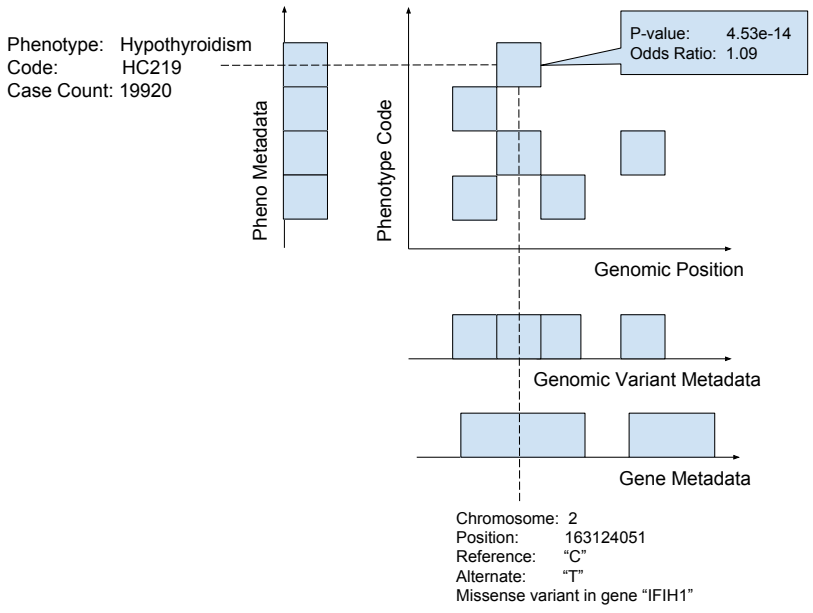
Stanford Rivas Lab Blog Post November 2017

The Rivas Lab at the Stanford University School of Medicine evaluated two databases—MongoDB and SciDB—for the purpose of serving as a backend for the Global Biobank Engine. Based on this evaluation, the Rivas Lab elected to proceed with SciDB. We discuss the results of our evaluation below.

Rivas Lab Requirements

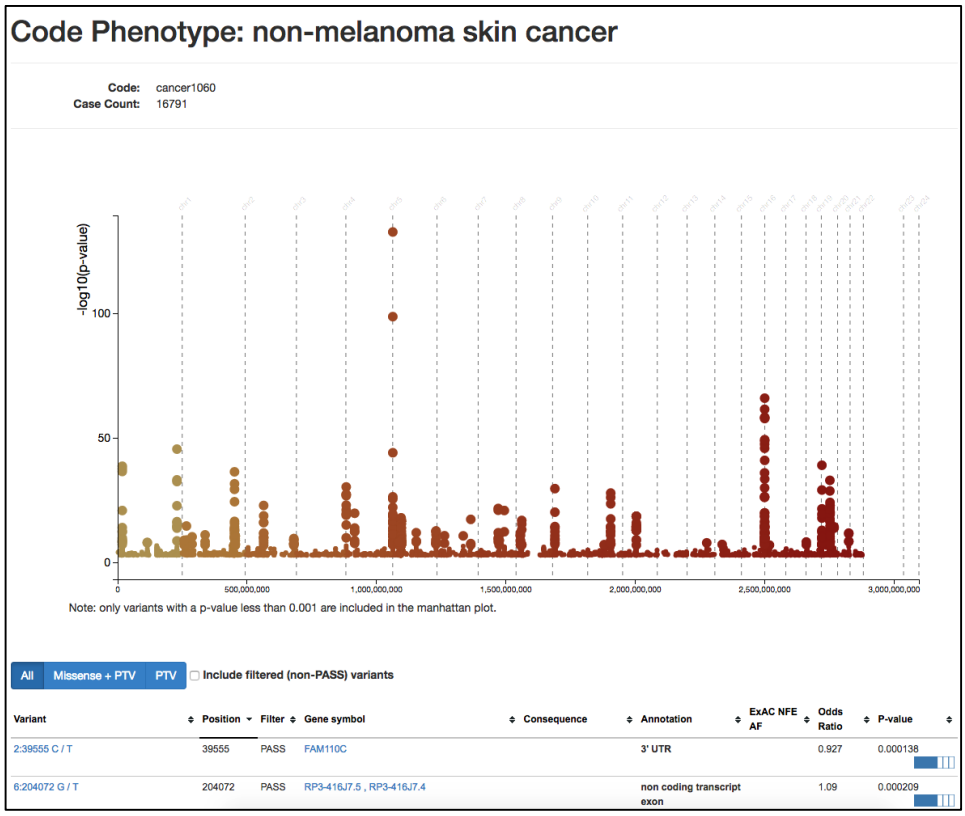
The Global Biobank Engine is an online service for browsing case-control association results from the UK Biobank genotype-to-phenotype association studies. The data structure is reminiscent of a large matrix with various disease Phenotype Codes - cancer, hypertension, arthritis, and so on - on the vertical axis and genomic variations on the horizontal. In this sparse matrix, phenotype-variant pairs are associated with a set of computed statistical likelihood scores. Both axes of the large matrix are “decorated” with additional metadata: information about the conditions investigated and additional annotations for genomic variants and regions. Note, the diagram is not scaled relative to the data volumes - the matrix is by far the largest component, exceeding 75% of the total data size:



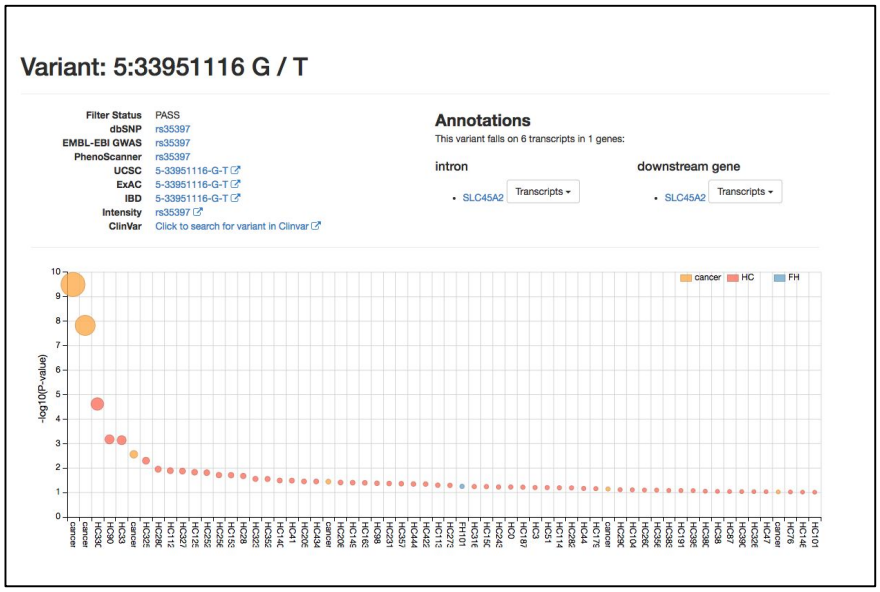
The objective of the browser is to make such data easily browsable and searchable. Two main views through the data correspond to the “horizontal” and “vertical” slices through the above matrix. The “Coding” view presents the most significant variants associated with a particular condition - a horizontal slice. At the same time, the “Variant” view presents the most significant traits for a

particular genomic locus - a vertical slice.

For example, a horizontal slice for phenotype “non-melanoma Skin Cancer” is presented below:



And a vertical slice for a particular variant looks like so. Now the x-axis represents various phenotype codes that are significantly associated with this variant - cancers, non-cancer conditions, and family history:



In both cases, the backend selects only the most significant associations to prevent display clutter and allow the page to load faster.

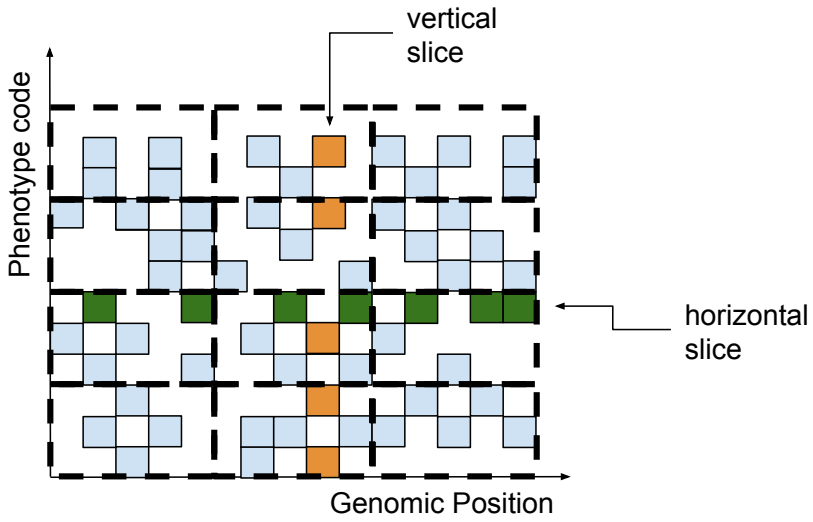
The overall requirements for the Biobank Engine infrastructure were as follows:

- Support rapid selection of “horizontal” and “vertical” slices interactively
- Support sub-slices on metadata: zeroing in on a specific genomic region, i.e. a gene
- Allow new phenotypes to be added to the data as more associations are collected
- Support additional background analytics on the data, which may be embarrassingly parallel (Fisher’s exact test, Bayesian model comparison) or not (principal component analysis)

The Rivas Lab team initially started out with a solution based on MongoDB, but then reached out to Paradigm4 to evaluate SciDB as a potential alternative.

The Rationale for SciDB

Since the data is clearly well-represented as a sparse matrix, SciDB's native array data model is a good fit. SciDB efficiently supports both sparse and dense data with its unified array data architecture. In particular, SciDB's multidimensional array clustering (MAC™) storage engine clusters the data into rectilinear chunks in the [Phenotype Code x Position] space:



This scheme ensures that both horizontal and vertical slices can be retrieved by only visiting a small fraction of the total chunks. That significantly reduces the number of required disk scans and allows for fast query response times, even on modest hardware. New phenotypes are easily added to the collection by updating the chunks at the edge of the matrix (top). Chunks that are not affected by writes are not disturbed.

SciDB-Py integrates SciDB nicely with Python, and makes it a convenient web portal backend to implement. SciDB also provides analytics capabilities, such as in-database linear algebra and the Streaming interface for executing custom code in parallel.

Having established this theoretical fit, we compared the two databases from a data model and a performance standpoint.

Comparing the Technologies

In a nutshell, we can illustrate the key points of comparison between the two systems below:

Aspect	SciDB	MongoDB
Data model	Multi-dimensional arrays. Unified sparse and dense model. Strongly typed schemas.	Collections of schema-less key-value documents
Scalability	Always runs as set of K instances. Data distributed automatically.	Manual sharding possible with separate load balancers
Storage Optimizations	Data clustering on dimensions Run length encoding Array attributes stored separately	"WiredTiger" storage engine, with support for various indexes. GridFS is recommended for documents exceeding 16MB
Joins (merging of datasets)	Many flavors of joins; all computed in parallel Inner, Left outer, Right outer and Full outer equi-joins	Left outer joins when one collection is unsharded
Transactional Data Integrity	Array-level ACID	Document-level ACID
Advanced Analytics & Functional Extensibility	Native Linear Algebra C++ User-Defined Functions Streaming interface for R, Python	JavaScript MapReduce framework

SciDB places a stronger emphasis on defining a data schema upfront and then requiring inserted data to conform to the schema. MongoDB is more flexible, allowing users to insert arbitrary heterogeneous documents into the same collection. The structured schema approach of SciDB requires the database creator to do somewhat more work while loading the data, but provides lots of optimizations for read queries down the road. This approach fits well with the "load once, read often" workload that the GBE project specifies. We shy away from making the claim that one technology is "better" than the other but we can evaluate the two systems in terms of *fitness* for this particular workload.

It quickly became apparent that the association data is very regular. Whether represented as a matrix, table, or simple text, each phenotype-variant association contains the following pieces of information:

- the phenotype code
- genomic position (usually encoded as a chromosome and a genomic coordinate)
- the association p-value (floating point)
- the association odds ratio (floating point)

All associations, without exception, contain these fields. There are no cases where extra fields are added and very rarely are scores missing. In short, this data is highly structured, which means SciDB is a better fit for it. MongoDB did offer more ease of use in dealing with less structured variant

and gene annotations, but those made up a small portion of the total data volume. The advantages offered by SciDB in dealing with the large matrix far outweighed the issues with the smaller annotations. We performed a few benchmarking experiments to verify this conclusion quantitatively.

Benchmark Results

A subset of the data was loaded on one medium-size server. We tried the systems one at a time on the same machine. Measuring the storage footprint and timing a few queries generally validated our conclusions, though some results did surprise us. We've summarized our findings below:

Metric	MongoDB 3.4	SciDB 16.9	SciDB vs. Mongo
Database Footprint	34 GB	13 GB	SciDB 2.6x smaller (without SciDB compression)
Query 1: lookup a cell by Phenotype code, chrom, pos	0.1 seconds (using indexes)	0.1 seconds (using dimensions)	a wash Mongo uses indices SciDB uses dimensions
Query 2 : count phenotype associations on chromosome 15 (vertical slice)	1.3 seconds (using indexes)	0.1 seconds (using dimensions)	SciDB 13x faster
Query 3: count all phenotype associations with p-value < 0.1	122.1 seconds (no index)	1.2 seconds (no p-value dimension)	SciDB ~100x faster
Challenge Query: join phenotypes with case count > 300 with significant pval, with VARIANTs where effect is "stop gained"	Not sure if possible	23 seconds	SciDB enabling capability

Storage Footprint

This useful barometer is far too often overlooked in benchmarks. A compact on-disk representation not only saves room but also requires fewer disk reads to access, which may yield faster queries.

We first performed all the indexing and clustering to optimize the queries in our benchmark, then measured disk footprint. We found that SciDB used 2.7x less disk space. This is likely due to SciDB's use of schemas, as opposed to Mongo's schema-less approach. Taking the "p-value" field as an example, SciDB only needs to store packed payloads of 8-byte floating point numbers whereas Mongo needs, for each entry, to keep track of the presence or absence of the p-value, as well as its data type. In short, imposing structure on the data allows for a compact format.

Query Timings

We found that while SciDB uses chunking and clustering, MongoDB can make aggressive use of multiple indexes. After indexes are created, both systems can quickly return a single cell out of the large matrix (Query 1). In this regard, the two systems perform at about the same level.

When it comes time to extracting slices of data as opposed to individual data points, SciDB wins thanks to the array clustering infrastructure. This is demonstrated by Query 2.

Query 3 demonstrates how well the database performs responding to ad-hoc queries, e.g. a hypothetical case where no indexing was performed on the p-value field but, suddenly, we wanted to get an answer to a previously unanticipated question. Surprisingly the runtime difference of 100x is substantial. This is because SciDB's storage places the "p-value" field in separate chunks from the other array attributes, whereas Mongo needs to fully scan all the documents in the collection and also search each document for a numeric field called "p-value".

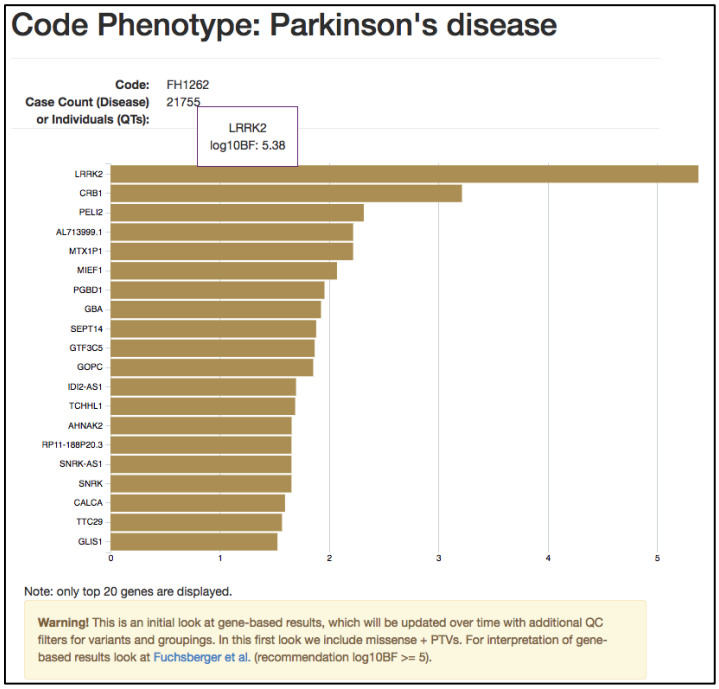
The Rivas Lab also presented Paradigm4 with a "challenge query" that consisted of first filtering all the variant metadata to match a certain criteria ("stop gained"), then filtering the phenotype metadata to select only studies with a high case count, then joining these to the matrix horizontally and vertically to extract only significant p-value associations that match those criteria. Rivas Lab staff pointed out this query might be difficult to implement in MongoDB, especially after the database is sharded - since Mongo does not fully support joins on sharded data. SciDB performed the query in less than a minute.

Advanced Analytics

After deploying SciDB and loading a substantial subset of the data, we've implemented multiple analytical workflows.

The Gene-level Streaming Workflow

We leverage SciDB Streaming to perform custom gene-level calculations in parallel. The user supplies an R or Python function to run as an argument, together with some filtering criteria, usually one or more phenotypes, and various clauses for variants (i.e. "effect = 'missense' and filter='PASS'"). SciDB then performs the required multi-dimensional slice, partitions the result by gene and evaluates the supplied function over each gene, using all available CPU cores, without having to export data out of the system. Finally, a summarized per-gene result is returned. Thus, complex user-supplied models can be evaluated over data in seconds. Below we show an example gene-level output for Parkinson's disease family history:



Linear Algebra

SciDB can also perform thresholded selection for a subset of variants, a subset of phenotypes, and potentially a subset of associations (analogous to the Challenge Query above) - to build a densified matrix of associations. Linear algebra methods can then be applied inside SciDB. We are currently working on linear algebra approaches using these data.

Summary

Bringing SciDB to bear on the problem provides multiple benefits. In the classical database sense, SciDB backs the web portal and serves up results interactively to many concurrent users. Its Multidimensional Array Clustering infrastructure ensures rapid query response times. At the same time, SciDB allows researchers to organize and evaluate complex scientific computations on the data in-situ, from large matrix linear algebra to various custom gene-level models.