Building Highly-Optimized, Low-Latency Pipelines for Genomic Data Analysis

Yanlei Diao, Abhishek Roy
University of Massachusetts Amherst
{yanlei,aroy}@cs.umass.edu

Toby Bloom
New York Genome Center
tbloom@nygenome.org

Abstract. Next-generation sequencing has transformed genomics into a new paradigm of data-intensive computing. The deluge of genomic data needs to undergo deep analysis to mine biological information. Deep analysis pipelines often take days to run, which entails a long cycle for algorithm and method development and hinders future application for clinic use. In this project, we aim to bring big data technology to the genomics domain and innovate in this new domain to revolutionize its data crunching power. Our work includes the development of a deep analysis pipeline, a parallel platform for pipeline execution, and a principled approach to optimizing the pipeline. We also present some initial evaluation results using existing long-running pipelines at the New York Genome Center, as well as a variety of real use cases that we plan to build in the course of this project.

1. INTRODUCTION

Genomics has revolutionized almost every aspect of life sciences, including biology, medicine, agriculture, and the environment. At the same time, next-generation sequencing has transformed genomics into a new paradigm of data-intensive computing [2]: the sequencing instruments are now able to produce billions of reads of a DNA sequence in a single run, raising the potential to answer biological questions with unprecedented resolution and speed. Large sequencing centers are already producing terabytes of genomic data each day. Figure 1 shows that the projected growth rates for data acquisition from sequencers will outpace the growth rates of processor/memory capacities as per Moore's law. This trend is making genomic data processing and analysis an increasingly important and challenging problem.

Despite the high volume, the deluge of genomic data needs to undergo complex processing to mine biological information from vast sets of small sequence reads. There has been intensive research on individual problems in genomic data analysis, such as assembly (e.g., [18, 37, 49]), alignment (e.g., [20, 22, 23, 26], SNP (single nucleotide polymorphism) detection (e.g., [7, 25]), and SNP genotyping (e.g., [6, 7]). On the other hand, a recent trend is that genomics is moving toward "deep analysis" with significantly increased complexity in data processing. Such deep analysis often requires sequence data to be transformed from the raw input through a large number of steps, where each step brings more biological meaning to the data or improves the quality of the derived information, until finally a high-level observation (e.g., regarding a population of objects under study) with a significant biological meaning can be derived. Our work is driven by the data processing needs of such deep analysis, characterized as follows:

1. *Deep analysis in genomics research requires a full pipeline that integrates many data processing and analysis tools.* Such a pipeline typically consists of a large number of steps, from the initial alignment (mapping short reads to the reference genome), to data cleaning for fixing many issues introduced by noisy input data (e.g., removing duplicate reads, recalibrating quality scores, and local re-alignment based on known SNP locations), to variant calling and genotyping, and further to advanced statistical mining. Based on our past work on building such pipelines for general genome research [27, 31] as well as specific research projects such as genome-wide association studies (GWAS) [41], we observe that there are usually at least 15-20 processing steps in the pipeline to ensure that (1) data is transformed into high-level biological information (e.g., genotypes, associations between mutations and phenotypes), and (2) such information is derived with all (most of) the noise removed from the input data—omitting the necessary data cleaning steps may have a severe negative impact on the quality of derived information.

2. *Deep analysis pipelines often take long to run, which entails a long cycle for algorithm and method development for genomics research and hinders future application for clinic use.* The long running time is due to several main factors: (1) Current bioinformatics pipelines are extremely inefficient with respect to I/O. This is because a data set of hundreds of GBs per sample is not reduced by processing, but rather

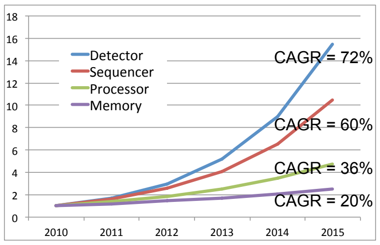


Figure 1: Genomic data grows faster than Moore's Law (figure courtesy of Katherine Yelick).

7th Biennial Conference on Innovative Data Systems Research (CIDR'15) January 4-7, 2015, Asilomar, California, USA.

increased in size through most of the analysis pipelines. In addition, all the data is stored in files currently, causing each processing step to read and write an entire file. Further, different steps in the pipeline often access data in different orders, hence requiring data to be sorted frequently. (2) Bioinformatics pipelines often involve complex analysis algorithms, ranging from Burrows-Wheeler transform, to grouping and complex statistical analysis, to methods based on geometric operations. Such algorithms are both CPU and IO intensive and can take days to weeks to run. Such long running algorithms include Mutect [4] for somatic variant calling and Theta [33] for complex cancer genome analysis, to name a few. (3) It is not easy to parallelize these pipelines because different processing steps permit different ways to partition the data and the decision of whether it is safe to partition data in a particular way requires deep understanding of how each algorithm works.

For all above reasons, current pipelines can take 3.4 to 23.4 days to complete for a test sample based on our initial profiling at the New York Genome Center, where the variation depends on the complexity of analysis algorithms used. Such long-running time is far from the desired turnaround time of one day for clinic use.

3. *Most small research labs do not have access to high-performance pipelines for genomic data analysis.* In current practice, research labs send DNA samples to a commercial sequencing company, and receives processed results including short read sequences and detected genomic variants, with a turnaround time of two months. Many research groups wish to reduce the long turnaround time, but lack computing resources and technical know-how to support full pipelines on high-volume data (100-200 GB per sample) by themselves.

An alternative is to use genome analysis centers that provide open access to the research community. However, performance problem persist. For instance, the popular Galaxy system [9] provides a web portal for assembling user pipelines, but internally uses a simple integration of existing tools with limited optimization and no parallel processing. Hence, its performance is not better than the profiling results show above (in practice, worse due to shared use). National centers like NCGAS [30] use supercomputers for genome analysis. However, NCGAS focuses on individual methods, e.g., assembly, but not a deep pipeline. Hence, it neglects problems related to data transfer between steps and frequent data shuffling, which is observed to be a main contributor to inefficiencies and delays. It further lacks sufficient support for spreading the data and computation of many analysis methods to a cluster of nodes.

A final option is to deploy an existing pipeline (e.g., from Galaxy) on the cloud. However, running existing (unoptimized) pipelines in the cloud can incur very high costs for research labs. In the cloud setting, a long running pipeline means higher payment as the pricing policy is based on CPU time and I/O frequency. Our previous experience with Amazon EC2 showed that running experiments for 8 days on 20 extra large instances (4 virtual cores, 15GB memory per instance) costed \$3,400, an average of over \$400 per day—this may not be a sustainable plan for most small research labs doing computational genomics.

Our work was motivated by a valuable lesson that we learned from big data industry: the success of big data providers like Google is due not only to their intelligent search algorithms, but also to powerful infrastructures that allow engineers to try any new idea, innovative or experimental, on thousands of nodes and obtain results within minutes or hours. Such early results and insights can significantly reduce the cycle of design, development, and validation of new algorithms. Hence, we anticipate that such powerful infrastructures, accessed for free or at reasonable costs by genomics researchers, will be a key enabler of many new models and algorithms developed for genomics and help advance this field at unprecedented speed as big data technology did for Internet companies.

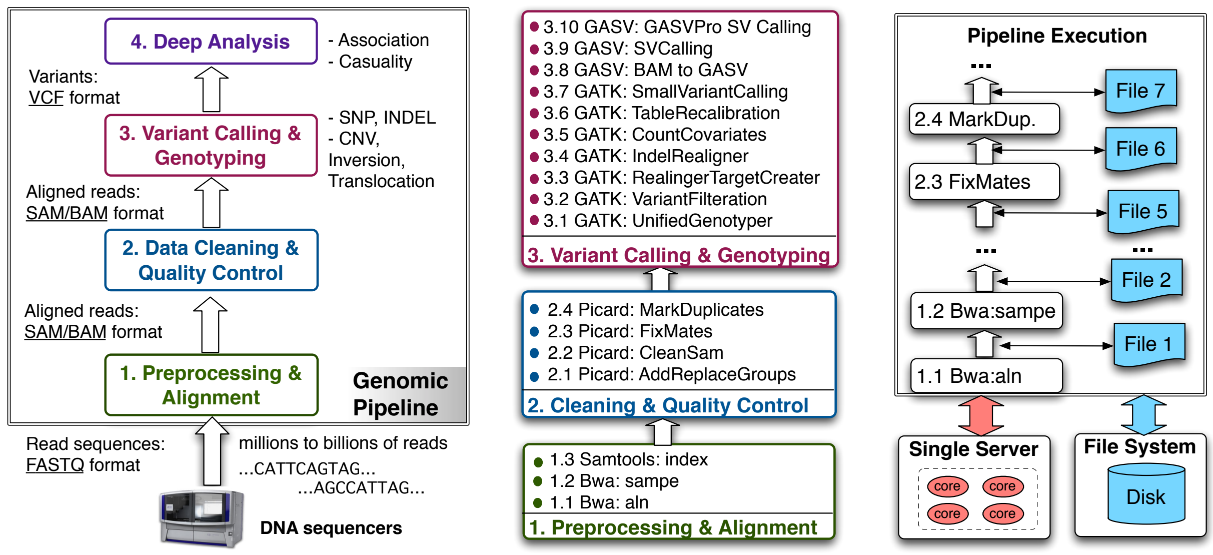
In this project, we bring the latest big data technology to the genomics domain, and innovate in this new domain to revolutionize its data crunching power. Our work includes several main efforts:

1. *We develop a deep pipeline for genomic data analysis and profile it to identify performance bottlenecks.* We set out to implement a full pipeline consisting of alignment, data cleaning and quality control, various types of variant calling (SNP, insertions, deletions, and structural variants), and examples of statistical analysis such as association mining over a population [41]. We further provide profiling results showing the CPU and I/O costs of the pipeline execution. Our profiling results reveal two types of long-running steps: those steps that are both CPU-intensive, due to expensive algorithms used, and I/O intensive, due to large files read and written; and those steps that are mostly I/O intensive, due to (repeated) sorting of large read files.

2. *We parallelize the pipeline in the big data infrastructure.* For those steps that incur high CPU and I/O costs, we parallelize them using the MapReduce model. In particular, we choose to use the open-source Hadoop system [13] for MapReduce processing, which forms the core of a large ecosystem of freely available software tools for big data analytics. Our first effort is to port existing genomic analysis programs into the distributed Hadoop file system (HDFS), with no or minimum change of existing analysis programs. To do so, we provide a new storage substrate that supports the same data access API as in existing programs but works on HDFS, and new features such as logical data partitioning and co-location. Our second effort is to parallelize many processing steps in the pipeline using the MapReduce framework (Hadoop). Such parallel processing supports not only embarrassingly parallel steps of the pipeline, but also many processing steps that partition data based on user-defined criteria (e.g., quality score recalibration). To mitigate I/O costs dominant in many data cleaning steps, we further explore streaming technology as a means to reduce I/O as well as total running time, and also columnar storage to replace the data files communicated between consecutive steps.

3. *We bring the principled approach for query optimization from the database area to bear on the process of optimizing genomic pipelines.* Query optimization in relational databases considers alternative equivalent implementations and chooses the most efficient one for a given workload. We explore a similar approach to pipeline optimization: We enhance the Hadoop system with alternative implementations of logical data partitioning and choose the most efficient one for each processing step. We propose cost-based optimization to choose the appropriate level (degree) of parallelism used in each processing step. We further identify opportunities in the pipeline where materialized views and across-step optimization can help reduce computation and I/O costs.

4. *Development and use cases.* Our system integrates a


4. Deep Analysis
- Association
- Casuality
Variants: VCF format
3. Variant Calling & Genotyping
- SNP, INDEL
- CNV, Inversion, Translocation
Aligned reads: SAM/BAM format
2. Data Cleaning & Quality Control
Aligned reads: SAM/BAM format
1. Preprocessing & Alignment
Genomic Pipeline
Read sequences: FASTQ format
millions to billions of reads
...CATTCAGTAG...
...AGCCATTAG...
DNA sequencers
3.10 GASV: GASVPro SV Calling
3.9 GASV: SVCalling
3.8 GASV: BAM to GASV
3.7 GATK: SmallVariantCalling
3.6 GATK: TableRecalibration
3.5 GATK: CountCovariates
3.4 GATK: IndelRealigner
3.3 GATK: RealingerTargetCreater
3.2 GATK: VariantFilteration
3.1 GATK: UnifiedGenotyper
3. Variant Calling & Genotyping
2.4 Picard: MarkDuplicates
2.3 Picard: FixMates
2.2 Picard: CleanSam
2.1 Picard: AddReplaceGroups
2. Cleaning & Quality Control
1.3 Samtools: index
1.2 Bwa: sampe
1.1 Bwa: aln
1. Preprocessing & Alignment
Pipeline Execution
2.4 MarkDup.
2.3 FixMates
1.2 Bwa:sampe
1.1 Bwa:aln
File 7
File 6
File 5
File 2
File 1
Single Server
core
File System
Disk

(a) Pipeline overview. (b) Example processing steps. (c) Existing architecture.

Figure 2: A pipeline for genomic data processing and deep analysis.

web portal, a deep analysis pipeline, and a platform for parallelization and optimization of the pipeline. While our current development focuses on parallelization and optimization of individual, expensive steps of the pipeline, in the long run we plan to deploy and evaluate our system using existing long-running pipelines (e.g., structural variant calling and cancer pipelines) at the New York Genome Center. To fully evaluate our parallel genomic processing pipeline and gain user feedback: we have planned several use cases, including variant calling and genotyping, analysis method development and comparison, population studies, and common disease studies, which we outline at the end of the paper.

2. A DEEP GENOMIC ANALYSIS PIPELINE

In this section, we outline our design of a deep pipeline for genomic data analysis, and present profiling results to reveal major performance bottlenecks.

2.1 A Pipeline for Genomic Data Analysis

To support the data processing needs of most bioinformatics users, we are assembling a pipeline for genomic data analysis which includes most common processing steps. Figure 2(a) shows an overview of the pipeline with a number of main phases of processing. Figure 2(b) shows some of the detailed processing steps in the first three phases.

The input to our pipeline is a large collection of short sequence reads for a single genome sample or a batch of them. There can be millions to billions of short reads for each genome sample. The input data is typically encoded in the FASTQ format [5], which is a text-based format for storing both a biological sequence (a nucleotide sequence) and its corresponding quality scores. At present, we support paired end reads encoded in FASTQ.

1. *Alignment:* After some preprocessing to change the data format (if needed), the first main task is alignment: the short reads are aligned against a reference genome, and each aligned read is assigned one or multiple mapped positions on the reference genome together with mapping quality scores. While there are many alignment algorithms available (e.g., [20, 22, 26]), our pipeline uses BWA [22] as the default algorithm due to its ability to support multiple aligned positions in the face of mapping ambiguity. The user can always customize our pipeline by choosing to use another aligner. The alignment result is encoded in the standard SAM/BAM format [42], where SAM includes attributes such as the mapped location and mapping score for each read, and BAM is the compressed, indexed binary format of SAM.

Figure 3 shows an example where the sequenced genome differs from the reference genome with two true mutations, $A \rightarrow C$ and $C \rightarrow A$. In this example, nine read sequences are aligned against the reference genome with up to five mismatches allowed per read—such mismatches must be allowed in order to detect mutations, which occur in every person's genome. The first four reads differ from the reference genome on the two bases where mutations occur among others, but the letters do not agree with the true genome. Most likely these reads have been mis-aligned to this fragment of the genome. The bottom five reads have the correct letters for the two bases where mutations occur, but have three additional mismatches, in relatively close positions, that differ from the true genome. Such mismatches can either come from errors in raw data or indicate that these reads should be aligned somewhere else. Such data quality issues will be addressed in the next phase of the pipeline.

2. *Data cleaning and quality control:* After alignment, the pipeline goes through a number of steps to remove the effect of noisy data from the rest of processing. For instance, *FixMates* (step 2.3 in Figure 2(b)) makes mate pair information consistent between a read and its pair, which is needed due to the limitations of alignment software. *MarkDuplicates* (2.4) flags duplicate reads that are those pair-end reads mapped to exactly the same start and end positions. These reads are unlikely to occur in a true random sampling based sequencing process, and mostly originate from DNA preparation methods. They cause biases that will skew variant calling results. We implement these steps using the Picard tools [38].

3. *Variant calling:* The next phase detects a range of variants against the reference genome. Our pipeline includes

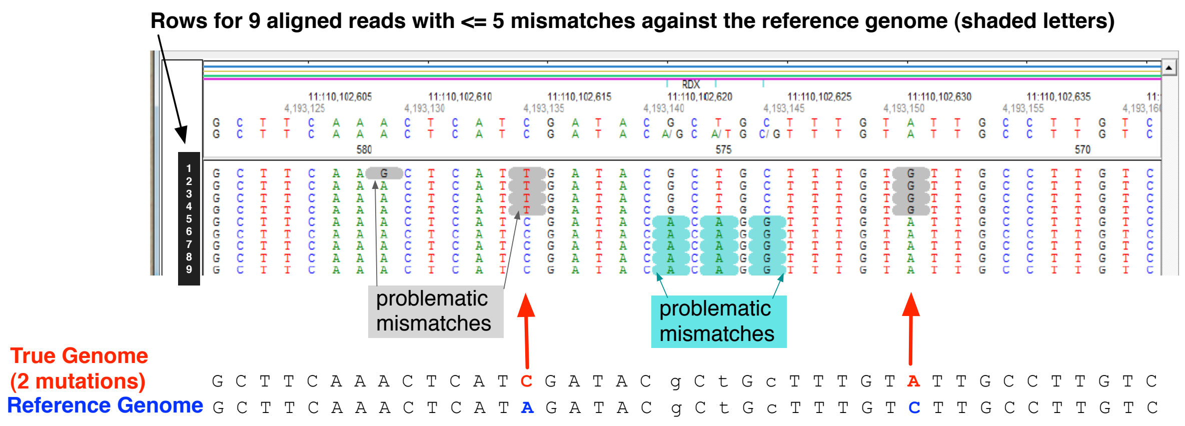

Rows for 9 aligned reads with <= 5 mismatches against the reference genome (shaded letters)
problematic mismatches
problematic mismatches
True Genome (2 mutations)
G C T T C A A A C T C A T C G A T A C g C t G c T T T G T A T T G C C T T G T C
Reference Genome
G C T T C A A A C T C A T A G A T A C g C t G c T T T G T C T T G C C T T G T C

Figure 3: Examples of alignments of read sequences against a reference genome.

two branches, for small variant calling and large structural variant calling, respectively, as described below. The output of this phase is in the Variant Call Format (VCF) [47].

Small variant calls include single nucleotide variants (SNPs) and small insertions/deletions (INDELs). Our design of the pipeline follows the best practice [10] that mixes variant calling and quality control steps. In particular, the *UnifiedGenotyper* (step 3.1) calls both SNPs and small ($\leq$20 bp) INDELs. *Variant Filtration* (step 3.2) flags suspicious variants based on quality scores, strand biases, etc. and flags variants in conserved regions. *Indel Realigner* (steps 3.3 and 3.4) takes the known SNP locations from the knowledge base and performs local realignment around those locations to minimize the total number of mismatched bases. *Quality score recalibration* (steps 3.5 and 3.6): Quality scores returned by the sequencer often differ from the actual error rates present in the data because they can be affected by many covariates such as the machine cycle, the position of a base within a read, neighboring bases, etc. As such, quality scores in input data can be recalibrated based on the empirical error rate in each data group, defined as the data having the same value of user-defined covariates.

Large structural variants (SV) include copy number variants (CNVs), inversion, translocation, etc. There is hardly any commercial software that can detect most of these variants. We studied the literature of SV detection including those techniques surveyed in [1] and other recent ones [4, 33, 44, 45, 48]. By default, our pipeline calls the GASV/Pro algorithms [44, 45] to detect structural variants because these algorithms integrate both the pair-end read signal and the read depth signal to detect structural variants including deletion, inversion, and translation, and can achieve much improved accuracy. We are currently investigating other more complex algorithms (e.g., [4, 33]) for inclusion into our pipeline. The user can customize our pipeline with any other SV calling algorithm.

4. Deep analysis: The next phase performs sophisticated statistical analysis over a population and produces high-level biological information, such as associations or causal relationships between genomic variants and phenotypes, or functional pathways in response to evolutionary, environmental, and physiological changes.

For instance, our pipeline includes an advanced genome-wide *association mining* algorithm [41]. It improves existing association mining methods [14, 46] to suit genomic data. In particular, the genome-wide association study presents several main differences from traditional association mining for transaction data: First, as genomic variations are rare in nature, the extremely low support for such variations makes existing algorithms highly inefficient or unable to complete. Second, the interestingness metric for association rules is usually confidence, which produces too many trivial and repetitive rules in the genomic domain and hides truly interesting ones. Third, large structural variants such as CNVs are never fully aligned across different patients. Hence, they cannot be used as a fixed vocabulary of items as in existing algorithms. Instead, they should be divided into small fragments and mined for association by considering proximity of these fragments. Our algorithm extends a recent one [24] to support a new interestingness metric, extremely low support, and proximity-aware mining. Over the course of this project, we will add more statistical algorithms based on the needs of our case studies.

2.2 Performance Issues Revealed in Profiling

We have performed an initial profiling study of the pipeline to understand its performance. The standard architecture for pipeline execution is shown in Figure 2(c). Each processing step takes a set of input files and writes an output file. The input files can be a combination of the following three types, depending on the computation needs in a processing step:

1. The reference genome is a digital nucleic acid sequence, where a letter 'A', 'C', 'T', or 'G' is assigned to each base for up to 3 billion bases. The file of the human reference genome (build 37) contains 3.2 GB in total.

2. The file of reads is a large set (usually, billions) of short reads of a test genome, where each base of the genome is read k times on the average. The number k is called the "*coverage*" of the sequencing process, with typical numbers in the range of 30 to 120 (where higher numbers are often used for cancer samples). The files of reads are typically hundreds of gigabytes with compression, but can also go over a 1 terabyte with compression.

   While the raw reads are usually encoded in the standard text-based format, FASTQ [5], they are transformed into SAM/BAM format [42] for processing at the beginning of the pipeline. The SAM format includes an optional header section and an alignment

Field	**Brief description**
QNAME	Pair name
FLAG	Integer representation of a bitwise flag
RNAME	Chromosome name
POS	Leftmost mapping position of the read
MAPQ	Mapping quality
CIGAR	Relationship between the base sequence of the read and the mapping coordinates
RNEXT	Chromosome name of the other read of the pair
PNEXT	Position of the other read of the pair
TLEN	Signed length of the read pair
SEQ	Base sequence of the read
QUAL	String representation of quality value for each base of the read

Read 1:	20GAVAAXX100126 \t 99 \t 1 \t 1263352 \t 60 \t 20M \t = \t 1263380 \t 48 \t AAGACGCCTCAGCCACGGCT \t *B@BFFABDHDHHDHIGGGIG* \n
Read 2:	20GAVAAXX100126 \t 147 \t 1 \t 1263380 \t 60 \t 20M \t = \t 1263352 \t -48 \t TAGTAATAAATTTGCCAATC \t *I <<< JIIHHHF?EADAD*?;= \n
	\t : Tab character, \n : End of line

Table 1: SAM record schema: Each record has eleven mandatory attributes and optional attributes. The table describes the mandatory attributes and shows SAM records for a read pair.

section. Each line in alignment section is a single read record. The read record contains eleven mandatory attributes (Table 1) and optional attributes in the form of `key:value_type:value` triplet. Blocks (default: 64KB) from the SAM file can be compressed and concatenated to create the corresponding BAM file. Thus BAM file format is the binary, compressed version of the human-readable SAM file format. If the reads in the BAM file are sorted by the genomic coordinates, then an index is created to map the genomic regions to the file offsets of the blocks in BAM file.

3. The third type of input file contains statistics of the reads or quality scores of these reads, which are computed from a previous processing step.

The output of a processing step can be one of the following two cases:

1. A *new* file of the reads, which contains both the original reads and additional information such as the mapped location(s) of each read against the reference genome, or a flag set to indicate that a read is a duplicate.

2. A new file of data statistics, which are computed from the reads or their quality scores.

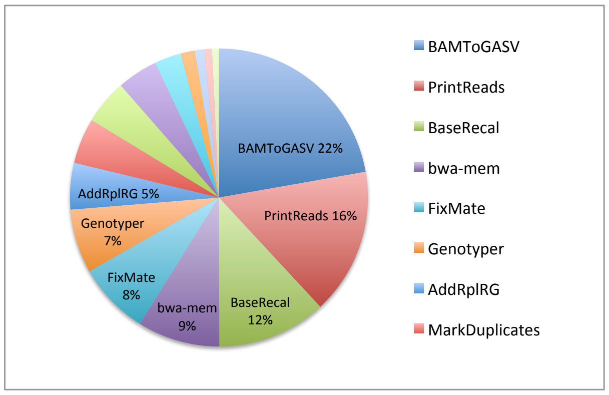
We ran two sets of experiments. The first set of experiments were conducted on a server at UMass Amherst, with 12 Intel Xeon 2.40GHz cores, 64GB RAM, 7200 RPM hard drive, and CentOS version 6.5. We used the NA12878 whole genome dataset (64x coverage) as input to the workflow. The input dataset had 1.24 billion read pairs in two FASTQ files, with 282GB each when uncompressed. We ran a pipeline from alignment (BWA), to data cleaning and quality control (Picard and GATK tools), and to variant calling including both small variant calls (GATK genotyping) and large variant calls (GASV/Pro), where data analysis programs were coded in C or Java. For those steps that have a multi-threaded implementation, e.g., BWA alignment and GATK genotyping, we ran the program with 12 threads. We measured performance using Linux tools `sar` and `perf`.

Some preliminary results are shown in Table 2 for some of the most expensive steps, including the running time, input and output files and sizes, bytes read and written, and CPU instructions incurred in each step. In addition, Figure 4(a) lists the processing steps in the percentage of total time, and Figure 4(b) shows CPU IOwait throughout the pipeline. Our main observations include the following.

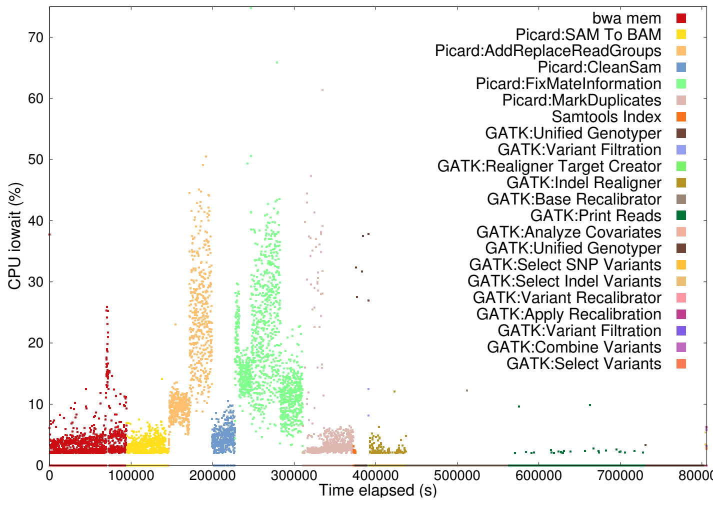
First, the pipeline takes 12.2 days (293 hours) to complete for a single genome with 64GB RAM and 12 cores. We highlight some steps with running time over 10 hours in Table 2, which are also shown in descending order of time in Figure 4(a). The cost may increase significantly if we add complex algorithms that themselves take days to weeks to run.

Second, the current bioinformatics pipeline is extremely inefficient with respect to I/O. (1) The read set of hundreds of GB per sample is not reduced by processing, but rather updated through many early steps in the analysis pipeline, as shown in the "Input" and "Output" columns of Table 2. This is because the reads need to be updated with the mapped locations, adjusted quality scores, flags, etc., but not aggregated in most steps, and additional processed data can be added. (2) There is a tremendous amount of data transfer between the many steps in a pipeline. Since all the data is stored in files currently, in each processing step an entire file is read and rewritten even when updates to small parts of data may suffice. (3) Different steps in the pipeline may access data in different orders. For instance, the step "AddorRplGroups" sorts all the reads by the mapped location, while the step "FixMateInfo" first sorts data by the read id for its own processing and then sorts data back by the mapped location. Such sorting is needed to prepare data in the right order for the next step. The combination of large files read and written in each step and a number of full sorts in the pipeline lead to significant I/O costs, as shown in the "Bytes (R/W) column" of the table.

Third, some of the processing steps are CPU intensive, as shown by the number of CPU instructions in the last column of the table. These operations include: *BWA alignment step*, which uses the Burrows-Wheeler transform and dynamic programming to match each read to the position(s) in the reference genome; *Quality score recalibration steps* (Base Recalibrator and Print Reads), which uses grouping and statistical computation over all the data to find the empirical quality score for the bases of reads; *Genotyping*, which looks at overlapping reads at each base of the reference and calculates likelihood of SNPs and small INDELs; and *BAM to GASV*, which runs geometric operations to identify



(a) Breakdown of total running time of 293 hours (12.2 days) on a whole genome sample (282GB, compressed). The piechart lists the processing steps in decreasing order the percentage of time taken. Numbers were obtained from a sever with 64GB RAM, 12 Intel Xeon 2.40GHz cores.



(b) CPU IOwait in processing steps throughout the pipeline.

Figure 4: Profiling results of pipeline execution on a whole genome dataset.

Step	Time (hours)	Input (in bytes)	Output (in bytes)	Bytes (R/W)	Instructions
Bwa mem	**26.26**	FASTQ (6.04E+11), Reference (3.15E+09)	SAM$_1$ (7.83E+11)	1.393E+12	2.955E+15
Picard: SAM To BAM	**14.31**	SAM$_1$	BAM$_1$ (2.36E+11)	1.019E+12	1.880E+14
Picard: AddOrRplGroups	**14.83**	BAM$_1$	BAM$_2^*$ (1.57E+11)	1.013E+12	1.814E+14
Picard: CleanSam	8.52	BAM$_2$	BAM$_3$	3.11E+11	1.120E+14
Picard: FixMateInfo	**23.07**	BAM$_3$	BAM$_4^*$ (1.61E+11)	1.581E+12	2.376E+14
Picard: MarkDuplicates	**14.45**	BAM$_4$	BAM$_5$ (1.63E+11)	6.374E+11	2.407E+14
Samtools Index	0.74	BAM$_5$	BAI$_1$ (9.02E+06)	1.555E+11	9.809E+12
Gatk: Unified Genotyper	4.53	BAM$_5$, Ref.	VCF$_1$ (1.09E+09)	1.651E+11	4.332E+14
Gatk: Variant Filtration	0.05	Reference, VCF$_1$	VCF$_2$ (1.12E+09)	1.972E+09	5.652E+11
Gatk: Realigner Target	0.37	Reference, VCF$_2$	Intervals$_1$ (1.42E+07)	3.802E+08	7.935E+12
Gatk: Indel Realigner	**12.93**	BAM$_5$, Ref., Intervals$_1$	BAM$_6$ (1.63E+11)	3.174E+11	1.882E+14
Gatk: Base Recalibrator	**34.83**	BAM$_6$, Ref., DBsnp$_1$	Recal$_1$ (9.02E+05)	1.703E+11	7.506E+14
Gatk: Print Reads	**46.57**	BAM$_6$, Ref., Recal$_1$	BAM$_7$ (3.29E+11)	4.929E+11	9.400E+14
Gatk: Unified Genotyper	**20.37**	BAM$_7$, Ref.	VCF$_3$ (1.03E+09)	3.302E+11	2.777E+15
Gatk: Select Variants	0.03	Ref., VCF$_3$	VCF$_4$ (8.82E+08)	8.144E+08	4.028E+11
BAM To GASV	**65.25**	BAM$_7$	PR_disc. (2.174E+09), PR_conc. (2.076E+10)	2.302E+11	1.406E+15
GASV/Pro	6.05	PR_disc., PR_conc.	SV, Coverage	2.117E+10	9.655E+13

Table 2: Performance measurements of a pipeline for variant calling. (* indicates a sorting step.)

concordant and discordant reads needed for structural variant discovery. Among them, the BWA aligner and genotyper were run with 12 threads, with reduced running time.

Finally, by combining the measurements in Table 2 and in Figure 4(b), we observe that the expensive steps in the pipeline roughly fall into two classes: (1) One class of long running steps are both CPU intensive (due to the algorithm used) and I/O intensive (due to large files read and written). Examples are the processing steps that run expensive algorithms on large data files, as mentioned above. For such steps, parallel execution with a large number of nodes will help reduce the running time, which we will discuss in §3.1. (2) Another class of long running steps involve large files and often a full sort, but are less CPU intensive in the analysis. These steps mostly perform data cleaning and quality control, such as AddRplReadGroups, FixMatesInfo, and MarkDuplicates. This set of steps exhibit high measurements of CPU IOwait, as shown in Figure 4(b), as well as of high disk queue lengths. To fix these I/O issues, we need to replace the current use of large files for data communication between processing steps and reduce sorting overheads, which we discuss in §3.2 and §3.3.

We also ran similar variant calling pipelines at New York Genome Center (NYGC) using five nodes, each with 20 dual-core 2.80GHz Intel Xeon processor and 256 GB RAM. Time measurements and profiling results on IO and CPU costs confirmed our observations above: The most expensive steps both CPU-intensive, due to expensive algorithms used, and I/O intensive, due to large files read and written; there are also a number of disk-based sorting steps (using NovoSort) of large read files. The NYGC team also ran special cancer analysis pipelines, involving complex algorithms that take long to run. Such long running algorithms include Mutect [4] for somatic variant calling and Theta [33] for complex cancer genome analysis, to name a few.

3. PARALLELIZATION AND OPTIMIZATION

In this section, we present our efforts to develop an optimized parallel processing platform for executing genomic pipelines using a cluster of compute nodes. Before delving into the details, we first articulate the key design criteria for parallelization and optimization of genomic pipelines, which were suggested by colleagues at the New York Genome Center. These design criteria fundamentally distinguish our work from other recent work that takes different approaches to parallelizing and optimizing genomic pipelines [19, 28, 39].

Complexity. *Regarding parallelism, bioinformatics pipelines can be far more complex than is often realized.* Such pipelines often consist of dozens of steps: (1) Some of the steps are not embarrassingly parallel: chunking the genome in different ways results in different results. As an example, the BWA aligner [22] is not completely locus-specific. If the genome is chunked in different ways, we can get different results, primarily because decisions about the best placement of reads with multiple mappings depends, in part, on where other reads are placed. (2) Some steps require that many genomes be processed together, rather than one at a time. Some steps, including the widely used GATK single nucleotide variant caller [10], process many genomes at once. The latest version suggests that joint genotyping be performed on all genomes in a project together, which can include thousands of genomes. That process requires splitting the genomes into small regions, and processing a single region of all genomes together. Co-processing subsets of the genomes together, and combining the results, produces different results. The co-processing eliminates false positives and provides verification for variant calls with insufficient quality or depth. Some structural variant callers like GenomeStrip [40], do the same.

Analysis Methods. *The infrastructure for parallelizing and optimizing these pipelines must permit any analysis methods that a scientist needs.* Given the complexity of the genomic pipelines, one may wonder whether it is a good idea to build new methods that are more amenable to parallelization. The problem with this approach is that almost all of existing analysis methods are probabilistic (i.e., there is no known "right" answer), and numerous heuristics are often embedded in these algorithms to handle idiosyncrasies of genome sequence found over time. As a result, no two aligners produce the same results. Variant callers, and particularly somatic variant callers and structural variant callers provide even less consistency. To overcome these issues, the common practice as used in the New York Genome Center is to run multiple somatic variant callers and structural variant callers on every cancer sample, compare them, and then use custom filtering methods to combine the results. In a concrete instance [16], after a deletion had been found in RNA from 10 patients, 3 different structural variant callers were required to validate the deletion in the 10 DNA genome samples. No two of those callers found all 10. It is therefore very important to scientists to use the methods they most trust. Substituting one variant calling method for another is not a simple matter: for many scientists, that requires months of validation and the results are often not acceptable. Thus, optimization approaches that require building new analysis methods are not considered acceptable to many scientists and genome research facilities.

Data Formats. A related question is whether we can improve the data formats to better enable parallelism and optimization. A single whole genome sequence file, at sufficient sequencing depth for cancer analysis, is often 500GB. In some formats, it is standard for such files to be over a TB, compressed. There is no question that these formats are inefficient, in both storage and processing capabilities. There are standards groups working on improvements to these formats (e.g., [11]). In many cases, from a purely technical viewpoint, databases would be better. But there are many standard methods in use throughout the bioinformatics community, and changing the pervasive data format standards requires changing all of those analysis methods that scientists depend on to get the best results available. *Until such time as new formats are agreed upon, developing new methods that use new formats is unlikely to gain adoption in the bioinformatics community.*

Effect of Hardware Advance. As CPU and memory capacities improve as per Moore's law, we would like to improve performance of the pipelines by utilizing all the cores and keeping as much data as possible in memory. However, as described above, there are very standard methods that require the analysis of hundreds to thousands of genomes together. They can appear as one step in the middle of a long analysis pipeline, in which some of the steps use one genome at a time, and yet other steps use multiple genomes at a time, but not chunked in the same ways, or not using the same set of genomes. These characteristics make in-memory processing models suitable only for reducing I/O in "local" parts of the computation, but unlikely to be applicable to full genomics pipelines. *A general infrastructure for genomic pipelines still needs to be designed with full capacity to process most data to and from disks.*

3.1 Parallelizing the Pipeline

We next discuss how we parallelize our pipeline over a cluster of nodes. This will improve performance, especially of those CPU and I/O intensive steps as described above. In this work, we choose to use the MapReduce (MR) cluster computing model, in particular, the open source Hadoop system [13]. This is because MR systems free the user from worrying about system-level details, such as partitioning, scheduling, and fault tolerance, while having proven success to scale to tens of thousands of cores. In contrast, existing genomic processing systems often require user manual work to do so: NCGAS [30] requires the user to parallelize his method using the MPI interface for supercomputers, and GATK [7] requires the user to partition the dataset manually in order to launch GATK instances on different nodes. In addition, Hadoop forms the core of a large ecosystem of open-source analytics products, which provides a zero-cost offering to the bioinformatics community for data analytics.

A New Distributed Storage System. To run existing software on Hadoop, the first issue to resolve is the incompatibility between the file system used in existing software and the distributed file system used in Hadoop. Modifying every existing software tool is not a realistic solution. To address this issue, we have developed a new storage substrate that provides the same data access interface, e.g., PicardTools' API for accessing SAM records, as for the current file system used in existing software, but directs the actual data access to the distributed Hadoop file system (HDFS). This solution uses a byte input stream that is created over HDFS and is passed to the constructor of a SAM record reader that existing analysis methods use.

The second issue is that the common data format for aligned reads, SAM [42], is often used in its binary compressed and indexed format, called BAM. As we place BAM data into the distributed HDFS, it is important that the binary compressed data be partitioned properly without corrupting the content. In our implementation, we deal with a range of issues, including making BAM header information available to all data nodes, handling those compressed BAM blocks that span two Hadoop chunks, and fixing BAM indexes to provide correct mapping from coordinates over the reference genome to the Hadoop chunks that contain read sequences that overlap with those coordinates. While our approach to porting BAM data into HDFS is somewhat similar to HadoopBAM [32], certain technical details differ. More importantly, we prefer to maintain our own implementation so that we have full flexibility to extend this storage substrate to support additional features, including logical partitioning and co-location, as discussed below.

Another important issue is that many processing steps of a genomic pipeline permit parallel processing based on logical data partitioning (which is formally defined shortly), e.g., by chromosome. Such partitioning is not supported by HDFS as chunks physically placed in a data node do not correspond to any logical partition of an input dataset. Our storage substrate provides logical partitioning by ensuring that the HDFS chunks placed in a data node corresponds to a user-defined partitioning criterion. This requires tagging each HDFS chunk with a logical-id and modifying the HDFS placement policy interface to control the placement of those chunks. In addition, our storage substrate can be extended to support co-location [8], e.g., co-locating HDFS chunks from different genome samples that belong to the same logical partitioning, which will be important for performance optimization in many processing steps that involve a (large) number of genome samples.

Finally, it is important to note that our storage substrate does not require the change of the SAM/BAM formats widely used in genomic analysis methods. In addition, the actual change of the programs of analysis methods is minimum.

Parallel Processing on Hadoop. To achieve scalability, a fundamental mechanism that MR systems use is (data) *partitioned parallelism*: input data is split into partitions, and these partitions are processed on different nodes in parallel. In the simplest case, data can always be simply partitioned based on the order in which it is stored, which is referred to as *physical partitioning* of the data. However, many processing steps may require data to be partitioned based on a logical condition that is specific to the analysis algorithm, and then further analysis can be performed in parallel in the logical partitions. This form of data partitioning is referred to as *logical partitioning*. Our parallel processing platform for genomic pipelines supports both forms of parallelism through the MR programming model, by encapsulating analysis software in a series of `map` and `reduce` functions. Then in a shared-nothing architecture, the MR system automatically implements: (1) parallel processing of `map`() on input partitions using a set of map tasks (mappers) in the cluster, which is a form of physically partitioned parallelism; (2) grouping the ⟨key, value⟩ pairs from all map output by key and sending grouped data to reducers; (3) parallel processing of `reduce`() on different groups using a set of reducers, which is a form of logically partitioned parallelism.

A key challenge in parallelizing a genomic pipeline is that most processing steps require logical partitioning of data. First of all, there is a wide range of opportunities to explore for parallelizing a genomic pipeline based on data partitioning, as summarized in Figure 5. Regarding the sequencing process, a run of a sequencer can produce billions of reads

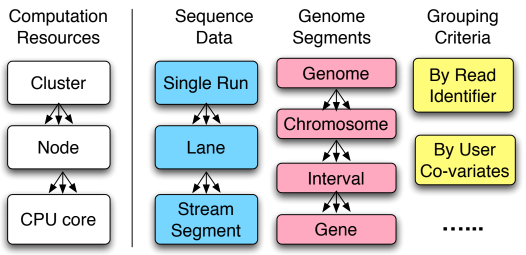


Figure 5: Opportunities for partitioned parallelism.

of multiple genome samples. The full set of data from a single run can be further partitioned into subsets according to the number of lanes (e.g., 96) used in sequencing. The per-lane data set can be further partitioned into steam segments. When the sequencing data is later separated for each genome sample and further aligned to the reference genome, we can also partition the per-genome data based on biological concepts, such as chromosomes, large intervals within a chromosome, and specific genes. Some analysis algorithms also have specific grouping requirements, e.g., group by read id in FixMateInfo, and group by user-defined covariates in BaseRecalibrator.

A second challenge is to determine the level (degree) of parallelism that each processing step should use. A first related question to ask is which level of parallelism is safe for a processing step, i.e., producing the same output as single-node execution. To answer this question, we are currently analyzing the most important steps used in our pipelines, by reading the algorithms described in the bioinformatics papers and consulting domain experts. This process can be expedited later if when publishing a new analysis method, a bioinformatician is asked to annotate the method with the finest granularity of partitioning that is deemed safe. Once we understand the safe way to partition a dataset, e.g., at the genome-level, we actually obtain a hierarchy of possible ways to partition data, e.g., by chromosomes, by large intervals within a chromosome, or by genes. Which level of parallelism to choose is another important question to answer. For maximum degree of parallelism within a step, we would like to choose the smallest granularity of partitioning. However, if we examine multiple steps within a pipeline, we may prefer to choose a larger granularity that is common to multiple consecutive steps, which will avoid data shuffling between steps in order to partition data in different ways. Hence, there is an optimization issue that we need to address (which is discussed more in the next section).

Finally, to port the entire genomic processing pipeline into the MR framework, we need to break the pipeline into a series of MR jobs, where each job consists of a `map`() and a `reduce`(). Specifically, our parallel processing platform encodes a number of decisions for the pipeline, including: (1) the number of MR jobs needed, (2) in each job, which subset of attributes from input data serve as the key for logical partitioning, and (3) in each job, which processing steps permit physical partitioning and hence can be encapsulated in `map`(), and which subset of steps require logical partitioning and hence should be encapsulated in `reduce`().

3.2 Techniques for Reducing I/O

As our profiling results show, a number of data cleaning and quality control steps are I/O intensive. Paralleling these steps using more nodes will not resolve the issue that the

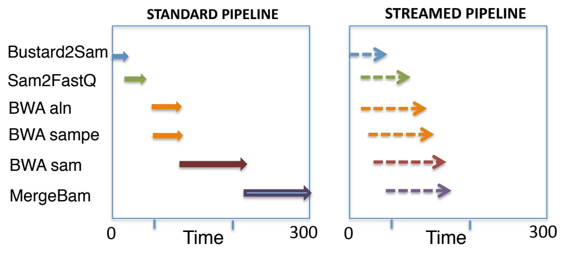


Figure 6: Visualization of compute progress in the standard pipeline (299 sec in total) vs. the streamed pipeline (163 sec).

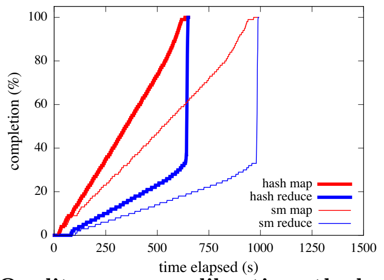


Figure 7: Quality score recalibration: the hash approach outperforms sort-merge for logical partitioning.

CPUs are mostly idle due to outstanding I/O activities. In our project, we explore a number of techniques to effectively reduce I/O.

Streaming through the Pipeline. Our first effort focuses on streaming data through a pipeline as a means of reducing I/O as well as completion time for the pipeline, without requiring alteration of existing software. We have implemented simple streaming protocols that send the output from one step directly to the servers running the next step, using Unix named pipes. Our initial tests, run on the first 5 steps of the pipeline, as depicted in Figure 6, showed an almost 50% decrease in latency, with almost no impact on throughput when the servers were not fully loaded. In addition, streaming allows different steps to run concurrently: as step 1 produces results, step 2 can start to make progress; the output of step 2 can trigger the processing of step 3, and so on—this is called *pipelined parallelism*, another form of parallelism our platform supports. Currently, we are building a prototype of a streaming scheduler that allows for pipelined parallelism throughout our analysis pipeline. An initial version is available online [12]. However, the current scheduler does not optimize for CPU utilization and other factors, which we plan to continue to work on in this project. We will further address issues such as controlling the rates of streaming from different steps.

Using A Column Store. The second effort is to a distributed, column-based storage system to replace the existing file system for transferring data between processing steps. To understand the benefits of a column store, consider a SAM file for aligned reads [42]. In the file, a collection of SAM records are stored consecutively, and each record for an aligned read contains 11 mandatory fields, including the read sequence, the quality scores of its bases, the aligned position, etc. In a column store, each attribute (or each disjoint partition of attributes, called a column family) of all the SAM records are stored together in a separate file. When a program needs to access only a subset of attributes of the SAM records, the column store has an efficient way (based on merge-sort) to combine the files to serve the data to the program. Of course, the exact savings depend on the number of attributes read and updated in each processing step of the pipeline. This change of storage architecture will provide benefits including reduced I/O, and effective compression. We are currently experimenting with a range of columnar storage choices, including HBase [15] and Parquet [35], that are available in the Hadoop family of software tools.

3.3 Pipeline Optimization

A main part of our research is to optimize genomic processing pipelines on our parallel processing platform. We propose to bring the principled approach to query optimization from the database area to bear on the process of pipeline optimization. In particular, our efforts include:

1. *Enhance Hadoop with alternative implementations of partitioned parallelism and choose the best one for each processing step.* Our prior work [21] showed that logical partitioning, i.e., "grouping data by key", is an expensive operation because it requires re-arranging all the data in the cluster into logical groups and distributing those groups to various reducers. Most MR systems, including Google's MR system and Hadoop use a *sort-merge* approach to the grouping operation. This sort-merge approach can incur high CPU and I/O overheads in the grouping operation, especially if the analysis in `reduce`() does not require data in sorted order. An example in our pipeline is quality score recalibration, which requires logical data partitioning but not data sorting.

To avoid unnecessary overheads, we have implemented an alternative, hash-based approach to logical partitioning in Hadoop [21]. This approach uses a series of hash-based algorithms in mappers and reducers to guide data shuffling to reducers and recognition of different groups of data in the reducers. Our initial effort to apply this hash approach to quality score recalibration showed 50% reduction of running time and dramatic reduction of I/O cost, as depicted in Figure 7. In current work, we are examining all processing steps in our pipeline, and choose the appropriate implementation between hashing and sort-merge based on the data partitioning and order requirements of those steps.

2. *Explore across-step optimization in the pipeline.* Our profiling results in §2.2 showed that different data access patterns used in different steps trigger global re-arrangement of data. If we can group several processing steps that share the same data access pattern, e.g., accessing reads in order of the aligned position, we can minimize the frequency of data re-arrangement through the pipeline. In our work, we examine the interaction among steps and re-group them when possible, to share data access patterns across steps—this is similar to re-ordering operators based on commutativity in relational databases. Further, we observe that although some processing steps require logical partitioning but not full sorting, if we use the sort-merge approach to logical partitioning, the sorted order will benefit the subsequent step. An example is MarkDuplicates that does not require sorting, but sorting all the reads based on the aligned position not only supports its required logical partitioning but also offers the right data access order for subsequent genotyping—this effect is similar to query optimization based on "interesting orders". We explore all for these opportunities in optimization.

3. *Address Degrees of Parallelism.* As mentioned in Section 3.1, it is challenging is to determine the best level (de-

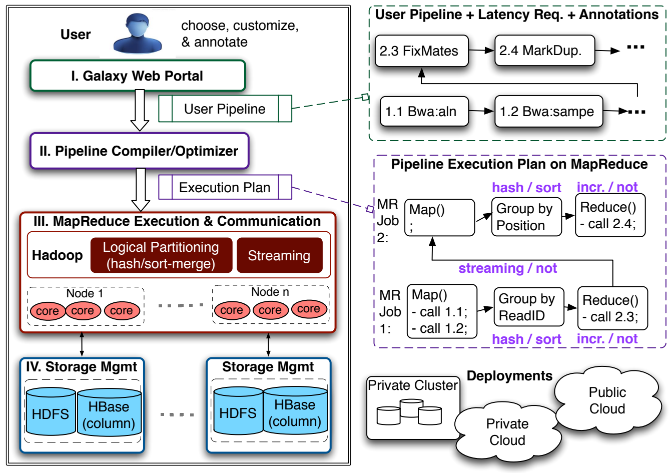


Figure 8: System architecture with a parallel processing backend.

gree) of parallelism that each processing step should use. For maximum degree of parallelism within a step, it may be better to choose the finest granularity of partitioning that an analysis method permits. However, if the next processing step employs a different level of parallelism, data needs to be repartitioned and shuffled over the cluster to prepare for the next step. Sometimes, the overhead of such data shuffling can outweigh the performance benefit that the previous step gained. In this case, if we examine multiple steps within a pipeline, it may be better to choose a larger granularity that is common to multiple consecutive steps, which will avoid data shuffling between steps in order to partition data in different ways. How to make appropriate choices for a long pipeline of processing steps poses an optimization problem. The approach that we are exploring is cost-based optimization, where we model the performance gains of each step using a particular level of parallelism, as well as the cost of data shuffling if the next step uses a different level of parallelism. Then cost-based optimization can suggest the best plan (or at least avoid the bad plans) for choosing the level of parallelism for each processing step.

4. *Explore materialized views.* Materialized views are another important mechanism for supporting different data access patterns. For instance, if the genome dataset is first sorted by the mapped location, then grouped by read id, and finally sorted by the mapped location again, it may be beneficial to store the the first sorted file as a materialized view and reuse it when such sorted order is required again. The difficulty lies in the fact that the processing step in between may have updated some attribute of a subset of records, and hence such updates must be reflected in the materialized view. A more general problem is that different sorting or grouping requirements can be considered as different “views” defined on an input genome dataset, and we would like to have the dataset (including the most recent updates) available in any view that the current processing step takes. This problem is an interesting extension of prior work on the view update problem as it mixes different data models, e.g., the relational model and the sequence model, and different data arrangement requirements. In our work, we will address these issues in the context of column stores.

4. DEVELOPMENT AND USE CASES

We are developing a full-fledged software system that implements our proposed platform and pipeline. Our system employs an architecture shown in Fig. 8. It integrates: (1) the popular Web portal, Galaxy [9], for building genomic data pipelines; (2) a pipeline compiler and optimizer, which transforms the user pipeline into an execution plan, which uses a (minimum) number of MapReduce jobs, and is optimized with the decisions for each step including the key for logical partitioning, degree of parallelism used, implementation of logical partitioning (hash versus sort-merge), and streaming output or not; (3) a Hadoop-based processing platform, which runs the parallel execution plan of the pipeline over a cluster of nodes, and employs extension of Hadoop with a new hash implementation for logical data partitioning, incremental processing, and streaming; (4) a storage system, which includes the Hadoop Distributed File System (HDFS), a columnar storage system such as Hadoop Database (HBase) or Parquet, and our new storage substrate that sits on top of HDFS and HBase/Parquet and supports existing analysis methods with minimum change, as well as additional features such as logical data partitioning and co-location in distributed storage.

Use Cases: Besides internal evaluation, we are building a set of real use cases for deployment and evaluation.

1. *Variant calling for individual samples*: Much of bioinformatics research is associated with the need to identify variants in individual genomes and to compare these variants across individuals. These variants can be SNPs, small INDELs, or large structural variants such as CNV (copy-number variants), inversion, translocation, etc. Our genomic data analysis pipeline will include algorithms for calling most types of variants. Such variant calling is relevant to a large number of biology and bioinformatics research labs.

2. *Analysis method development and comparison*: One of the major functions of the bioinformatics group at NYGC is development of new methods and comparisons of existing analysis methods to determine accuracy, sensitivity, and specificity of each under varying circumstances. In one recent comparison of variant callers, we have less than 30% overlap in calls under some circumstances. Understanding the differences is key to making accurate calls and finding the true causal variants in a disease. One of our current projects is to use this analysis to build a method that uses calls from multiple methods and combines their results to improve accuracy. One major hurdle we face is that some of those initial methods take over two weeks to run on one genome. If every iteration of an experiment takes two weeks, getting to the final results seems interminable. The algorithms are not embarrassingly parallel, hence not amenable to simple partitioning. The new techniques in our system for parallelizing these methods and pipelines will be applied to support this research.

3. *Population studies*: In population studies, large numbers of genomes or exomes are analyzed to identify a “reference genome” for a population, that is, to identify variants in the genome that are more common or less common in this population than in others. NYGC recently processed over 600 whole exomes from one population. The scientists running the study had a deadline of 45 days to complete the analysis, from initial DNA samples through variant calling, requiring both high-throughput and reasonably fast turnaround times to accomplish. Furthermore, often in large studies such as this, some initial sequencing will fail, requir-

ing a second round of sequencing. The faster that early stages of analysis can detect problems, the faster re-sequencing can be initiated, impacting final delivery of results. We will evaluate our parallel processing for both throughput and response time. When a user deadline is coming up, we will also use streaming to optimize for latency of his results.

4. *Common disease studies*: Our last use case shows the broader application of our work beyond basic biology research. Finding the variants that lead to susceptibility to common diseases such as diabetes, autism, and Alzheimer's disease can require studies of thousands or even tens of thousands of samples. Variant calling and structural variant calling phases of analysis can require that hundreds of samples be processed together, leading to very large data sets in analysis. Currently the leading variant callers take days to complete analysis of just one step on one hundred genomes. End-to-end analysis from raw data through variant calls can take weeks, and the need to analyze the samples together can make simple partitioning of data untenable. We can evaluate our system using batches of genome data and report in both throughput and response time of each batch of data.

5. RELATED WORK

National projects on genomic data processing. Several large projects for developing genomic processing software take different approaches or have different focuses from ours. The National Center For Genome Analysis Support (NCGAS) [30] focuses on genome-level assembly and analysis software running on supercomputers. Our approach differs fundamentally as (1) we support a deep pipeline, not just individual tools like assembly, and (2) our underlying processing infrastructure is the "big data" infrastructure, which uses a large cluster of hundreds of nodes with commodity hardware and open-source software that deals automatically with parallelization and load balancing. In contrast, for most analysis methods, NCGAS requires users to parallelize using the MPI programming interface. Galaxy [9] is an open, web-based platform for biomedical research. However, its processing backend is merely an integration of existing software tools with limited innovation for high performance, scalability, or low latency.

Parallel genomic data processing: In earlier work, Simpson et al. [43] used ABySS to assemble a human genome using a cluster of 168 cores (21 computers), in about 3 days of wall clock time, and Jackson et al. [17] assembled a Drosophila melanogaster genome from simulated short reads on a 512-node BlueGene/ L supercomputer in less than 4 hours of total elapsed time. These efforts require access to a specific type of hardware resource, hence not widely applicable. More recent work explored MapReduce and in particular, Hadoop for parallel genomic data processing. Crossbow [19] implements a parallel pipeline for alignment and SNP detection using Hadoop. However, it supports only two specific algorithms, Bowtie for alignment and SOAPsnp for SNP calling. Furthermore, its implementation requires some modification of existing software and avoids porting SAM/BAM data properly into HDFS, hence very inefficient regarding I/O. Similarly, Seal [39] supports three specific processing steps, alignment using BWA, mark duplicates, and base quality score recalibration, using significantly modified code and without porting data into HDFS. In comparison, our system offers a general parallel framework for supporting many other processing steps, a full storage system based on HDFS, as well as new optimizations. GATK [7, 29] supports the MapReduce interface but not distributed parallelism. It can parallelize within a single multi-threading process, or asks the user to *manually* divide data based on the chromosome and then run multiple GATK instances. Hadoop-BAM [32] provides access to reads in binary, compressed BAM format stored in HDFS, which is similar to our HDFS-based storage, but without advanced features such as logical partitioning and co-location. ADAM [28] provides a set of formats, APIs, and processing stage implementations for genomic data. The implementation uses Avro for explicit data schema access, Parquet for columnar storage, and Spark for in-memory processing of genomic data. While the initial results are promising, ADAM requires reimplementing genomic analysis methods using Scala and RDD's (in-memory fault-tolerant data structures used in Spark). As discussed in §3, our collaboration with the New York Genome Center results in fundamentally different design criteria: we aim to provide a general parallel processing framework that can take any analysis method that a scientist provides, with no or minimum change of the data format and program used.

Cloud computing for genomic data processing: Most of published work on porting genome analysis to the cloud has focused on single analysis tools, e.g., alignment [19]. In prior work, a team at the Broad Institute ported into the Amazon cloud individual steps in the Picard primary pipeline [38], the GATK unified genotyper, and GenomeStrip structure variant caller, but not a full pipeline. The study discovered that porting entire pipelines introduced numerous challenges not previously identified, including frequent data shuffling due to different data access and partitioning requirements in various steps, as well as effects of virtualization and shared resources. We plan to address these issues in this project.

Big data analytics using MapReduce (e.g., [3,34,36]) has been intensively studied lately. This line of work differs from ours in several key aspects: (1) It does not focus on the specific data model and computing algorithms for genomic data analysis. (2) It usually deals with a single task, such as a query computing aggregates or a data mining algorithm, but not a full pipeline of steps with different data processing needs in terms of CPU resources needed, data access patterns, and types of parallelism permitted. Hence, our project is addressing a new problem in the MapReduce framework, for the specific genomic data type and processing needs.

6. CONCLUSIONS

In this paper we presented the initial design of a test genomic processing pipeline, performance measurements that reveal bottlenecks in current pipeline execution, and a general Hadoop-based parallel platform for pipeline execution and optimization. We also shared fundamental design criteria when parallelizing large genomic data pipelines, and highlighted the key research questions to consider in parallelization and optimization. We finally presented some initial results on parallel execution and optimization of a test pipeline, as well as a number of real use cases that we are currently developing for deployment and evaluation.

7. REFERENCES

[1] C. Alkan, B. P. Coe, and E. E. Eichler. Genome structural variation discovery and genotyping. *Nature reviews. Genetics*, 12(5):363–376, May 2011.

[2] M. Baker. Next-generation sequencing: adjusting to data overload. *Nature Method*, 7(7):495–499, 2010.

[3] B. Chattopadhyay, L. Lin, W. Liu, et al. Tenzing a SQL implementation on the MapReduce framework. *PVLDB*, 4:1318–1327, 2011.

[4] K. Cibulskis, M. S. Lawrence, S. L. Carter, et al. Sensitive detection of somatic point mutations in impure and heterogeneous cancer samples. *Nature Biotechnology*, 31(3):213–219, 2013.

[5] P. J. A. Cock, C. J. Fields, et al. The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic Acids Research*, 38(6):1767–1771, Apr. 2010.

[6] T. I. H. Consortium. The international hapmap project. *Nature*, 426:789–796, 2003.

[7] M. A. DePristo, E. Banks, R. Poplin, et al. A framework for variation discovery and genotyping using next-generation dna sequencing data. *Nat Genet*, 43(5):491–498, 2011.

[8] M. Y. Eltabakh, Y. Tian, F. Özcan, et al. Cohadoop: Flexible data placement and its exploitation in hadoop. *Proc. VLDB Endow.*, 4(9):575–585, June 2011.

[9] Galaxy: An open, web-based platform for data intensive biomedical research. `https://main.g2.bx.psu.edu/`.

[10] Best practice variant detection with gatk. `http://http://www.broadinstitute.org/gatk/`.

[11] The global alliance for genomics and health. `http://genomicsandhealth.org`.

[12] Pipeline execution manager with streaming. `https://github.com/nnovod/PEMstr`.

[13] Hadoop: Open-source implementation of mapreduce. `http://hadoop.apache.org`.

[14] J. Han and J. Pei. Mining frequent patterns by pattern-growth: Methodology and implications. *SIGKDD Explorations*, 2(2):14–20, 2000.

[15] Hadoop database (hbase): A distributed, column-oriented data store. `http://hadoop.apache.org`.

[16] J. N. Honeyman, E. P. Simon, N. Robine, et al. Detection of a recurrent dnajb1-prkaca chimeric transcript in fibrolamellar hepatocellular carcinoma. *Science*, 343:1010–1014, 2014.

[17] B. Jackson, P. Schnable, and S. Aluru. Assembly of large genomes from paired short reads. In *Proceedings of the 1st International Conference on Bioinformatics and Computational Biology*, pages 30–43. Springer-Verlag, 2009.

[18] M. Kasahara and S. Morishita. *Large-scale Genome Sequence Processing*. Imperial College Press, 2006.

[19] B. Langmead, M. Schatz, J. Lin, et al. Searching for SNPs with cloud computing. *Genome Biology*, 10(11):R134+, Nov. 2009.

[20] B. Langmead, C. Trapnell, M. Pop, et l. Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome biology*, 10(3), 2009. R25.

[21] B. Li, E. Mazur, Y. Diao, A. McGregor, and P. J. Shenoy. A platform for scalable one-pass analytics using mapreduce. In *SIGMOD Conference*, pages 985–996, 2011.

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

[23] H. Li and R. Durbin. Fast and accurate long-read alignment with burrows-wheeler transform. *Bioinformatics*, 26(5):589–595, 2010.

[24] J. Li, A. W.-C. Fu, and P. Fahey. Efficient discovery of risk patterns in medical data. *Artificial Intelligence in Medicine*, 45(1):77–89, 2009.

[25] R. Li, Y. Li, X. Fang, H. Yang, et al. SNP detection for massively parallel whole-genome resequencing. *Genome Research*, 19(6):1124–1132, June 2009.

[26] Y. Li, A. Terrell, and J. M. Patel. Wham: a high-throughput sequence alignment method. In *SIGMOD Conference*, pages 445–456, 2011.

[27] K. Lindblad-Toh, M. Garber, and O. Z. et al. A high-resolution map of human evolutionary constraint using 29 mammals. *Nature*, 478(7370):476–482, Oct. 2011.

[28] M. Massie, F. Nothaft, C. Hartl, et al. Adam: Genomics formats and processing patterns for cloud scale computing. Technical Report UCB/EECS-2013-207, UC Berkeley, 2013.

[29] A. McKenna, M. Hanna, E. Banks, et al. The genome analysis toolkit: A mapreduce framework for analyzing next-generation dna sequencing data. *Genome Research*, 20(9):1297–1303, 2010.

[30] National center for genome analysis support. `http://ncgas.org/`.

[31] T. C. G. A. R. Network. Comprehensive genomic characterization defines human glioblastoma genes and core pathways. *Nature*, 455(7216):1061–1068, Sept. 2008.

[32] M. Niemenmaa, A. Kallio, A. Schumacher, et al. Hadoop-BAM: directly manipulating next generation sequencing data in the cloud. *Bioinformatics (Oxford, England)*, 28(6):876–877, Mar. 2012.

[33] L. Oesper, A. Mahmoody, and B. Raphael. Theta: Inferring intra-tumor heterogeneity from high-throughput dna sequencing data. *Genome Biology*, 14(7):R80, 2013.

[34] C. Olston, B. Reed, U. Srivastava, et al. Pig latin: a not-so-foreign language for data processing. In *SIGMOD Conference*, pages 1099–1110, 2008.

[35] Parquet: a columnar storage format for the hadoop ecosystem. `http://parquet.incubator.apache.org`.

[36] A. Pavlo, E. Paulson, A. Rasin, et al. A comparison of approaches to large-scale data analysis. In *SIGMOD Conference*, pages 165–178, 2009.

[37] P. A. Pevzner, H. Tang, and M. S. Waterman. An eulerian path approach to dna fragment assembly. *Proceedings of the National Academy of Sciences*, 98(17):9748–9753, 2001.

[38] Picard tools: Java-based command-line utilities for manipulating sam files. `http://picard.sourceforge.net/`.

[39] L. Pireddu, S. Leo, and G. Zanetti. Mapreducing a genomic sequencing workflow. In *Proceedings of the 2nd Int'l workshop on MapReduce and its applications*, MapReduce '11, pages 67–74, 2011. ACM.

[40] H. RE, K. JM, N. J, and M. SA. Discovery and genotyping of genome structural polymorphism by sequencing on a population scale. *Nature genetics*, 43:269–276, 2011.

[41] A. Roy, Y. Diao, E. Mauceli, et al. Massive genomic data processing and deep analysis. *PVLDB*, 5(12):1906–1909, 2012.

[42] Sam: a generic format for storing large nucleotide sequence alignments. `http://samtools.sourceforge.net/`.

[43] J. Simpson, K. Wong, S. Jackman, et al. Abyss: a parallel assembler for short read sequence data. *Genome Research*, 19:1117–1123, 2009.

[44] S. S. Sindi, E. Helman, A. Bashir, and B. J. Raphael. A geometric approach for classification and comparison of structural variants. *Bioinformatics*, 25(12), 2009.

[45] S. S. Sindi, S. Onal, L. Peng, et al. An integrative probabilistic model for identification of structural variation in sequence data. *Genome Biology*, 13(3), 2012.

[46] R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. In *SIGMOD Conference*, pages 1–12, 1996.

[47] Variant call format. `http://vcftools.sourceforge.net/specs.html`.

[48] R. Xi, A. G. Hadjipanayis, et al. Copy number variation detection in whole-genome sequencing data using the Bayesian information criterion. *Proceedings of the National Academy of Sciences*, 108(46):E1128–E1136, Nov. 2011.

[49] D. R. Zerbino and E. Birney. Velvet: algorithms for de novo short read assembly using de bruijn graphs. *Genome Res.*, 18(5):821–9, 2008.