Persona: A High-Performance Bioinformatics Framework

Stuart Byma and Sam Whitlock, *EPFL;* Laura Flueratoru, *University Politehnica of Bucharest;*
Ethan Tseng, *CMU;* Christos Kozyrakis, *Stanford University;*
Edouard Bugnion and James Larus, *EPFL*

https://www.usenix.org/conference/atc17/technical-sessions/presentation/byma

This paper is included in the Proceedings of the
2017 USENIX Annual Technical Conference (USENIX ATC '17).

July 12–14, 2017 • Santa Clara, CA, USA

ISBN 978-1-931971-38-6

Persona: A High-Performance Bioinformatics Framework

Stuart Byma* Sam Whitlock* Laura Flueratoru† Ethan Tseng‡
Christos Kozyrakis§ Edouard Bugnion* James Larus*

Abstract

Next-generation genome sequencing technology has reached a point at which it is becoming cost-effective to sequence all patients. Biobanks and researchers are faced with an oncoming deluge of genomic data, whose processing requires new and scalable bioinformatics architectures and systems. Processing raw genetic sequence data is computationally expensive and datasets are large. Current software systems can require many hours to process a single genome and generally run only on a single computer. Common file formats are monolithic and row-oriented, a barrier to distributed computation.

To address these challenges, we built Persona, a cluster-scale, high-throughput bioinformatics framework. Persona currently supports paired-read alignment, sorting, and duplicate marking using well-known algorithms and techniques. Persona can significantly reduce end-to-end processing times for bioinformatics computations. A new *Aggregate Genomic Data* (AGD) format unifies sample data and analysis results, while enabling efficient distributed computation and I/O.

In a case study on sequence alignment, Persona sustains 1.353 gigabases aligned per second with 101 base pair reads on a 32-node cluster and can align a full genome in $\sim$16.7 seconds using the SNAP algorithm. Our results demonstrate that: (1) alignment computation with Persona scales linearly across servers with no measurable completion-time imbalance and negligible framework overheads; (2) on a single server, sorting with Persona and AGD is up to 2.3$\times$ faster than commonly used tools, while duplicate marking is 3$\times$ faster; (3) with AGD, a 7 node COTS network storage system can service up to 60 alignment compute nodes; (4) server cost dominates for a balanced system running Persona, while long-term data storage dwarfs the cost of computation.

*EPFL
†U. Politehnica of Bucharest *(work done during EPFL internship)*
‡Carnegie Mellon University *(work done during EPFL internship)*
§Stanford University

1 Introduction

In 2001, the approximate cost of sequencing a whole human genome was $100 million. In 2017, the cost of Whole Genome Sequencing (WGS) is rapidly approaching $100 [25], a faster-than-Moore's Law improvement. Low-cost sequencing is a key enabler of personalized medicine, which tailors treatments for patients to their genetic makeup, promising better diagnoses and more effective therapies.

The genomic data produced by modern sequencing machines, however, is unusable in its raw form. A large amount of pre-processing must be done before analysis. Depending on the sequencing parameters, raw data for one human cell genome can range from several gigabytes to hundreds of gigabytes. The data analysis and storage problems are already challenging and will continue to grow with the increasing ambition of doctors and researchers to sequence more humans and other organisms.

WGS processing consists of a number of steps, including read alignment (matching short snippets of genomic data against a known reference), sorting, indexing, duplicate marking and variant calling (determining where a patient has mutations/differences in their genome). For a typical human genome, processing reads and writes tens to hundreds of gigabytes of data and can require many hours with current tools. Computational costs were minor when sequencing was rare and expensive. However, as sequencing becomes an integral part of medical diagnosis and treatment, fast and efficient processing is invaluable for timely diagnosis and treatment.

Many existing tools run in parallel on a single multicore computer but are not designed to scale to server clusters or cloud computing (though there are significant efforts in this direction; see §7). A crucial challenge in scaling is that genomic data is stored in multiple file formats, none of which are appropriate for parallel or distributed computation. Sequencing machines produce raw genomic data in one file format (FASTQ [8]) while aligned data uses a different format (SAM/BAM [31]),

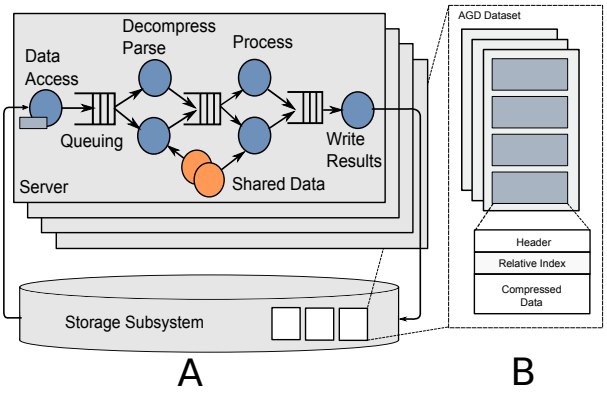


Figure 1: **A.**Persona architecture. Processing genomic data across multiple servers using a distributed dataflow framework. **B.**The Aggregate Genetic Data format stores data in columns to facilitate distributed processing.

which results in data duplication. Downstream analysis produces more files with different formats. In addition, common file formats are mainly row-oriented, which precludes efficient field access and frustrates data partitioning.

This state of affairs requires a new computing architecture to deal with the coming deluge of genomic data. We need a software architecture that runs effectively across computers ranging from a single machine to a cluster, so that genomic data processing can be performed in environments ranging from doctors' offices to hospitals and regional "gene banks".

To accomplish this, we require appropriate file formats that enable: (1) scalable, parallel access from multiple servers; (2) efficient use of both read and write bandwidth; (3) flexibility, to support the multiple phases in a genomics analytics pipeline; Additionally, scaling requires the efficient use of compute resources in terms of throughput and latency, which implies: (1) saturating compute resources of a server at all times, which requires data and task partitioning; (2) when possible, distribute computation across multiple servers; (3) scheduling this work, while avoiding stragglers [10]; (4) overlapping I/O with compute to hide latency.

In this paper, we present Persona, a scalable, high-performance framework for bioinformatics workloads, and the *Aggregate Genomic Data* (AGD) format. Figure 1 shows Persona and AGD at a high level. The goal of Persona and AGD is to provide a complete solution for bioinformatics computations, including (but not limited to) read alignment, sorting, duplicate marking, filtering, and variant calling. A secondary goal is extensibility — both Persona and AGD are designed and implemented in a way that allows straightforward integration of new capabilities (*e.g.,* different alignment algorithms or new data fields). Currently, Persona integrates well-known algorithms from the bioinformatics community, including those from BWA-MEM [30], SNAP [47], Samblaster [14], and samtools [31], so users can be confident in the results produced.

This paper makes the following contributions: (1) To address the limitations of disparate monolithic row-oriented files, the AGD format is a column-oriented file structure designed for compute, storage and I/O bandwidth efficiency, offering selective field and random access, distributed computation support, and unified storage of all genomic data for a given patient; (2) To run efficiently across single computers and moderate-sized clusters, we use distributed dataflow. Persona is built on Google TensorFlow [1], a state-of-the-art distributed dataflow framework. TensorFlow's coarse-grain dataflow minimizes framework overheads, yet, when augmented by a simple fine-grain mechanism, allows efficient use of all CPU resources in a cluster. We show that decoupling I/O granularity from task granularity in read alignment is necessary to maximize I/O bandwidth and balance work on modern multicore architectures; (3) We demonstrate linear scaling to the saturation point of our testbed storage cluster. We perform WGS alignment for a typical dataset in $\sim$16.7 seconds, a near order of magnitude improvement over existing solutions; (4) We demonstrate that the architecture is balanced from a total cost of ownership perspective, with the cost dominated by compute servers. Assuming full occupancy over 5 years, the cost of alignment is as little as 6.07¢. However, the long-term overall costs are likely to be dominated by storage.

Persona, AGD, and benchmarking scripts are freely available [13].

The rest of this paper is organized as follows: §2 provides some background in relevant algorithms and file formats. §3 describes the new AGD format and §4 describes the architecture of Persona. §5 evaluates our solution on a 32-server compute cluster attached to a scale-out storage subsystem. §6 provides some insight into bioinformatics workloads, and analyzes the TCO for different cluster options. Finally, we discuss related work in §7 and conclude in §8.

2 Background

The explosion of interest in and use of genomic data has been made possible by Next-Generation Sequencing (NGS) [6]. NGS machines, through a biochemical process called shotgun sequencing, read a genome by chopping long DNA strands into small pieces and reading these short snippets, which typically consist of 100 to 200 bases (A,T,C,G). The short snippets of a genome are called *reads* and must be *aligned* — reassembled into a full, coherent genome — before further analysis.

2.1 Bioinformatics Computations

Since our case study focuses on alignment, we provide some additional background.

To form a coherent genome, the reads in a raw dataset must be aligned to a reference genome (about 3 billion base pairs for a human). An *aligner* takes an individual read and attempts to find the most likely match in the reference sequence. Insertions, deletions, and mismatches between the bases are allowed, since genomes can have small mutations and the sequencing machines regularly misread base pairs. A read from a sequencing machine consists of three data fields: the bases (A,C,T,G or N, which is an ambiguous base), a quality score for each base indicating the machine's confidence, and metadata uniquely identifying the read. Datasets typically ensure that each base in the sample is overlapped by many reads — this is called *coverage* and is typically 30 to 50×. Raw datasets are typically single-ended, where each read is independent, or paired-ended, where reads are aligned as pairs with some gap between them. Reads are produced in arbitrary order.

Common algorithms for performing alignment include Smith-Waterman [43], an exact, dynamic programming algorithm, and BLAST (Basic Local Alignment Search Tool) [3], which uses seed-and-extend heuristics to locate short common words between sequences and extend them to reach a threshold. These approaches are expensive computationally, especially considering that modern read datasets with 50× coverage can contain billions of reads. Newer aligners, for example BWA-MEM [30], Bowtie [29], NovoAlign [37] and SOAP [32], rely on heuristics and algorithmic techniques such as tree-based indexing of the reference to speed up alignment. Others, such as SNAP [47], use hash-based indexing of the reference and are designed for multicore scalability. Alignment throughput is measured in bases aligned per second, a read-length agnostic measure.

Other expensive operations follow alignment. Downstream processing usually requires datasets to be sorted by read ID or aligned location in the genome. In addition, some downstream steps are more efficient with random access to the dataset. Sorting and indexing common data formats (§2.2) is often very time-consuming.

Once data is aligned, sorted and indexed, further filtering of data may take place. The preceding steps are usually followed by *variant calling*, another expensive process that compares the reassembled genome to the reference and attempts identify mutations. Common variant calling tools include GATK [34] and FreeBayes [16].

This is a sample of all the commonly used tools in bioinformatics; readers are referred to [38] for a more comprehensive survey.

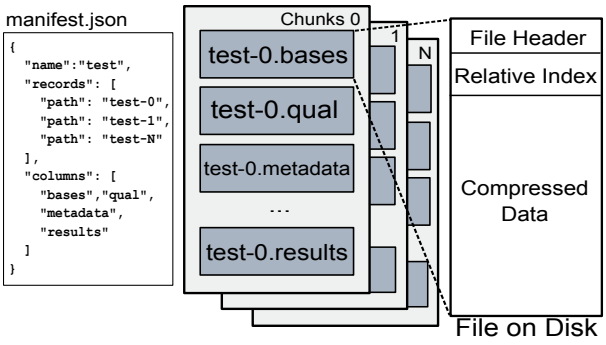


Figure 2: A dataset in AGD format.

2.2 File Formats

The canonical format produced by sequencing machines is FASTQ [8], an ASCII text format containing a delimited list of reads. FASTQ delimits reads by the `@` character, which makes parsing nontrivial as `@` is also an encoded quality score value. FASTQ files are usually distributed in a compressed format to save disk space.

The de facto standard for read and aligned data is the Sequence Alignment Map (SAM) format [31], or more often its binary, compressed version BAM. Variant calling results use the standard VCF format [9].

Typically, a dataset is stored in one FASTQ/SAM/BAM file, so these files are very large (50 to 100+ GB). While FASTQ just holds raw read data from a sequencer, SAM/BAM stores both the read and alignment data. The files are row oriented, so accessing a specific field requires reading all preceding entries, or generating a separate index file.

3 Aggregate Genomic Data Format

The *Aggregate Genomic Data* (*AGD*) format is a new extensible format for storing and manipulating genomic data designed to support the high I/O demands of Persona. AGD is designed for high-throughput read and write performance and to easily partition genomic data for parallel processing across one or more computers. Persona provides efficient utilities to export/import AGD to/from existing formats (SAM/BAM/FASTQ).

An AGD dataset is a table of records, each of which contains one or more fields (*i.e.,* a relational table). AGD stores the data in an indexed, column-store format (Figure 2). Record fields are stored by columns, which in turn are divided into large granularity *chunks* that reside in disk files. A descriptive manifest *metadata* file holds an index describing the columns, chunks, and records in an AGD dataset, in addition to other relevant data such as the names and sizes of contiguous reference sequences to which the dataset reads have been aligned. The manifest is implemented as a simple `JSON` file, which can be re-

constructed from the set of chunk files it describes. As an illustrating example, Persona uses three columns to store bases, quality scores, and metadata, and a fourth to store alignment results.

Operations on a genome dataset do not always require all elements in a record. For example, some duplicate marking schemes only require results, not base pairs or quality scores. In contrast to the row-oriented format of both FASTQ and SAM, each AGD column can be read independently and its data processed independently and simultaneously.

Moreover, AGD is extensible. A new record field (one or more columns) can be easily added by writing the column chunk files and adding appropriate entries to the metadata file. For example, Persona appends alignment results to a new AGD column. Any required parsing functions for a new column may be added to Persona. Columns can also be row-grouped, indicating that record indices align in those columns.

AGD columns are split into chunks containing varying number of records, enabling optimization for different storage subsystems. A chunk file contains a header, index, and data block (Figure 2). AGD specifies the record type in the chunk header, which informs applications how the data is stored (*e.g.,* what type of parsing to apply to each record). The index is relative, with offsets to records being generated by summing preceding index values. For more efficient random access, an absolute index can be generated on the fly.

AGD applies two techniques to reduce the size of the dataset: block compression of the data block and base compaction. The type of compression may be selected on a column-by-column basis. For example, a user may compress the bases column with *gzip* while using LZMA for the metadata. This flexibility allows tradeoffs between compressed file size and decompression time, which allow a user to balance the frequency of access against the size of a column. Our implementation uses *gzip*, as it has a good compression without being too compute-intensive. An additional optimization of *base compaction* is applied to the base reads column, which stores base characters using 3 bits each, with 21 bases in a 64-bit word.

The choice of chunk size is an important factor to maximize I/O performance. Larger chunk sizes have better compression ratios and lower overhead due to large contiguous reads from local storage. However, smaller chunk sizes decrease the I/O and decompression latency during which processing cores may stand idle.

4 System Architecture

We use a coarse-grain dataflow execution model for Persona. The major functions of the system — I/O, computation, and system management — are separated into

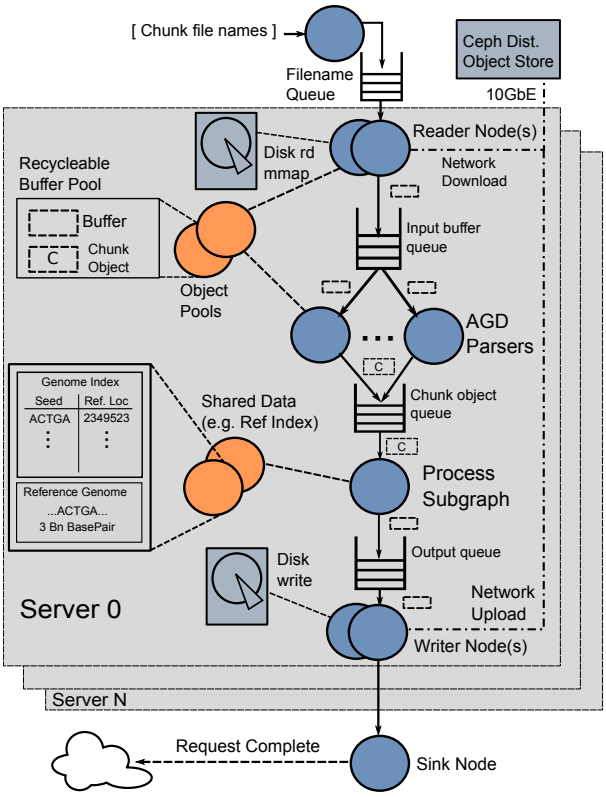


Figure 3: Persona dataflow architecture.

dataflow kernels. Each kernel can be mapped to available hardware resources (servers, cores, threads, or accelerators). This computation model simplifies the design, implementation, and deployment of the system, and allows for simple integration of new processing steps. In particular, the explicit flow between kernels simplifies performance and bottleneck analysis and makes it easy to adjust queuing for flow control and load balancing. Dataflow semantics mean that independent tasks always execute in parallel, both at the multicore and server levels.

We use Google TensorFlow as our underlying dataflow execution engine [1]. Although designed for machine learning, the core of TensorFlow is a generic dataflow engine. In TensorFlow, dataflow operators are called *nodes*, which are assembled into computation *graphs* using a Python API. Underlying *kernel* implementations of nodes are written in C++ and compiled alongside the runtime framework.

We demonstrate that it is possible to use the Google engine in a different context with minimal overhead (1%). To achieve this low overhead, Persona: (1) Uses a coarse-grain dataflow execution model between kernels, while adding a *fine-grain* execution model within compute-intensive kernels; (2) Uses pools of reusable objects to buffer data and implement a zero-copy archi-

tecture; (3) Controls memory usage by limiting the size of object pools and the length of the queues between kernels; (4) Balances the parallelism of I/O and alignment to keep all CPU threads busy.

4.1 Persona Architecture

Persona consists of two layers: a set of TensorFlow dataflow operators that read, parse, write, and operate on AGD chunks, and a thin Python library that stitches these nodes together into optimized subgraphs for common I/O patterns and bioinformatics functions.

Figure 3 shows an instance of a Persona graph on a single server. AGD chunk file names are consumed by reader nodes that read data from disk or network sources. AGD parsers decompress read chunks, enqueuing them for the *process* subgraph. The *process* subgraph contains the compute-intense operations — alignment, sorting, duplicate marking, variant calling, etc. The writer nodes store results from the process stage. Shared data objects and pools provide recyclable buffers for AGD chunks and results, and other shared objects such as the multi-gigabyte reference indexes required for some aligners.

Individual dataflow nodes and queues can be stitched together using the Python API however the user desires. However, certain configuration patterns are more efficient. The following subsections describe subgraphs that Persona uses to achieve high performance.

4.2 I/O Input Subgraph

The input subgraph is designed to keep the process subgraph fed with data while incurring minimal overhead. Reader nodes are implementations that read AGD chunks from storage. Currently, Persona supports a local disk or the Ceph object store [46] — other storage systems can be supported simply by writing the interface into a new Reader dataflow node. For disk files, Reader nodes `mmap` AGD chunk files, producing a handle to a read-only mapped file memory region. For network files, Reader nodes request the chunk files from a storage system (*e.g.,* Ceph), putting each into a recyclable buffer obtained from a buffer Pool. Once a chunk has been read, it passes via a queue to an AGD Parser node, which decompresses and parses the chunk into a useable, in-memory *chunk object*. Chunk objects are then passed to the process subgraph via a central queue.

4.3 Process Subgraphs

Process subgraphs implement the bioinformatics operations on the AGD chunk objects. We describe the implementation of several major functions and variants that are currently implemented in Persona. In our experience, since the I/O and parallel execution are provided by Persona, integrating existing tools is usually simple.

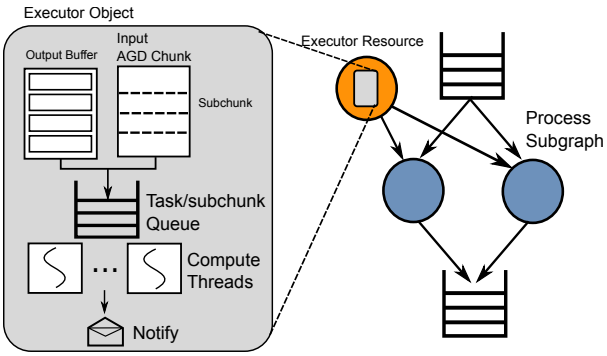


Figure 4: To abstract and share threads in a coherent way between parallel compute-intense kernels, a thread-owning executor object is provided via a resource.

SNAP Alignment The Persona SNAP aligner node uses the SNAP short read aligner [47], an open source tool that is highly optimized for modern servers with a large amount of memory and many cores. To attain maximum performance, each core in the system should be running the SNAP algorithm continuously on AGD chunks, however we found the granularity of AGD chunks, being optimized for storage, is too coarse for threads and produces work imbalance that leads to stragglers. To remedy this, execution of the alignment algorithm is delegated to an *executor* resource that owns all of the threads, and implements a fine-grain task queue (Figure 4). Multiple parallel aligner nodes then feed chunks to this *executor*, and wait for them to be completed. All cores in the system are thus kept running continuously doing meaningful work.

When executed, the aligner node receives chunk objects containing reads (base pairs and quality scores), a handle to a buffer pool of output objects, and a handle to the *executor* resource. The chunk object and output buffer are logically divided into subchunks and placed in the *executor* task queue as (subchunk, buffer) pairs. Once a full chunk is completed, the originating aligner node is notified, and the result buffer is placed in the subgraph output queue.

BWA-MEM Alignment BWA-MEM [30] is another popular read alignment tool that uses the Burrows-Wheeler transform to efficiently find candidate alignment positions for reads. We integrate BWA-MEM in the same manner as SNAP, using the *executor* resource with a fine-grain task queue (Figure 4). We call BWA-MEM alignment functions directly, with only several lines of cosmetic code changes required. For single-read alignment, this approach is straightforward, however for

paired reads, BWA-MEM incorporates a single-threaded step over sets of reads to infer information about the data. This leads to better alignment results, but separates the computationally intense multithreaded step into two parts. Therefore, the *executor* resource for BWA paired alignment divides the system threads among these tasks. We find a balance empirically, but because the computation times are data dependent, some efficiency is lost.

Sorting and Duplicate Marking Persona also integrates full dataset sorting by various parameters, including mapped read location and read ID. The sort implementation is a simple external merge sort, where several chunks at a time are sorted and merged into temporary file "superchunks". A final merge stage merges superchunks into the final sorted dataset. Persona sort is several times faster than samtools sorting of SAM/BAM files (§5).

Duplicate marking is a process of marking reads that map to the exact same location on the reference genome. This step is often performed since duplicate data can disrupt downstream statistical methods. Persona duplicate marking uses an efficient hashing technique based on the approach used by Samblaster [14].

4.4 I/O Output Subgraph

The output subgraph mirrors the input subgraph, with Writer nodes writing AGD chunks to disk or a Ceph object store, with an optional compression stage. In general, the process subgraph is responsible for ensuring AGD chunks to write are properly formatted for a given AGD column, as the Writer nodes are generic.

Persona also implements an output subgraph for the common SAM/BAM format for compatibility with tools that have not been integrated or do not yet support AGD.

4.5 Memory Management and Queuing

Proper memory management is necessary to efficiently use the underlying server hardware. In particular, it is important to avoid freeing, reallocating, and copying memory and to avoid bringing in too much data, which sits idle, or too little data, which stalls the pipeline.

We avoid using TensorFlow tensors directly for storing data, as they are not amenable to byte strings or raw buffers. Instead, we pass tensors of *handles*, which are identifiers for *resources* stored in the TensorFlow Session. The resources in Persona are the pools and their objects (buffers, chunk objects, shared read-only objects) as shown in Figure 3. With this technique, Persona performs no unnecessary copies.

Because computations in bioinformatics tend to be compute- or memory-bound, the input subgraph generally runs ahead of the alignment subgraph, quickly filling the process subgraph input queue. Persona controls memory pressure by limiting the queue length and therefore the number of objects passed around. The total quantity of objects is the sum of the queue lengths and the number of dataflow nodes that use an object. Overall memory use in Persona is stable after the input queues are filled. Because of the relatively coarse granularity of AGD chunks, default queue lengths are set to the number of parallel downstream nodes they feed, but can be tuned lower for low-memory systems.

Queue capacity is kept at a level that ensures there is always data to feed the process subgraph, but the individual servers do not have too many AGD chunks in their pipelines, which can lead to stragglers. A server can become a straggler if its queue contains "expensive" chunks with high compute latency. Work stealing [5] is an alternative to avoid stragglers, but the approach of bounding the queues is simpler and incurs less communication in a distributed system.

4.6 Discussion

Using TensorFlow as a general dataflow engine was a key design decision that had many benefits, but also led to some challenges. Bioinformatics data is not particularly amenable to storage in tensors. Initially, we had stored strings of bases, qualities and metadata in `string` type tensors, however this led to large amounts of small memory allocations, and constant data copying since the `std::string` type owns its data. This prompted the decision to move to the recyclable buffer pooling strategy outlined in the previous subsections. In an ideal world, the dataflow engine and runtime of TensorFlow would be separate from the Tensor data type and allow arbitrary types.

The execution semantics of TensorFlow also caused some issues when trying to maximize performance, especially in the multithreaded aligner kernels. Because graphs are executed in steps, there is necessarily a delay between one execution of a kernel and the next. Therefore, parallelism must be used in the graph to ensure that threads do not sit idle between executions. However, ad-hoc sharing of threads between these multiple kernels via the TensorFlow CPU device threadpool becomes difficult due to the way we need to split AGD chunks to reduce thread-level stragglers. The solution to this was the method described in §4.3, where all threads executing a given task are owned by a shared resource that can be fed with work by multiple kernels.

Despite these difficulties, we were still pleased overall with TensorFlow. The framework provides numerous features that greatly ease development and optimization, such as node-level profiling, graph visualization, and runtime statistics including current queue states or any other variable one wishes to track. We were also

pleasantly surprised at how seamlessly the implementation was able to overlap disk or network I/O with computation. We also found that the dataflow semantics in general enforce a fairly high degree of code separation and modularity, which makes for seamless extension for new support (*e.g.,* different I/O subsystems).

5 Evaluation

5.1 Experimental Setup

We use a cluster of typical datacenter machines, each with two Intel Xeon E5-2680v3 CPU chips at 2.5GHz and 256 GBytes of DRAM. With 12 cores per socket nd hyperthreading enabled, each node has 48 logical cores. All machines run Ubuntu 16.04 Xenial Linux. Each machine includes 2 SSDs in RAID1 configuration for the OS, 6 SATA disks (4TB, 7200 RPM, 6 GB/s), a hardware RAID controller, and 10GbE network interface. For single-node (local) experiments, we store the input data on a 20 TB RAID0 disk array. For distributed (cluster) experiments, we store the AGD dataset in a Ceph distributed object store [46] spread over 7 servers. The Ceph cluster is configured to use 3-way replication and each of its 7 nodes has 10 disks. The compute and storage are connected by a 40GbE-based IP fabric consisting of 8 top-of-rack switches and 3 spine switches.

Persona accesses Ceph objects via the Rados API. Using the `rados bench` tool, we measure the peak Ceph read throughput of our configuration at 6 GB/s, with sequential reads and evenly distributed data.

In all our experiments, we use half of a paired-end whole genome dataset from Illumina [12] (ERR174324), which consists of 223 million single-end 101-base reads, and is 18 GB in gzipped-FASTQ format and 16 GB in AGD format. The use of single-end read data is an arbitrary choice; Persona's integrated aligners and AGD also support paired-end alignment. The reference genome to which we align the dataset is the common hg19 human genome [23]. As mentioned in §2, alignment throughput is measured in bases aligned per second.

5.2 Persona Configuration

All execution uses the TensorFlow direct session, unmodified. For cluster-wide execution, Persona launches a TensorFlow instance per compute server. Within each server, the first stage in the TensorFlow graph fetches a chunk name from the manifest server; the latter is implemented as a simple message queue. Unless noted, the AGD chunk size is 100,000, grouped into 2231 chunks. At this chunk size, both the bases and the qualities are ∼3.5 MB. As our performance analysis focuses mainly on alignment, we read only these two columns of each chunk, totaling ∼7 MB per chunk.

	SNAP	AGD Single Node	Speedup
Disk(Single)	817 sec	501 sec	1.63
Disk(RAID)	494 sec	499 sec	0.99
Network	760 sec	493.5 sec	1.54
Data Read	18GB	15GB	1.2
Data Written	67GB	4GB	16.75

Table 1: Dataset Alignment Time, Single Server

5.3 I/O Behavior of AGD

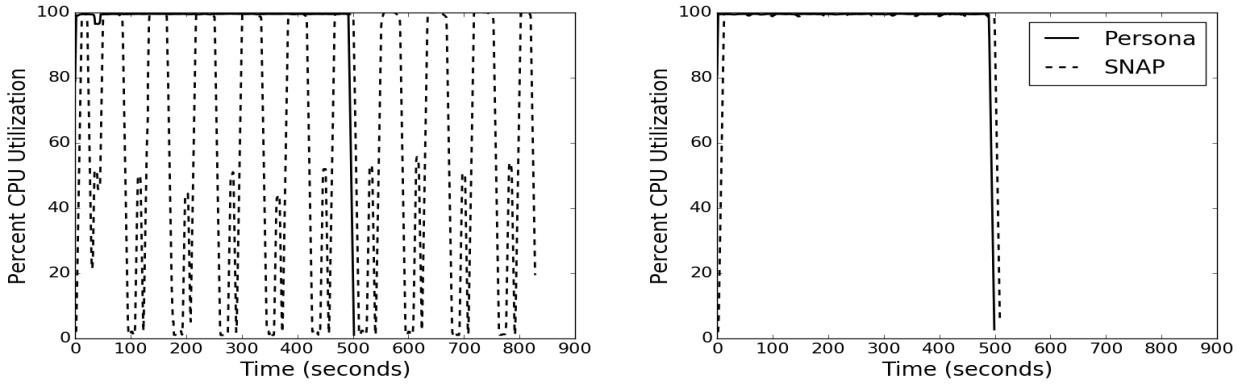
We first study the I/O behavior of Persona and AGD. I/O behavior in Persona is fundamental, since we can never assume a given patient's genome data will already be in memory (or that it even fits in memory). We perform alignment using different disk I/O configurations, using the SNAP alignment subgraph and comparing to the SNAP standalone program. We use SNAP instead of BWA because it has higher throughput and is better able to exercise the I/O subsystem. The *single disk* configuration stores the genome (and the results) on a single local disk. The *RAID0* configuration uses a hardware RAID0 array of 6 disks to increase bandwidth. Both SNAP and Persona are tuned for best performance, and use 47 aligner threads.

Figure 5 provides a characterization of the CPU utilization using a single disk and the full RAID0 configuration. Both systems overlap I/O and decompression with alignment: SNAP uses an ad-hoc combination of threads, whereas Persona leverages dataflow execution. Figure 5a and Figure 5b show that Persona is CPU bound in both configurations, but that SNAP can only use the CPU resource fully in the RAID0 configuration.

In particular, Figure 5a shows a cyclical pattern with SNAP where the operating system's buffer cache write-back policy competes with the application-driven data reads; during periods of writeback, the application is unable to read input data fast enough and threads go idle.

Table 1 summarizes the difference in terms of the amount of I/O traffic required as well as the impact on execution time. While the column-orientation of AGD has a marginal benefit in terms of data input, it has a 16.75× impact on data output, and a 1.63× speedup for the single-disk configuration. When the storage subsystem provides sufficient bandwidth, as for the RAID0 configuration, the performance of SNAP and Persona are nearly identical. Persona, however, does at least the same amount of work with less hardware and eliminates the disk I/O bottleneck.

The benefits of column-orientation of AGD are not limited to local disks. Table 1 also shows the speedup of 1.54× when the data is stored on Ceph network-attached



(a) Single Disk CPU

(b) RAID 0 CPU

Figure 5: Comparison of SNAP (GZIP'd FASTQ) and Persona (AGD) in CPU utilization with single disk and RAID0.

storage[1].

Finally, Table 1 shows that, by overlapping I/O with computation in meaningful-sized pieces, the performance of Persona is nearly identical to SNAP and CPU bound in three very different storage configurations.

5.4 Single-node CPU Alignment

We characterize the thread scaling behavior for Persona in both the SNAP and BWA-MEM aligners, while comparing them to their standalone baselines, with single-end alignment. These experiments show that Persona imposes negligible core-scaling overhead on the subsystems we have integrated, and avoids thread and I/O saturation issues by efficient overlapping.

Figure 6 shows the scalability of standalone SNAP and BWA-MEM compared to Persona as a function of the number of provisioned aligner threads on the 48 core server. The experiments were measured on the RAID0 configuration so that SNAP has enough I/O bandwidth. For SNAP, Figure 6 shows clearly: (1) a near-linear speedup for up to 24 threads, corresponding to the 24 physical processor cores of the server; (2) that, beyond 24 cores, the 2nd hyperthread increases the alignment rate of a core by 32%. At 48 threads however, contention with I/O scheduling causes a drop in performance in SNAP. Persona is less sensitive to operating system kernel thread scheduling decisions because of TensorFlow's built-in queue abstractions.

BWA scales fairly well to 24 threads, but afterwards suffers from high memory contention after hyperthreading kicks in, something we can not fix without significant changes to the codebase. However, because Persona avoids setting up and tearing down threads for different steps of processing, Persona's BWA-MEM subgraph scales slightly better with more threads than the standalone program.

[1]SNAP does not natively support reading from Ceph, so we use the `rados` utility to pipe the dataset in gzipped FASTQ format, and pipe the resulting SAM file into Ceph.

5.5 Cluster Scalability

Figure 7 shows the throughput of two different systems as a function of the number of nodes. "Actual" represents the measured performance of Persona using the SNAP alignment node, reported in gigabases aligned per second for a single genome (*i.e.,* a measurement of latency). "Simulation" is the ideal speedup line based on the maximum local server performance of ∼45.45 megabases aligned per second (see §5.4).

Persona scales linearly up to the available 32 nodes by making efficient use of all compute resources, hiding all I/O latencies and addressing the straggler problem through shallow queues. Again, we use SNAP because the higher throughput is better able to exercise the I/O subsystems. When considering BWA-MEM, alignment throughput may be lower per node, but may scale to higher numbers of servers. We reiterate that our point is not to compare BWA-MEM to SNAP, but to show that Persona is able to scale to a high number of servers while keeping process subgraphs fully supplied with data.

Using 32 servers and the SNAP process subgraph, Persona aligns the genome in 16.7 seconds, from the beginning of the request to when all results are written back to the Ceph cluster. This corresponds to 1.353 gigabases aligned per second. As far as we are aware, this represents the fastest whole genome alignment to date.

We use a different methodology to test the scalability of the storage cluster. For this, we deploy multiple "virtual" TensorFlow sessions per server and replace the CPU-intensive SNAP algorithm with a stub that simply suspends execution for the mean time required to align a chunk, and then output a representative (but obviously

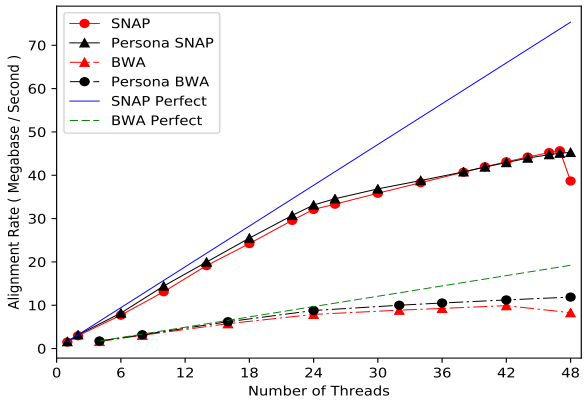


Figure 6: Throughput scaling across cores. Persona adds no measurable overhead.

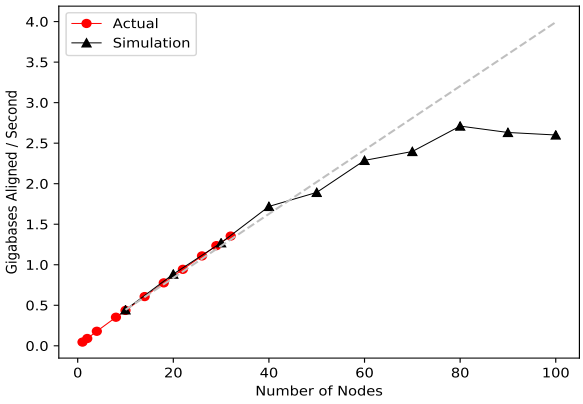


Figure 7: Actual cluster throughput up to 32 servers using the Persona SNAP aligner. Simulated throughput while scaling to 100 servers.

Tool	Time	Speedup
Persona	556 sec	1.0×
Samtools	856 sec	1.54×
Samtools w/ conversion	1289 sec	2.32×
Picard	2866 sec	5.15×

Table 2: Dataset Sort Time in Seconds, Single Server

incorrect) result.

Figure 7 shows the results in the "Simulation" line. We first validate that the simulation matches the "Actual" measurements up to 32 nodes. We then observe that the Ceph cluster scales to ∼60 nodes without loss of efficiency. Beyond 60 nodes, and for an AGD chunk size of 100,000 reads, write performance of the alignment results limits performance.

5.6 Sorting and Duplicate Marking

We also compare Persona in sorting performance to Samtools [31] and Picard [27], standard utilities for sorting SAM/BAM files. Table 2 shows the results when configuring Samtools to use all 48 cores available. Picard does not have an option for multithreading. Samtools requires sorting input in BAM format; we include both sort and sort + conversion times. Persona can directly process aligned results in AGD, performing up to 2.32 times faster than Samtools when considering the file conversion time. Persona's sort implementation is currently naive, using `std::sort()` across chunks, and we believe these results can be improved substantially.

We compare Persona's duplicate marking performance to Samblaster [14], whose algorithm we have used in our implementation. Samblaster can mark duplicates at 364,963 reads per second, while Persona, which uses Google's optimized dense hashtable, can mark duplicates at 1.36 million reads per second. Note that Persona also uses less I/O since only the results column needs to be read/written from the AGD dataset.

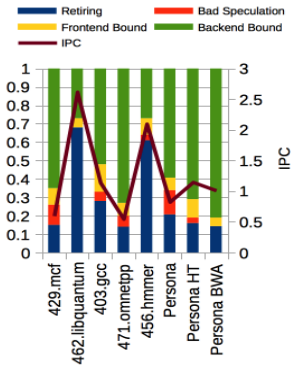
5.7 Conversion and Compatibility

To support existing sequencer output formats and other tools that have not yet been integrated, Persona can import FASTQ and export BAM formats at high throughput. FASTQ is imported to AGD at 360 MB/s, while BAM format files are produced from AGD at 82 MB/s.

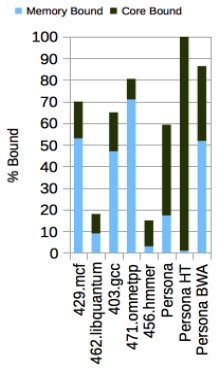
6 Discussion

Our performance analysis focuses on alignment, as it is the most compute-intense step we have yet integrated into Persona. As this is a primary bottleneck for analysis, we used Intel's VTune Amplifier [41] to profile both BWA-MEM and SNAP while running in Persona, to identify any possible avenues for improvement. Figures 8a–8b summarize our findings, while comparing to several relevant SPEC benchmarks.

Both aligner profiles share some similarity, in that they are heavily CPU backend-bound (*i.e.,* many cycles stalled due to lack of resources for accepting μOps into the pipeline, like D-cache misses or occupied functional units). With SNAP, we see that the issue is due to the core and not memory access — this is due to short but frequent calls to a local alignment edit distance function that has a small instruction mix and many data dependent instructions and branches. In BWA-MEM, the system is much more memory bound. VTune reports that this is due mostly to cache misses and DTLB misses, and our findings corroborate previous analyses [48]. This also helps explain our improved thread scaling — by restricting primary functions to sets of cores, we reduce thread



(a) Analysis breakdown.



(b) Core and memory bound levels.

Figure 8: Workload analysis (with and without Hyper-threading) compared to several SPEC benchmarks.

interference in the memory hierarchy.

6.1 TCO of Cluster Architectures

Personalized medicine has become practical because of dramatic decreases in the cost of genome sequencing. In light of these decreases, it is worth considering the cost contributions of storage and computation. We consider three cases: a single system attached to a single NGS sequencer, our own balanced cluster, and a nation-wide solution. We limit the analysis to alignment, the most expensive computation we have yet integrated into Persona.

First, Figure 5 shows the performance of single server, where genomic data is stored, aligned, and processed on a local machine. A single server can align ∼144 full sequences per day. Considering the total cost of ownership (TCO) of the server over 5 years, this implies a cost of 4.1¢ per alignment, assuming full utilization. Note that this scenario has limited genome storage capacity.

Second, there are economies of scale for sequencing, and a more likely scenario would be a regional center providing sequencing and processing services. A small cluster and network storage subsystem, as we have used in our experiments, could support 5173 alignments per day. Figure 7 shows that our storage cluster can sustain the I/O requests of a cluster of twice this size, offering expansion capacity. Table 3 summarizes the cluster compute and storage costs over a 5 year lifetime. For the network fabric cost, we determine the per-port cost of the 8-TOR, 3-spine architecture deployed in our physical cluster, and multiply by the number of ports used. Table 3 shows that, assuming the system is fully loaded, the TCO of a genome alignment on such a regional cluster is 6.07¢, higher than above because of the larger storage subsystem needed to support the throughput.

Item	Unit cost	Units	Total
Compute Server	$8,450	60	$507K
Storage server	$7,575	7	$53K
Fabric ports	$792	67	$53K
Total			$613K
TCO(5yr) [21]			$943K
Cost/Alignment (100% Utilization)			6.07¢

Table 3: Cluster TCO and alignment costs. The storage cluster has 126 TB of usable capacity, corresponding to approximately 6,000 sequenced genomes.

Third, a nation-wide solution would be needed to support initiatives such as Genomics England's 100,000 Genomes [17]. For this, additional storage is required as our balanced cluster has a usable capacity of 126 TB, which can store 6,000 in AGD format (1 days worth of sequencing). One can use the 60:7 ratio of compute to storage machines as a "not to exceed" scaling guide. The TCO model of Table 3 can be adjusted to estimate the capacity and throughput requirements of a deployment.

Storage is the dominant cost of a cluster and of genome processing. With our current high-throughput storage subsystem, the cost per genome for storage is $8.83, two orders of magnitude higher than the alignment cost. Genomes that are not being actively processed could be stored in tiered storage system using slower, lower-cost storage and erasure coding [11]. Currently, using Amazon Glacier storage ($0.007 GB/month [4]), a full genome could be stored for 5 years for $6.72, only slightly less expensive than locally hosted storage. Note that with higher coverage datasets, storage amounts and cost would increase.

Computation is far from the dominant contribution to the cost of sequencing a genome. Storage, while more expensive, is still far from a significant expense, but if the cost of sequencing continues to decline at its faster-than-Moore's-Law rate, storage may become the limiting factor in widespread genome sequencing. Novel compression for genomic data, such as reference-based compression [15], will likely be required.

7 Related Work

Because of its potential, bioinformatics and genomics have been the topic of much research. Large organizations such as the Broad Institute have established pipelines (Genome Analysis Toolkit [34]), a system similar to Persona. GATK also employs sharding for parallel data access (i.e. HDFS), but uses the standard SAM/BAM formats, often merging multiple input files into single files, which can limit scalability. Recently,

GATK has also been ported a cloud environment, Google Genomics [26]. Microsoft also advertizes cloud-based genomics capabilities [35]. However, these companies have not released details of their internal systems architectures, so it is unclear how they compare.

In terms of file formats, the recent ADAM format [33] is most similar to AGD. It also uses a column store format to achieve better compression. In addition, data is serialized using a common framework (Avro) that supports multiple languages and is easily parsed. ADAM relies on Spark and HDFS for distributed computation, again restricting users to a single storage subsystem type. In terms of performance, ADAM claims a $\sim 2\times$ speedup over Picard in single node sorting, whereas Persona achieves a $\sim 5\times$ speedup. HDF5 [44] is a general purpose hierarchical file format that can also support a bioinformatics schema similar to ADAM. In contrast to AGD, it restricts users to MPI for multiprocessing and is difficult to tune for high performance. TileDB [39] is a system that stores multi-dimensional array data in fixed size data tiles, similar to HDF5 but superior in write performance and concurrency. TileDB "chunking" is similar to AGD, but it employs a more rigid data model and is generally much more complex. Parallel access is implemented using MPI as in HDF5. Futhermore, GenomicsDB [22] is built on TileDB to store genomic variant data in 2D arrays, columns and rows correspond to genome positions and samples, respectively.

AGD differs substantially from these formats in that it is simple and requires only a way to store keyed chunks of data. The AGD API to access chunk data can simply be layered on top of different storage or file systems, using those system's APIs for parallel access, distribution, replication, etc.

Distributed alignment has been explored before, for example CloudBurst [42], which uses Hadoop MapReduce. They also find that the problem scales linearly and that distribution can result in significant speedups. CloudBurst reports 7 million reads aligned to one human chromosome in 500 seconds using 96 cores (5256 bases aligned per second per core), however a direct performance comparison is difficult because the alignment algorithm is different, the read size is different (36 base pairs versus our 101), and the cluster architecture and CPU were different. Cloud-Scale BWAMEM [7] is a distributed aligner that can align a genome in $\sim$80 minutes over 25 servers, but requires different file formats for single (SAM) or distributed computation (ADAM). SparkBWA [2] is similar, scaling alignment out over a Spark cluster, but not achieving linear scaling. ParSRA [19] shows close to linear scaling using a PGAS approach, but relies on FUSE to split input files among nodes. Eoulsan [28] uses MapReduce to perform several pipeline steps and supports different aligners. Pmap [24] uses MPI to scale several different aligners across servers and claims linear scaling.

Other efforts include SAND [36], where alignment is divided into stages for reads, candidate selection and alignment on dedicated clusters using algorithms similar to BLAST. There have also been efforts to distribute BLAST computation itself [40]. Others have shown that aligning reads to a reference genome scales linearly [20]. merAligner [18] implements a seed-and-extend algorithm that is highly parallel at all stages, but uses fine-grained parallelism more amenable to supercomputing systems rather than the clusters or datacenters that Persona targets. GENALICE Map [45] reports 92 million bases aligned per second on a single machine, faster than even SNAP, however it is a closed-source proprietary product.

In contrast to previous work, Persona and AGD provide a *general* high-performance framework that facilitates *linear* core and server scale out of not only alignment but many bioinformatics processes. Persona has *negligible* overhead, and does not restrict users to specific storage systems or parallel patterns. The dataflow architecture can support different models of parallelism, while the Python API allows user composable pipelines. AGD provides scalable, high-bandwidth access to data. Both Persona and AGD are also extensible, making it easy to integrate new or existing tools and data schemas.

8 Conclusion

In this paper, we demonstrate that existing state-of-the-art bioinformatics tools can be embedded in a distributed dataflow framework based on Google TensorFlow, yielding a composable bioinformatics pipeline that scales linearly with near-zero overhead. In addition, we propose a new data format for genomic data (AGD) that allows for efficient data partitioning and distribution across clusters.

When using the SNAP algorithm, Persona aligns a peak throughput of 1.353 gigabases per second on 32 servers. It can align a 223 million read dataset in $\sim$16.7 seconds. As far as we are aware, this represents the fastest genomic sequence alignment system to date.

When scaled up, alignment can be very cost-efficient, at only 6.07¢ per alignment, showing that bioinformatics computing can be both fast and cost effective. Costs for sequencing, at least in the near future, will be dominated by the cost of consumables and data storage.

Persona and AGD are under active development, with work ongoing to integrate comprehensive data filtering and variant calling. The goal of Persona is to bring the many disparate bioinformatics tools and algorithms into a single, high-performance, yet easy-to-use system that will meet the needs of both small-scale research and large-scale personalized medicine. We look forward to working with the systems and bioinformatics communities to achieve this end.

Acknowledgements

We thank the anonymous reviewers for their constructive feedback and our shepherd, Fred Douglis for his suggestions. This work was supported in part by the NanoTera *YINS* project, Microsoft-EPFL Joint Research Center, and a grant from VMware.

References

[1] ABADI, M., AGARWAL, A., BARHAM, P., BREVDO, E., CHEN, Z., CITRO, C., CORRADO, G. S., DAVIS, A., DEAN, J., DEVIN, M., GHEMAWAT, S., GOODFELLOW, I. J., HARP, A., IRVING, G., ISARD, M., JIA, Y., JZEFOWICZ, R., KAISER, L., KUDLUR, M., LEVENBERG, J., MANÉ, D., MONGA, R., MOORE, S., MURRAY, D. G., OLAH, C., SCHUSTER, M., SHLENS, J., STEINER, B., SUTSKEVER, I., TALWAR, K., TUCKER, P. A., VANHOUCKE, V., VASUDEVAN, V., VIÉGAS, F. B., VINYALS, O., WARDEN, P., WATTENBERG, M., WICKE, M., YU, Y., AND ZHENG, X. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *CoRR abs/1603.04467* (2016).

[2] ABUÍN, 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), e0155461.

[3] ALTSCHUL, S. F., GISH, W., MILLER, W., MYERS, E. W., AND LIPMAN, D. J. Basic local alignment search tool. *Journal of Molecular Biology 215*, 3 (1990), 403 – 410.

[4] AMAZON, I. Amazon glacier pricing. `https://aws.amazon.com/glacier/pricing/`. Accessed: 10-16-2016.

[5] BLUMOFE, R. D., AND LEISERSON, C. E. Scheduling Multithreaded Computations by Work Stealing. *J. ACM 46*, 5 (1999), 720–748.

[6] BROWN, S. M. *Next-Generation DNA Sequencing Informatics*. Cold Spring Harbor Laboratory Press Cold Spring Harbo, 2013.

[7] CHE, Y.-T., 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 (Poster). *HiTSeq* (2015).

[8] COCK, P. J., FIELDS, C. J., GOTO, N., HEUER, M. L., AND RICE, P. M. The sanger fastq file format for sequences with quality scores, and the solexa/illumina fastq variants. *Nucleic acids research 38*, 6 (2010), 1767–1771.

[9] DANECEK, P., AUTON, A., ABECASIS, G., ALBERS, C. A., BANKS, E., DEPRISTO, M. A., HANDSAKER, R. E., LUNTER, G., MARTH, G. T., SHERRY, S. T., ET AL. The variant call format and vcftools. *Bioinformatics 27*, 15 (2011), 2156–2158.

[10] DEAN, J., AND BARROSO, L. A. The tail at scale. *Commun. ACM 56*, 2 (2013), 74–80.

[11] DIMAKIS, A. G., GODFREY, B., WU, Y., WAINWRIGHT, M. J., AND RAMCHANDRAN, K. Network coding for distributed storage systems. *IEEE Trans. Information Theory 56*, 9 (2010), 4539–4551.

[12] EBERLE, M. A., FRITZILAS, E., KRUSCHE, P., KALLBERG, M., MOORE, B. L., BEKRITSKY, M. A., IQBAL, Z., CHUANG, H.-Y., HUMPHRAY, S. J., HALPERN, A. L., KRUGLYAK, S., MARGULIES, E. H., MCVEAN, G., AND BENTLEY, D. R. A reference dataset of 5.4 million human variants validated by genetic inheritance from sequencing a three-generation 17-member pedigree. *bioRxiv* (2016).

[13] EPFL VSLC-DCSL. Persona - A High-Performance Bioinformatics Framework. `https://github.com/epfl-vlsc/persona`.

[14] FAUST, G. G., AND HALL, I. M. Samblaster: fast duplicate marking and structural variant read extraction. *Bioinformatics* (2014), btu314.

[15] FRITZ, M. H.-Y., LEINONEN, R., COCHRANE, G., AND BIRNEY, E. Efficient storage of high throughput dna sequencing data using reference-based compression. *Genome research 21*, 5 (2011), 734–740.

[16] GARRISON, E., AND MARTH, G. Haplotype-based variant detection from short-read sequencing. *arXiv preprint arXiv:1207.3907* (2012).

[17] GENOMICS ENGLAND (NHS). The 100,000 Genome Project. `https://www.genomicsengland.co.uk`, 2016.

[18] GEORGANAS, E., BULUÇ, A., CHAPMAN, J., OLIKER, L., ROKHSAR, D., AND YELICK, K. A. merAligner: A Fully Parallel Sequence Aligner. In *Proceedings of the 29th IEEE International Symposium on Parallel and Distributed Processing (IPDPS)* (2015), pp. 561–570.

[19] GONZLEZ-DOMNGUEZ, J., HUNDT, C., AND SCHMIDT, B. parsra: A framework for the parallel execution of short read aligners on compute clusters. *Journal of Computational Science* (2017), –.

[20] GUO, S., AND PHAN, V. A distributed framework for aligning short reads to genomes. *BMC Bioinformatics 15*, S-10 (2014), P22.

[21] HAMILTON, J. Overall data center costs. Accessed: 08-13-2016.

[22] HEALTH, I., AND SCIENCES, L. Genomicsdb. `https://github.com/Intel-HLS/GenomicsDB/wiki`. Accessed: 05-05-2017.

[23] HG19 Human Genome Download. `http://hgdownload.cse.ucsc.edu/goldenPath/hg19/bigZips/`. Accessed: 06-20-2016.

[24] HPC LAB – OSU. Parallel Sequence Mapping Tool. `http://bmi.osu.edu/hpc/software/pmap/pmap.html`, 2016.

[25] ILLUMINA, INC. Illumina NovaSeq. `https://www.illumina.com/systems/sequencing-platforms/novaseq.html`, 2017.

[26] INC., G. Broad institute gatk on google genomics. https://cloud.google.com/genomics/gatk. Accessed: 08-13-2016.

[27] INSTITUTE, B. Picard. https://broadinstitute.github.io/picard/. Accessed: 08-10-2016.

[28] JOURDREN, L., BERNARD, M., DILLIES, M.-A., AND LE CROM, S. Eoulsan: a cloud computing-based framework facilitating high throughput sequencing analyses. *Bioinformatics 28*, 11 (2012), 1542.

[29] LANGMEAD, B., TRAPNELL, C., POP, M., AND SALZBERG, S. L. Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome Biology 10*, 3 (2009), 1–10.

[30] LI, H., AND DURBIN, R. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics 25*, 14 (2009), 1754–1760.

[31] LI, H., HANDSAKER, B., WYSOKER, A., FENNELL, T., RUAN, J., HOMER, N., MARTH, G., ABECASIS, G., DURBIN, R., ET AL. The Sequence Alignment/map Format and SAMtools. *Bioinformatics 25*, 16 (2009), 2078–2079.

[32] LI, R., YU, C., LI, Y., LAM, T. W., YIU, S.-M., KRISTIANSEN, K., AND WANG, J. SOAP2: an improved ultrafast tool for short read alignment. *Bioinformatics 25*, 15 (2009), 1966–1967.

[33] MASSIE, M., NOTHAFT, F., HARTL, C., KOZANITIS, C., SCHUMACHER, A., JOSEPH, A. D., AND PATTERSON, D. A. Adam: Genomics formats and processing patterns for cloud scale computing. *University of California, Berkeley Technical Report, No. UCB/EECS-2013 207* (2013).

[34] MCKENNA, A., HANNA, M., BANKS, E., SIVACHENKO, A., CIBULSKIS, K., KERNYTSKY, A., GARIMELLA, K., ALTSHULER, D., GABRIEL, S., DALY, M., ET AL. The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Research 20*, 9 (2010), 1297–1303.

[35] MICROSOFT. Microsoft genomics. https://enterprise.microsoft.com/en-us/industries/health/genomics/. Accessed: 08-13-2016.

[36] MORETTI, C., THRASHER, A., YU, L., OLSON, M., EMRICH, S. J., AND THAIN, D. A Framework for Scalable Genome Assembly on Clusters, Clouds, and Grids. *IEEE Trans. Parallel Distrib. Syst. 23*, 12 (2012), 2189–2197.

[37] NOVOCRAFT TECHNOLOGIES SDN BHD. NovoAlign. http://www.novocraft.com/products/novoalign/, 2016.

[38] PABINGER, S., DANDER, A., FISCHER, M., SNAJDER, R., SPERK, M., EFREMOVA, M., KRABICHLER, B., SPEICHER, M. R., ZSCHOCKE, J., AND TRAJANOSKI, Z. A survey of tools for variant analysis of next-generation genome sequencing data. *Briefings in Bioinformatics 15*, 2 (2013), 256.

[39] PAPADOPOULOS, S., DATTA, K., MADDEN, S., AND MATTSON, T. G. The TileDB Array Data Storage Manager. *PVLDB 10*, 4 (2016), 349–360.

[40] PELLICER, S., CHEN, G., CHAN, K. C., AND PAN, Y. Distributed sequence alignment applications for the public computing architecture. *IEEE transactions on nanobioscience 7*, 1 (2008), 35–43.

[41] REINDERS, J. *VTune performance analyzer essentials*. Intel Press, 2005.

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

[43] SMITH, T. F., AND WATERMAN, M. S. Identification of common molecular subsequences. *Journal of molecular biology 147*, 1 (1981), 195–197.

[44] THE HDF GROUP. Hierarchical data format version 5. http://www.hdfgroup.org/HDF5, 2000-2010.

[45] TOLHUIS, B., LUNENBERG, J., AND KARTEN, H. Ultra-fast, accurate and cost-effective ngs read alignment with significant storage footprint reduction. http://www.genalice.com/wp-content/uploads/2013/07/GENALICE-poster-HiTSeq-2013.pdf. Accessed: 08-13-2016.

[46] WEIL, S. A., BRANDT, S. A., MILLER, E. L., LONG, D. D. E., AND MALTZAHN, C. Ceph: A Scalable, High-Performance Distributed File System. In *Proceedings of the 7th Symposium on Operating System Design and Implementation (OSDI)* (2006), pp. 307–320.

[47] ZAHARIA, M., BOLOSKY, W. J., CURTIS, K., FOX, A., PATTERSON, D. A., SHENKER, S., STOICA, I., KARP, R. M., AND SITTLER, T. Faster and More Accurate Sequence Alignment with SNAP. *CoRR abs/1111.5572* (2011).

[48] ZHANG, J., LIN, H., BALAJI, P., AND FENG, W. C. Optimizing burrows-wheeler transform-based sequence alignment on multicore architectures. In *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing* (2013), pp. 377–384.