Distributed Visualization for Genomic Analysis

Alyssa Morrow
Anthony D. Joseph, Ed.
Nir Yosef, Ed.

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2017-82
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-82.html

May 12, 2017

Acknowledgement

Frank A. Nothaft, Justin Paschall, George He, Devin Petersohn, Michael Heuer

Distributed Visualization for Genomic Analysis

Alyssa Morrow

May 12, 2017

Submitted to the Department of Electrical Engineering and Computer Sciences,of California at Berkeley

Abstract

The transition from Sanger to second and third generation sequencing technologies in the past decade has led to a dramatic increase in the availability of genomic data. The 1000 Genomes Project provides over 1.6 terabytes of variant data and 14 terabytes of alignment data, laying the foundation for large-scale exploration of human variation across 2,504 individuals [1]. Sequencing is useful beyond identifying DNA variation: the ENCODE Consortium project has collected 20TB of sequencing data across various assays, which has enabled novel insights into the role of epigenetics in human disease [2]. However, current genomic visualization tools are intended for a single-node environment and cannot scale to terabyte scale datasets. To enable visualization of terabyte scale genomic datasets, we develop Mango. Mango is a visualization tool that selectively materializes and organizes genomic data to provide fast in-memory queries driving genomic visualization. Mango materializes data from persistent storage as the user requests different regions of the genome, and efficiently organizes data in-memory using interval arrays, an optimized data structure derived from interval trees. This interval based organizational structure supports ad hoc queries, filters, and joins across multiple samples at a time, enabling exploratory interaction with genomic data. When used in conjunction with Apache Spark, Mango allows users to query large datasets and predictive models built from such datasets, while exploring results in real time.

Chapter 1

Introduction

Next generation sequencing technologies have greatly decreased the cost of genetic sequencing, leading to a dramatic increase in the availability of genomic data [3]. The Human Genome Project was a 15 year effort to sequence a single human, costing over $2.7 billion [4]. Today, sequencing for an individual costs approximately $1000, and can be completed in days [5]. This dramatic decrease in cost and time has led to projects such as the 1000 Genomes Project [1] and The Cancer Genome Atlas [6], generating 15 TB and 2 PB, respectively. Next generation sequencing technologies have also led to an explosion of new assays, characterizing the transcriptome and epigenome. Projects like the ENCODE Consortium have produced open access datasets 20 TB in size [2]. This combination of high throughput DNA sequencing and epigenomic datasets has introduced new challenges of how to compute, interact and learn from large genomics datasets.

This advent of big data was not novel to the field of genomics. Amidst completion of the Human Genome Project in 2003, Google MapReduce introduced a fault-tolerant, distributed programming model that allowed common data reduction tasks to be run in a distributed environment [7]. Google MapReduce was closely tied to Google File System (GFS), which stored large datasets in partitioned replicates across executing machines, reducing network bandwidth requirements for transferring data during compute [8]. After the introduction Google MapReduce

and GFS, Apache Hadoop and Hadoop Distributed File System (HDFS) were created as an open source alternative to Google's solution to distributed computing [9]. Apache Hadoop's open source implementation of fault-tolerant, commodity hardware computing allowed all cloud supporters, such as AWS and Microsoft Azure, to provide easy access to distributed computing on commodity machines, democratizing access to large-scale data analysis.

Although MapReduce pioneered an accessible programming model for distributed computing on commodity hardware, Apache Spark modified the existing model to move lineage of data transformations from persistent storage to an in-memory structure, called Resilient Distributed Dataset (RDD). Apache Spark's optimization of in-memory processing removed overhead of IO intensive transactions between data transformations inherent in Apache Hadoop, providing near real time analytics [10, 11]. Apache Spark's fast in-memory transactions and adaptive fault tolerance model led to a myriad of downstream applications. One such application included ADAM, a genomic data processing system intended for preprocessing, storage and batch analyses of large genomic datasets [12, 13]. ADAM provides cheaper genome alignment and preprocessing, decreasing costs of previous best practice genomic pipelines by 66% [14]. More specifically, the combination of Apache Spark and ADAM allowed users to perform iterative analyses on genomic datasets in-memory, reducing latency between analyses.

However, as sequencing and genomic preprocessing costs continue to fall, the ability to perform fast and scalable interactive analytics on genomic data has not kept up. Traditionally, genome browsers have served as a "data reduction" tool, summarizing overwhelmingly large genomic datasets into single tracks in a browser window [15]. Such visualizations aid clinicians and researchers in making decisions from these genomic datasets. However, existing visualization tools are bottlenecked by the following criteria:

1. **Interactive Analytics**. The ability to interactively query data with sub-second response times is crucial for exploratory data analysis (EDA). Liu and Heer conclude that visual

latency in 500ms increments degrades the rate at which users can make observations from visualizations [16]. An example analysis where interactive response times improve analysis is de novo variant discovery. Currently, visualization tools such as IGV and IGB require external systems to query for de novo variants in the proband of high coverage pedigrees, incurring significant overhead from switching between single-node systems. However, optimizing coordinate queries on these variant datasets in a unified distributed environment allows users to quickly query de novo regions and view the results within the same system. This interactive analysis allows users to quickly iterate on variant calling methods based on visual output of initial results.

Enabling interactive analytics and visualizations on genomic data requires a coordinate-based distributed system that can access data quickly while enabling fast, iterative analytics on such data. With the existence of such a system, we can explore and solve many interesting problems regarding genomic analysis. However, systems such as Hadoop MapReduce and Apache Spark are intended for batch processing of large datasets, and do not natively support low latency, fine-grained queries that are intended for interactive analysis.

2. **Visualization**. Current genomic visualization software is computationally constrained by the amount of genomic data they can quickly display. Current visualizations, such as track layouts and Circos diagrams, are optimized only to view individual genomes [17]. Specifically, alignment record track layouts, or pileups, require approximately 1.3 kilobytes per read under GA4GH schemas [18]. Using these schemas, data transfer for four high coverage alignment file at 40,000 bp is 2.4 GB, already exceeding the recommended Apache 2 Server REST sizes of 2GB [19]. Data transfer sizes grows linearly with the number of files viewed. Leveraging a distributed coordinate optimized system would

allow for more powerful data reduction methods, supporting population scale visualizations such as interactive heat maps, SNP comparisons, and summary statistics that can be computed in real time.

3. **Location independent data-serving**: Current desktop browsers require specific file formatting and preprocessing prior to visualization. To view large files on a desktop application, regions of interest (ROI) must first be sub-selected from high coverage files using tools such as bedtools, vcftools and igvtools [20, 21, 22]. Desktop tools such as the Integrative Genome Viewer (IGV) and Integrated Genome Browser (IGB) require presorting and indexing of VCF and BAM files, creating mandatory preprocessing prior to visualization. Similarly, web based genome browsers require uploading genomic files in specific legacy format, incurring overhead from upload times. Using a schema independent endpoint would abstract away requirements of preprocessing, sorting and indexing, allowing users to swap out abstracted data endpoints with local files during visualization.

This thesis introduces Mango, an interactive genomic visualization tool intended to drive EDA on large genomic datasets. Preliminary results show Mango's efficient use of in-memory indexing supports sub-second query for visualizations on 800 GB of high coverage alignment data from the Platinum dataset [23]. Mango addresses the following problems in state of the art genomic visualization tools:

1. **Scalable visualizations**: We use Apache Spark to optimize data locality by implementing per-partition coordinate-based data organization and partition across sorted chromosomal locations for quick access of multiple samples around a given genomic loci. This coordinate-based system can be leveraged to quickly produce visualizations that encourage exploration and hypotheses of population genomics. An important visualization for genome wide association studies include the ability to group samples and study aggregate

single nucleotide polymorphisms (SNPs) between groups. Such interactive visualizations would initially provide higher level views and the ability to zoom in on specific regions and samples.

2. **Location independent data-serving**: While current browsers depend on data-aware positioning, Mango removes the constraint of location aware data partitioning. Mango uses REST integration to separate the visualization and data-serving layers, making visualizations agnostic to data location. Therefore, users can interact with local and staged datasets seamlessly.

3. **Single Genomic Pipeline for Exploratory Data Analysis (EDA)**: Mango is built on the Apache Spark and ADAM infrastructure, allowing users to efficiently preprocess and query datasets in the same environment. This design removes the constraint of using different tools to join, query and visualize data.

Mango and its dependencies are all open source, freely available projects licensed under the Apache 2 license. Mango is available at https://github.com/bigdatagenomics/mango.

Chapter 2

Related Work

This chapter explains current state of the art tools for visualizing genomic datasets. Several different genome browsers exist, each supporting a range of functionality and features. Genome browsers can be classified into the following three categories:

1. **Desktop Application**: Genome browsers used for browsing personal data are often built as desktop applications. Desktop applications integrate visualization and data access and are optimized for end-to-end performance. Popular browsers include IGV [21], IGB [24], and Savant [25].

2. **Local Web Application**: Local web applications are primarily optimized for visualization of genomic data in a web browser. These tools are built in popular front-end languages such as JavaScript, HTML and AngularJS. Commonly used front-end browsers include pileup.js [26], igv.js [27], and JBrowse [28].

3. **Online Web Application**: Online web applications support visualization of open access datasets through a web portal. Visualizations in online browsers are constructed from static, curated datasets that are controlled by an external organization. Examples of online browsers include the UCSC Genome Browser [29].

In the following sections, we discuss each category of genome browsers and the corresponding strengths and weakness of each category.

Desktop Applications

State of the art genome browsers intended to run in a desktop environment include IGV [21], IGB [24], and Savant [25]. In this section, we assess the strengths and weakness of each tool in terms of intended use case, feature availability, scalability, and performance.

Integrative Genome Viewer (IGV)

IGV is a genome visualization tool built to run on a single-node. It is well supported by a wide community of engineers, and thus supports robust visualization of multiple data types, including NGS sequence alignment, genome annotation, and array-based datasets [21]. Because IGV's primary use case is single-node visualization, IGV puts primary focus on minimal memory footprint by precomputing data tiles for multiple resolutions, which decreases memory consumption at low resolution regions. IGV achieves efficient memory allocation by precomputing multiple zoom levels for each track, and fetches each data tile based on the base pair resolution requested by the user [21]. To minimize the computational burden of data tiling on large files, IGV uses a hybrid approach, tiling data at low resolution, computing tiles for high resolution views on the fly. This bypasses the need to store all zoom levels in-memory before the user views them. IGV's data tiling approach allows users to view regions with low latency after tiles are computed. However, initial computation of data tiles is incurred during runtime and is transparent to the user. To eliminate transparency of such overhead, IGV supports additional file formats that optimize data tiling at run time. One example of this is the tiled data frame (TDF) file format, which is computed from alignment files and summarizes coverage over the whole genome. Computation of TDF files reduces file size and removes the requirement for

coordinate-based joins at run time [21].

Because IGV runs on a single-node, it is limited in the file sizes it can display. Alternatively, IGV can run remotely, allowing users to view files on computers with sufficient memory resources to store large files. Although files on IGV can be stored in arbitrary locations, individual files are read from a single-node endpoint, enforcing hardware specific constraints of data storage limits and I/O performance.

Integrated Genome Browser (IGB)

Similar to IGV, the Integrated Genome Browser (IGB) is a desktop tool intended for viewing NGS sequencing data, annotation and array data [24]. IGB puts greater emphasis on user experience when traversing zoom levels and genomics coordinates. One mechanism IGB uses to re-orient user focus when traversing datasets is one-dimensional, animated semantic zooming, which centralizes the user focus to the center of the viewing screen while panning and zooming [24]. Like IGV, IGB has the ability to run on remote machines. However, IGB does not support data tiling at low resolutions, inhibiting users from visual information gain at large genomic regions.

Savant

Similar to IGV and IGB, Savant is a genome browser intended for a single-node environment. However, Savant puts greater emphasis on the ability to easily run on a remote server, allowing users to run queries on datasets without having to physically move files. Savant uses data tiling to view large genomics ranges, and additionally computes read coverage from alignment files on the fly when viewing alignments at zoomed out ranges. Savant supports coverage precomputation, similar to TDF generation in IGV. Preprocessing times for precomputing coverage for a high coverage file is 40 minutes. Queries in Savant run on average in sub-second latency, with worst case queries of up to 2 seconds [25].

Table 2.1: Functional comparison of three categories of genome browsers. 'X' indicates existence of functionality.

Trait	Desktop	Local Web	Online Web
Data Expressibility	X	X	-
Modularity	-	X	-
End-to-End Optimization	X	-	-

Local Web Applications

Although tools like IGV, IGB and Savant provide full end-to-end data access and visualization, tools such as JBrowse [28], pileup.js [26] and igv.js [27] are lightweight local applications that only support data visualization, accessing data over a REST API or through local computer I/O. Front end browsers enforce strong stack modularity, requiring developers to serve genomic data, independent of front-end architecture. However, this removes the potential for integrated optimization between front-end and back end, incurring higher latencies when integrating with a modular back end.

Online Web Applications

Tools such as the UCSC Genome Browser eliminate single-node scalability constraints by providing an elastic back end server which stores datasets to be viewed by the user [29]. UCSC Genome Browser stages large amounts of precomputed annotation data that can be searched and visualized by the user. Staged datasets include 1000 Genomes datasets and TCGA. However, UCSC Genome Browser has limited bandwidth for users to upload personal files, limiting the user to explore data prestaged by the browser.

We summarize the functionality of these three categories of genome browsers in Table 2.1, comparing features relating to data expressibility, modularity, and end-to-end optimizations. The first feature, dataset expressibility, allows browsers to easily import and adapt to new

datasets, agnostic of location or data schema. Examples of location-based expressibility include the ability to access data either locally or remotely. Schema-based expressibility is the ability to retrieve data independent of storage and file format. Desktop applications allow interchange between local and remote resources, providing location-based expressibility. However, desktop applications do not provide any support for schema-based expressibility, handing only legacy file formats. Local web applications allow back end implementation to dictate dataset expressibility. However, local web applications offer no potential solutions to expressibility, requiring the back-end to address such problems.

The second feature, modularity, allows users to interchange stack layers of browser infrastructure based on personal preference and computational requirements. Examples of genome browser modularity include replacing front end visualizations based on the datasets being viewed, or being able to move data storage endpoints based on file size requirements. However, modularity and end-to-end optimization are mutually exclusive features. While desktop applications and online web applications implement end-to-end optimizations for low latency, architecture boundaries are unclear, thus inhibiting users from reconfiguring the browser based on specific needs. Local web applications enforce modularity, allowing flexibility in the use of a back-end based on data requirements.

In the design of Mango, we sacrifice end-to-end optimization for a modular stack. Shown in Figure 3.1, we introduce a layered architecture, allowing users to substitute out layers of the stack for custom architectures. This modular architecture is explained in the next chapter.

Chapter 3

Architecture

This chapter explains the architecture of Mango. Mango is a distributed visualization tool built on Apache Spark [10] and ADAM [12]. It uses a layered stack architecture, divided into client, server and cluster components. Figure 3.1 depicts Mango's architecture.

The cluster layer builds upon data layout patterns and genomic transformations introduced in ADAM [12], and implements optimizations for fine-grained genomic data access. The service layer provides standalone data access to genomic endpoints, supporting standardized schemas such as GA4GH schemas [18]. The client layer provides genomic visualizations, built on pileup.js [26].

Cluster

The cluster layer of the stack is built on Apache Spark and ADAM. Apache Spark is a distributed general purpose cluster computing framework intended for fast in-memory data processing of large datasets [10]. Apache Spark is fault-tolerant, allowing Apache Spark to gracefully handle failures from commodity hardware. With the increase in accessibility to cloud computing in environments such as AWS and Microsoft Azure, Apache Spark serves as a low cost, low maintenance computing option over single node custom architecture.

Apache Spark is a general framework, and does not natively support genomic file formats

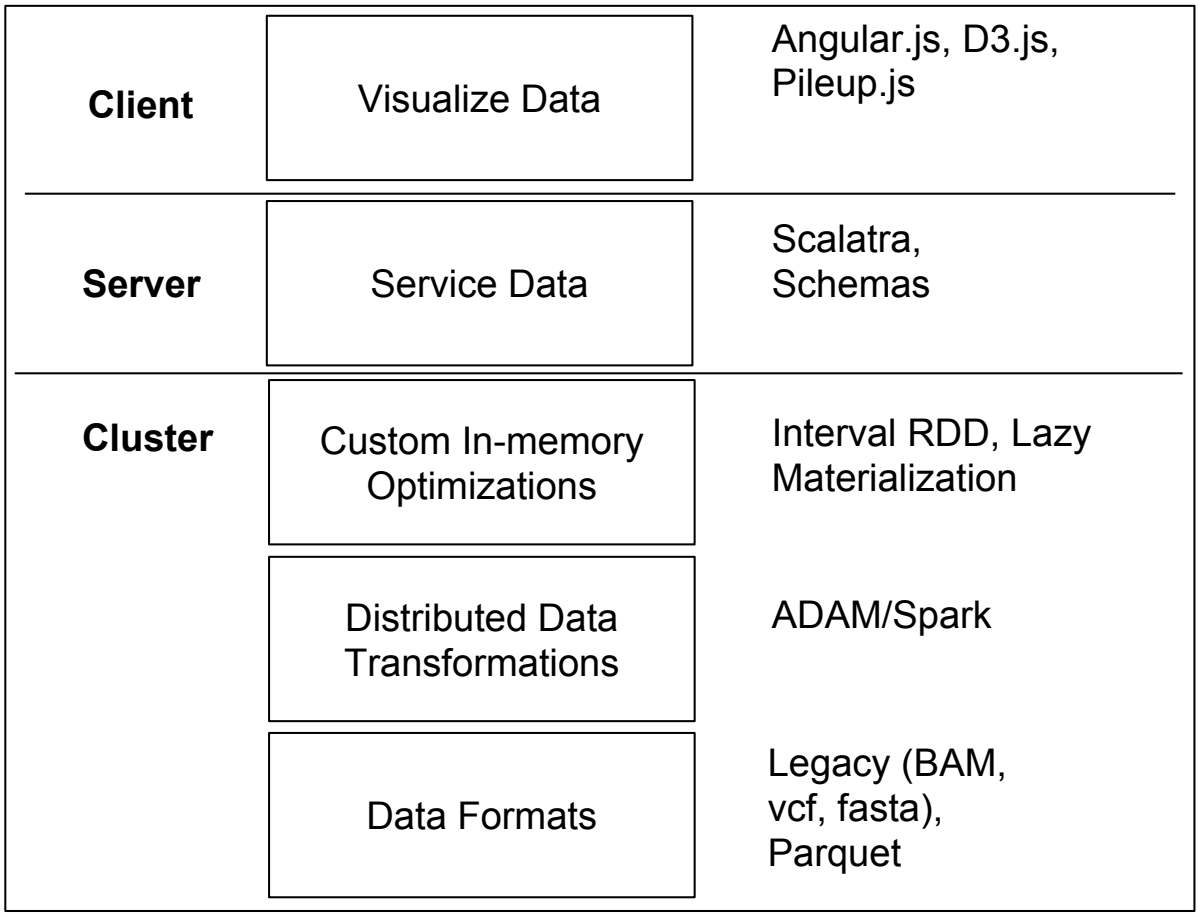

Figure 3.1: Mango Stack Architecture

Table 3.1: Legacy file formats supported in genome browsers and ADAM equivalent formats.

Data	File Format	ADAM and Mango Support
Variant Calls	VCF	ADAM VariantContext, VCF
Alignment Data	BAM	ADAM AlignmentRecord, BAM
ChIP-seq, RNA-seq	BAM, WIG	ADAM Coverage, BAM
Numeric data, features	BED, WIG	ADAM Feature, BED
Annotations	GTF	ADAM Feature, GTF

and transformations. ADAM is a framework for processing genomic data, and is built to run on Apache Spark. ADAM uses distributed abstractions in Apache Spark to batch process raw genomic datasets in parallel. More importantly, ADAM provides standardized schemas intended for various types of genomic datasets, presented in a narrow waist design that can be easily accessible by external applications. Schemas that are used for genomic visualization from ADAM include alignment, features, and variant schemas. ADAM and Mango also support legacy file formats, including VCF, BAM, BED and GTF files. However, legacy formats can be transformed into ADAM format and viewed in Mango. Table 3.1 lists file formats supported in IGV and corresponding file format supported in Mango.

Although ADAM and Apache Spark provide fast data transformations in a distributed environment, neither tool is optimized for fine-grained, low latency queries. To modify these systems to support low latency queries, we adopt two strategies for selective access of genomic regions from persistent storage and low latency in-memory queries. The first adaptation, lazy materialization, is a technique that fetches fine-grained genomic regions from persistent storage, using Apache Parquet predicates to selectively access genomic regions from persistent storage [30]. Granularity of genomic region size can be adopted to the amount of memory available in a given environment. The second adaptation, Interval RDD, are optimized for interval-keyed lookups, and are compatible with Apache Sparks RDD interface.

Lazy Materialization

Lazy materialization is a data management layer used in Mango that selectively accesses records from persistent storage based on the genomic region requested. This structure fetches data from storage upon each query and materializes the selected range as well as surrounding regions for fast subsequent access to specific loci. Lazy materialization aggressively manages memory resources to purge least recently accessed materialized regions. This design is motivated by visualization applications' tendency to access data in small, localized areas. For genomics, users might be interested in a particular gene, and will look in the immediate area around that particular gene rather than jump around to random locations in the genome. This observation of visualization informed navigation patterns is not unique to genomics and was observed more generally by Battel et. al. [31]. Due to these fine-grained, strategic access patterns in visualization tools, batch caching strategies traditionally used in Apache Spark would incur memory overhead of 300,000x from loading the whole genome, but only visualizing a subselected 10,000 bp region. By constructing a caching layer that scales to application requests, system memory requirements only need to scale to the fraction of dataset requested.

IntervalRDD

In genomics, record indexing and filtering is dictated by interval-keyed records. Figure 3.2 demonstrates a query on interval-keyed records, called overlapping range queries. Querying over interval-keyed records requires finding all records whose segment lies within or overlaps that interval.

Existing solutions to filtering interval-keyed records in distributed environments such as Apache Spark are:

1. **Linear traversal**: Store unsorted records and filter all records by start and end interval value.

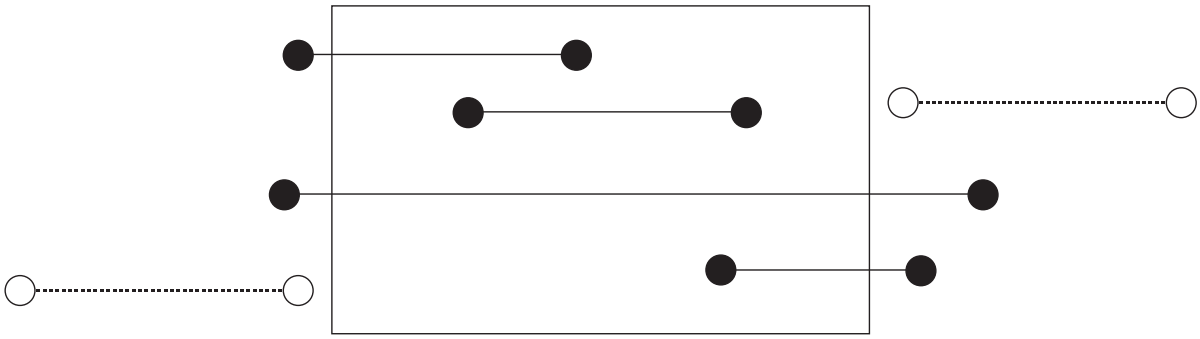

Figure 3.2: Overlapping Range Query: In querying interval-keyed records, only the black segments overlapping or contained in the query boundaries should be returned. All dotted segments should be omitted.

2. **Indexed traversal**: Store sorted records, indexed by start value, in a persistent adaptive radix tree structure.

3. **Interval Tree traversal**: Store interval keyed records in an interval tree structure.

4. **Interval Array traversal**: Binary search and fetch records stored by interval in an array.

Performing linear traversal (Option 1) in Apache Spark's RDD requires a full data scan. Another consideration for sorting interval keyed records in Apache Spark is the Indexed RDD [32]. The IndexedRDD project [32] uses persistent adaptive radix trees (PARTs) [33] to optimize point queries on RDDs. However, PART's are optimized for one-dimensional keys, and thus can also incur a worst case time complexity of full data scans on interval-keyed records. Interval trees are an efficient sorting schema for fetching interval-keyed records in $O(logn+m)$ time, where n is the total number of records and m is the density of records overlapping a given

Table 3.2: Query Complexity for Overlap Range Queries: n is the number of interval-keyed records and m denotes record density at a given interval.

Method	Creation	Query	Worst Case Query	Memory
Linear	$O(1)$	$O(n)$	$O(n)$	$O(n)$
PART	$O(nlogn)$	$O(n)$	$O(n)$	$O(n)$
Interval Tree	$O(nlogn)$	$O(logn + m)$	$O(logn + m)$	$O(n)$
Interval Array	$O(nlogn)$	$O(logn + m)$	$O(logn + m)$	$O(n)$

query [34]. Table 3.2 shows the creation and runtime complexities of the options explained above.

The ideal structure for range lookups of interval-keyed records is the interval tree. However, in practice, interval trees incur high space complexity and memory consumption, which are generated from the objects required for tree design. An alternative to the interval tree is an interval array, which maintains a sorted list of interval-keyed records and runs an expanding binary sort on records, terminating when the search reaches non-overlapping intervals. Therefore, we have implemented interval arrays as the primary interval-keyed lookup structure. We maintain average query complexity of $O(logn + m)$ for interval-keyed records, while storing all data in a simple array structure.

Server and Client

The remaining two layers of the stack are the server and client. Genomic endpoints in the server layer are exposed through a common API for external access to genomic datasets. Mango integrates standardized endpoint schemas built from GA4GH through the alignment record endpoint [35, 18]. The remaining client layer implements commonly used genomic visualizations to be visualized in a web browser. Mango implements a front-end interactive browser using pileup.js [26].

Chapter 4

Experiments

This chapter explains two use cases for the integration of genomic visualization into large-scale EDA, and assesses the runtime and feature compatibility of Mango to drive such analyses.

De Novo Variant Exploration

One use case for genomic EDA is discovery of de novo mutations in the proband. De novo mutation discovery has become an increasingly important in analysis of the effect of mutational processes in development of neurodevelopmental diseases [36]. Additional applications of de novo variant discovery include pediatric diagnosis of rare diseases [37]. In this EDA experiment, we analyze six high coverage alignment files from the Illumina Platinum datasets [23]. Individuals assessed in this analysis are shown in Figure 4.1. Two trios for NA12877 and NA12878 are queried in parallel for de novo mutations.

Preprocessing

Six raw BAM files totaling 691.8 GB from the Platinum Genomes dataset were converted to ADAM format and loaded in Hadoop's Distributed File System across 200 partitions. File conversion ran on 36 Intel E5-2670 2.6 GHz 8 core machines with 480 GB RAM. Mean conversion times for each alignment file was 15.89 seconds per file with standard deviation of 3.21 seconds.

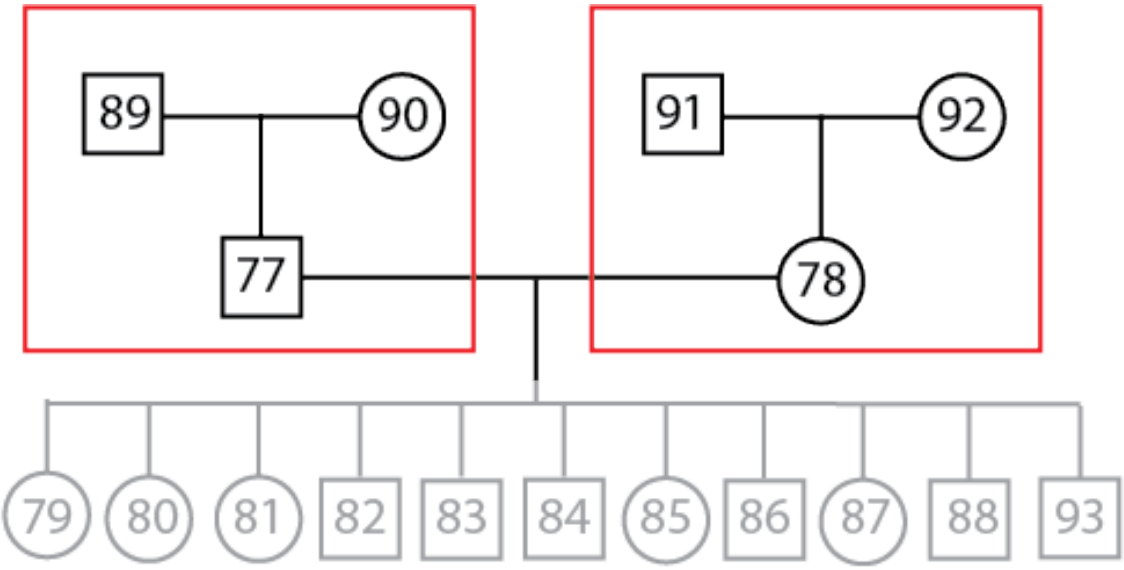

Figure 4.1: Platinum Pedigree for NA128* individuals. De novo analysis is run on the NA12877 and NA12878 trios, highlighted in red.

Total ADAM file sizes for the six Platinum individuals was 821.9 GB. The six files were variant called through avocado, a distributed variant calling tool [38], resulting in 3.525 GB of variant data. Final variant files for each trio were joined using Gnocchi [39], averaging a merge time of 41 seconds per trio. Final variant file sizes in ADAM format were 1.365 GB.

De Novo Variant Queries

Merged variant files were queried for de novo mutations in the proband of each trio. Genomic regions were binned into 1000 bp regions and scanned for de novo mutations called in avocado that were absent in the parents. Final results were filtered by mutations overlapping genes or in the vicinity of enhancer regions upstream from a given gene. Final genome wide densities for de novo variants were 74593 regions for NA12877 and 82823 regions for NA12878. Query times for de novo discovery were 4.2 minutes and 11.7 minutes for NA12877 and NA12878, respectively.

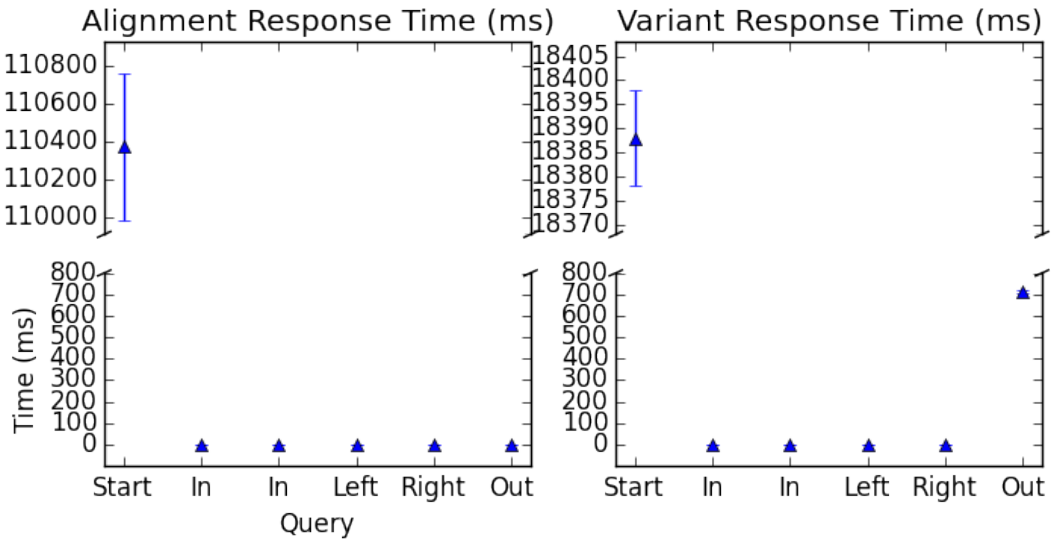

Figure 4.2: Average response times in Mango for Alignment and Variant data from Platinum datasets.

Analysis of De Novo Mutations with Mango

From the resulting de novo queries, genomic region 78309000-78310821 on chromosome 1 was isolated as a hot spot for plausible de novo mutations in NA12878. This region was isolated due to direct overlap with the protein coding GIPC2 gene. Mango was then run on 637 cores on Intel E5-2670 2.6 GHz 8 core machines with 1.7 TB RAM. To assess latency times in Mango, we adopt a common query exploration access pattern, zooming in on the GIPC2 region of interest, and panning left and right, then zooming in and out [31]. Timing results are shown in Figure 4.2.

Overhead of initial load time averages 110 seconds for 821 GB of alignment data and 20 seconds 1.5 GB of variant data. High latency results from range query scans in Apache Parquet files. However, subsequent requests are cached, and thus achieve average response times of 1ms.

Figure 4.3 shows initial variant and alignment resolution of variant hot spot at a 233 bp range. Figure 4.4 demonstrates detailed annotation of de novo variants in NA12878 at 59 bp,

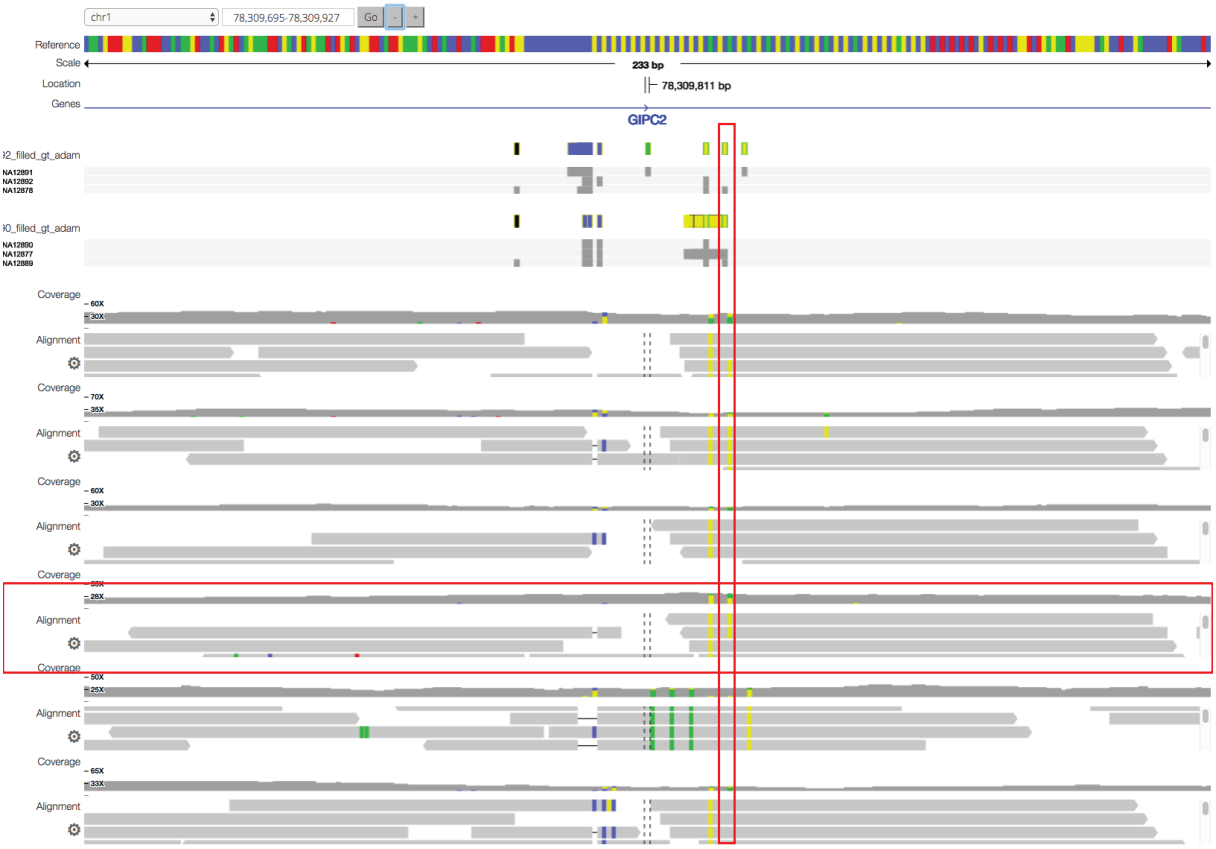

Figure 4.3: Initial resolution of variants at 233 bp, demonstrating potential de novo sites for NA12878, outlined in red.

highlighted in red. From further visual analysis of raw alignment data for NA12878, we see that the specific genotype for this potential de novo mutations for NA12878 at this region are uncertain, as calls at this location display similar proportions of the 'G' and 'T' alternate allele.

At large ranges (more than 40,000 bp), variants are binned by region to give coordinate-based summary of variant density. This is demonstrated in the final zoomed out query of 60,000 bp, and is explained in Figure 4.5.

Figure 4.4: High resolution image of 59 bp, demonstrating potential de novo sites for NA12878, outlined in red. Final variant calls at this site for NA12878 have potential alternates, given base changes in the raw alignment data.

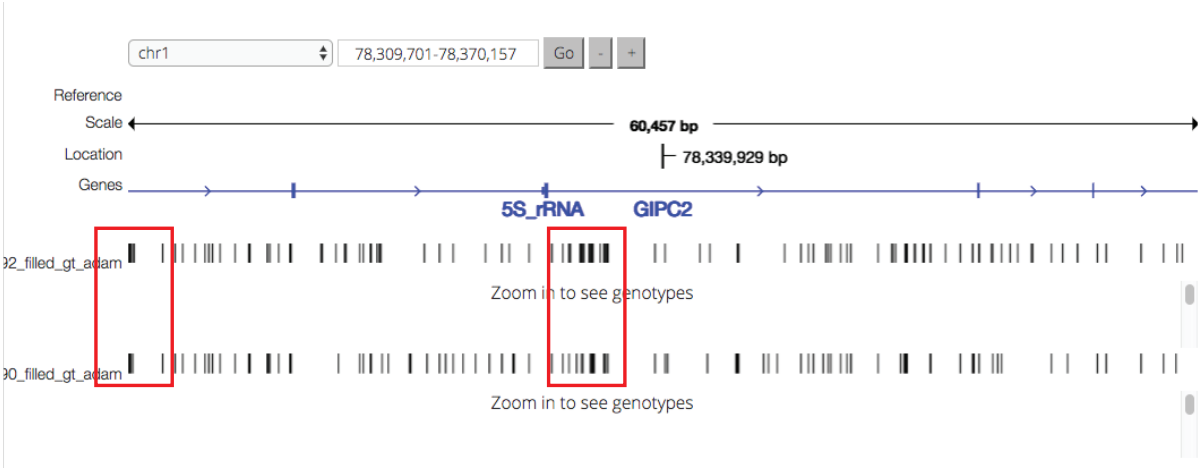

Figure 4.5: Final query of 60,000 bps zoomed out to GIPC2 gene. Regions of high density variants are outlined in red. From this image, users can summarize hotspots for differential variation in the NA12878 trio around 78,309,000 bp.

EDA for Epigenetic Analysis

Although Mango provides a powerful tool for visualizing large datasets, using Mango as an integrated step in EDA allows for users to investigate specific questions while visually exploring interesting regions of the genome. Here, we use the combinatorial power of Apache Spark, ADAM and Mango to demonstrate a use case of EDA in a distributed environment. This use cases addresses the question of investigating differentiation in binding patterns of transcription factors (TFs) in seven cell types.

This pipeline queries ChIP-seq for TF CEBPB and corresponding DNase-seq data from seven cell types from the ENCODE consortium. The investigated cell types include A549, H1-hESC, HCT116, HeL1-S3, HepG2, IMR-90 and K562.

Preprocessing

This section explains required preprocessing steps for DNase-seq and ChIP-seq datasets. Thirty-three BAM files of DNase-seq data were taken for the seven cell types listed above from the

Table 4.1: Cell type specific technical and biological replicate files. File sizes include raw BAM and processed ADAM files.

Cell Type	Replicates	Raw BAM Size (GB)	Final Size (GB)
A549	3	5.1	0.34
H1-hESC	2	4.1	0.47
HCT116	2	3.3	0.35
HeLa-S3	2	17	1.1
HEPG2	3	28.6	1.1
HeLa-S3	2	17	1.1
HepG2	3	28.6	1.1
IMR-90	2	5.1	0.45
K562	19	47.7	2
Total	**33**	**110.9**	**9.275**

ENCODE Consortium [2]. Each cell type had a combination of technical and biological replicates. The replicates and file sizes for each cell type are shown in Table 4.1.

Raw DNase-seq alignment data for each cell type were aggregated together into two ADAM formatted coverage files, one representing positive strands and the other representing negative strands. Negative strands were shifted one bp towards the 5 direction to account for bias in binding [40]. This preprocessing pipeline was executed in Apache Spark using ADAM on 384 cores with 120GB RAM. This preprocessing pipeline leads to approximately a 10x compression size from original BAM files to coverage files stored in ADAM format.

Preprocessing Transcription Factor Peak Datasets

Narrowpeak files representing peaks of transcription factor binding sites for the seven cell types were processed from the ENCODE consortium into ADAM format using ADAM on a single core on a Intel E5-2670 2.6 GHz machine with 20 GB RAM. Total time for processing these files totaled 118s, with a mean of 16.9s per file and standard deviation of 3.5s. The final representation of narrowpeak files in ADAM format totaled 4.1MB.

Query Explanations

In this EDA pipeline, three queries on the preprocessed ChIP-seq and DNase-seq datasets were run prior to visualization in Mango. These queries include the following:

1. Find regions in ChIP-seq that have the highest consistent binding across all seven cell types.

2. Find regions where binding sites have low accessibility, measured from DNase-seq.

3. Find regions where binding sites have consistently high accessibility from DNase-seq across cell types.

All three queries were run in 5 minutes 57 seconds on 385 virtual cores totaling 1.8 TB memory.

These genomic regions from all three queries were used as a driver for ad-hoc queries in Mango. Using these query results, we chose chromosomes most commonly identified in the query sets, chromosome 7 and 11, and preloaded these chromosomes into Mango with 385 cores and 1.4 TB RAM. Total preloading time for chromosomes 7 and 11 had a mean of 87.5 seconds with a sample standard deviation of 2.1 seconds.

Upon load time, Mango displays density of peak files, displayed in Figure 4.6. Here, we can see high density binding regions across the genome.

Next, we navigate the region of the genome that had the top hit for Query 1, or highest binding sites that are consistent across all cell types. Initial visualization, shown in Figure 4.7, demonstrates a summary of accessibility and binding regions in a 250,000 bp region. In Figure 4.7, both features and accessibility coverage are binned to provide efficient visualizations at large regions. A high resolution region on chromosome X is shown in Figure 4.8. At high resolution, DNase-seq footprinting across all cells is clearly evident.

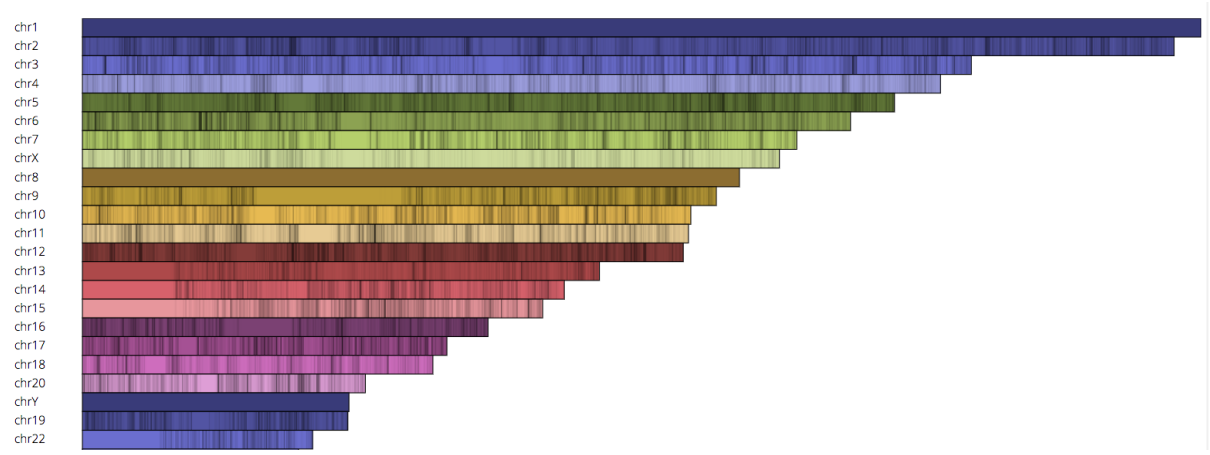

Figure 4.6: Initial loading screen in Mango. Black bars indicate high density binding regions for transcription factor CEBPB in seven different cell types.

Run time for visualizing the region with most binding sites across celltype from query 1 is shown in Figure 4.9. This query fetches from chromosome 7, which is a region from a preloaded chromosome. Therefore, mean initial fetch times for DNase- and ChIP-seq peaks averages around 500ms interactive latencies, with maximum times of 1695 and 1711 ms, respectively. Here, we first visualize a region sized 250,000 bps and follow a similar zooming pattern in Battel et. al. [31], starting at a zoomed out region and navigating to higher resolution windows. Mango's binning strategy reduces the need to collect data at these large regions. The initial latency is 62315 and 7791 bps when a new zoom region is computed.

Next, the top hit from query 2 was loaded into the browser from chromosome X, shown in Figure 4.10. This chromosome was not preloaded, so we incur 18.7 seconds of latency while loading raw data from persistent storage. However, subsequent query times match the caching patterns from query 1.

Evaluation of Coverage

Coverage in Mango allows users to receive summary statistics of alignment, variant and feature data at zoomed out ranges. Because raw alignment data is too large to collect and communicate

s

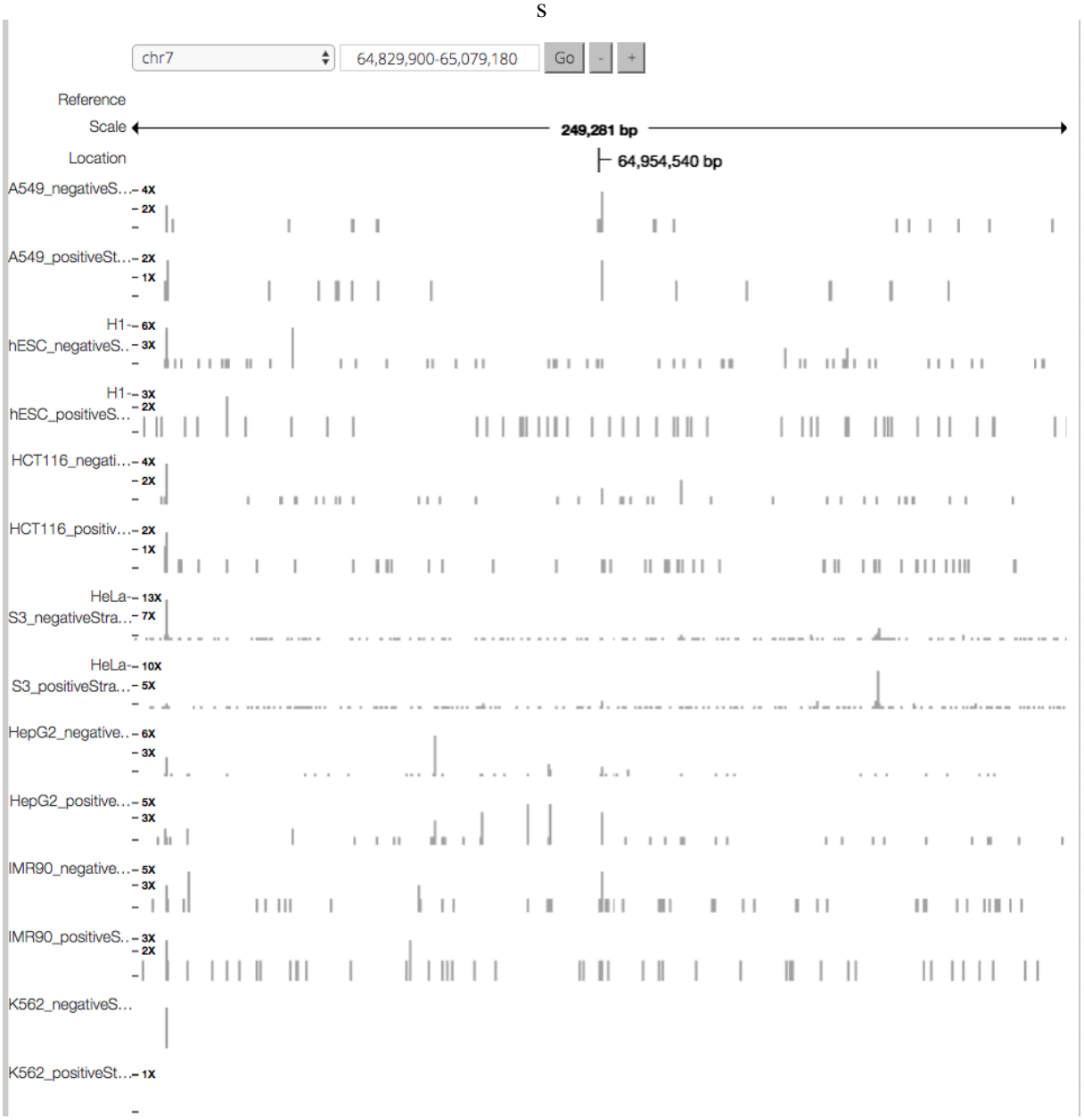

Figure 4.7: Zoomed out region of 250,000 bp of binding sites with consistently accessibly regions across seven cell types. Region is binned into 100 bp bins.

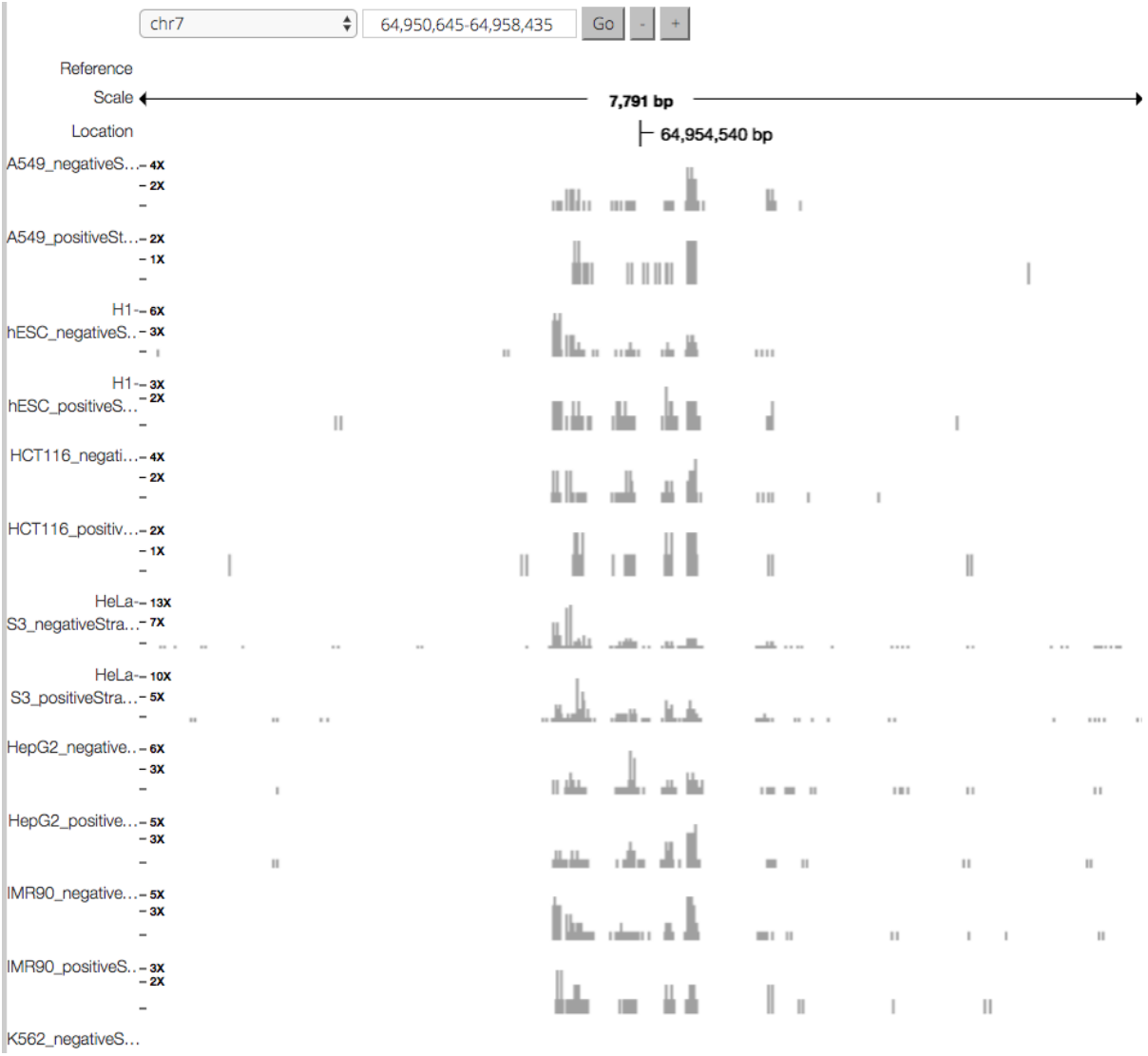

Figure 4.8: Initial loading screen in Mango. Black bars indicate high density binding regions for transcription factor CEBPB in seven different cell types.

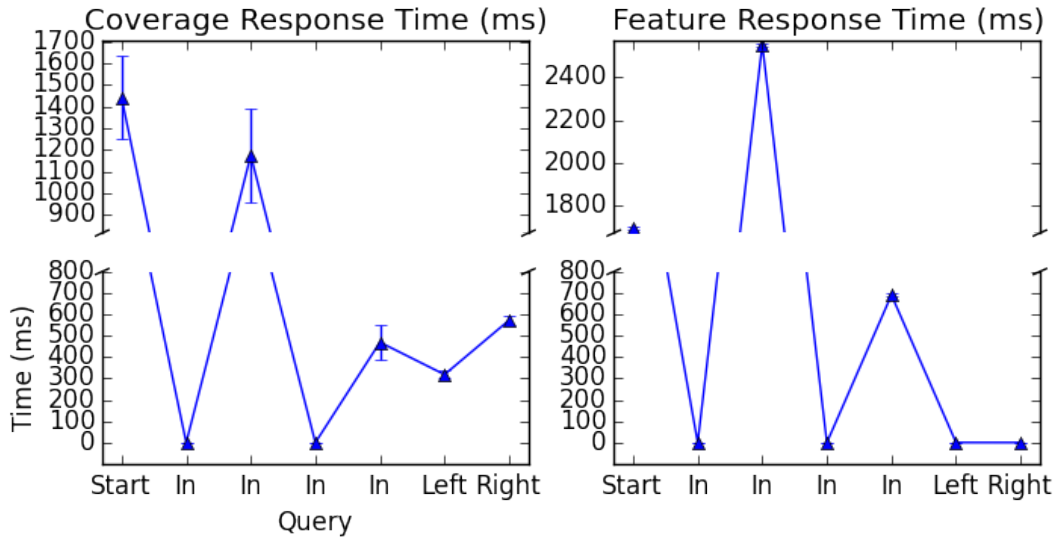

Figure 4.9: Average response times in Mango for coverage and feature data from ENCODE datasets. Response times at query 3 increases in latency due to transition to a higher resolution layer.

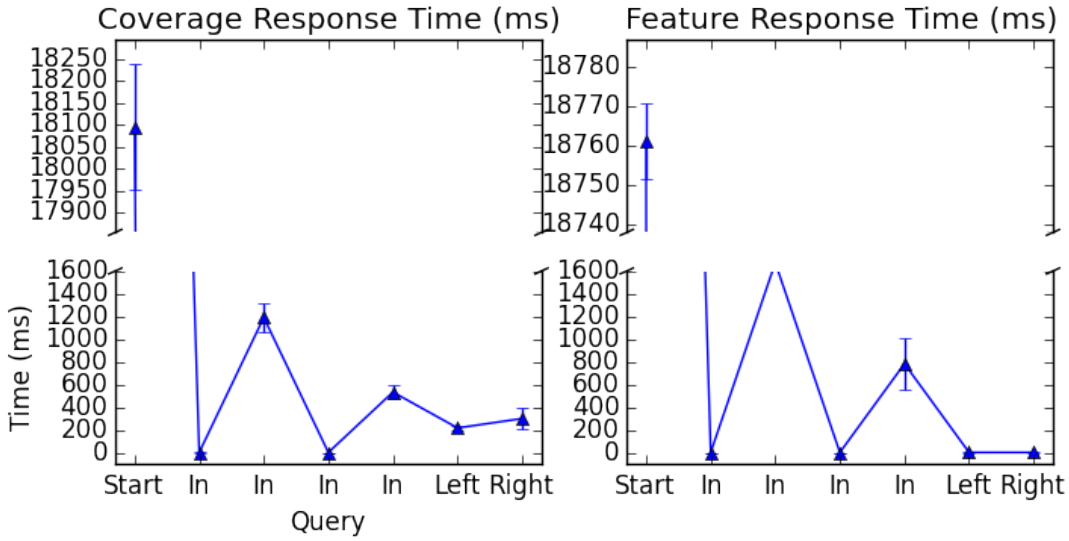

Figure 4.10: Average response times in Mango for Coverage and Feature data from ENCODE datasets.

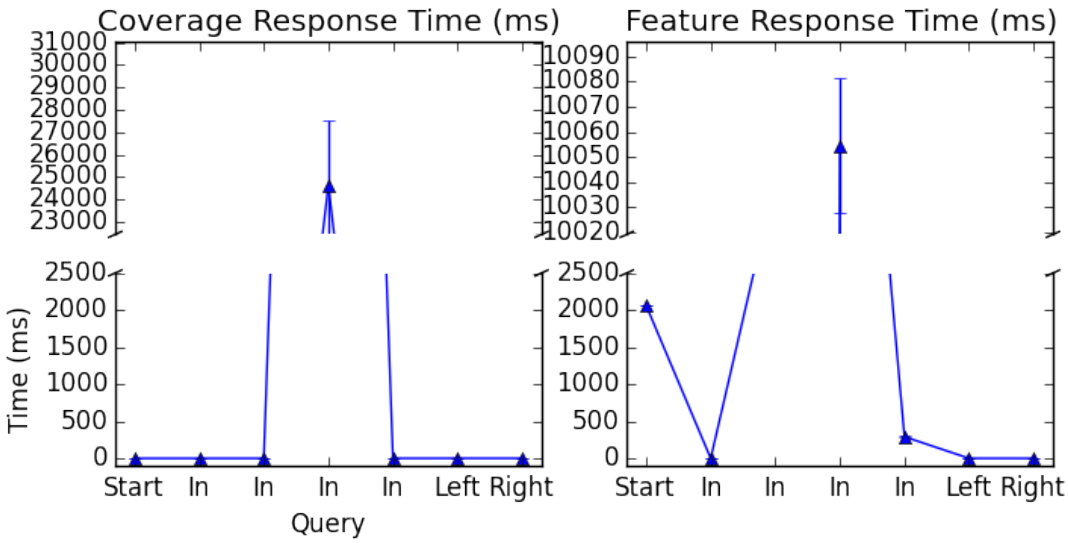

Figure 4.11: Average response times in Mango for raw alignment data and feature data from ENCODE datasets. Increased latency at query 4 indicates initial load of alignment data. Alignment data is not shown at resolution lower than 40,000 bp, generating a spike in load time for coverage and features at the fourth query.

at large genomics regions, coverage summarizes these data set sizes up to genomic ranges the size of the entire genome. In this section, we evaluate runtime on the same query set shown in Figure 4.10 with raw alignment data in place of coverage data. Here, we load in the data aggregated from the original 33 raw DNase-seq BAM files into seven different tracks. The runtime for these queries are shown in Figure 4.11. Raw ADAM sizes for the 33 aggregated files totals 103.8 GB. Preload time for chromosomes 7 and 11 totaled 4 minutes and 34s.

At low resolution, the user no longer is able to access summary data regarding DNase-seq data. Initial load time of alignment data at 32000 bp is 31 seconds, as alignment data is not cached during preloading due to extensive memory overhead.

Interactive Model Serving for Regulatory Genomics

The last use case we demonstrate is that of model serving for genomic workloads in a visualization environment. This use case is to address the issue of pipeline segmentation in training and analyzing machine learning models on genomic workloads. Currently, there is no infrastructure available for users to upload and predict on their models, allowing them to visually annotate prediction results in real time. This use case demonstrates an end-to-end pipeline that efficiently computes and predicts on user-provided DNA sequence while maintaining seamless integration between processing steps. In previous work, we used a scalable parallel algorithm to efficiently compute protein binding sites without the use of specialized hardware, allowing us to flexibly use hardware independent systems such as Apache Spark for training [41]. We utilize this algorithm to predict the binding affinity of the EGR1 protein using ChIP-seq and DNA sequence, and provide a full analysis pipeline to visualize results. This novel pipeline allows users to interactively predict on a ROI in a query genome.

For preliminary results, we train a model predicting protein binding sites using the kernel approximation featurization method described in [41]. This model was solved using Keystone MLs Weighted Least Squares estimator, which accounts for large class imbalance of positive binding sites across the genome [42]. This model was trained in a distributed environment from 200 million sequences extracted from transcription factor EGR1 ChIP-seq binding regions, generating a 4 MB model to be tested on new datasets. Training the model took 58 minutes parallelized across 16 nodes with Intel E5-2670 2.6 GHz 8 core CPU, 256 GB RAM and 4 1 TB HDFS hard drives. The resulting model was saved and loaded into Mango to visualize predictions on raw datasets. Raw ChIP-seq peaks and sequencing data were loaded and visualized in Mango. Regions sized 2000 bp, 2000 bp, 4000bp, 8000 bp and 16000 bp were queried, predicting binding results at user request. Minimum, maximum and mean times to predict on new sequences using the model are shown in Table 4.2.

Table 4.2: Minimum, maximum and mean prediction response times for transcription factor EGR1.

Min	Max	Mean
3.09ms	3.98s	402.31ms

Figure 4.13 visualizes the resulting predictions in Mango on a 10,000 bp region. Here, we predict the binding affinity of EGR1 on the GM12878 cell type. Visualization of EGR1 binding predictions suggest that the algorithm predicts more binding sites than are actually bound by ChIP-seq experiments. These visual insights dictate modifications to subsequent prediction algorithms.

Concordance with Local Genome Browsers

IGV is a well supported genome browser commonly used in the research and clinical genetics communities. For this reason, we compare Mango and IGV on a a single machine with 24 Xeon processors and 256 GB of RAM. For this workload, we analyze the runtimes of the NA12878 trio of sorted and indexed alignment files, sized 310 GB. We analyze a query pattern on an initial 1000 bp region and traversal of surrounding regions. Total run times for each query are shown in Figure 4.12. Mango and IGV start up times are comparable. IGV initial load times, however, range from 2.8 seconds, while Mango initial load times are 25 s. Overhead in Mango is due to IO overhead from Hadoop access with Hadoop-BAM [43]. Subsequent request for cached regions are comparable between Mango and IGV.

Here, we do not include runtimes of ADAM file formats for a local machine due to extensive overhead from Apache Parquet predicates. Although initial response times for Mango range 24 seconds, subsequent requests meet interactive thresholds of 500ms.

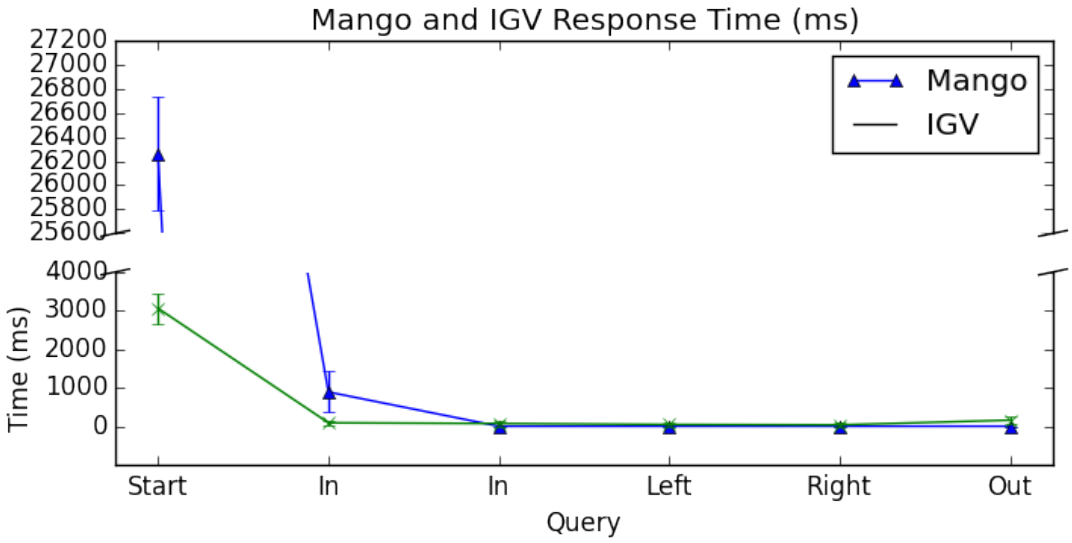

Figure 4.12: Average response times in Mango for raw alignment data and feature data from ENCODE datasets. Although Mango incurs initial 24 second latency on data load, subsequent response times are concordant with IGV.

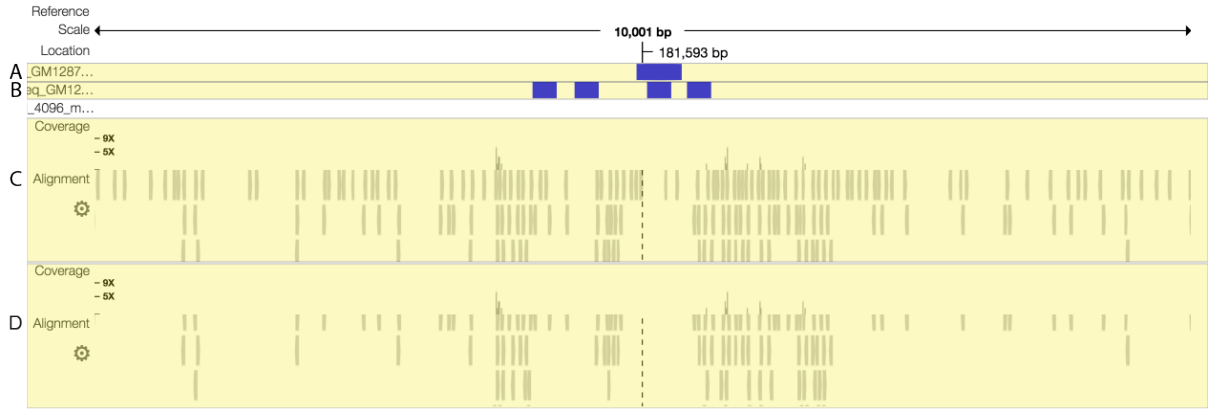

Figure 4.13: Visual representation of predictions for EGR1 TF with raw datasets and labels. (A) shows model predictions. (B) shows true labels indicating binding sites. (C) and (D) display raw ChIP-seq datasets. From this image, a researcher learns that the presented model has high false positive rates, suggesting implementation of stricter class imbalance.

Chapter 5

Feature Compatibility

Mango implements many features available in other genome browsers. In Table 5, four main features are listed and evaluated for inclusion across IGV, IGB, Mango, and Savant. These features were listed in the analysis by Robinson et. al. [21]. Although Mango does not currently support access to public hosted datasets, Mango is the only tool that allows users to scale out to multiple nodes in a distributed environment.

Feature	Mango	IGV	IGB	Savant
Alignment	X	X	X	X
Multiple Resolution and Binning	X	X	-	X
Multinode Scalability	X	-	-	-
Public Hosted Datasets	-	X	X	-

Table 5.1: Feature comparison of Mango and common genome browsers.

Chapter 6

Conclusion

This thesis introduces Mango, a visualization tool intended for large genomic workloads. Mango takes advantage of existing distributed systems intended for batch processing and modifies them to support fine-grained, low latency queries. We have analyzed the ability of Mango to execute important EDA pipelines under a single system.

Future Work

Integration of exploratory data tools into the genomics community depends on intuitiveness and robustness. Thus, we design our future work around integrating scalable EDA pipelines for use in the genomics community.

1. **Notebook Integration**: Improving the integration of the EDA pipeline to a single interface would enhance the efficiency of executing EDA pipelines in a single system. In the future, we plan to implement user facing abstractions in Mango into a notebook form, where users can execute queries and interact with results in a single environment.

2. **Intuitive visualizations**: Although Mango supports traditional track visualization of commonly viewed genomic formats, scalable visualizations for population level analysis must be developed for large-scale analysis. Effective integration of heat map structures to

visualize population SNP density and Manhattan plots for GWAS visualization is crucial to make conclusions from large population studies.

With notebook integration and intuitive population scale visualizations, we will have a fully integrated genomic EDA pipeline for large-scale genomic analysis.

Conclusion

In this thesis, we have introduced a data independent, horizontally scalable visualization tool intended for genomic workloads. We have identified key weaknesses in current tools used for visualization and address scalability and system unification required for an efficient EDA stack. We have demonstrated, under two unique use cases, compelling full stack EDA analyses that are simplified under a single system. Most importantly, we have published a tool that abstracts away programming semantics, allowing users to interact with large-scale genomic data in an intuitive and interactive environment.

Bibliography

[1] The 1000 Genomes Project Consortium, "A global reference for human genetic variation," *Nature*, vol. 526, pp. 68–74, 10 2015.

[2] The ENCODE Project Consortium, "An integrated encyclopedia of dna elements in the human genome," *Nature*, vol. 489, pp. 57–74, 2012.

[3] J. Shendure and H. Ji, "Next-generation dna sequencing," *Nat Biotech*, vol. 26, pp. 1135–1145, 10 2008.

[4] International Consortium Completes Human Genome Project, "National human genome research institute," 2015. Web. 6 Oct. 2015.

[5] Illumina, "An introduction to next generation sequencing," 2016.

[6] J. N. Weinstein, E. A. Collisson, G. B. Mills, K. R. M. Shaw, B. A. Ozenberger, K. Ellrott, I. Shmulevich, C. Sander, J. M. Stuart, and e. a. Cancer Genome Atlas Research Network, "The cancer genome atlas pan-cancer analysis project.," *Nature Genetics*, vol. 43, pp. 1113–1120, 2013.

[7] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, (San Fransisco, CA), 2004.

[8] H. G. Sanjay Ghemawat and S.-T. Leung, "The google file system," in *19th ACM Symposium on Operating Systems Principles*, (Lake George, NY), 2003.

[9] Apache, "Hadoop." http://hadoop.apache.org, 2017.

[10] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, (Berkeley, CA, USA), pp. 10–10, USENIX Association, 2010.

[11] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, (San Jose, CA), pp. 15–28, USENIX, 2012.

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

[13] F. A. Nothaft, M. Massie, T. Danford, Z. Zhang, U. Laserson, C. Yeksigian, J. Kottalam, A. Ahuja, J. Hammerbacher, M. Linderman, M. J. Franklin, A. D. Joseph, and D. A. Patterson, "Rethinking data-intensive science using scalable analytics systems," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, (New York, NY, USA), pp. 631–646, ACM, 2015.

[14] F. Nothaft, "Scalable genome resequencing with adam and avocado," *Technical report, UCB/EECS-2015-65, EECS Department, University of California, Berkeley*, 2015.

[15] T. S. Furey, "Comparison of human (and other) genome browsers," *Human Genomics*, vol. 2, no. 4, pp. 266–70, 2006.

[16] Z. Liu and J. Heer, "The effects of interactive latency on exploratory visual analysis," in *IEEE Transactions on Visualization and Computer Graphics*, 2014.

[17] M. P. Schroeder, A. Gonzalez-Perez, and N. Lopez-Bigas, "Visualizing multidimensional cancer genomics data," *Genome Medicine*, vol. 5, no. 1, p. 9, 2013.

[18] ga4gh, "ga4gh-schemas." https://github.com/ga4gh/ga4gh-schemas, 2017.

[19] A. Foundation, *Apache HTTP Server Version 2.4*, 2017. LimitRequestBody.

[20] A. Quinlan and I. Hall, "Bedtools: a flexible suite of utilities for comparing genomic features," *Bioinformatics*, vol. 26, pp. 841–842.

[21] J. T. Robinson, H. Thorvaldsdttir, W. Winckler, M. Guttman, E. S. Lander, G. Getz, and J. P. Mesirov, "Integrative genomics viewer," *Nature Biotechnology*, vol. 29, pp. 24–26, 2011.

[22] P. Danecek, A. Auton, G. Abecasis, C. A. Albers, E. Banks, and M. A. DePristo, "The variant call format and vcftools," *Bioinformatics*, pp. 2156–2158, 2011.

[23] M. Eberle and et al., "A reference data set of 5.4 million phased human variants validated by genetic inheritance from sequencing a three-generation 17-member pedigree.," *Genome Research*, vol. 27, pp. 157–164, 2017.

[24] N. Freese, D. Norris, and A. Loraine, "Integrated genome browser: Visual analytics platform for genomics," *Bioinformatics*, vol. 32, pp. 2089–95, 2016.

[25] M. Fiume, V. Williams, A. Brook, and M. Brudno, "Savant: genome browser for high-throughput sequencing data," *Bioinformatics*, vol. 26, no. 16, p. 1938, 2010.

[26] D. Vanderkam, B. Aksoy, I. Hodes, J. Perrone, and J. Hammerbacher, "pileup.js: a javascript library for interactive and in-browser visualization of genomic data," *Bioinformatics*, vol. 32, no. 15, pp. 2378–2379.

[27] igvteam, "igv.js." https://github.com/igvteam/igv.js, 2017.

[28] M. E. Skinner, A. V. Uzilov, L. D. Stein, C. J. Mungall, and I. H. Holmes, "Jbrowse: A next-generation genome browser," *Genome Research*, vol. 19, pp. 1630–1638, 2009.

[29] D. Karolchik, A. Hinrichs, and W. Kent, "The ucsc genome browser," *Current protocols in bioinformatics / editoral board, Andreas D Baxevanis*, 2009.

[30] Apache, "parquet." http://parquet.incubator.apache.org, 2017.

[31] L. Battle, R. Chang, and M. Stonebraker, "Dynamic prefetching of data tiles for interactive visualization," *Technical report, MIT-CSAIL-TR-2015-031, EECS Department, MIT*, 2015.

[32] Amplab, "Indexedrdd." https://github.com/amplab/spark-indexedrdd, 2016.

[33] V. Leis, A. Kemper, and T. Neumann, "The adaptive radix tree: Artful indexing for main-memory databases," pp. 38–49, ICDE, 2013.

[34] H. Samet, *The Design and Analysis of Spatial Data Structures*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1990.

[35] B. Paten, M. Diekhans, B. J. Druker, S. Friend, J. Guinney, N. Gassner, M. Guttman, W. James Kent, P. Mantey, A. A. Margolin, M. Massie, A. M. Novak, F. Nothaft,

L. Pachter, D. Patterson, M. Smuga-Otto, J. M. Stuart, L. Vant Veer, B. Wold, and D. Haussler, "The nih bd2k center for big data in translational genomics," *Journal of the American Medical Informatics Association*, vol. 22, no. 6, p. 1143, 2015.

[36] J. A. Veltman and H. G. Brunner, "De novo mutations in human genetic disease," *Nat Rev Genet*, vol. 13, pp. 565–575, 08 2012.

[37] R. Ribiero, "Advances in treatment of de-novo pediatric acute myeloid leukemia," *Curr Opin Oncol.*, vol. 26, pp. 656–62, 2014.

[38] BigDataGenomics, "avocado." https://github.com/bigdatagenomics/avocado, 2017.

[39] BigDataGenomics, "Gnocchi." https://github.com/bigdatagenomics/gnocchi, 2017.

[40] C. A. Meyer and X. Liu, "Identifying and mitigating bias in next-generation sequencing methods for chromatin biology," *Nature Reviews*, vol. 15, pp. 709–721.

[41] A. Morrow, V. Shankar, D. Petersohn, A. Joseph, B. Recht, and N. Yosef, "Convolutional sinks for transcription factor binding site prediction," in *NIPS workshop*, 2016.

[42] E. Sparks, S. Venkataraman, T. Kaftan, M. Franklin, and B. Recht, "Keystoneml: Optimizing pipelines for large-scale advanced analytics," in *arXiv preprint*, 2016.

[43] M. Niemenmaa, A. Kallio, A. Schumacher, P. Klemel, E. Korpelainen, and K. Heljanko *Bioinformatics*, vol. 28, pp. 876–877, 2012.