Accelerating Variant Calling on Human Genomes Using a Commodity Cluster

Praveen Rao
praveen.rao@missouri.edu
University of Missouri-Columbia
USA

Arun Zachariah
azachariah@mail.missouri.edu
University of Missouri-Columbia
USA

Deepthi Rao
raods@health.missouri.edu
University of Missouri-Columbia
USA

Peter Tonellato
tonellatop@health.missouri.edu
University of Missouri-Columbia
USA

Wesley Warren
warrenwc@missouri.edu
University of Missouri-Columbia
USA

Eduardo Simoes
simoese@health.missouri.edu
University of Missouri-Columbia
USA

ABSTRACT

Variant calling is a fundamental task that is performed to identify variants in an individual's genome compared to a reference human genome. This task can enable better understanding of an individual's risk to diseases and eventually lead to new innovations in precision medicine and drug discovery. However, variant calling on a large number of human genome sequences requires significant computing and storage resources. While access to such resources is possible today (e.g., through cloud computing), reducing the cost of analyzing genomes has become a major challenge. Motivated by these reasons, we address the problem of accelerating the variant calling pipeline on a large number of human genome sequences using a commodity cluster. We propose a novel approach that synergistically combines data and task parallelism for different stages of the variant calling pipeline across different sequences with minimal synchronization. Our approach employs futures to enable asynchronous computations in order to improve the overall cluster utilization and thereby, accelerate the variant calling pipeline. On a 16-node cluster, we observed that our approach was 3X-4.7X faster than the state-of-the-art Big Data Genomics software.

CCS CONCEPTS

• **Computing methodologies**; • **Applied computing → Genomics**;

KEYWORDS

Human genomes, variant calling, cluster computing, futures

ACM Reference Format:
Praveen Rao, Arun Zachariah, Deepthi Rao, Peter Tonellato, Wesley Warren, and Eduardo Simoes. 2021. Accelerating Variant Calling on Human Genomes Using a Commodity Cluster. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM '21), November 1–5, 2021, Virtual Event, QLD, Australia.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3459637.3482047

CIKM '21, November 1–5, 2021, Virtual Event, QLD, Australia

ACM ISBN 978-1-4503-8446-9/21/11…$15.00
https://doi.org/10.1145/3459637.3482047

1 INTRODUCTION

Genomics is regarded as a Big Data science [49]. It is projected that between 100 million-2 billion humans could be sequenced by 2025 producing between 2-40 exabytes of genome data [49]. With the cost of human whole-genome sequencing (WGS) falling below $1,000 [15], WGS in large-scale studies and clinical practice have become economically feasible. In fact, new genome sequencing initiatives have been launched around the world [9, 11, 16] to understand COVID-19 susceptibility and severity in individuals [33].

The (diploid) human genome contains 6 billion base pairs of deoxyribonucleic acid (DNA) [5]. During DNA sequencing, a sample DNA is sliced into shorts fragments and read as a sequence of bases (a.k.a. *reads*). Due to sequencing errors, a position in the genome is covered by multiple DNA fragments resulting in millions of reads. In essence, a human genome sequence can produce hundreds of gigabytes of data. Such large sizes of human genome sequences pose technical challenges for efficient storage, processing, and analysis.

Recently, companies such as Microsoft [13], Databricks [7], and NVIDIA [14] are providing new tools and services to customers for accelerating analytics on genomics data. Several open source projects have emerged (e.g., GATK4 [1], ADAM-Cannoli [8, 44, 45]) that employ cluster computing frameworks, Apache Spark [52] and Apache Hadoop [51], to manage and analyze large volumes of genome data. Thus, there is growing interest in advancing the state-of-the-art in processing human genomes efficiently at scale.

Once an individual's genome is sequenced, it must be analyzed. Variant calling is a fundamental task that is performed to identify variants in an individual's genome compared to a reference human genome such as single nucleotide polymorphisms (SNPs), short insertions/deletions (indels), copy number variation, and other structural variants [18]. Identifying these variants will enable better understanding of an individual's risk to diseases and eventually lead to new innovations in precision medicine and drug discovery.

A variant calling pipeline consists of several stages [36, 50] including reading the genome sequence data, performing alignment of reads with a reference genome, additional pre-processing steps, and finally, invoking a variant caller to produce raw variants. The raw variants are further processed by variant filtering and annotation steps. The variant calling pipeline involves several computationally intensive tasks and requires significant computing and storage resources to analyze large number of human genome sequences. While access to such resources is possible today through cloud

computing, reducing the cost of analyzing genomes has become a key challenge [40, 44, 45].

Motivated by the aforementioned reasons, we propose a novel approach called AVAH (**A**ccelerating **VA**riant Calling on **H**uman Genomes) by leveraging asynchronous computations and cluster computing technologies for faster execution of the variant calling pipeline. The key contributions of our work are as follows:

- AVAH distributes the task of executing the variant calling pipeline on the input sequences across the cluster nodes. It synergistically combines task parallelism and data parallelism for different stages in the pipeline by launching asynchronous computations using futures [35]. These computations are executed in a sliding window manner on small groups of sequences to control the degree of parallelism and improve cluster utilization.
- AVAH employs two strategies for synchronizing different stages of the variant calling pipeline: The first strategy uses a synchronization barrier after every stage of the pipeline; the second, however, only uses a synchronization barrier at the end of the final stage of the pipeline. While both strategies improve the overall cluster utilization, the latter is designed to achieve the best performance.
- AVAH is built atop Apache Spark and Apache Hadoop and leverages the APIs of existing Big Data Genomics software that enable data parallelism. Through a detailed performance evaluation on a 16-node commodity cluster, we show that AVAH is significantly faster than ADAM-Cannoli, the state-of-the-art Big Data Genomics software.

2 RELATED WORK AND MOTIVATION

2.1 Variant Calling Pipelines

A typical variant calling pipeline for an individual's DNA sample [21, 36] involves a set of stages: (a) reading raw unmapped reads (e.g., in FASTQ format [4]), (b) alignment of the reads with a reference genome (e.g., using BWA [38]) to obtain mapped reads (e.g., in BAM format [22]), (c) marking duplicate reads in the mapped reads, (d) base quality score recalibration (BQSR) [17] and local realignment around the indels (to correct sequencing errors and improve accuracy of downstream processing), and finally, (f) variant calling (e.g., using FreeBayes [19, 34], GATK HaplotypeCaller [12]). The output file in VCF format [23] contains raw variants. Finally, variant filtering and annotation is applied on the raw variants. In fact, there are several pipelines for variant discovery [10, 36, 50].

In the interest of space, we discuss prior work closely related to DNA variant calling pipelines using Apache Hadoop/Spark. CloudBurst [48], CloudAligner [42], SEAL [46], and BigBWA [25] used Apache Hadoop for speeding up the computationally-intensive alignment stage. Hadoop-BAM was developed to support Hadoop-based parallel I/O [43] for sequencing data. Later, SparkBWA [26] employed Apache Spark to speed up alignment using BWA [38]. Cloud Scale-BWAMEM [29] also used Apache Spark to speed up alignment and was adapted for FPGAs [28]. Recently, PipeMEM [53] used Spark pipes to speed up alignment using BWA-MEM [37].

The Broad Institute developed the GATK Best Practices Workflows [20], which have been widely adopted for variant discovery. Halvade [31] parallelized the variant calling pipeline of GATK using

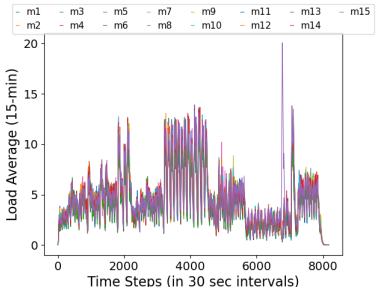


Figure 1: ADAM-Cannoli's cluster utilization

MapReduce [30]. Later, GATK4 was built atop Apache Spark for parallelization [1]. Parabricks was developed to accelerate GATK pipelines using GPUs [14]. Nothaft et. al. [44] created ADAM and Cannoli as part of the Big Data Genomics project [8]: ADAM was designed to enable the processing of large genomic datasets using Spark's primitives. ADAM supports read transformations and correcting errors in aligned reads. Cannoli leverages pipes and parallelizes the alignment process and variant calling by reusing existing tools. Together, we refer to them as ADAM-Cannoli. ADAM-Cannoli is considered the state of the art and was significantly faster than GATK4 [44]. Our goal is similar to that of ADAM-Cannoli – to accelerate the entire variant calling pipeline.

2.2 Motivation

While most of the prior work has focused on accelerating certain stages of the pipeline, we investigate how to accelerate the entire variant calling on a large collection of genome sequences.

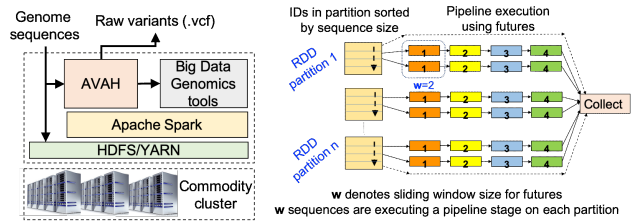
We tested ADAM-Cannoli on a 16-node cluster in CloudLab [32] using 98 paired-end[1] whole genome sequences [3]. The variant calling pipeline was executed without BQSR/indel realignment. ADAM-Cannoli processed one sequence-at-a-time employing data parallelism across all the 15 Spark worker nodes and took close to 69 hours to process all the sequences. We observed that the cluster utilization on the worker nodes was modest at best. Figure 1 shows an example of the 15-minute load average (measured every 30 seconds) on the cluster nodes. Then we executed ADAM-Cannoli on multiple sequences simultaneously (e.g., 2 at-a-time, 4 at-a-time, 8 at-a-time) to increase the load average. Unfortunately, the time taken was higher than before for each case (i.e., 90+ hours) due to resource contention among the Spark jobs. These observations clearly indicate that a new approach is needed that (a) improves the cluster utilization when executing on a large number of genome sequences and (b) executes the variant calling pipeline faster. By achieving higher throughput for variant calling, we can lower the cost of analyzing human genome sequences especially in a cloud computing environment.

3 OUR APPROACH

Motivated by the aforementioned reasons, we propose AVAH to accelerate the variant calling pipeline for human genomes. Our approach draws inspiration from asynchronous computations and the futures abstraction [35]. A *future* is a result of an asynchronous

[1]In paired-end DNA sequencing, a fragment is read in both directions, which enables more accurate alignment and indel detection [24].

computation that may or may not be available yet. It enables non-blocking operations. Several software systems like Ray [39], CIEL [41], and Dask [47] have implemented the futures abstraction for efficient large-scale distributed execution.



(a) (b)

Figure 2: (a) Architecture (b) AVAH's processing model

AVAH's architecture is shown in Figure 2(a). It is built atop Apache Spark and Hadoop. It also leverages the APIs of existing Big Data Genomics software (e.g., Adam-Cannoli, GATK4) that run on Apache Spark to execute the variant calling pipeline. Genome sequences, intermediate files produced by the pipeline, the reference genome, additional data needed during BQSR/indel realignment, and the raw variants can be stored in HDFS. This enable stages of the pipeline to be re-executed on different nodes if required.

Consider a single sample variant calling pipeline supported by ADAM-Cannoli. Table 1 shows the four stages of the pipeline that are invoked sequentially on a human genome. Recall that our goal is to speed up the pipeline execution on a large collection of genomes. AVAH models each stage of the pipeline as an atomic task. Tasks are executed as asynchronous computations using futures. Tasks representing the same pipeline stage managed by a Spark executor (on a worker node) are managed via a sliding window approach. This controls the degree of parallelism and reduces resource contention. Tasks across different executors are controlled by synchronization barriers executed on the Spark driver process.

Table 1: Tasks in a variant calling pipeline

Stage	Stage Description
1	f_I: interleave paired-end FASTQ files to produce a `.ifq` file
2	f_A: align against a reference genome to produce a `.bam` file
3	f_D (or f_{D+}): produce a Parquet [6] file by marking duplicates on mapped reads without pre-processing (or marking duplicates with BQSR/indel realignment)
4	f_V: invoke a variant calling method

We begin with our first approach called AVAH$_x$ that uses a synchronization barrier at the end of each pipeline stage. Algorithm 1 summarizes the steps involved. Each genome sequence in HDFS is identified by a sequence ID. AVAH$_x$ reads an input file containing a list of sequence IDs and the corresponding sequence sizes as a resilient distributed dataset (RDD). (In Spark, an RDD is a distributed immutable collection of items that can be operated in parallel.) The RDD can be *repartitioned* (e.g., by using hash/range partitioning) for load balancing along with sorting the sequence IDs in each partition by size. (See Line 1.) The sorting enables the sliding window of futures, which is introduced later in Algorithm 2, to slide faster. Next, AVAH$_x$ invokes a map operation on each RDD partition along with the appropriate pipeline stage. The map operation is executed on a worker node on the set of sequences identified by the partition. For example, in Line 2, AVAH$_x$ invokes the first stage of the pipeline on the sequences. The map operation on all the partitions returns an RDD containing tuples of sequence IDs and status of the execution of a stage on that sequence (i.e., success or failure). Each task/stage on a sequence can be executed in a data parallel manner using existing Big Data Genomics software. The *collect* call executed by the Spark driver after the map operation completes on all the partitions acts as a synchronization barrier before the next stage can begin. (See Lines 2-5.)

Algorithm 1 AVAH$_x$: Our first approach

Input: p: partitioning function; k: num. of partitions; w: sliding window size for futures

1: S ← read('sequence-list.txt').repartition_and_sort(p, k)
2: r_I ← S.mapPartitions(execFutures(f_I, w)).collect()
3: r_A ← r_I.mapPartitions(execFutures(f_A, w)).collect()
4: r_D ← r_A.mapPartitions(execFutures(f_D or f_{D+}), w).collect()
5: r_V ← r_D.mapPartitions(execFutures(f_V), w).collect()
6: **return** r_V /* Raw variant files are stored in HDFS */

Rather than executing one sequence at a time, AVAH$_x$ maintains a sliding window of outstanding futures representing tasks on the sequence IDs in a single partition that is processed by a worker node. The steps are shown in Algorithm 2. The total number of outstanding futures on a partition is at most w, i.e., the window size. This controls the degree of parallelism and reduces resource contention among different tasks. The sliding window advances by one sequence ID when the future at the beginning of the window completes. As the sequence IDs are sorted by size, it is expected that the sequence denoted by the head will tend to finish faster than the others to allow the window to slide faster. Thus, AVAH$_x$ *synergistically combines task parallelism and data parallelism* for different stages in the pipeline across different sequences.

Algorithm 2 execFutures: A sliding window approach

Input: P: RDD partition containing sequence IDs; f: task to execute; w: sliding window size

1: **for** j=0 to w-1 **do**
2: $ret[j]$ ← Future($f(P[j].sequenceID)$)
3: $head$ ← 0
4: **while** $head \leq P$.length() - w **do**
5: await($ret[head]$)
6: $ret[head + w]$ ← Future($f(P[head + w].sequenceID)$)
7: $head$ ← $head$ + 1
8: await_all($ret[head], \ldots, ret[P.length() - 1]$)

Our second approach called AVAH$_y$ uses a synchronization barrier only at the end of the last stage of the pipeline. An illustration is shown in Figure 2(b). Algorithm 3 summarizes the steps involved. Essentially, AVAH$_y$ chains the map operations that are applied on the partitions for the different pipeline stages. Finally, it invokes a single collect call after the variant calling stage. (See Line 2.) Like

AVAH$_x$, it also uses the sliding window approach (execFutures) for managing futures. Compared to AVAH$_x$, AVAH$_y$ has minimal synchronization among the pipeline stages and is designed to provide better cluster utilization and faster execution. Furthermore, in AVAH$_y$, *two sequences can be executing two different stages of the variant calling pipeline at a given time.*

Algorithm 3 AVAH$_y$: Our second but improved approach

Input: p: partitioning function; k: num. of partitions; w: sliding window size for futures

```
1: S ← read('sequence-list.txt').repartition_and_sort(p, k)
2: r_V ← S.mapPartitions(execFutures(f_I, w))
          .mapPartitions(execFutures(f_A, w))
          .mapPartitions(execFutures(f_D or f_D+), w)
          .mapPartitions(execFutures(f_V), w).collect()
3: return r_V /* Raw variant files are stored in HDFS */
```

4 PERFORMANCE EVALUATION

We compared the performance of AVAH$_x$, AVAH$_y$, and ADAM-Cannoli. As ADAM-Cannoli [44] ran faster than GATK4, we did not include GATK4 in the comparison.

4.1 Implementation, Setup and Dataset

We implemented AVAH$_x$ and AVAH$_y$ in Scala and built the code using Scala 2.12.8. (The APIs of ADAM-Cannoli were used for different pipeline stages.) The code was run with Apache Spark 3.0.0, Apache Hadoop 3.2.0, and OpenJDK 8. We ran the experiments on a 16-node cluster set up in two different data centers of CloudLab [32]: Clemson and Wisconsin. The nodes were physical machines running Ubuntu 16.04 and connected by a Gigabit Ethernet network (10 Gbps). Each node in Clemson had 2 Intel E5-2660 10-core CPUs (2.20 GHz) and 256 GB RAM. Each node in Wisconsin had 2 Intel E5-2660 10-core CPUs (2.60 GHz) and 160 GB RAM. (Each core had 2 hardware threads.) Due to limited root filesystem storage on the nodes, we mounted additional local block storage (striped across multiple physical disks) on each node. HDFS was set up using the local block storage of the nodes. We also installed ADAM 0.33, Cannoli 0.11, FreeBayes 1.3.2, and BWA 0.7.17 on the nodes.

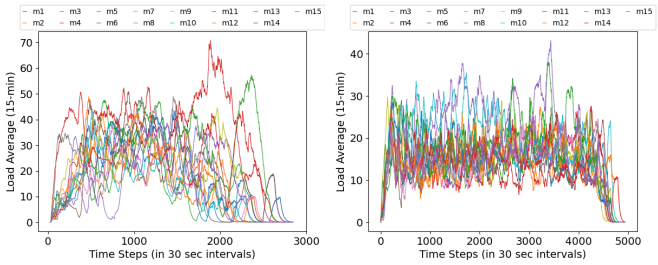
We downloaded 98 human whole genome sequences from the 1000 Genomes Project [3, 27]. The total size of these low-coverage (paired-end) sequences was 632 GB (in compressed form). The minimum and maximum size of the sequences were 2.2 GB and 15.4 GB, respectively.

Table 2: Time comparison (best time shown in bold)

Data center	Time taken (no preprocessing)		Our best speedup
	ADAM-Cannoli	AVAH$_y$	
Clemson	68 h 51 min	**22 h 13 min**	3.10X
Wisconsin	46 h 7 min	**15 h 1 min**	3.07X
Data center	Time taken (w/ BQSR/indel realignment)		Our best speedup
	ADAM-Cannoli	AVAH$_y$	
Clemson	127 h 34 min	**39 h 55 min**	3.17X
Wisconsin	147 h 45 min	**31 h 24 min**	4.70X

4.2 Performance Results

We tested two versions of the variant calling pipeline: (a) with no pre-processing in Stage 3 (f_D) and (b) with BQSR/indel realignment in Stage 3 (f_{D+}). Note that BQSR/indel realignment are computationally intensive and require more I/O due to additional files that must be processed. Hence, the latter version always required more time to complete. (We used GRCh38 [2] as the reference genome and a sliding window size of 2/3.) The input RDD containing the sequence IDs and their sizes had 15 partitions. Table 2 shows the time taken by the approaches. (In the interest of space, AVAH$_x$ timings are not shown. We observed that AVAH$_y$ was at least 2X faster than AVAH$_x$, which in turn was faster than ADAM-Cannoli.) For both versions, AVAH$_y$ was the fastest as it is designed to synergistically combine both task and data parallelism for different stages of the pipeline using futures and minimal synchronization barriers. On the other hand, ADAM-Cannoli only exploited data parallelism. For the first pipeline version, AVAH$_y$ was 3X faster than ADAM-Cannoli. For the second pipeline version, AVAH$_y$ was between 3X-4.7X faster than ADAM-Cannoli.



(a) No preprocessing (f_D)

(b) With BQSR/indel realignment (f_{D+})

Figure 3: Cluster utilization of AVAH$_y$ in Clemson

While both AVAH$_x$ and AVAH$_y$ improved the overall cluster utilization compared to ADAM-Cannoli, AVAH$_y$ yielded the best results. Figure 3 shows the 15-minute load average of AVAH$_y$ (for both pipeline versions) measured every 30 seconds on the Spark worker nodes in Clemson. Similar trends were observed in Wisconsin and are not shown in the interest of space. These results demonstrate the effectiveness of AVAH$_y$ for accelerating the variant calling pipeline on human genomes.

5 CONCLUSION

We developed a novel approach to accelerate the variant calling pipeline on human genomes using futures to synergistically combine data and task parallelism for different stages of the pipeline across different sequences. Our best approach introduces minimal synchronization barriers and significantly improves the cluster utilization. As a result, it was significantly faster than ADAM-Cannoli. We believe our work provides an effective template for accelerating different variant calling pipelines using cluster computing. Our code is available at https://github.com/MU-Data-Science/EVA.

Acknowledgments: This work was supported by the National Science Foundation under Grant No. 2034247.

REFERENCES

[1] GATK4. https://github.com/broadinstitute/gatk.

[2] Genome Reference Consortium Human Build 38. https://www.ncbi.nlm.nih.gov/assembly/GCF_000001405.26/, 2013.

[3] 1000 Genomes Phase 3 Release. https://www.internationalgenome.org/data-portal/data-collection/phase-3, 2015.

[4] FASTQ Files Explained. https://support.illumina.com/bulletins/2016/04/fastq-files-explained.html, 2016.

[5] Size Matters: A Whole Genome is 6.4B Letters. https://www.veritasgenetics.com/our-thinking/whole-story, 2017.

[6] Apache Parquet. https://parquet.apache.org/documentation/latest/, 2018.

[7] Building the Fastest DNASeq Pipeline at Scale. https://databricks.com/blog/2018/09/10/building-the-fastest-dnaseq-pipeline-at-scale.html, 2018.

[8] Big Data Genomics. https://github.com/bigdatagenomics/, 2020.

[9] COVID-19 Genomics UK Consortium. https://www.cogconsortium.uk/, 2020.

[10] DNA-Seq Analysis Pipeline. https://docs.gdc.cancer.gov/Data/Bioinformatics_Pipelines/DNA_Seq_Variant_Calling_Pipeline, 2020.

[11] Genomics on a Mission: Meeting the COVID-19 Challenge. https://www.genomecanada.ca/en/news/genomics-mission-meeting-covid-19-challenge/, 2020.

[12] HaplotypeCaller in a Nutshell. https://gatk.broadinstitute.org/hc/en-us/articles/360035531412-HaplotypeCaller-in-a-nutshell, 2020.

[13] Microsoft Genomics. https://www.microsoft.com/en-us/genomics/, 2020.

[14] NVIDIA Clara Parabricks. https://developer.nvidia.com/clara-parabricks, 2020.

[15] The Cost of Sequencing a Human Genome. https://www.genome.gov/about-genomics/fact-sheets/Sequencing-Human-Genome-cost, 2020.

[16] The COVID Human Genetic Effort. https://www.covidhge.com/, 2020.

[17] Base Quality Score Recalibration (BQSR). https://gatk.broadinstitute.org/hc/en-us/articles/360035890531-Base-Quality-Score-Recalibration-BQSR-, 2021.

[18] Ensembl Variation - Variant Classification. https://m.ensembl.org/info/genome/variation/prediction/classification.html, 2021.

[19] FreeBayes, a Haplotype-Based Variant Detector. https://github.com/freebayes/freebayes, 2021.

[20] Genome Analysis Toolkit. https://gatk.broadinstitute.org/hc/en-us, 2021.

[21] Germline Short Variant Discovery (SNPs + Indels). https://gatk.broadinstitute.org/hc/en-us/articles/360035535932-Germline-short-variant-discovery-SNPs-Indels-, 2021.

[22] Sequence Alignment/Map Format Specification. https://samtools.github.io/hts-specs/SAMv1.pdf, 2021.

[23] The Variant Call Format (VCF) Version 4.2 Specification. https://samtools.github.io/hts-specs/VCFv4.2.pdf, 2021.

[24] What is Paired-End Sequencing? https://www.illumina.com/science/technology/next-generation-sequencing/plan-experiments/paired-end-vs-single-read.html, 2021.

[25] Abuin, J. M., Pichel, J. C., Pena, T. F., and Amigo, J. BigBWA: Approaching the Burrows-Wheeler Aligner to Big Data Technologies. *Bioinformatics 31*, 24 (2015), 4003–4005.

[26] Abuin, J. M., Pichel, J. C., Pena, T. F., and Amigo, J. SparkBWA: Speeding up the Alignment of High-Throughput DNA Sequencing Data. *PLoS ONE 11*, 5 (2016).

[27] Auton, A., and et.al. A Global Reference for Human Genetic Variation. *Nature 526*, 7571 (2015), 68–74.

[28] Chen, Y.-T., Cong, J., Fang, Z., Lei, J., and Wei, P. When Apache Spark Meets FPGAs: A Case Study for Next-Generation DNA Sequencing Acceleration. In *Proc. of the 8th USENIX Conference on Hot Topics in Cloud Computing* (2016), pp. 64–70.

[29] Cong, J., Lei, J., Li, S., Peto, M., Spellman, P., Wei, P., and Zhou, P. CS-BWAMEM: A Fast and Scalable Read Aligner at the Cloud Scale for Whole Genome Sequencing. In *High Throughput Sequencing Algorithms and Applications (HITSEQ)* (2015).

[30] Dean, J., and Ghemawat, S. MapReduce: Simplified Data Processing on Large Clusters. In *Proc. of the 6th Symposium on Operating Systems Design and Implementation* (2004), pp. 137–149.

[31] Decap, D., Reumers, J., Herzeel, C., Costanza, P., and Fostier, J. Halvade: Scalable Sequence Analysis with MapReduce. *Bioinformatics 31*, 15 (2015), 2482–2488.

[32] Duplyakin, D., Ricci, R., Maricq, A., Wong, G., Duerig, J., Eide, E., Stoller, L., Hibler, M., Johnson, D., Webb, K., Akella, A., Wang, K., Ricart, G., Landweber, L., Elliott, C., Zink, M., Cecchet, E., Kar, S., and Mishra, P. The Design and Operation of CloudLab. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)* (2019), pp. 1–14.

[33] Fricke-Galindo, I., and Falfan-Valencia, R. Genetics Insight for COVID-19 Susceptibility and Severity: A Review. *Frontiers in Immunology 12* (2021), 1057.

[34] Garrison, E., and Marth, G. Haplotype-Based Variant Detection from Short-Read Sequencing, 2012.

[35] Halstead, R. H. MULTILISP: A Language for Concurrent Symbolic Computation. *ACM Transactions on Programming Languages and Systems 7*, 4 (1985), 501–538.

[36] Koboldt, D. C. Best Practices for Variant Calling in Clinical Sequencing. *Genome Medicine 12*, 1 (2020), 91.

[37] Li, H. Aligning Sequence Reads, Clone Sequences and Assembly Contigs With BWA-MEM. *arXiv e-prints* (Mar. 2013), arXiv:1303.3997.

[38] Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., and R. Durbin, e. a. The Sequence Alignment/Map Format and SAMtools. *Bioinformatics 25*, 16 (2009), 2078–2079.

[39] Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., Elibol, M., Yang, Z., Paul, W., Jordan, M. I., and Stoica, I. Ray: A Distributed Framework for Emerging AI Applications. In *Proc. of the 13th USENIX Conference on Operating Systems Design and Implementation* (2018), pp. 561–577.

[40] Muir, P., Li, S., Lou, S., Wang, D., Spakowicz, D. J., Salichos, L., Zhang, J., Weinstock, G. M., Isaacs, F., Rozowsky, J., and Gerstein, M. The Real Cost of Sequencing: Scaling Computation to Keep Pace with Data Generation. *Genome Biology 17*, 1 (2016), 53.

[41] Murray, D. G., Schwarzkopf, M., Smowton, C., Smith, S., Madhavapeddy, A., and Hand, S. CIEL: A Universal Execution Engine for Distributed Data-Flow Computing. In *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)* (USA, 2011), pp. 113–126.

[42] Nguyen, T., Shi, W., and Ruden, D. CloudAligner: A Fast and Full-Featured MapReduce Based Tool for Sequence Mapping. *BMC Research Notes 4*, 1 (2011), 171.

[43] Niemenmaa, M., Kallio, A., Schumacher, A., Klemela, P., Korpelainen, E., and Heljanko, K. Hadoop-BAM: Directly Manipulating Next Generation Sequencing Data in the Cloud. *Bioinformatics 28*, 6 (2012), 876–877.

[44] Nothaft, F. A. *Scalable Systems and Algorithms for Genomic Variant Analysis*. PhD thesis, UC Berkeley, ProQuest, 2017.

[45] Nothaft, F. A., Massie, M., Danford, T., Zhang, Z., Laserson, U., Yeksigian, C., Kottalam, J., Ahuja, A., Hammerbacher, J., Linderman, M. D., Franklin, M. J., Joseph, A. D., and Patterson, D. A. Rethinking Data-Intensive Science Using Scalable Analytics Systems. In *Proc. of the 2015 ACM SIGMOD International Conference on Management of Data* (2015), pp. 631–646.

[46] Pireddu, L., Leo, S., and Zanetti, G. SEAL: A Distributed Short Read Mapping and Duplicate Removal Tool. *Bioinformatics 27*, 15 (2011), 2159–2160.

[47] Rocklin, M. Dask: Parallel Computation with Blocked algorithms and Task Scheduling. In *Proc. of the 14th Python in Science Conference* (2015), K. Huff and J. Bergstra, Eds., pp. 130 – 136.

[48] Schatz, M. C. CloudBurst: Highly Sensitive Read Mapping with MapReduce. *Bioinformatics 25*, 11 (2009), 1363–1369.

[49] Stephens, Z. D., Lee, S. Y., Faghri, F., Campbell, R. H., Zhai, C., Efron, M. J., Iyer, R., Schatz, M. C., Sinha, S., and Robinson, G. E. Big Data: Astronomical or Genomical? *PLOS Biology 13*, 7 (2015), 1–11.

[50] Supernat, A., Vidarsson, O. V., Steen, V. M., and Stokowy, T. Comparison of Three Variant Callers for Human Whole Genome Sequencing. *Scientific Reports 8* (2018).

[51] White, T. *Hadoop: The Definitive Guide*, 1st ed. O'Reilly Media, Inc., 2009.

[52] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. Spark: Cluster Computing with Working Sets. In *Proc. of the 2nd USENIX Conference on Hot Topics in Cloud Computing* (2010), pp. 10–10.

[53] Zhang, L., Liu, C., and Dong, S. PipeMEM: A Framework to Speed Up BWA-MEM in Spark with Low Overhead. *Genes 10*, 11 (2019).