BIOINFORMATICS

Vol. 00 no. 00 2013
Pages 1–4

ADAM: A Data Format And Pipeline For Cloud Scale Genomic Processing

Matt Massie [1,*], Frank Austin Nothaft [1,*], Christopher Hartl [2], Christos Kozanitis [1], André Schumacher [3], Timothy Danford [4], Carl Yeksigian [4], Jey Kottalam [1], Anthony D. Joseph [1], and David Patterson [1*]

[1]Department of Electrical Engineering and Computer Science, University of California, Berkeley
[2]The Broad Institute of MIT and Harvard
[3]International Computer Science Institute (ICSI), University of California, Berkeley
[4]GenomeBridge, Cambridge, MA

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Associate Editor: XXXXXXX

ABSTRACT

Motivation: Current genomics data formats and processing pipelines are not designed to scale well to large datasets. As the quantity of genetic data continues to increase, it is desirable to be able to process this data in the cloud. We introduce a new set of file formats that are designed for cloud computing and that replace the BAM and VCF standards.

Results: On a high coverage (60×) 250GB NA12878 BAM file, we are able to perform Sorting and Duplicate Marking in under 50 minutes on 100 node cluster. On a single node, we are twice as fast as Picard and Samtools. ADAM files are up to 25% smaller than their equivalent BAM files.

Availability: The ADAM project website is at `http://adam.cs.berkeley.edu`. ADAM is open source under the Apache 2 license, is deployed through Maven, and the source is available through GitHub.

Contact: {massie,fnothaft,pattrsn}@berkeley.edu

1 INTRODUCTION

The Sequence/Binary Alignment/Map (SAM/BAM) and Variant Call Format (VCF) file formats were designed before multi-node processing and cloud computing were in vogue—thusly, they were designed for single node processing (Li *et al.*, 2009). As noted by McPherson (2009), as next generation sequencing (NGS) methods continue to improve, the latency of alignment and variant calling is becoming increasingly costly for scientists who are using genomics in a clinical setting, or who are working on very large datasets. To address this issue, it is desirable to be able to compute on genomic data across many machines, but characteristics of the BAM and VCF file formats practically limit scalability to 8 nodes (Niemenmaa *et al.*, 2012).

As the scalability limitations are inherent to the file formats[1], we choose to rethink file formats for genomics, instead of looking to incrementally improve either BAM or VCF. We view the following points as critical design questions:

- BAM and VCF optimize for row-oriented, flat file access. Is this access pattern the best pattern for genomics?
- Scientists want to process data from many different programming languages. How can we avoid incompatibilities between data processed using different languages or libraries?
- Genomic data will be processed on various computing systems, including single workstations, large dedicated clusters, and using cloud computing services. How can we jointly optimize for these diverse platforms?
- The file formats we use must be flexible enough to support new fields, but should not become disorganized. Is there a better way to support this extensibility than attribute maps?

To address these problems, we introduce ADAM, a set of formats, APIs, and processing stage implementations for genomic data. ADAM is fully open source under the Apache 2 license, and is implemented on top of Avro (Apache, 2013) and Parquet (Twitter and Cloudera, 2013) for data storage. Our reference pipeline is implemented on top of Spark, a high performance in-memory map-reduce system (Zaharia *et al.*, 2010). This combination provides the following advantages:

1. Avro provides explicit data schema access in C/C++/C#, Java/Scala, Python, php, and Ruby;
2. Parquet allows efficient distributed access and use by database systems like Impala and Shark; and
3. Spark improves performance through in-memory caching and reducing disk I/O.

These changes lead to significant performance improvements, and improve the format's cross-platform portability. On a single node, we are able to speedup sort and duplicate marking by 2×. More importantly, on a 250 Gigabyte (GB) high (60×) coverage human genome, this system achieves a 50× speedup on a 100 node

*to whom correspondence should be addressed

[1] This is discussed in more detail in §2.

computing cluster, fulfilling the promise of scalability of ADAM. We provide an extensive review of ADAM's performance in §4.

The ADAM format provides explicit schemas for read and reference oriented (pileup) sequence data, variants, and genotypes. As the schemas are implemented in Apache Avro—a cross-platform/language serialization format—they eliminate the need for the development of language-specific libraries for format decoding/encoding, which eliminates the possibility of library incompatibilities. Additionally, any application that implements the ADAM schema is compatible with ADAM. This prevents applications from being locked into a specific tool or pattern. The ADAM stack is inspired by the "narrow waist" of the Internet Protocol (IP) suite (see Figure 1). We consider this stack model to be the greatest contribution of ADAM.

In this paper, we start by introducing why a new approach is necessary, and what our new approach is in §2. We review the design and performance of our pipeline in §3 and §4. Finally, we provide a comprehensive comparison of ADAM against other data formats and discuss the future growth of ADAM in §5.

2 APPROACH

As we noted in the introduction, instead of trying to extend the BAM and VCF file formats, we choose to reimagine these formats. We choose to reimagine BAM and VCF because we feel that these file formats are too limiting. This knowledge comes from several attempts to adapt the BAM and VCF file formats for distributed processing (Niemenmaa *et al.*, 2012). These adaptations have been limited in their ability to scale to larger cluster sizes due to several issues intrinsic to BAM and VCF:

- BAM and VCF both depend on centralized headers, which must be globally distributed; and
- BAM and VCF both have irregular record sizes, which limits the efficiency of row-parallel reading.

These characteristics limit the performance of applications that are performing parallel access to all records of a BAM or VCF file. However, this points at another severe problem in bioinformatics: modern file formats lock users into a single processing platform or technique. There are several reasons that traditional flat file formats are not ideal for genomics. Several important access patterns that are difficult to implement on a flat, row-oriented file include:

- **Database Queries:** Extending BAM to provide standard database queries required a significant extension (Kozanitis *et al.*, 2013). No such extension exists for VCF data.
- **Efficient Predication:** Typically, a user would only like to process certain records. Currently, for both BAM and VCF, a user must load all records and then perform a filter, which is inefficient. BAM supports the inclusion of an index file (BAM Index, BAI), but this index file accelerates to a minor subset of filtering patterns[2].
- **Projection of Specific Fields:** It is typical for an application to only read some fields of a BAM record across all records, especially for statistics collection tools like SAMtools' `flagstat` (Li *et al.*, 2009). Row-oriented formats like BAM and VCF cannot support these access patterns.

Moving forward, we seek to meet two major goals: we want to provide an extensible architecture that will allow for efficient future extension of the file format, and we want to provide an initial implementation of that architecture that addresses the problems introduced above. In the remainder of this section, we review our high-level architecture that provides extensibility, and then we provide a technical overview of our reference pipeline.

2.1 Architectural Overview

The primary takeaway from our criticisms of BAM and VCF are that the formats are difficult to specialize for certain processing patterns without changing the implementation of the formats themselves. This issue is similar to the problems addressed during the development of the Open Systems Interconnection (OSI) model and Internet Protocol (IP) stack for networking services (Zimmermann, 1980). Specifically, the developers of the OSI model and the IP stack needed to make many different technologies and systems function in unison—to do this, they introduced the concept of a "narrow waist," which guaranteed that a new protocol or technology would be compatible with the rest of the system if it implemented one specific interface.

We draw significant inspiration from the development of networking standards—we believe that our largest contribution is the introduction of a stack model for genomics, where the explicit ADAM schema is the "narrow waist" in our stack. This allows components to be cleanly interchanged as long as they implement the ADAM schema. Our stack is pictured in Figure 1.

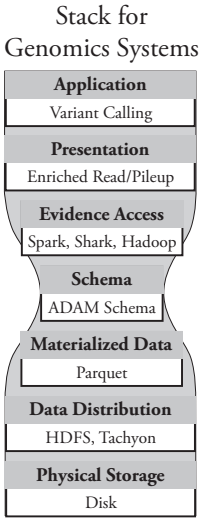

Fig. 1. A Stack Model for Genomics

[2] Typically, the first step of variant calling involves a filter that removes low quality reads. These filters are based on position-independent data (e.g. mapping quality, boolean flags) and cannot be accelerated with the BAI.

The seven layers of our stack model are decomposed as follows, traditionally numbered from bottom to top:

1. **Physical Storage:** This layer coordinates data writes to physical media, usually magnetic disk.
2. **Data Distribution:** This layer manages access, replication, and distribution of the genomics files that have been written to disk.
3. **Materialized Data:** This layer encodes the patterns for how data is encoded and stored. This provides read/write efficiency and compression.
4. **Data Schema:** This layer specifies the representation of data when it is accessed, and forms the narrow waist of the pipeline.
5. **Evidence Access:** This layer implements efficient methods for performing common access patterns such as random database queries, or sequential/parallel reading of records from a flat file.
6. **Presentation:** The presentation layer provides the application developer with efficient and straightforward methods for querying the characteristics of individual portions of genetic data.
7. **Application:** Applications like variant calling and alignment are implemented in this layer.

A well defined software stack has several significant advantages. By limiting application interactions with layers lower than the presentation layer, application developers are given a clear and consistent view of the data they are processing. By divorcing the API from the data access layer, we unlock significant flexibility. With careful design in the data format and data access layers, we can seamlessly support conventional flat file access patterns, while also allowing easy access to data with database methods. By treating the compute substrate and storage as separate layers, we also drastically increase the portability of the APIs that we implement.

2.2 Reference Implementation

In this paper, we present a reference implementation of a read processing pipeline (`adam-cli`). This implementation is fully open source and is designed for a distributed, in-memory map-reduce framework.

3 METHODS

ADAM contains formats for storing read and reference oriented sequence information, and variant/genotype data. The read oriented sequence format is forwards and backwards compatible with BAM/SAM, and the variant/genotype data is forwards and backwards compatible with VCF. We provide two APIs:

1. A data format/access API implemented on top of Apache Avro and Parquet
2. A data transformation API implemented on top of Apache Spark

In this section, we provide brief introductions to these two APIs. We then introduce the data representations, discuss the content of each representation, and introduce the transforms that we provide for each representation.

3.1 Data Format and Schema

The data representation for the ADAM format is described using the open source Apache Avro data serialization system (Apache, 2013). The Avro system also provides a human readable schema description language that can auto-generate the schema implementation in several common languages including C/C++/C#, Java/Scala, php, Python, and Ruby. This flexibility provides a significant cross-compatibility advantage over the BAM/SAM format, where the data format has only been implemented for C/C++ through Samtools (Li *et al.*, 2009) and for Java through Picard (Picard, 2013). Additionally, there are known incompatibilities between these two implementations.

We layer this data format inside of the Parquet (Twitter and Cloudera, 2013) column store. Parquet is an open source columnar storage format that was designed by Cloudera and Twitter. It can be used as a single flat file, a distributed file, or as a database inside of Hive, Shark, or Impala. Columnar stores provide significant compression (Abadi *et al.*, 2006), and are a very good match for genomics access patterns.

Figure 2 shows how ADAM in Parquet compares to BAM. We remove the file header from BAM, and instead distribute these values across all of the records stored. This dissemination eliminates global information and makes the file much simpler to distribute across machines. This distribution is effectively free in a columnar store, as the store just notes that the information is replicated across many reads. Parquet achieves additional data parallelism by writing data to disk in regularly sized row groups[3].

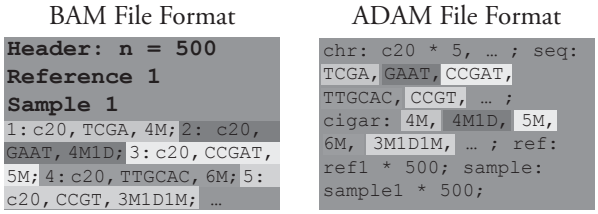

Fig. 2. Comparative Visualization of BAM and ADAM File Formats

The data types presented in this section make up the narrow waist of our proposed genomics stack in Figure 1. We provide the following datatypes:

- **ADAMRecord:** This datatype is for storing read data, and provides similar semantics to the read storage in SAM/BAM;
- **ADAMPileup:** This datatype stores reference-oriented pileup data, broken out by individual read base;
- **ADAMGenotype:** This datatype represents the genotype of a single chromosome of a single sample; and
- **ADAMVariant:** This datatype represents a single variant allele which segregates at a site.

A full description of the fields of the data types is available in the ADAM technical report (see Massie *et al.*, 2013, §5). We note that the internal representations of the data may change at any point in time—for the most up to date schema, we refer readers to the ADAM repository.

3.2 Data Transformations

In this section, we provide a brief discussion of the transforms and utilities implemented inside of ADAM. For an extended description of the *Sorting*, *Mark Duplicates*, *BQSR*, and *Indel Realignment* algorithms, we refer readers

[3] This allows for parallelism across both rows and columns.

to the appendix of Massie *et al.* (2013) which discusses these algorithms in more detail.

4 PERFORMANCE

Table 1 previews the performance of ADAM for *Sort* and *Mark Duplicates*.

Table 1. *Sort* and *Mark Duplicates* Performance on NA12878

Software	EC2 profile	Wall Clock Time
Picard 1.103	1 `hs1.8xlarge`	17h 44m
ADAM 0.5.0	1 `hs1.8xlarge`	8h 56m
ADAM 0.5.0	32 `cr1.8xlarge`	33m
ADAM 0.5.0	100 `m2.4xlarge`	21m
Software	**EC2 profile**	**Wall Clock Time**
Picard 1.103	1 `hs1.8xlarge`	20h 22m
ADAM 0.5.0	100 `m2.4xlarge`	29m

By using dictionary coding, run length encoding with bit-packing, and GZIP compression, ADAM files can be stored using less disk space than an equivalent BAM file. Typically, ADAM is up to 75% of the size of the compressed BAM for the same data. If one is performing pileup aggregation, compression can improve to over 50%.

5 DISCUSSION

Table 2. Comparison of File Formats

	BAM	ADAM
Size	1.0×	0.75-0.9×
Scalability	<8 machines	>100 machines
	VCF	**ADAM**
Scalability	<8 machines	>100 machines

5.1 Dedicated Clusters vs. Cloud Computing

ADAM is designed to fit all computing platforms well.

6 CONCLUSION

This paper presents ADAM, a new data storage format and processing pipeline for genomics data.

ADAM makes use of efficient columnar storage systems to improve the lossless compression available for storing read data, and uses in-memory processing techniques to eliminate the read processing bottleneck faced by modern genomics pipelines. On top of the file formats that we have implemented, we also present APIs that enhance developer access to read, pileup, genotype, and variant data.

We are currently in the process of extending ADAM to support SQL querying of genomic data, and extending our transformations API to more programming languages.

ADAM promises to improve the development of applications that process genomic data, by removing current difficulties with the extraction and loading of data and by providing simple and performant programming abstractions for processing this data.

ACKNOWLEDGEMENT

The authors would like to thank their many colleagues who provided feedback on early implementations of ADAM, and on drafts of the manuscript.

Funding: This research is supported in part by NSF CISE Expeditions award CCF-1139158 and DARPA XData Award FA8750-12-2-0331, a NSF Graduate Research Fellowship (**FIXME**) and gifts from Amazon Web Services, Google, SAP, Apple, Inc., Cisco, Clearstory Data, Cloudera, Ericsson, Facebook, GameOnTalis, General Electric, Hortonworks, Huawei, Intel, Microsoft, NetApp, Oracle, Samsung, Splunk, VMware, WANdisco and Yahoo!.

REFERENCES

Abadi, D., Madden, S., and Ferreira, M. (2006). Integrating compression and execution in column-oriented database systems. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 671–682. ACM.

Apache (2013). Avro. `http://avro.apache.org`.

Kozanitis, C., Heiberg, A., Varghese, G., and Bafna, V. (2013). Using genome query language to uncover genetic variation. *Bioinformatics*.

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

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

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

McPherson, J. D. (2009). Next-generation gap. *Nature Methods*, **6**, S2–S5.

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

Picard (2013). Picard. `http://picard.sourceforge.org`.

Twitter and Cloudera (2013). Parquet. `http://www.parquet.io`.

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

Zimmermann, H. (1980). OSI reference model–the ISO model of architecture for open systems interconnection. *IEEE Transactions on Communications*, **28**(4), 425–432.