

Interactive Exploration on Large Genomic Datasets

Eric Tu



Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2016-111

<http://www.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-111.html>

May 16, 2016

Copyright © 2016, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Acknowledgement

I would like to thank my advisor, David Patterson, for his invaluable advice throughout my graduate career, and Anthony Joseph, for his guidance in the Big Data Genomics research group. I would also like to thank Frank Nothaft, who brought me onto the Big Data Genomics research group in the AMPLab during my undergraduate career and has mentored me throughout the years. I thank Alyssa Morrow, for her unyielding hard work and dedication in working together with me. Finally, I would like to thank my friends and family, who have always supported and inspired me.

Interactive Exploration on Large Genomic Datasets

by Eric Tu

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences,
University of California at Berkeley, in partial satisfaction of the requirements for the
degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

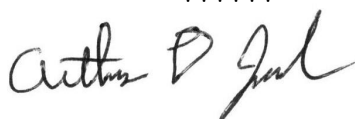
Committee:



Professor David Patterson
Research Advisor

May 15, 2016

(Date)



Professor Anthony Joseph
Second Reader

May 15, 2016

(Date)

Interactive Exploration on Large Genomic Datasets

by

Eric Tu

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor David Patterson, Chair

Professor Anthony Joseph

Spring 2016

Abstract

Interactive Exploration on Large Genomic Datasets

by

Eric Tu

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor David Patterson, Chair

The prevalence of large genomics datasets has made the the need to explore this data more important. Large sequencing projects like the 1000 Genomes Project [1], which reconstructed the genomes of 2,504 individuals sampled from 26 populations, have produced over 200TB of publically available data. Meanwhile, existing genomic visualization tools have been unable to scale with the growing amount of larger, more complex data. This difficulty is acute when viewing large regions (over 1 megabase, or 1,000,000 bases of DNA), or when concurrently viewing multiple samples of data. While genomic processing pipelines have shifted towards using distributed computing techniques, such as with ADAM [4], genomic visualization tools have not.

In this work we present Mango, a scalable genome browser built on top of ADAM that can run both locally and on a cluster. Mango presents a combination of different optimizations that can be combined in a single application to drive novel genomic visualization techniques over terabytes of genomic data. By building visualization on top of a distributed processing pipeline, we can perform visualization queries over large regions that are not possible with current tools, and decrease the time for viewing large data sets. Mango is part of the Big Data Genomics project at University of California-Berkeley [25] and is published under the Apache 2 license. Mango is available at <https://github.com/bigdatagenomics/mango>

Table of Contents

1. Introduction	2
1.1 UCSC Genome Browser	4
1.2 Integrated Genomics Viewer	5
2. Motivation	7
3. Architecture	10
3.1 Architecture Summary	10
3.2 ADAM Summary	11
3.3 Mango Stack	12
4. Visualization Techniques	16
5. Evaluation	17
5.1 Workload Description	17
5.2 Local Parity	19
5.3 Horizontal Scalability	21
6. Future Work	24
6.1 Advanced Prefetching Mechanisms	24
6.2 Achieving Interactive Latency	25
6.3 Notebook Form Factor	25
7. Conclusion	25
8. Bibliography	26

1. Introduction

Genome visualization tools enable users such as clinicians and researchers to gain insight into genomic data through a visual interface, a more intuitive alternative to bioinformatics command-line tools.

Genomic data takes many different forms, such as alignment data (short snippets of DNA output by sequencer machines), variant data (areas of the genome where the genetic makeup of a person differs from the population average), and annotation data (interesting areas on the genome such as the nucleotide bases coding for a gene). All these data types are referenced by their location on the genome. Visualizations can take the form of both aggregated/summary statistics when looking at large regions, or an in depth look at data objects when looking at small regions.

Ideally, genomic visualizations can provide a powerful means of discovering underlying trends in genomic data. During variant interpolation analysis, for example, a user wants to find correlations between variations in a person's genetic makeup to the physical expression of traits in that person.

Visualization can serve as a way to propose hypotheses during variant interpolation. Typical workflow involves starting with a hypothesis, viewing the data to see if the data supports that hypothesis, then using the result of the visualization to iteratively refine and ask better questions about the data. Through this analytic workflow, users can arrive at a meaningful conclusion.

Another use case for genomic visualization is in the callset refinement stage of the GATK (Figure 1) [18], the current most commonly used pipeline for genomic analysis. The GATK is split into three stages: preprocessing of raw sequencing data, variant discovery from that analysis-ready sequencing data, and refinement of the variants discovered. The analysis to refine the variant callset is currently done using command line tools, using text transformation Unix utilities such as sed and awk run over terabyte-sized genomic datasets. Visualization can provide a much more natural means of viewing and stringing together filters on data to uncover the important genetic variants in the data.

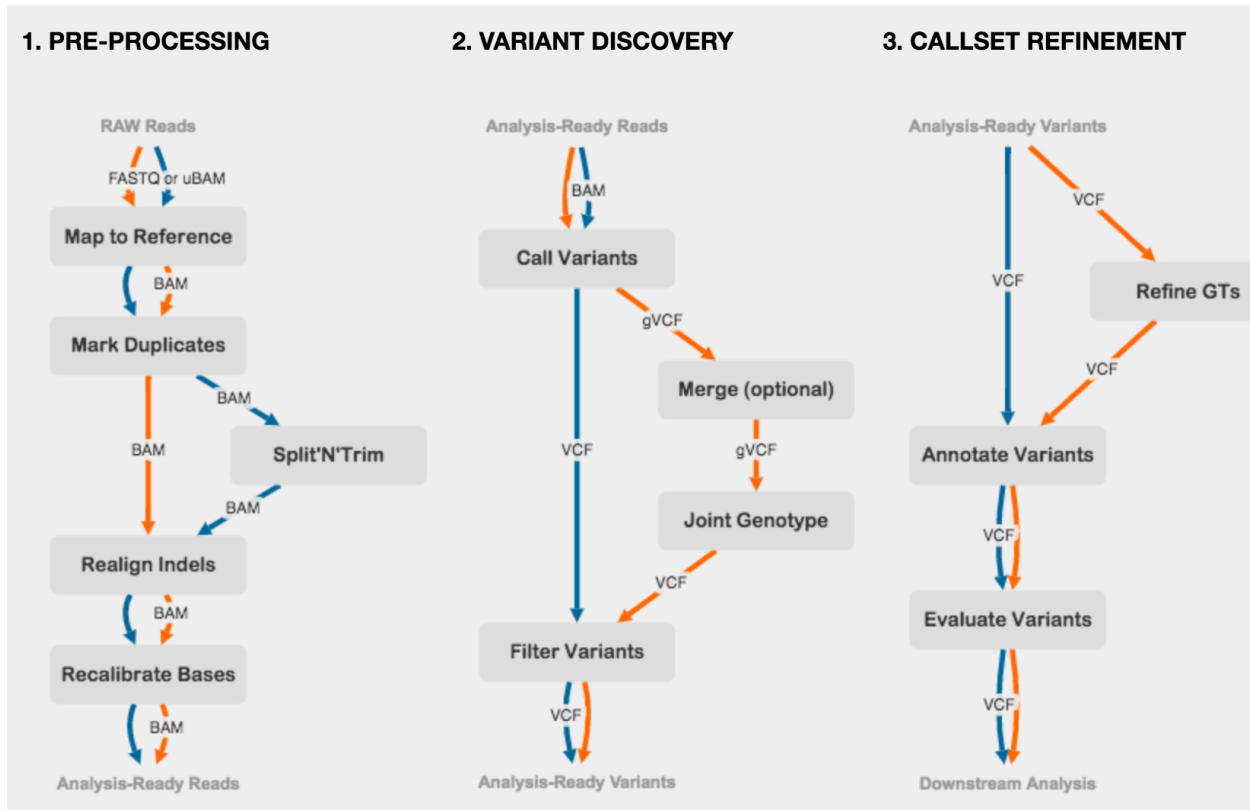


Figure 1. The Genome Analysis Toolkit, the most commonly used genomic pipeline [26]. Genomic visualization can improve analysis in the callset refinement stage.

To better describe different tools within the space, we start by defining four desirable criteria.

Functionality: What a user can use a visualization tool for? A wide variety of genomic data types exist, as do storage formats for those data types. Because genomic data is highly complex, users may want to filter by several different fields at once or recompute data through the tool itself. We define a visualization tool with high functionality as one that supports many different operations on different types of common data formats.

Intuitiveness: How visually informative and usable is the tool? Genomic visualization tools attempt to leverage the number of pixels on a screen to strike a balance between simple visual representation and robust conveyance of information. For example, viewing a megabase (Mb) of a chromosome can be visually overwhelming if each base is displayed. An open question in the research area is how to create visually meaningful that summarize many different dimensions or data.

Performance: How responsive is the tool to user input? In modern web applications response times must be subsecond to be considered interactive. Typically, a delay of 500ms results in decreased user activity and discovery of trends in the data [2].

Scalability: Can a tool support genome datasets of 100's to 100,000's of samples? How does a tool scale to both the storage needed to store genome datasets, and the computation needed to perform analysis over these datasets?

Two popular genome browsers are the UCSC Genome browser [6] and the Integrated Genomics Viewer [3]. We begin by briefly summarizing features of these browsers in terms of the four criteria above.

1.1 UCSC Genome Browser

The UCSC Genome browser [6] (Figure 2) is available as a web service, and offers access to a database of sequence data and annotations for over 40 species. Users can also upload their own files and view them through the web service. The UCSC Genome browser can be thought as a catalog of curated genomic data.

Functionality: The UCSC Genome browser contains a large number of tracks that can display many types of gene annotations from different sources on the same screen at once. These tracks may include expression, gene, sequencing, and regulation information. These tracks are precomputed previously, and from a preselected list of data, and clicking on objects in the tracks takes users to a new web page containing detailed information on that webpage.

Intuitiveness: The UCSC Genome browser utilizes a “track view” of genomic data, where all genomic data is aligned along a one-dimensional genomic axis. Information such as the reference genome, sequenced reads, variants, and genomic features are displayed as horizontal tracks, or rows, aligned along a genomic coordinate axis. Users navigate through their data by moving along the genomic axis, moving left or right, and zooming in and out of regions of interest. Tracks are selected from a predefined menu, which toggles these tracks on and off.

Performance: The UCSC Genome browser is implemented as a web service. Its views, while static, return within a few seconds, displaying different amount of objects based on the range of the region. For catalog-esque browsers like the UCSC Genome browser, computation and bandwidth is limited servicing data from a MySQL database.

Scalability: The UCSC Genome browser supports viewing curated datasets, and users can upload datasets up to 500MB in size. Larger datasets require users to upload data into their own web servers, and hook into the UCSC Genome browser. Visualizations are precomputed, and data is serviced from a traditional MySQL database, which does not scale to user requests in interactive latencies.

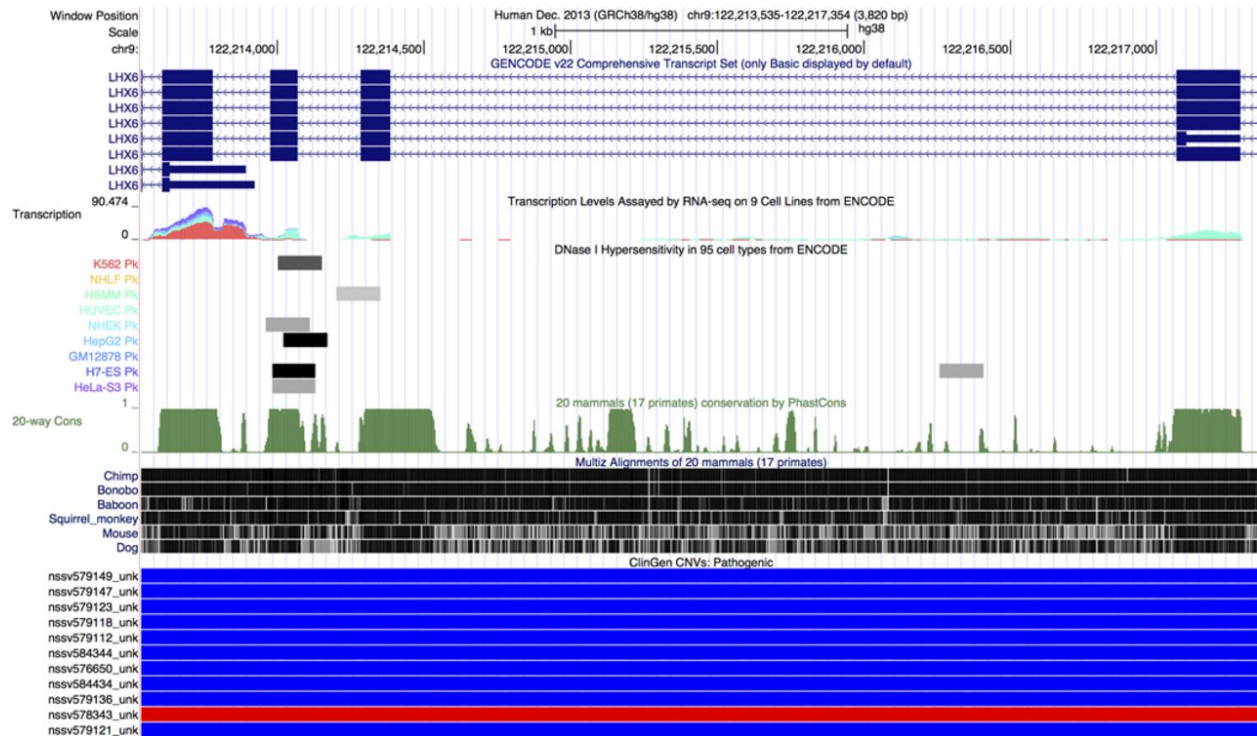


Figure 2. UCSC Genome Browser [6]. Shown here are multiple tracks of catalogued information overlaid over the same genomic coordinates.

1.2 Integrated Genomics Viewer

The Integrated Genomics Viewer [3] (Figure 3), is a desktop application that allows visualization of a wide variety of data formats. Researchers use the results of experiments to verify data and analyze trends.

Functionality: IGV allows users to load data from common legacy genomics formats. The tool concurrently displays many fields of the type of data being displayed, as opposed to the approach taken with the UCSC Genome browser, where the fields/information of the data object are obtained by viewing separate web pages. These different genomics formats can be viewed side-by-side and navigated concurrently.

Intuitiveness: IGV, like the UCSC Genome browser, displays data in tracks. However, unlike the UCSC Genome Browser, each individual interaction does not yield a refresh to a different web page. Instead, IGV allows interactive panning of the data. Interactions typically involve panning on small regions less than 1,000 base pairs (bp), and zooming in and out at higher resolutions. Different information is displayed at different resolutions.

Performance: IGV implements a special multiresolution format to scale to the size of the data being displayed. This multiresolution format allows quick zooming and panning of data, which can contain multiple samples, but must be computed via an external command line tool. However, high resolution views of the data are limited to regions of a few thousand bases, with

variant data being displayed at regions less than 1,000 bp, and alignment data being displayed at regions less than 40,000 bp. Ultimately, for applications like IGV, computation and storage is limited by the power of the computer being run on. Performance of this tool will be further explored in the evaluation section of this report.

Scalability: IGV is a single node application, so storage and computation resources are limited to those of a given node. While IGV can load in any dataset that fits on local disk, the browser hangs and crashes when fetching regions of very large files (>100GB). These issues are heightened when concurrently loading and viewing multiple large files.

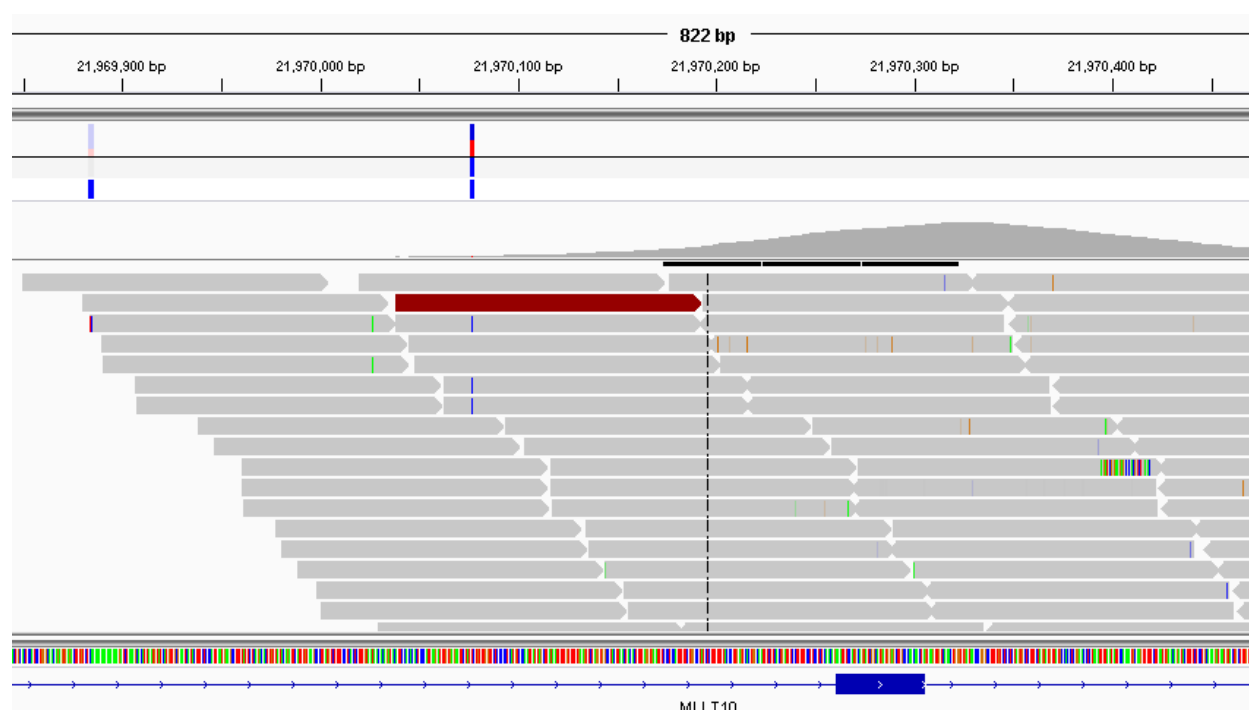


Figure 3. Integrated Genome Browser. Shown here are overlapping snippets of DNA (reads) and the reference nucleotide bases compared in tracks over the same genomic coordinates.

While genomic visualization is a research area in and of itself, the current tools that perform visualizations have similar shortcomings, namely performance. These visualization tools are intended for a single computer environment and lack computational resources to provide interactive speeds.

Both IGV and UCSC Genome Browser require data to be viewed at very high resolution, making it difficult to explore course trends in data. Technical limitations force tools to compromise features for latency. When loading large regions, IGV takes several seconds to load information, and hangs when loading multiple files at once.

These technical limitations constrain genomic data exploration. A lack of low resolution views of data means that users must already know what to look for. Given the large expanse of a genome, getting lost in nonsensical data is a real issue. Even if a user does know where to look, current genome browsers provide basic statistics and display the data, but don't allow ad-hoc analyses to further explore a dataset. Any additional filter or data transformations need to be done outside the genome browser on the source files and reloaded back in.

As a result, popular genome browsers are meant for referencing or checking work, and often only display curated datasets. To achieve acceptable latencies, browsers must preprocess data and precompute summary statistics or tiled views of data. Other tools such as iobio [8] use streaming and sampling techniques that limit the type of computation that can be performed and may limit the accuracy of the visualization. There is no room for ad-hoc analysis.

Furthermore, the amount of genomic data produced is only increasing. Large sequencing projects such as the Department of Veterans Affairs' Million Veterans Project [7] expects to generate over three orders of magnitude more data than the 1000 Genomes Project [1].

2. Motivation

The question then, is to ask whether these technical limitations, namely low latency computations on very large datasets, have been solved before in other areas.

Cluster computing frameworks are able to scale to large datasets. Iterative batch processing systems such as MapReduce, Hadoop, and Spark [9, 12, 13] distribute parallel tasks on a cluster with fault tolerance. Query execution engines such as Cloudera's Impala [11] provide a SQL interface on top of Hadoop to quickly run read-mostly queries on a distributed dataset.

Scientific systems such as ADAM [4] use these types of frameworks (Spark) on commodity hardware to accelerate genomic analyses, demonstrating a 28X speedup over non-distributed genomic pipelines. ADAM also uses commodity formats such as Apache Parquet [5], a columnar store, and Apache Avro [10], a commodity schema and data serialization system, to minimize storage of large genomic datasets on disk.

Given the applicability of big data technologies to scientific systems, we believe the same technologies can be applied to scientific visualization systems.

However, existing big data analytics are not optimized for interactive data exploration, with big data analytics consisting of summary statistics from large scale batch jobs, not low latency queries. While users want to glean as much insight into data as possible, there is not much notion of visually exploring data in an application setting, with visualizations typically being summaries of a long-running jobs (Figure 4).

This workflow is not interactive, a direct result of computing frameworks which are not optimized for serving data to visualization applications. Big data visualizations focus on reporting large scale results from batch cluster computing frameworks, but do not focus on

interactive latencies. This long latency is because the large batch operations run are often compute/write heavy, as opposed to read-heavy.

Interactive queries on business intelligence, read-heavy workloads exist, such as with Impala [11] or SparkSQL [14], but none were built for the needs of a visualization application, which requires more user interaction than displaying a static representation of a computed result. Most visualizations reside in a notebook form factor, such as in Hue [15] (Figure 4), and are made with primarily data reproducibility in mind, not data exploration.

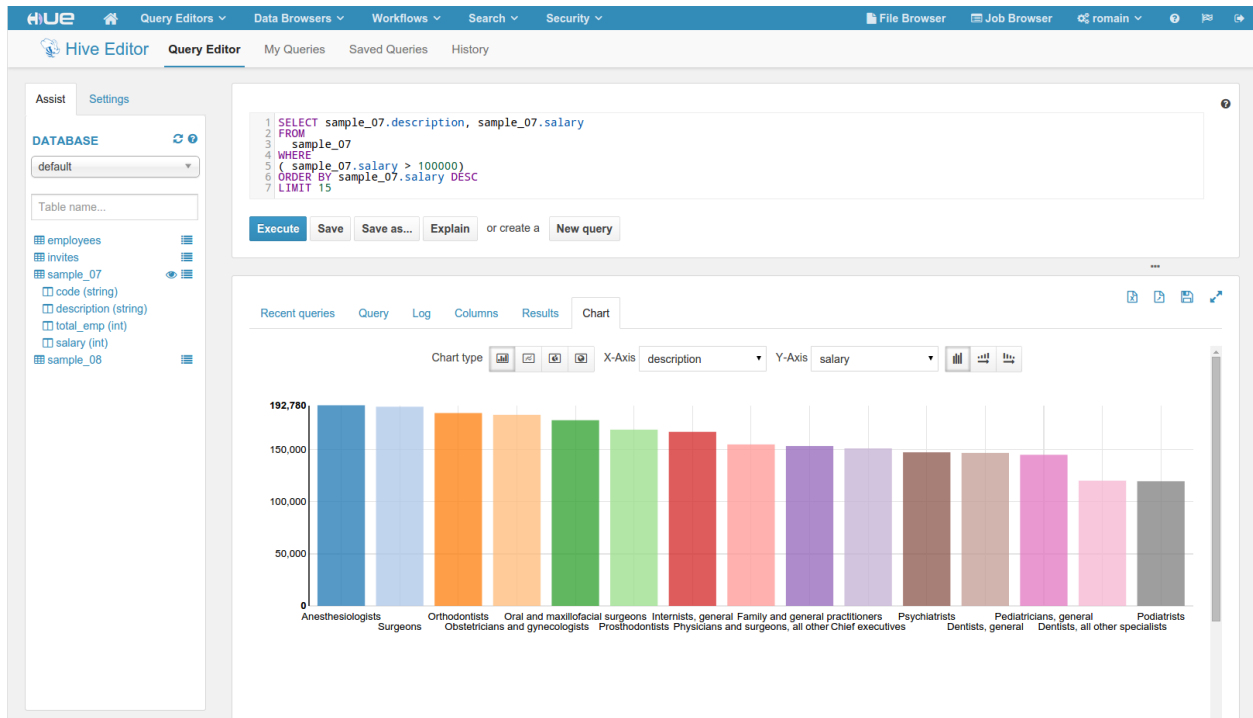


Figure 4: Apache Hue. Summary chart for the salaries of different occupations.

Meanwhile, exploratory data visualizations provide low latencies with high user interaction, but typically reflect a static dataset, and do not scale. These datasets are small (typically a CSV file), but typically highly complex in interaction, and differ from big data workloads, which are large but focus more on summary graphs. Tools such as Tableau [16], and visualizations built on top of D3.js [17], allow trends to be discovered within data through building custom visualizations. Figures 5 and 6 show custom visualizations developed using these tools. These visualizations help users raise hypotheses about data and enable further investigation through additional interaction beyond an initial static view.

Users typically performing exploratory data analysis do not know what they are looking for and are able to sift through the dataset at interactive speeds, as opposed to big data workloads, where users are looking at summary statistics and are unable to dig further without running a high latency job. This subsecond interactive speed is necessary to allow for meaningful user exploration, with complex interactions such as brushing and linking, and panning and zooming.

With increased interactions, a delay of even 500ms results in decreased user activity and less effective discovery of data trends [2].

The differences then, between exploratory data visualizations and big data analytics can be thought of as the tradeoff between specialized visualizations on small datasets, and generalized computation on large datasets.

In this report, we aim to combine the two, providing the power and scalability of big data analytics with the interactivity of exploratory data visualizations in the context of genomics. In doing so, we can address all four desirable criteria of genomic visualizations proposed in Section 1. Commonly used big data technologies allow easier development of features, which can improve *functionality*. Exploratory data visualizations increase *intuitiveness* by showing more informative views and interactions with data. Big data technologies improve *performance* by latency through faster distributed computation. Finally, big data technologies provide *scalability*, enabling support for very large datasets.



Figure 5: Tableau. This figure shows a brushing and linking interaction, where any interaction on one of the three custom data visualizations will be reflected on the other two.

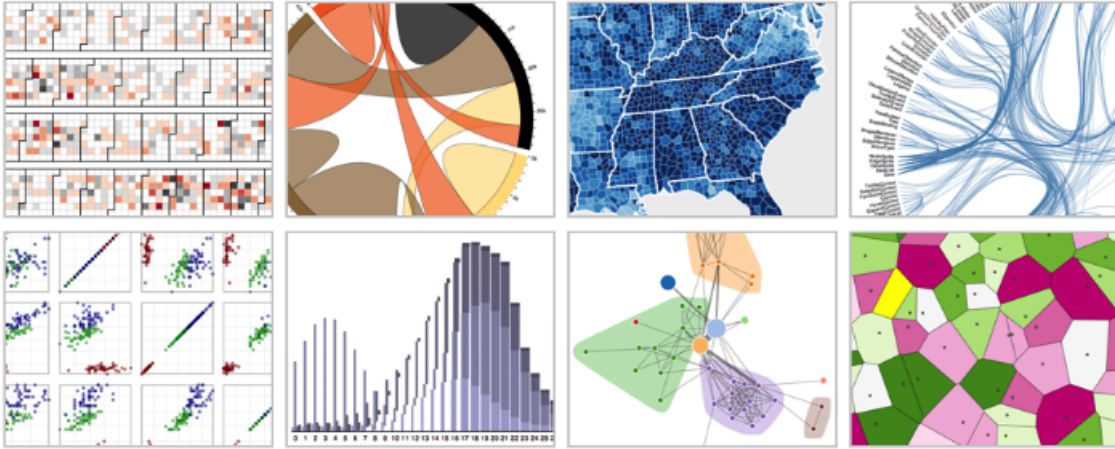


Figure 6: The custom visualizations used in D3.js. Each box represents a different style of visualization on a different dataset.

3. Architecture

3.1 Architecture Summary

We build Mango on top of the ADAM stack [4]. ADAM provides a set of core APIs to bring in genomic data into commodity big data technologies, storing data on disk using Parquet [5]. On top of this base, we build optimizations for accessing highly selective subregions of data with low latency and intelligent caching to materialize likely viewed regions of data prior to user request. All these services have a frontend that contains interactive user-facing genomic visualizations, built on lightweight web technologies such as D3.js [17].

We build on top of ADAM for practicality and extensibility. A user who already has datasets on a machine or cluster can simply point Mango to the data and start using it. Because Parquet is the main building block, Mango also easily plugs into the Hadoop [12] ecosystem, which supports Parquet as the default columnar storage format. This format can be transferred to other Hadoop based pipelines other than ADAM.

Although the Spark platform was not initially intended for handling low-latency visualization workloads, maintaining the ability to integrate visualization into an existing ecosystem allows us to utilize the same genomic algorithms provided by ADAM in a unified environment. From a usability standpoint, we believe building on top of Spark simplifies development of genomic visualizations because we are able to combine both pipelined workloads and visualization workloads with the same language and frameworks.

3.2 ADAM Summary

ADAM is a distributed genomics analysis pipeline that provides a well supported platform upon which to build genomic visualization. ADAM [4] uses Apache Avro [10] to clearly define data schemas, which are stored on disk in Apache Parquet [5]. Genomic data is then loaded into ADAM, which leverages Spark's distributed execution [13] and fault tolerance to perform computation.

As Figure 7 shows, ADAM uses a “narrow waist” of schemas to segregate an application's use of genomic formats from underlying data distribution mechanisms. ADAM supports both legacy genomic formats (e.g., VCF, BED, SAM, BAM [19]) and Parquet formats, and provides a toolkit to convert between the two. While Parquet formats reduce end-to-end computation cost 63% (utilizing Amazon EC2 compute) compared to legacy formats, support for legacy formats is important to bridge the worlds of genomics to big data systems.

Parquet is an important building block of Mango. We make extensive use of Parquet predicates and projections to only read in the fields and blocks of data we need to display. Legacy formats used in existing genome browsers do not have such support for predicates and projections. Parquet also allows for significant compression of data on disk, offering about 1.25x compression on gzipped BAM files and 1.66x compression on gzipped VCF files.

These files are loaded via ADAM APIs into Spark RDDs, resilient distributed datasets, upon which genomic algorithms can be run. Specifically, these algorithms correspond to the preprocessing stage of the GATK pipeline (Figure 1).

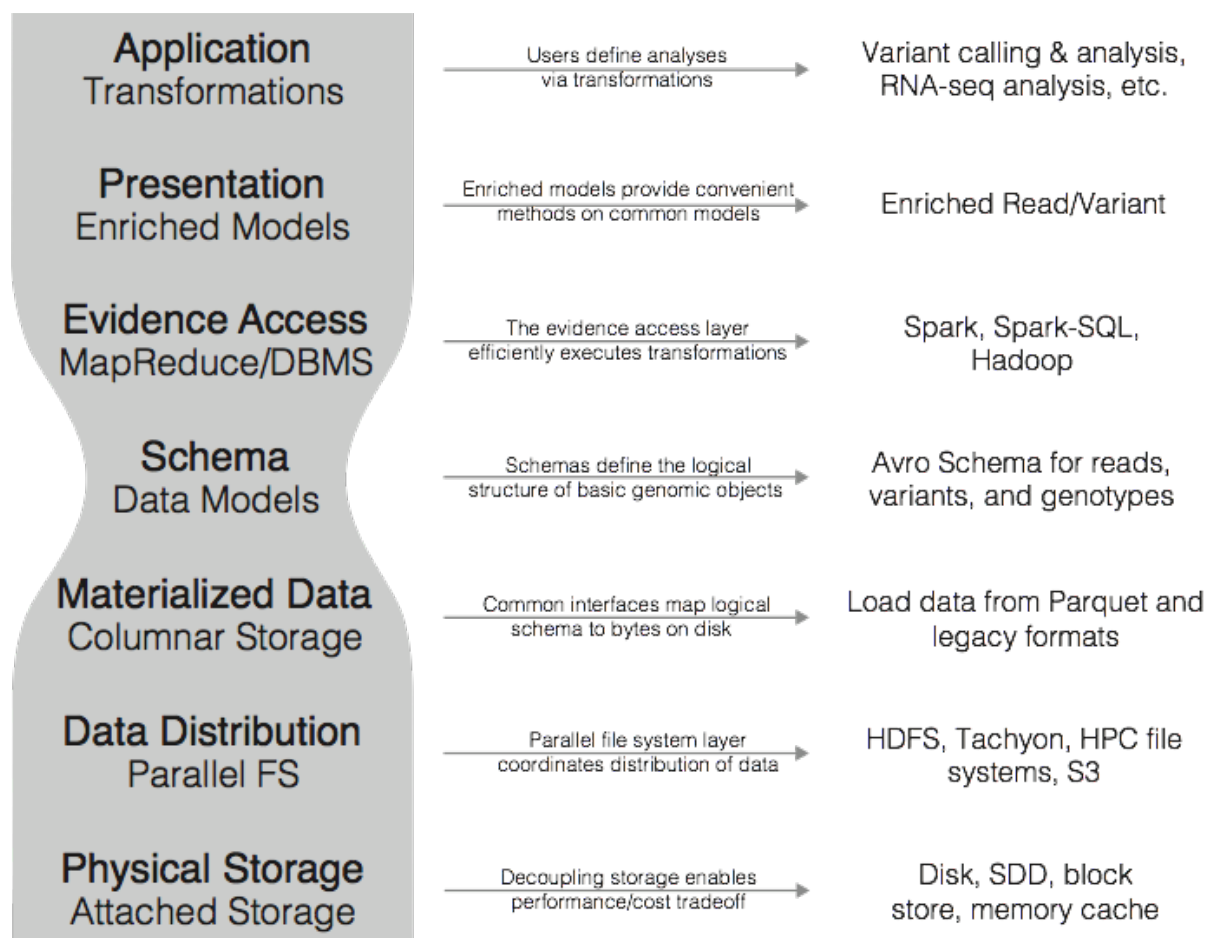


Figure 7: ADAM Stack of different open source technologies

3.3 Mango Stack

Mango also utilizes a stack-oriented model, as Figure 8 shows. The elements of the stack are put into three functional groups: Cluster, server/master node, and end host/client. Mango supports modularity in its visualization and optimizations, allowing different levels of the stack to be replaced or used in existing genomic pipelines. For example, while Mango provides novel visualizations, these can be swapped out in favor of different frontend options.

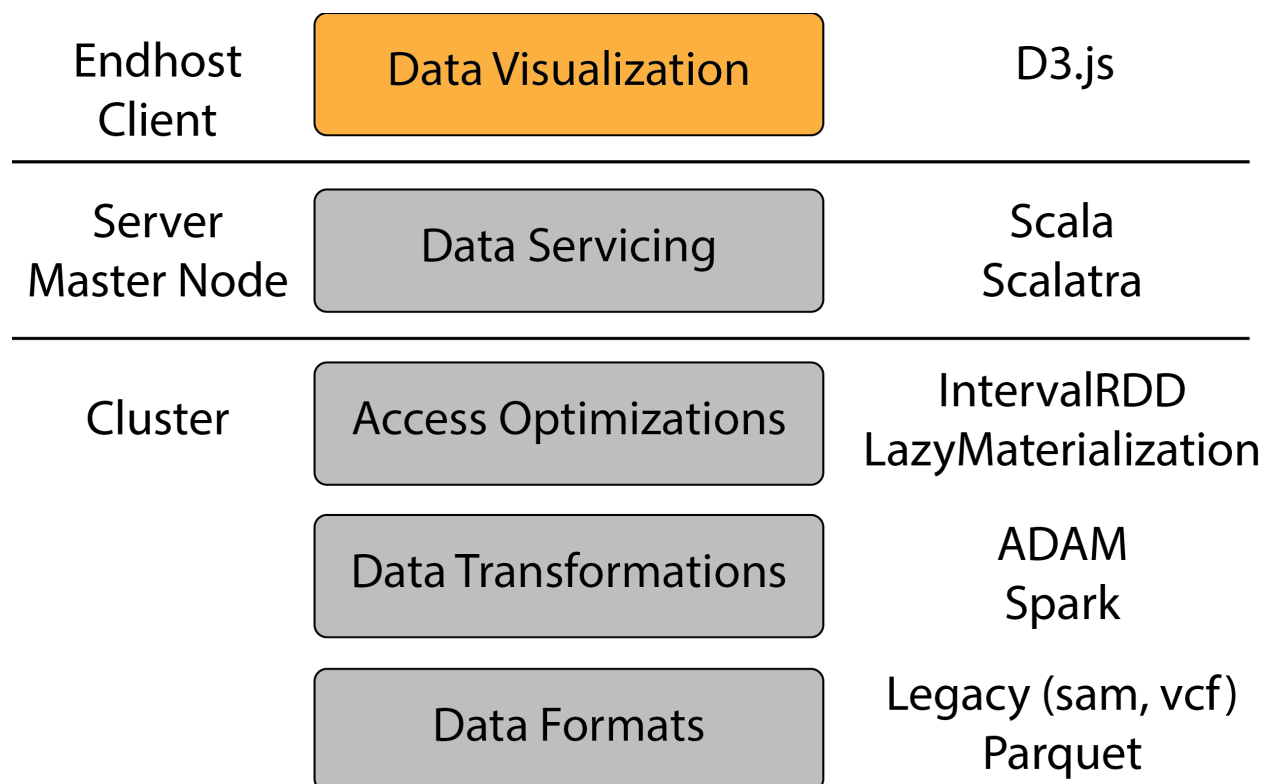


Figure 8. Mango Stack. The elements (boxes) are put into three functional groups shown on the left: Cluster, server/master node, and end host/client. The technologies used to build the stack elements are shown on the right.

3.3.1 Cluster

The cluster group includes the ADAM core and Parquet, combined with data access optimizations. Our primary contribution at the cluster layer is the implementation of the *Interval RDD*, a version of Spark's resilient distributed dataset (RDD) optimized for two-dimensional range queries, and a caching layer called Lazy Materialization.

Interval RDD

We implement a variant of Spark's RDD, called an *Interval RDD*, that organizes records into an interval tree and optimizes overlapping range queries. Performing this type of query in Spark's primary abstraction for data (an RDD) currently requires a full scan of data. Consider the query which gets all 2-dimensional segments overlapping or contained within the start and end value of an interval:

```
RDD.filter(rec => rec.start < interval.end && rec.end > interval.start)
```

Even if records are sorted by start value, worst case performance requires all data to be scanned. Meanwhile, interval trees provide $O(\log(n))$ search for intervals within a given range. Interval trees insert and delete nodes in the same manner as a binary search tree. However, every node in

an interval tree stores the maximum value found in its subtree. Therefore, during search, a given subtree is only traversed if the range minimum is less than the maximum of the subtree.

Genomic range lookups are a key use case for Interval RDDs. Records are keyed by intervals defined over an axis of information. For genomic data, the axis is the location on a chromosome, and the key is the tuple of the start and end nucleotide base positions on that chromosome. Common use cases for filtering an overlapping range include inspecting sequencing coverage over a chromosomal region, visualizing a range of the genome, and comparing different samples to a reference genome.

Lazy Materialization

To efficiently cache and monitor data being queried by the client, we implement a cache manager called Lazy Materialization that enables incremental data fetching and formatting into a working set. This working set, which can be an Interval RDD or any Spark primitive, is lazily populated with data that is stored in memory. The motivation behind this is a workload common to visualization applications—have low latency and access a very selective interval of data to visually present to a user.

Lazy Materialization fetches data from storage per query, then uses materialized views to bring into memory the immediate region around the queried interval. The layer keeps track of past queries and keeps in memory the data associated with them. While initial queries take the overhead time to load from storage, future queries on the same interval will instantly return from being cached in memory.

In addition, we implement a basic prefetching algorithm to asynchronously fetch and materialize data during idle periods of user interaction. After outstanding user requests have been completed, the prefetching scheme materializes regions of data to the left and right of the currently viewed region into the working set. Parameters for prefetching include 1,000 bp to the left and right of the current window for high resolution raw data views and 100,000 bp to the left and right for summary statistics.

The motivating factor behind this design is that visualization applications tend to access data in localized areas. Because of the coordinate nature of genomic data, 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.

Lazy Materialization dynamically fetches data depending on what a user requests and does not require the entire dataset to be loaded immediately. As opposed to running batch operations or transformations over an entire dataset, users are not sure what data they may access over a session in a visualization application. By constructing a caching layer that scales to user requests in a visualization application with a dynamically constructed working set, we can use only the memory and compute resources we need for our workload.

Working Set Primitives

As mentioned above, the Lazy Materialization structure is either backed by an Interval RDD or a Spark primitive. In this project, we use different structures for different types of data.

Alignment data is loaded using Interval RDDs. Common operations over alignment data involve querying a region for all overlapping two-dimensional segments, and performing computation over the data. The Interval RDD tackles both these considerations by respectively storing data in an optimized structure, and provides the same rich procedural processing semantics as RDDs. This means that ADAM's APIs and algorithms for RDDs can be applied.

Variant data is loaded using *DataFrames*, introduced in Spark as part of the SparkSQL package [14]. DataFrames provide the semantics of relational processing within the Spark ecosystem, while providing a query optimizer to speed up simple data access patterns. Because variant data is the summary and result of computed alignment data, access patterns are simpler than alignment data and fall more in line with declarative queries.

Variants are typically single point mutations, not intervals. In addition, visualization involves looking at regions of the genome that have a high frequency of variants. DataFrames are a good fit for these two properties: DataFrames provide optimized filter and count operations that can quickly query a region of variant data or return a count of how many variants are in that region.

3.3.2 Server/Master Node

The main function of the data servicing layer is servicing client requests and formatting data into JSON for client consumption.

To service data, we use Scalatra [20], which provides a lightweight web application framework for Scala. We choose Scalatra because of its simple integration with both the Scala programming language, and with frontend technologies. Using a Scala-based framework allows for Scala, Spark, and web service code to be written within the same class.

Scalatra provides specialized templating styles that provide access to Scala functions and compile into HTML. This means that Scala and Spark code can be launched from within a web template.

We also implement our bookkeeping structures within this group. Information relating to a user's current session, such as what regions of data he/she has queried for, is stored here, as is the information for the locations of different files that are loaded throughout the course of application use.

3.3.3 End host/Client

The end host/client group include front end technologies used to generate visualization. We use D3.js, which provides custom visualizations within a frontend environment. D3.js supports HTTP requests for JSON data, which provides a simple abstraction upon which to ask for data. We currently use HTTP requests as the interface between the server/master node and the end host/client.

4. Visualization Techniques

Here we describe the client front-end portion provided with Mango and the usefulness of the visual elements displayed.

Mango's first aim is to provide feature parity with IGV. This goal means that Mango supports viewing the same type of data as IGV (reads, variants, genomic features/annotations, reference) in track format, and can display multiple tracks of data at once.

The main difference comes with what increased performance enables. Mango allows regions greater than 40 kbp to be viewed on the fly without precomputation. Because of this expanded ability for visualization, we can begin to ask what exactly are meaningful visualizations at low resolution views. While Mango can display the same exact data as one would look at in IGV, just over a larger region, we dedicated the bulk of this increased performance to computing aggregate statistics over data.

This primarily means displaying frequencing/count information. At low resolutions (>40 kbp), displaying all records provides little intuition to the user, and *will also crash modern web browsers*, because millions of objects are being displayed at a time. Displaying frequencing/count information provides users with an idea of where to look. For alignment data, this means regions with a high frequency of mismatching bases (the base on this read differs from the base on the reference genome at this position) as Figure 10 shows. For variant data, this means regions with a high frequency of non reference/reference variants. (This variant differs from the base on the reference genome at this position).

As such, records are not displayed until a user zooms into a high resolution, specifically 1,000 bases. Individual records are overlaid with colors that that display the mismatching bases on reads (Figure 9), and the non ref/ref variants. For reads, users can filter by fields such as mapping quality and choose to only look at mismatching bases, insertions/deletions, or the reads themselves. For variants, users can filter by the type of variant (ref/ref, ref/alt, alt/alt) as well as mapping quality.



Figure 9. All reads records being displayed, with colors overlaid for mismatching bases. Near the top of the page, the count/frequency of reads at a position is shown, overlaid with the count/frequency of mismatching bases.

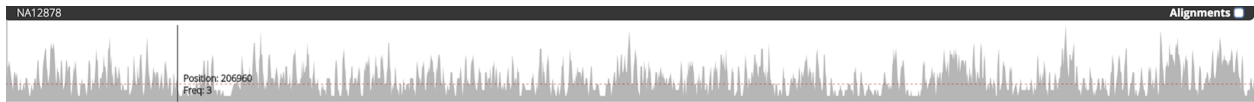


Figure 10. Low resolution view of reads, displaying the count/frequency of reads at a given position. Users can zoom in further to regions of interest.

We also strove to improve user intuition with initial interaction. Current tools provide no clearly suggested initial interaction with the tools. Users do not know what to look for, and what can be looked for first. Mango strives to provide visual cues, through form autocomplete boxes and summary visualizations that provide an overview at the data loaded. We do this by visualizing genomic metadata, such as sequence dictionaries that provide information about the relative size of chromosomes.

5. Evaluation

Local experiments to evaluate parity against single node genome browsers were run on a 2015 MacBook Pro with a 3.1 GHz Intel Core i7 processor and 16 GB 1867 MHz DDR3 memory. Evaluation of Mango in a distributed environment was performed on a 64-node cluster, each with Intel E5-2670 2.6 GHz 8 core CPU, 256 GB RAM and 4 1-TB HDFS hard drives.

5.1 Workload Description

Evaluation criteria described in this section assess query patterns for both variant and alignment workloads. Chosen query patterns for genomic data were modeled after typical scientific workload trends discovered in a user study of 18 domain scientists in ForeCache [24]. These

assumptions model scientific query patterns as an iteration of zooming in and panning across a given region of interest. The assumptions made in scientific workload exploration include the following:

- Users initially view zoomed out regions of their data and hone in on smaller regions during exploration.
- Users exhibit zooming and panning in a non-random pattern. This feature implies that previously viewed regions inform the user of which regions to view next.

Following these assumptions, we constructed separate workloads for both variant and alignment data. Figure 11 demonstrates the chosen query patterns for both datasets.

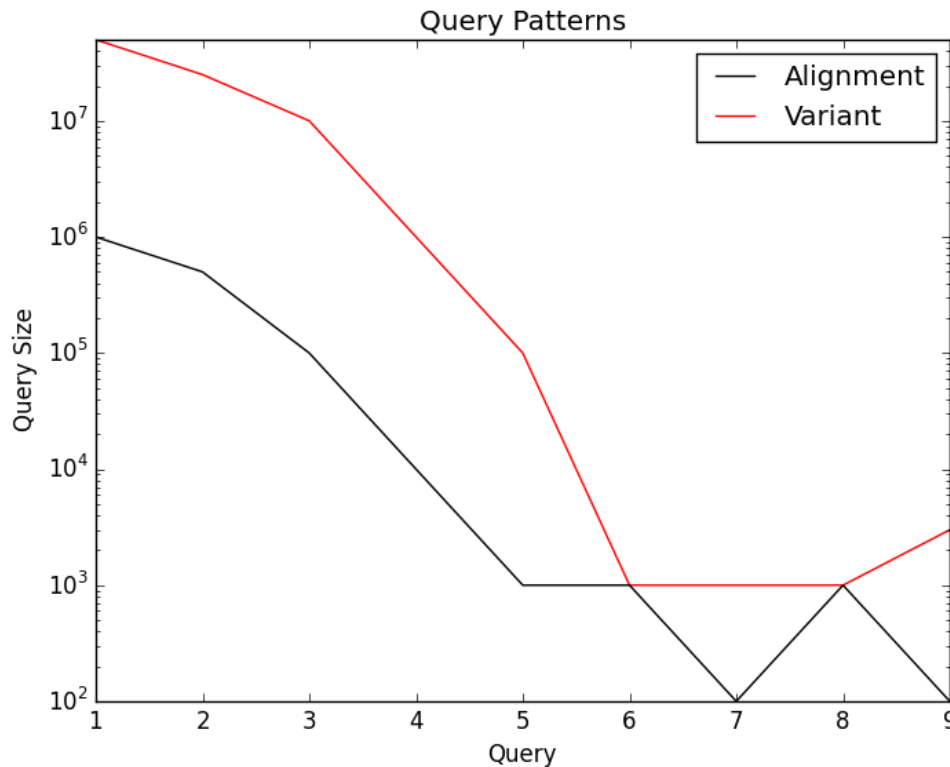


Figure 11. Query zoom levels across variant and alignment data workloads for the nine queries.

For both workloads, queries 7 to 9 indicate panning out of the current view region at high (100 to 3,000 bp) resolution.

The variant workload, shown in red, is initialized at a 50,000,000 bp region on one chromosome. The query zooms in to a region of 1,000 bp (queries 1 through 6), pans (queries 7 and 8), then zooms out to 3,000 bp (query 9). The variant workload is tested at lower resolution than alignment data. This difference is because variant data summarizes alignment data, and therefore displays more information in a smaller number of pixels.

The alignment data workload, as Figure 11 shows in black, has a higher resolution query pattern than variant data. Alignment workload starts at 1,000,000 bp as the coarsest level of granularity,

and zooms in to 100 bp (queries 1 through 5), then panning (query 6 and 8), and zooming (queries 7 and 9). Because alignment data is used to analyze pointwise mutations, the resolution for alignment data is often much higher.

5.2 Local Parity

In this section, we demonstrate local comparison to IGV, the current state of the art genome browser. Like Mango, IGV is intended to visualize personal variant and alignment data. For variant workloads, we demonstrate latency on a 18-GB variant file from 2504 samples of chromosome 20 from the 1000 Genomes Project [1]. For alignment workloads, we demonstrate latency on a 318 MB single sample consisting of alignment data from chromosome 20. For both workloads, chromosome 20 was chosen due to its availability of medium coverage alignment data from 1000 Genomes Project. Chromosome 20 is a smaller chromosome, consisting of only 62,435,965 bases.

5.2.1 Alignment Data

Figure 12 shows latency on a single node for an alignment workload against IGV, Mango, and Mango without prefetching. Figure 12 illustrates that IGV does not show any data until the 10K query, which takes 9,411 ms. Otherwise, IGV provides a constant latency of around 2,000 milliseconds. Mango without prefetch shows a spike in latency for the 1K Pan query, as this query triggers a cache miss and requires retrieving raw data from persistent storage. However, there still exists a spike in latency at the 1K Pan query with prefetching enabled. This increase may be due to the overlapping between prefetch and user queries, as the preformatting of raw alignment data was not yet complete.

In cases for Mango with and without prefetching enabled, there is a dip in latency at the 10K query that rises back after the first 1K query. This drop is due to low latency summary statistic computation (frequency) being computed from 1M down to 10K. Higher resolution data at 1K triggers collecting all data records, a more costly operation.

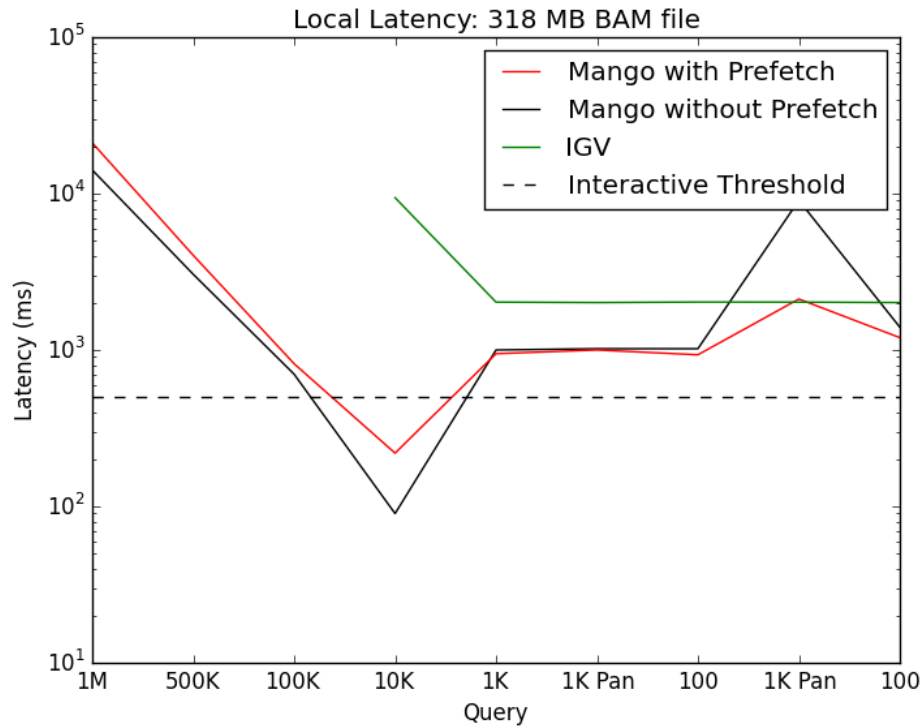


Figure 12. Local response to alignment data workload. IGV begins displaying data at 1K resolution. Mango incurs higher initial overhead, but provides the ability to view extremely large regions.

5.2.2 Variant Data

Figure 13 shows local latency results on the variant workload. Average latency is slightly higher compared to a method without prefetching, because the working set of data is larger with prefetching. As a result, fetching the same region over the larger working set results in the Spark job having to check more blocks of memory for data satisfying the specified predicate. This behavior is exacerbated by the unionAll operation in DataFrames, which increases the number of tasks when querying on the unioned dataset. This higher prefetch latency is more noticeable locally because a local machine has fewer executors than a cluster to distribute the larger number of Spark tasks. These issues can potentially be addressed through more intelligent tuning of Spark parameters and prefetching sizes (see Section 6.1).

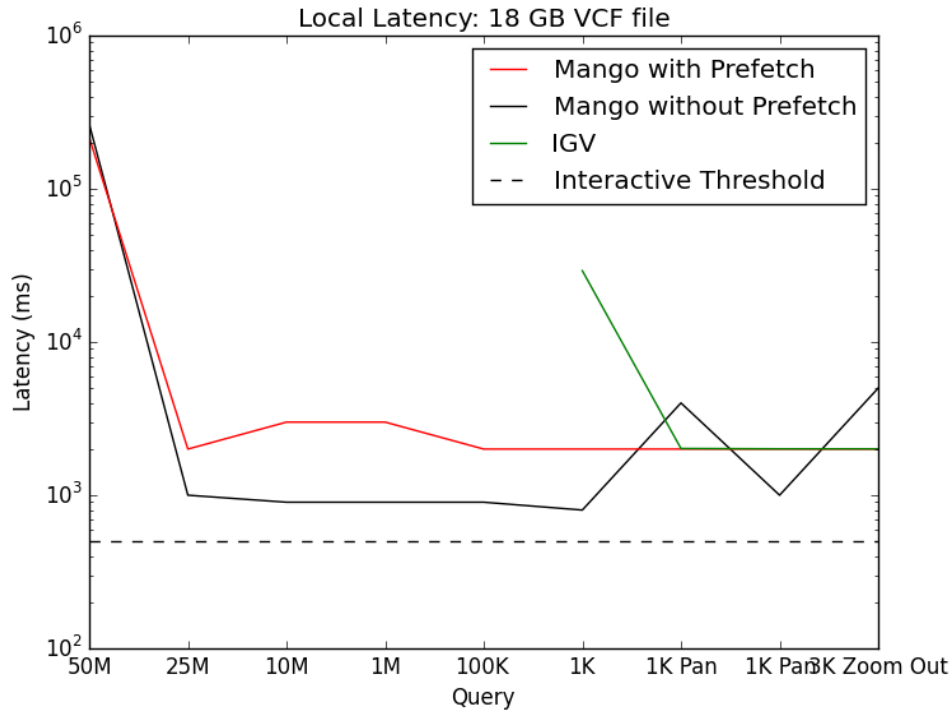


Figure 13. Local Variant Workload. IGV begins displaying data at 1K resolution. Mango incurs higher initial overhead, but provides the ability to view extremely large regions.

5.3 Horizontal Scalability

Because there is no explicit comparative software that supports distributed genome browsing, we compare Mango in a distributed setting across a variable set of computational resources with and without the prefetching mechanism.

Tests on both variant and alignment workloads run against 8, 16, 32, 64, 128, and 256 executors, each with 12 GB memory, where we assigned each to one CPU core.

5.3.1 Alignment Data

Alignment data workload was run against two 250 GB high coverage full genome alignment files of samples NA12878 and NA12891 from the 1000 Genomes Project. These two were the largest files produced by the 1000 Genomes Project. Results query sample NA12878 across chromosome 20.

Figure 14 shows latency results from chromosome 20 against the alignment workload. The dip at the 10K query and spike at 1K Pan can be explained similarly to the local evaluation by speedup from summary statistics and cache miss, respectively. However, cache misses incur a much higher cost in a distributed setting due to fetching remotely from HDFS. Iterative addition of executors for alignment data shows significant benefit when loading from persistent storage, shown in latency reduction in queries 1M and 1K Pan from 8 to 256 executors.

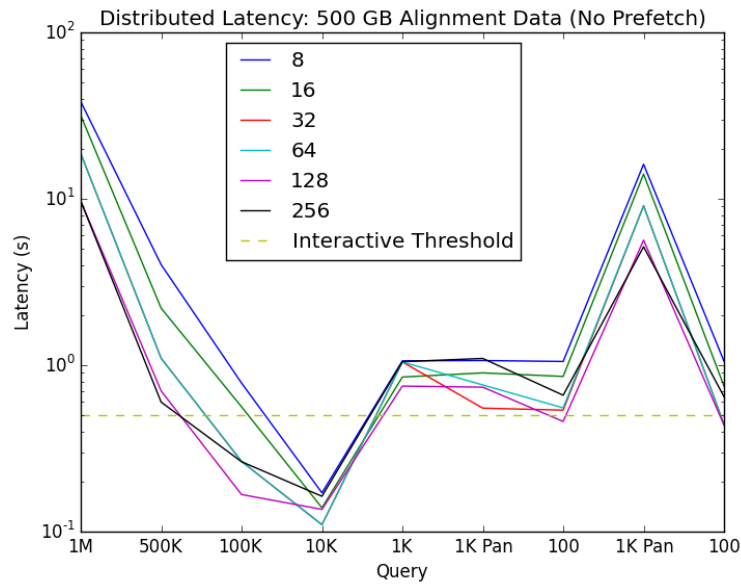


Figure 14. Alignment query workload with no prefetching. Increasing the number of executors reduces latency.

Figure 15 shows the latency results from alignment data with prefetching enabled. With prefetching enabled, 1K Pan does not incur real time data fetching from persistent storage, thus maintaining close to interactive thresholds.

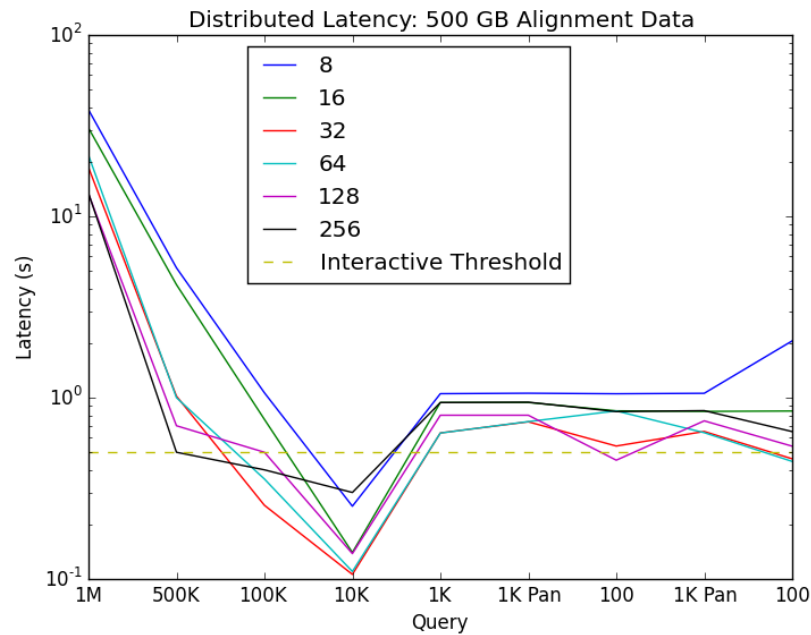


Figure 15. Alignment query workload with prefetching. Increasing the number of executors reduces latency.

5.3.2 Variant Data

Figures 16 and 17 show latency results for the variant workload without and with prefetching, respectively, over the variants for 2,504 people across chromosome 1. This represents worst case performance, because this file being queried for is 106 GB, the largest variant file from the 1000 Genomes Project.

Similarly to the local variant workload, we see a spike in latency in the 1K and 1K Pan queries around 9 seconds which are triggered by cache misses. Unlike the alignment record workload, variant queries maintain approximately interactive threshold with 16 or more executors, with diminishing benefit past 32 executors. Figure 17 demonstrates latency with prefetching enabled. Latency for prefetching is greater due to the DataFrame issue previously mentioned, but is less of an issue due to having more executors present to handle tasks.

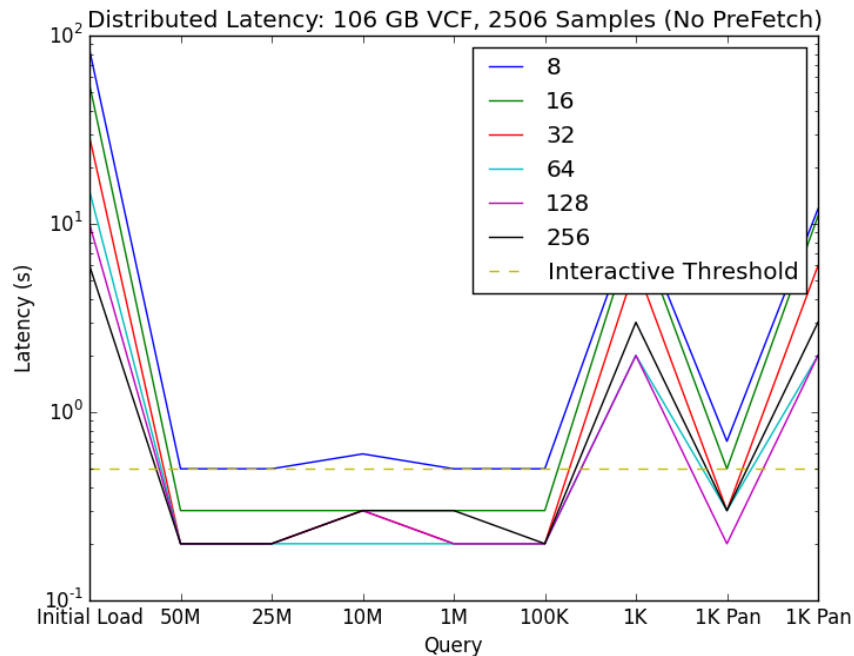


Figure 16. Variant query workload with no prefetching. There is diminishing benefit past 32 executors.

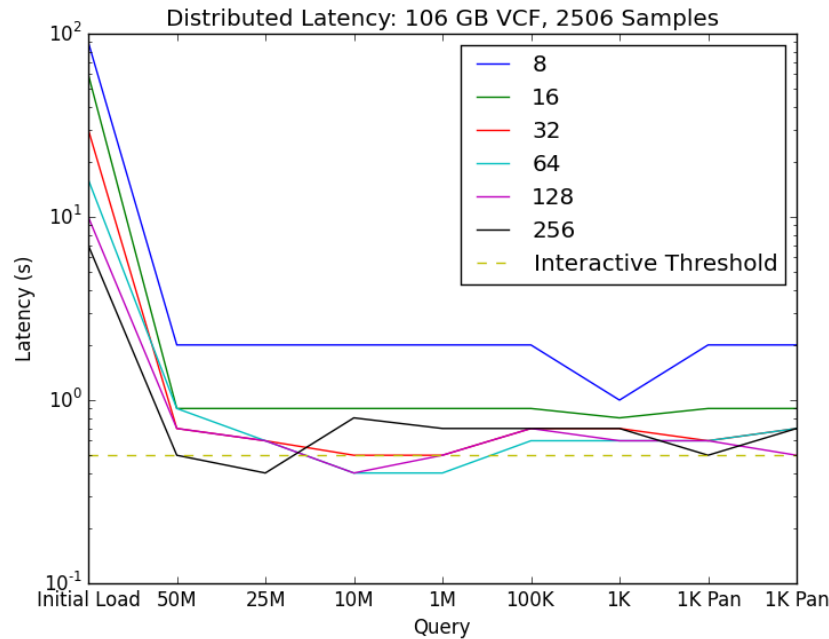


Figure 17. Variant query workload with prefetching. There is diminishing benefit past 32 executors.

6. Future Work

6.1 Advanced Prefetching Mechanisms

Prefetching mechanisms implemented in this report were simple, only fetching regions to the left and right of the currently viewed region to speed up panning and zooming interactions. We can improve prefetching mechanisms by completing a user study and building a prefetching model based on past user interactions. From user-defined data, we can determine the size of the region to materialize and the maximum latency allowed for running prefetching jobs. These user interactions can be modeled via common visualization prediction algorithms. Hotspot models recommend regions that are viewed frequently by other users [22]. Momentum models prefetch data with the assumption that a user's next move will be the same as his/her last [23]. Coupled with the amount of compute and storage resources given to Mango, we can better determine the size and location of prefetched regions.

A second strategy to improve prefetching includes a recommendation model that materializes regions of data similar to regions the user has viewed. ForeCache [24] calculates l2 distance between data tiles to recommend regions similar to what the user is currently viewing. With a genome browser, we can define more specific prediction methods because we do not support such a general workload. Such suggestion algorithms would include patterns in insertions, deletions, and distribution in mismatch density.

6.2 Achieving Interactive Latency

While Mango outperforms existing tools over large regions of data, latencies do not always meet the 500ms threshold required for interactive applications. Currently, while a region of data may be persisted on a cluster and return quickly when queried, that query still requires a Spark job to be run, leading to latency on the order of hundreds of milliseconds. We can avoid this redundant request by caching the results of the server at the client. If the data being requested already exists in the cache, the application can simply locally access that data without waiting for a round trip call to the server. We can implement the same prefetching mechanisms described for the server to further improve the utility of a client cache.

6.3 Notebook Form Factor

Reproducibility is important in scientific analysis. While Mango provides an interactive genome browser interface, user interaction is difficult to reproduce. We can extend Mango to a notebook form factor that visually displays the result of a query. This form factor requires integration into an existing notebook tool such as the Spark notebook [21], which can already perform analysis from ADAM.

7. Conclusion

In this report we presented Mango, a genome browser that leverages distributed computing for interactive genomic exploration. Mango scales horizontally, achieving latencies comparable to local state of the art browsers on both variant and alignment workloads, while optimizing for latency with additional compute resources in a distributed environment.

We demonstrated the optimizations required to accommodate a visualization workload of high data selectivity and low latency in a distributed environment. Mango selectively materializes data on user request and organizes the data efficiently for future access.

With the rise of available genomic data, it is crucial that clinicians and researchers have sufficient tools to interactively analyze their data. Mango brings scalable visualization that will enable the necessary speed and features needed to meaningfully explore these increasing datasets.

8. Bibliography

- [1] 1000 Genomes Project Consortium. "A global reference for human genetic variation." *Nature* 526.7571 (2015): 68-74.
- [2] Z. Liu, and J. Heer. "The effects of interactive latency on exploratory visual analysis." *Visualization and Computer Graphics, IEEE Transactions on* 20.12 (2014): 2122-2131.
- [3] H. Thorvaldsdóttir, J. T. Robinson, and J. P. Mesirov. "Integrative Genomics Viewer (IGV): high-performance genomics data visualization and exploration." *Briefings in bioinformatics* 14.2 (2013): 178-192.
- [4] F. A. Nothaft, M. Massie, T. Danford, Z. Zhang, U. Laserson, C. Yeksigian, J. Kottalam, A. Ahuja, J. Hammerbacher, M. Linderman, M. Franklin, A. D. Joseph, and D. A. Patterson. "Rethinking data-intensive science using scalable analytics systems." *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 2015.
- [5] Apache. Parquet. <http://parquet.incubator.apache.org>.
- [6] M.L. Speir, A.S. Zweig, K.R. Rosenbloom, B.J. Raney, B. Paten, P. Nejad, B.T. Lee, K. Learned, D. Karolchik, A.S. Hinrichs, and S. Heitner. (2016) The UCSC Genome Browser database: 2016 update. *Nucleic Acids Res.*, 44, D717–D725.
- [7] U. D. of Veterans Affairs. Million veteran program (mvp). <http://www.research.va.gov/mvp>.
- [8] iobio. <http://iobio.io>
- [9] M. Zaharia, M. Chowdhury, M.J. Franklin, S. Shenker, and I. Stoica. "Spark: Cluster Computing with Working Sets." *HotCloud* 10 (2010): 10-10.
- [10] Apache. Avro. <http://avro.apache.org>.
- [11] M. Kornacker, A. Behm, V. Bittorf, T. Bobrovitsky, C. Ching, A. Cho, J. Erickson, M. Grund, D. Hecht, M. Jacobs, and I. Joshi. "Impala: A Modern, Open-Source SQL Engine for Hadoop." *CIDR*. 2015.
- [12] Apache. Hadoop. <http://hadoop.apache.org>.
- [13] J. Dean, and S. Ghemawat. "MapReduce: simplified data processing on large clusters." *Communications of the ACM* 51.1 (2008): 107-113.

- [14] M. Armbrust, R.S. Xin, C. Lian, Y. Huai, D. Liu, J.K. Bradley, X. Meng, T. Kaftan, M.J. Franklin, A. Ghodsi, and M. Zaharia. "Spark sql: Relational data processing in spark." *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 2015.
- [15] Hue. <http://gethue.com>
- [16] Tableau. <http://www.tableau.com/>
- [17] M. Bostock, V. Ogievetsky, and J. Heer. "D³ data-driven documents." *Visualization and Computer Graphics, IEEE Transactions on* 17.12 (2011): 2301-2309.
- [18] A. McKenna, M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis, A. Kernytsky, K. Garimella, D. Altshuler, S. Gabriel, M. Daly, and M.A. DePristo. "The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data." *Genome research* 20.9 (2010): 1297-1303.
- [19] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, and R. Durbin. "The sequence alignment/map format and SAMtools." *Bioinformatics* 25.16 (2009): 2078-2079.
- [20] Scalatra. <http://www.scalatra.org/>
- [21] Spark Notebook. <http://spark-notebook.io/>
- [22] R. Li, R. Guo, Z. Xu, and W. Feng. "A Prefetching Model Based on Access Popularity for Geospatial Data in a Cluster-based Caching System." *Int. J. Geogr. Inf. Sci.*, 26(10):1831-1844, Oct. 2012.
- [23] S. Yesilmurat and V. Isler. "Retrospective Adaptive Prefetching for Interactive Web GIS Applications." *Geoinformatica*, 16(3):435-466, July 2012.
- [24] L. Battle, C. Remoco, and M. Stonebraker. "Dynamic Prefetching of Data Tiles for Interactive Visualization." *Computer Science and Artificial Intelligence Laboratory Technical Report MIT-CSAIL-TR-2015-031* (2015).
- [25] Big Data Genomics. <http://bdgenomics.org/>
- [26] GATK Best Practices. https://www.broadinstitute.org/gatk/guide/bp_step.php