avocado: A Variant Caller, Distributed

Frank Austin Nothaft, Peter Jin, Brielin Brown
Department of Electrical Engineering and Computer Science
University of California, Berkeley
{fnothaft,phj,brielin}@berkeley.edu

ABSTRACT

In this paper, we present `avocado`, a distributed variant caller built on top of ADAM and Spark. `avocado`'s goal is to provide both high performance and high accuracy in an open source variant calling framework. To achieve this, we implement both local assembly and pileup-based single nucleotide polymorphism (SNP) calling. A key innovation presented in our work involves the development of heuristics for when to choose more expensive assembly-based methods instead of pileup-based methods. Additionally, we introduce the concept of "significant statistics," a tool for performing incremental joint variant calling.

Categories and Subject Descriptors

Applied Computing [**Life and Medical Sciences**]: Computational Biology—*Sequencing and Genotyping Technologies*; Applied Computing [**Genomics**]: Computational Genomics; Computing Methodologies [**Distributed Computing Methodologies**]: Distributed Algorithms—*MapReduce Algorithms*

General Terms

Algorithms, Performance

Keywords

Variant Calling, Genotyping, Genomics Pipeline, Local Assembly, Distributed Computing, MapReduce

1. INTRODUCTION

Modern genomics processing pipelines can be divided into four primary ordered stages:

1. **Sequencing:** Gathering of read data from DNA
2. **Alignment:** Alignment of read data against reference genome
3. **Variant Calling:** Statistical determination of differences against reference genome
4. **Variant Annotation:** Annotation of impact of variation

Currently, to run a genomics pipeline end-to-end for a single high coverage genome[1] consumes approximately 100 hours [17]. Of this 100 hour figure, both alignment and variant calling consume approximately 50 hours each.

Although some applications that use genomic data are latency insensitive (for example, population genomics), many medical applications like genomic medicine, or genomic classification of viral outbreaks [16] are latency sensitive. However, it is unacceptable to sacrifice accuracy in the pursuit of speed. Recent work has focused on the problem of accelerating alignment [19]; in this paper, we address accelerating variant calling.

As noted above, it is unacceptable to sacrifice accuracy for performance. To achieve improved performance, we implement several enhancements:

- Current pipelines are penalized by I/O performance, we address this by using an in-memory MapReduce framework [20] to reduce I/O pressure
- Additionally, we leverage the new ADAM data format [12], a high performance file format for distributed genomics
- Finally, we achieve high accuracy at a low performance cost by using high fidelity assembly-based methods only on complex segments of the genome

In this paper, we discuss this system, related work, and perform a performance analysis. We start with a discussion of the related work in §2. We then describe our architecture and algorithms in §3. Finally, we analyze the performance of our system in §4, and propose future research directions in §5.

[1]High coverage refers to having on average >30× bases aligned to each location in the reference genome.

2. RELATED WORK

There has been significant work related to variant calling, and towards accelerating the genomic processing pipeline. In this section, we discuss other variant callers, and tools that we use in our evaluation.

2.1 ADAM

ADAM [12] is a new data format for genomics meant to replace the Sequence/Binary Alignment/Map (SAM/BAM) formats for read data [11], and the Variant Call Format (VCF) for variant/genotype data [3]. The original SAM/BAM/VCF formats were designed for single-node processing, and do not easily distribute across several machines. Although a library was designed for processing BAM/VCF data in Hadoop [14], this API does not scale well past 8 nodes. ADAM achieves scalability beyond 100 machines by eliminating the central file header, and by using the Parquet data store which is optimized for parallel data access [18].

In the process of developing `avocado`, we contributed 3,500 lines of code (LOC) to the ADAM project. This contribution comprised the variant and genotype format, code for calculating normalized variant data from genotypes, and converters to/from the VCF format. Additionally, this contribution included code for translating between read and reference oriented views of data.

2.2 The Genome Analysis Toolkit

The Genome Analysis Toolkit (GATK) [13, 4] is a variant calling framework released by the Broad Institute of Harvard and MIT.

2.3 Samtools Mpileup

Samtools Mpileup is a tool for Single Nucleotide Polymorphism (SNP) calling and genotyping aligned read data. Given a set of reads from several individuals aligned to reference chromosomal position (a *pileup*), Mpileup determines

- Is there statistically significant evidence that some individuals in the population have a non-reference allele at this position? (SNP calling)
- Given that there is a SNP, which individuals are homozygous for the reference base, which are heterozygous, and which are homozygous for the non-reference base? (genotyping)

Since reads from a single individual will contain sequencing errors that do not represent true genetic variation, samtools leverages the alignment and read quality from several individuals, and calls a site an variant if the probability that all individuals in the sample are homozygous to the reference is small enough [9].

2.4 FreeBayes

[6]

2.5 SNAP

SNAP is a high performance short-read aligner that is optimized for longer read lengths, and for distributed computing [19]. At the current moment, we have not integrated with SNAP, but long term, we plan to incorporate SNAP as the aligner in our read-alignment and variant calling pipeline. This is significant, as variant callers typically optimize to correct for the error models of the aligners that they coexist with.

SNAP leverages the increasing length of reads to build a large alignment index, which is similar to the method used by BLAST [2], and which is dissimilar to the Burrows-Wheeler transform based methods used by BWA [10]. Aligners which use the Burrows-Wheeler transform perform very well in the absence of mismatching data—however, they cannot handle mismatches or inserts well. BWA has bad failure modes for reads with mismatches within the first 20 bases of the read, and Bowtie [8] does not handle insertions when aligning. As SNAP will have better performance when aligning indels, it is likely that we will be able to omit the local realignment stage from ADAM [12]—this is significant as local realignment is the most expensive phase of read processing before variant calling.

2.6 SMaSH

SMaSH is a benchmarking suite for alignment and variant calling pipelines [17], and was a key tool used for the evaluation of `avocado`.

Traditionally, variant calling pipelines have been evaluated on concordance[2], and through using venn diagrams [7]. This is because genomics rarely has access to *ground truth*: typical sequencing methods have insufficient fidelity to detect all variants clearly, and extremely high fidelity sequencing methods are too expensive/slow to be used in clinical practice. However, this method is fraught with risk: concordance is not a good metric to use if variant callers or aligners are making similar systemic errors.

To address this problem, SMaSH leverages synthetic data which by definition has known ground truth, and rigorously verified mouse and human genomes. The human genomes and validation data come from the 1000 Genomes project, a project which surveyed the genomes of 1000 individuals using multiple sequencing technologies [15]. On top of this curated data, SMaSH provides a VCF based interface for determining the precision of a variant calling pipeline. Novel to this benchmarking suite, the authors introduced a "rescue" phase, which is used to resolve ambiguities that are caused by the VCF specification.

It is worth noting that SMaSH does not include any datasets that are designed for joint variant calling. Because of this, we fall back on concordance as a metric for evaluating our joint variant calling algorithms which are described in §3.3.

3. ARCHITECTURE

When architecting `avocado`, we made a conscientious decision to prioritize modularity and extensibility. There are several reasons behind this design choice:

- Current variant calling pipelines are not meant to be extended. Because of this, anyone looking to proto-

[2]Identifying overlap between the call set of multiple variant calling pipelines.

type a new variant calling algorithm must implement their own variant calling infrastructure, which is a significant impediment to variant calling research.

- Although we have limited specialization in our current pipeline (we specialize towards local assembly and pileup based SNP calling), long term we plan to add increasingly specialized variant calling algorithms.
- Similarly, it is known that modern variant callers perform poorly when calling structural variants (SVs). This is a critical area that we hope to attack through specialized variant calling algorithms.

To improve modularity, we pushed as much functionality into the ADAM stack as possible [12]. ADAM implements several important transformations that are used on the read processing frontend, including sort-by-reference position, duplicate marking, base quality score recalibration (BQSR), and local realignment (see §2.5). Additionally, we have moved portions of the variant calling stack into ADAM—after genotyping samples, we use the ADAM pipeline to transform genotypes into variant calls. We include a brief discussion of the merits of this approach in Appendix B.

A diagram of the `avocado` architecture is included in Figure 1.

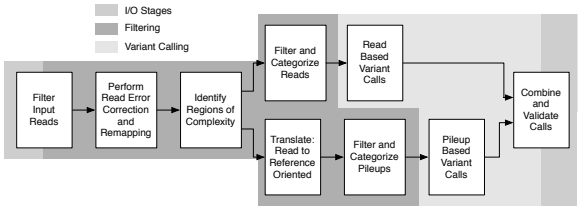

Figure 1: System Architecture

`avocado` is roughly divided into four stages:

1. **Read pre-processing:** Applies transformations from the ADAM pipeline, including sort, duplicate marking, BQSR, and local realignment. Operates strictly on read data.
2. **Filtering:** Selects the best algorithm for calling variants on a segment of data; transforms data to be reference-oriented if necessary. This is discussed in §3.4.
3. **Variant calling:** Calls genotypes for samples in input set on either read-oriented or reference-oriented data. These algorithms are discussed in §3.1 and §3.2.
4. **Variant filtering:** Here, variant data is calculated from genotypes using the ADAM API [12]. Additionally, variants can be filtered for quality.

All stages are designed to be configurable, and easy to replace. Specifically, the filtering and variant calling stages are designed with a specific eye towards modularity. For variant calls, we provide a base class, and two specialized subclasses for variant calls that operate on either read or reference oriented data. A variant calling algorithm can be added by implementing one of these class interfaces, and then registering the call with a filter. For filters, we provide two filtering stages: the first operates on read data, and is used to implement the filter described in §3.4. The second filtering stage operates on pileup (reference oriented) data—currently, we use this to filter out pileups that have no mismatch evidence. For both of these filters, we provide a simple interface for developers to implement.

To improve pipeline performance, we made a significant optimization to the reads-to-pileup transformation step. At the start of transforming reads into reference oriented data, the reads are sorted by reference position. However, due to the partitioning used by Spark [20], after a naïve transformation of reads to single reference oriented bases, the reference oriented bases are no longer sorted. This makes the ensuing grouping of bases by position very expensive due to significant disk shuffle I/O. Instead, we perform a quasi-streaming transform on the read data. Here, the sorted read blocks are chunked together. All of the reads in this window are then converted into pileup bases, and then grouped together into rods. This leads to significantly better performance, but has tricky edge cases: reads that border the end of a group must be duplicated into both read groups.

3.1 Local Assembly

Given a partition of reads, we can group them by their starting locus in intervals of W, creating regions of length $W + L - 1$ where L is the read length. Within each region, we can evaluate the likelihood of observing the reads given the reference haplotype:

$$\begin{aligned}\mathcal{L}(H^{\text{ref}}) &\equiv \mathbf{P}(\{r_i\}|H^{\text{ref}}) \\ &= \prod_i \mathbf{P}(r_i|H^{\text{ref}})\end{aligned} \tag{1}$$

where $\mathbf{P}(r|H)$ is obtained from aligning the read and the candidate haplotype by a pairwise HMM alignment [5]. Note that, in practice, all probabilities are computed in units of logarithm base 10, so products become sums, etc.

If a reference haplotype likelihood is below a fixed threshold, the region corresponding to the haplotype is marked *active*. Each *active region* is assembled independently and in parallel.

The assembly of an active region starts by splitting all reads assigned to the region into k-mers, where k is a fixed parameter for all assemblies. A read generates $L - k + 1$ total k-mers. Each k-mer is uniquely identified by the substring of its originating read. Because of coverage overlap and sequence repeats, some k-mers will be duplicates; these are consolidated, and the duplication factor is recorded as the k-mer multiplicity.

The k-mers define edges in the completed k-mer assembly graph. Within a read, each adjacent pair of k-mers have an overlapping substring of length $k-1$; these are seeded as the initial vertices in the k-mer graph. Because there are duplicated k-mers, some vertices will be "merged," connecting the graph. Unlike an exact de Bruijn graph, which connects all

overlaps between k-mers, we only connect the overlaps found in the reads, performing a simple form of read threading.

Once the k-mer graph is complete, we perform a depth-first traversal with an upper bound on the total path multiplicity, defined as the sum of the edge multiplicities, to enumerate a set of possible paths. The traversal begins at a graph source, and a completed path must also end at a sink. Each path is an assembled haplotype to be evaluated.

From the assembled haplotypes, we order them according to the haplotype likelihood:

$$\mathcal{L}(H_j) = \prod_i \mathbf{P}(r_i|H_j). \tag{2}$$

Among the ordered haplotypes, we pick the top scoring haplotypes and ignore the low scoring ones. The likelihood of observing the reads $\{r_i\}$, given a pair of haplotypes H_j and $H_{j'}$, is defined to be [1]:

$$\begin{aligned}\mathcal{L}(H_j, H_{j'}) &\equiv \mathbf{P}(\{r_i\}|H_j, H_{j'}) \\ &= \prod_i \left[\frac{\mathbf{P}(r_i|H_j)}{2} + \frac{\mathbf{P}(r_i|H_{j'})}{2}\right].\end{aligned} \tag{3}$$

We compute the posterior probability of observing the pair of haplotypes H_j and $H_{j'}$ from the haplotype pair likelihood and a haplotype pair prior probability:

$$\mathbf{P}(H_j, H_{j'}|\{r_i\}) = \frac{1}{Z}\mathcal{L}(H_j, H_{j'})\mathbf{P}(H_j, H_{j'}) \tag{4}$$

where Z is a normalization:

$$Z = \sum_j \sum_{j'} \mathcal{L}(H_j, H_{j'})\mathbf{P}(H_j, H_{j'})$$

and where we obtain the prior $\mathbf{P}(H_j, H_{j'})$ by aligning the haplotype pair with the same pairwise HMM alignment as above, and taking the product of the prior probabilities for each SNP and indel event [1].

We choose the maximum a priori estimate among haplotypes with any variants as the called non-reference maternal and paternal haplotype pair:

$$(H^{\text{nonref}}_{\text{mat}}, H^{\text{nonref}}_{\text{pat}}) = \arg \max_{H_j, H_{j'}: n_{\text{var}}(H_j, H_{j'})>0} \mathbf{P}(H_j, H_{j'}|\{r_i\}). \tag{5}$$

Similarly, we may define the reference haplotype pair as $(H^{\text{ref}}, H^{\text{ref}})$. The error probability of calling the non-reference haplotype pair is:

$$\begin{aligned}&\mathbf{P}_{\text{error}}(H^{\text{nonref}}_{\text{mat}}, H^{\text{nonref}}_{\text{pat}}) \\ &= \frac{\mathbf{P}(H^{\text{ref}}, H^{\text{ref}})}{\mathbf{P}(H^{\text{nonref}}_{\text{mat}}, H^{\text{nonref}}_{\text{pat}}) + \mathbf{P}(H^{\text{ref}}, H^{\text{ref}})}.\end{aligned} \tag{6}$$

The quality score of all variants present in the nonreference haplotype pair is defined as the Phred scaling of $\mathbf{P}_{\text{error}}$.

3.2 Genotype Calling

For one sample at a site, we can estimate the genotype likelihood based on the number of reads that match the reference genome and the quality of each of these reads. Let the number of reads at site a be k and the ploidy be m. Without loss of generality assume the first $l \leq k$ bases match the reference, and the rest do not. Let ϵ_j be the error probability of the jth read base. We have that,

$$\begin{aligned}\mathcal{L}(g) =& \frac{1}{m^k} \prod_{j=1}^{l} [(m-g)\epsilon_j + g(1-\epsilon_j)] \\ &\times \prod_{j=l+1}^{k} [(m-g)(1-\epsilon_j) + g\epsilon_j]\end{aligned} \tag{7}$$

Here we are only leveraging the data of a single sample. Performance can be improved by considering statistics associated with other samples. If a study involves sequencing many individuals from a population, it is beneficial to run them jointly in order to determine population parameters like per-locus allele frequency spectrum (AFS) and minor allele frequency (MAF) (§ 3.3). However, when genotyping a single individual computational time can be saved by looking these parameters up in a database such as dbSNP. If the population MAF at site a is ϕ_a, the likelihood can be compensated by the prior probability of seeing a non-reference allele, and the genotype is

$$\hat{g}_a = \arg\max_g \frac{\mathcal{L}(g)P[g|\phi_a]}{\sum_g \mathcal{L}(g)P[g|\phi_a]} \tag{8}$$

where $P[g|\phi] = \binom{m}{g}\phi^g(1-\phi)^{m-g}$ is the pmf of the binomial distribution.

3.3 Joint Variant Calling

When genotyping several individuals one may wish to genotype them jointly while determining population allele statistics, especially when said individuals are from a specific population of interest. In this case, we can use the EM procedure of Samtools Mpileup. Given the data for several individuals and using the likelihood in (7), we can infer the population MAF per site via iteration of

$$\phi_a^{(t+1)} = \frac{1}{M}\sum_{i=1}^{n} \frac{g\mathcal{L}(g)P[g|\phi_a^{(t)}]}{\sum_g \mathcal{L}(g)P[g|\phi_a^{(t)}]} \tag{9}$$

where n is the number of individuals, $M = \sum_{i=1}^{n} m_i$ is the total number of chromosomes and P is the binomial likelihood described above. This population MAF can then be used in genotyping as above.

3.4 Algorithm Selection

As discussed in §3, we seek to improve the performance of our variant caller without reducing accuracy by directing the variant caller to use higher accuracy methods in areas that show increased complexity. Loosely, we define a complex region as an area that is highly similar to other areas of the genome, or where it is likely that a complex variant (such as an indel) is segregating. To identify those regions, we use the following heuristics:

- Areas that are highly similar to other areas of the genome can be distinguished by low mapping quality. Reduced mapping quality indicates that alignment

found several areas where the read could map with similar quality.

- Complex variants lead to a change in coverage over the effected area. Deletions will see reduced coverage, and insertions lead to increased coverage.

We implemented our filter by stepping a window across the reference genome. The window was stepped by 1000 base pairs. In each 1000 base pair window, we would compute the average coverage, and mapping quality. If a window violated either of the mapping quality or the coverage threshold, we would flag it as high complexity and pass it to the assembler. If a window did not violate either of those two thresholds, we built pileups out of the data on this interval, and then used the genotyping methods described in §3.2.

We provided several tuning parameters to the end user. Specifically, the end user could set the target mapping quality and coverage deviation percentage (percent change from mean coverage). Table 1 summarizes the sensitivity of these parameters.

Table 1: % Reads in High Complexity Region

MapQ,Cov	0.2	0.4	0.6
40	88%	54%	20%
50	90%	56%	22%
60	98%	91%	88%

As can be noted, mapping complexity begins to saturate as the mapping quality threshold increases to 60[3], and as the coverage variation coefficient decreases to 0.2.

4. EVALUATION

4.1 Accuracy

4.2 Performance

5. FUTURE WORK

Since assembly depends on the exact sequences of the input data, the quality of the reads is critical for performing an accurate assembly. One error correction method is spectral filtering, which depends on the observation that the distribution of k-mers from all reads with respect to multiplicity is bimodal, one due to the Poisson sampling of reads (with high multiplicity), the other due to errors (with low multiplicity), so that splitting the reads between the two modes and keeping the reads near the mode with higher multiplicity serves to prune the poor quality reads. Errors in the read data lead to *spurs*, which are spurious short sections of the graph connected to the source or the sink. We do not remove spurs, although it may slightly improve performance. Empirically, we have found during other work that utilizing mate pair information greatly improves the quality of an assembly. We also do not employ mate pair threading, which requires collecting the insert size distribution of the data, but we expect that implementing it has the potential to vastly improve the accuracy of variant calls.

[3]Phred-scaled 60 is equivalent to a probability of $P = 0.999999$ that the read is correctly mapped.

6. CONCLUSION

APPENDIX

A. AVAILABILITY

`avocado` is open source and is licensed under the Apache 2 license. The source code is available at:

`http://www.github.com/bigdatagenomics/avocado`

B. GENOTYPE/VARIANT REFACTORING

Traditionally, variant calling has involved genotyping n samples, and then determining which of the genotypes of these samples contain true variants. This involves the calculation of a *variant quality* score[4], and filtering variants.

Semantically, all of the information necessary for the creation of variant data can be extracted from the genotypes of the samples. Therefore, the step of packing/unpacking genotype data to create variant calls is unnecessary. To reduce the likelihood of making errors, we have migrated code to do this into the ADAM [12] framework. Then, to get variant calls, we must just genotype the samples we see, and call the ADAM library.

This does not preclude joint variant calling—although joint variant calling implies that multi-sample data is being used to influence the filtering of variants, this is a bit of a misnomer. Practically, as discussed in §3.3, joint variant calling involves the use of data from multiple samples to refine genotype probabilities across a population to support genotypes that are frequently seen in this population.

C. REFERENCES

[1] C. A. Albers, G. Lunter, D. G. MacArthur, G. McVean, W. H. Ouwehand, and R. Durbin. Dindel: Accurate indel calls from short-read data. *Genome Research*, 21:961–973, 2011.

[2] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

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

[4] M. A. DePristo, E. Banks, R. Poplin, K. V. Garimella, J. R. Maguire, C. Hartl, A. A. Philippakis, G. del Angel, M. A. Rivas, M. Hanna, et al. A framework for variation discovery and genotyping using next-generation dna sequencing data. *Nature genetics*, 43(5):491–498, 2011.

[5] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge Univ Press, 1998.

[6] E. Garrison and G. Marth. Haplotype-based variant detection from short-read sequencing. *arXiv preprint arXiv:1207.3907*, 2012.

[4]Loosely defined as the likelihood that there is *at least* one genotype with this variant out of all haplotypes seen in the call set.

[7] T. Jiang, G. Sun, Y. Wu, W. Wang, J. Hu, P. Bodily, L. Tian, H. Hakonarson, W. E. Johnson, et al. Low concordance of multiple variant-calling pipelines: practical implications for exome and genome sequencing. 2013.

[8] B. Langmead, C. Trapnell, M. Pop, S. L. Salzberg, et al. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol*, 10(3):R25, 2009.

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

[10] H. Li and R. Durbin. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009.

[11] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, R. Durbin, et al. The sequence alignment/map format and SAMtools. *Bioinformatics*, 25(16):2078–2079, 2009.

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

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

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

[15] N. Siva. 1000 genomes project. *Nature biotechnology*, 26(3):256–256, 2008.

[16] E. S. Snitkin, A. M. Zelazny, P. J. Thomas, F. Stock, D. K. Henderson, T. N. Palmore, J. A. Segre, et al. Tracking a hospital outbreak of carbapenem-resistant Klebsiella pneumoniae with whole-genome sequencing. *Science translational medicine*, 4(148):148ra116–148ra116, 2012.

[17] A. Talwalkar, J. Liptrap, J. Newcomb, C. Hartl, J. Terhorst, K. Curtis, M. Bresler, Y. S. Song, M. I. Jordan, and D. Patterson. SMASH: A benchmarking toolkit for variant calling. *arXiv preprint arXiv:1310.8420*, 2013.

[18] Twitter and Cloudera. Parquet. `http://www.parquet.io`.

[19] M. Zaharia, W. J. Bolosky, K. Curtis, A. Fox, D. Patterson, S. Shenker, I. Stoica, R. M. Karp, and T. Sittler. Faster and more accurate sequence alignment with SNAP. *arXiv preprint arXiv:1111.5572*, 2011.

[20] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, page 10, 2010.