling that reproduces the functionalities of the standard `apt-probeset-genotype` tool, with the added scalability of a distributed implementation. In less than one day, the MapReduce version enables the analysis of thousands of data samples *in the same batch*, thus eliminating the error-inducing batch effects mentioned in section 2. Moreover, our implementation is much more suitable for repeated analysis with varying configuration parameters, not only because of the speed advantages, but also because of its greater flexibility: while `apt-probeset-genotype` is a monolithic piece of software that runs the whole process from input CEL files to the final genotype calls, our MapReduce workflow acts in stages that save intermediate results to HDFS. Thus, if a parameter affecting the final GC step is changed, only that step needs to be re-executed.

With the lack of substantial increases in density in SNP arrays since 2007 [16] and the continuously decreasing costs of next generation sequencing (NGS) [24], which could soon make it the first choice for genotyping [7], it might appear that microarrays are becoming irrelevant. On the contrary, microarrays are still widely used, especially in medical research where studies can reach sizes of hundreds of thousands of individuals [12]. Therefore, SNP genotype calling will continue to require scalable and efficient methods for data analysis such as the one presented in this work.

In addition to scalability, the decision to base the workflow on Hadoop provides another benefit in the possibility to run the workflow on a cloud infrastructure as the one offered by Amazon. However, we expect there to be two main problems stopping users from taking advantage of this possibility: first, the privacy issues associated with the remote storage of confidential clinical patient data; second, the cost and time required to transfer the data. For instance, consider a GC run on 5000 samples: since the average CEL file size for 6.0 arrays is about 66 MB and that of output files is 89 MB, with a 10 Mbps connection at 80% network utilization it would take more than 9 days to get the results, for a total cost (data transfer only) of $562 [5].

In the near future, we plan to add more features (e.g., a distributed application for generating allele-allele plots – see figure 1 – for all probesets) and release the full source code as open source.

7. ACKNOWLEDGMENTS

We would like to thank Francesco Cucca and CNR-IRGB for kindly allowing us to perform scalability tests with their genotyping data. We would also like to thank our colleagues Maria Valentini and Ilenia Zara and CNR-IRGB's Serena Sanna for sharing their expertise on genotype calling and GWAS and for their precious feedback at all stages of development.

8. REFERENCES

[1] Dynamic model based algorithms for screening and genotyping over 100 K SNPs on oligonucleotide microarrays. *Bioinformatics*, 21(9):1958–63, 2005.

[2] D. Abrahams and R. W. Grosse-Kunstleve. Building hybrid systems with Boost.Python. *C/C++ Users Journal*, 21(7):29–36, 2003.

[3] Genome-wide human SNP array 6.0 – data sheet. `http://www.affymetrix.com/support/technical/datasheets/genomewide_snp6_datasheet.pdf`.

[4] Affymetrix power tools. `http://www.affymetrix.com/partners_programs/programs/developer/tools/powertools.affx`.

[5] Aws import/export. `http://aws.amazon.com/importexport/`.

[6] B. Bolstad, R. Irizarry, M. Åstrand, and T. Speed. A comparison of normalization methods for high density oligonucleotide array data based on variance and bias. *Bioinformatics*, 19(2):185–193, 2003.

[7] A. Coombs. The sequencing shakeup. *Nature Biotechnology*, 26(10):1109–1112, 2008.

[8] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. In *OSDI '04: 6th Symposium on Operating Systems Design and Implementation*, 2004.

[9] M. Gaggero, S. Leo, S. Manca, et al. Parallelizing bioinformatics applications with MapReduce. *CCA-08: Cloud Computing and its Applications*, 2008.

[10] Google Protobuf: protocol buffers. `http://code.google.com/p/protobuf/`.

[11] J. N. Hirschhorn and M. J. Daly. Genome-wide association studies for common diseases and complex traits. *Nature Reviews Genetics*, 6(2):95–108, 2005.

[12] T. J. Hoffmann, Y. Zhan, M. N. Kvale, et al. Design and coverage of high throughput genotyping arrays optimized for individuals of East Asian, African American, and Latino race/ethnicity using imputation and a novel hybrid SNP selection algorithm. *Genomics*, 98(6):422–430, 2011.

[13] M. J. Huentelman, D. W. Craig, A. D. Shieh, et al. SNiPer: improved SNP genotype calling for Affymetrix 10K GeneChip microarray data. *BMC genomics*, 6(1):149, 2005.

[14] D. R. Kelley, M. C. Schatz, and S. L. Salzberg. Quake: quality-aware detection and correction of sequencing errors. *Genome Biology*, 11(11):116, 2010.

[15] J. M. Korn, F. G. Kuruvilla, S. A. McCarroll, et al. Integrated genotype calling and association analysis of SNPs, common copy number polymorphisms and rare CNVs. *Nature Genetics*, 40(10):1253–1260, 2008.

[16] T. LaFramboise. Single nucleotide polymorphism arrays: a decade of biological, computational and technological advances. *Nucleic Acids Research*, 37(13):4181–4193, 2009.

[17] B. Langmead, K. D. Hansen, and J. T. Leek. Cloud-scale RNA-sequencing differential expression analysis with Myrna. *Genome Biology*, 11(8):83, 2010.

[18] B. Langmead, M. C. Schatz, J. Lin, et al. Searching for SNPs with cloud computing. *Genome Biology*, 10(11):134, 2009.

[19] J. Leek, R. Scharpf, H. Bravo, et al. Tackling the widespread and critical impact of batch effects in high-throughput data. *Nature Reviews Genetics*, 11:733–739, 2010.

[20] S. Leo and G. Zanetti. Pydoop: a Python MapReduce and HDFS API for Hadoop. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 819–825, 2010.

[21] W.-m. Liu, X. Di, G. Yang, et al. Algorithms for large-scale genotyping microarrays. *Bioinformatics*, 19(18):2397–2403, 2003.

[22] A. Matsunaga, M. Tsugawa, and J. Fortes. Cloudblast: combining MapReduce and virtualization on distributed resources for bioinformatics applications. In *Fourth IEEE International Conference on eScience*, pages 222–229, 2008.

[23] S. A. McCarroll, F. G. Kuruvilla, J. M. Korn, et al. Integrated detection and population-genetic analysis

of SNPs and copy number variation. *Nature Genetics*, 40(10):1166–1174, 2008.

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

[25] S. Naitza, E. Porcu, M. Steri, et al. A genome-wide association scan on the levels of markers of inflammation in Sardinians reveals associations that underpin its complex regulation. *PLoS Genetics*, (1):e1002480.

[26] L. Pireddu, S. Leo, and G. Zanetti. SEAL: a distributed short read mapping and duplicate removal tool. *Bioinformatics*, 27(15):2159–2160, 2011.

[27] A. Pluzhnikov, J. E. Below, A. Konkashbaev, et al. Spoiling the whole bunch: quality control aimed at preserving the integrity of high-throughput genotyping. *American Journal of Human Genetics*, 87(1):123–128, 2010.

[28] N. Rabbee and T. P. Speed. A genotype calling algorithm for Affymetrix SNP arrays. *Bioinformatics*, 22(1):7–12, 2006.

[29] S. Sanna, M. Pitzalis, M. Zoledziewska, et al. Variants within the immunoregulatory CBLB gene are associated with multiple sclerosis. *Nature Genetics*, 42(6):495–7, 2010.

[30] M. C. Schatz. CloudBurst: highly sensitive read mapping with MapReduce. *Bioinformatics*, 25(11):1363–1369, 2009.

[31] G. A. Thorisson, A. V. Smith, L. Krishnan, and L. D. Stein. The international HapMap project web site. *Genome Research*, 15(11):1592–1593, 2005.

[32] M. Valentini, I. Zara, and M. Muggiri. Comparison of two strategies for genotype calling. Poster – ESHG 2011, Amsterdam, May 25-31, 2011.

[33] S. van der Walt, S. Colbert, and G. Varoquaux. The NumPy array: a structure for efficient numerical computation. *Computing in Science Engineering*, 13(2):22 –30, 2011.