2015

Web service platform to provide access to maize diversity data

Abhinav Vinnakota
Iowa State University

Recommended Citation

Vinnakota, Abhinav, "Web service platform to provide access to maize diversity data" (2015). *Graduate Theses and Dissertations*. 14734.
https://lib.dr.iastate.edu/etd/14734

Web service platform to provide access to maize diversity data

by

Abhinav Vinnakota

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Leslie Miller, Major Professor
Carson Andorf
Samik Basu

Iowa State University

Ames, Iowa

2015

DEDICATION

I would like to dedicate this work to my parents: Mr. Gupta Vinnakota and Mrs. Radha Vinnakota for their constant support throughout my life, especially during my Master’s Education away from home.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

GBS	Genotyping by sequencing
HDF5	Hierarchical Data Format version 5
API	Application Program Interface
DNA	Deoxyribonucleic Acid
REST	Representational State Transfer
JSON	JavaScript Object Notation
NSF	National Science Foundation
NAM	Nested Association Mapping
SNP	Single-Nucleotide Polymorphism
NCBI	National Center for Biotechnology Information
VCF	Variant Call Format
IDE	Integrated Development Environment
PAVs	Present and Absent Variations
JAR	Java Archive
URI	Uniform Resource Identifier
PHP	Hypertext Preprocessor
IUPUC	International Union of Pure and Applied Chemistry

ACKNOWLEDGMENTS

I would like to thank my committee chair, Dr. Leslie Miller, and my committee members, Dr. Carson Andorf, and Dr. Samik Basu, for their guidance and support throughout the course of this research.

In addition, I would also like to thank my friends, colleagues, the department faculty and staff for making my time at Iowa State University a wonderful experience. I want to also offer my appreciation to those who were willing to evaluate my work and give advice, without whom, this thesis would not have been possible.

ABSTRACT

Maize is one of the most important crops in the world. The maize genome is very complex and has huge genetic diversity. To understand the genome and its diversity the genes were sequenced. New approaches for sequencing like genotyping by sequencing (GBS) were developed which led a flood of diversity data. This led to a data storage and accessibility problem. The available data was stored in HDF5 format. Accessing the data through various tools was very complicated. The data was not easily accessible on the internet.

We tackle all these issues and provide a better experience for the user. In particular, we create a big data database with modern techniques including web service tools and an API to query the data and present it in a format useful to the maize community.

CHAPTER I

INTRODUCTION

Maize is one of the most important crops in the world. In America, 90 percent of the grain produced comes in the form of maize [1]. It is used as human food, animal feed and also has many industrial uses. Maize was chosen as a model research plant in the early 20th century partly because its traits were ideal for genetics experiments. The maize genome is very complex and has huge genetic diversity. Two maize varieties can show as much DNA sequence variation as two different species [2]. Some genes in one maize variety may not be present in another maize variety. For this reason, United States Department of Agriculture's (USDA) Plant Introduction Station in Ames, Iowa holds over 19,000 different samples of corn from around the world. Due to its importance scientists, breeders and seed companies worked on improving maize production. They created hybrid vigor, or heterosis, when two different inbred varieties are crossed to produce more robust hybrid offspring. The hybrids can greatly increase crop productivity and are used in commercial corn production. Scientists are still trying to fully understand the molecular basis of heterosis. Maize genomic sequences can be used to help them better understand the reasons behind heterosis's success.

Sequencing is the process of determining the order of nucleotide bases (A – adenine, C - cytosine, T - thymine, and G - guanine) within a stretch of DNA. Publically, the maize research community has one available reference sequence (B73 line) [3]. In 2011, Elshire *et al.* developed a new approach for sequencing genomic data called genotyping by sequencing (GBS) [11]. It is a sequencing procedure that provides a large number of markers across the genome at low cost per sample. A genetic marker is a gene or DNA sequence with a known location on a chromosome that can be used to identify individuals or species or phenotypes (observable characteristics). This approach works well for species with high diversity and large genomes like maize. The maize genome has an approximate size of around 2.4 billion base pairs [4]. GBS has led to a large scale genome sequencing by different research groups and companies. The GBS data produced is very large. The most recent data set used for this work contains 1.7E10 data points (17,280 lines at 955,690 SNPs. SNPs are single-nucleotide substitutions of one base for another that occur in more than one percent of the general population). There exists larger datasets, but they are not publicly available.

The maize community would benefit from getting access to this data. Researchers and breeders would be able to use this data to create better crops. As the data is being sequenced in different places, they are stored in different formats. Some data exists in a flat file consisting of tens of gigabytes. It is a cumbersome process if you want to figure out which nucleotide is present in a particular line at a specific position in a specific chromosome. In the original format, you either had to browse through the flat files manually to look up for the value or you needed to learn a new tool which the creator of the dataset developed. If you plan on looking at the same value in a different dataset you will have to learn another new tool or browse the dataset manually again. Browsing through these large datasets is not a simple task. Depending upon the format, the text file may contain more than 10,000 columns. Browsing through that file and identifying the specific position is a time consuming task for the researchers and breeders. At the moment, almost every research group has their own tool for their datasets. This requires a steep learning curve for anyone trying to use the data. In addition, a lot of the tools are still under active development or do not have long term funding. Lastly, running the tools on one's own personal computer is very resource intensive and will take a considerable time to load the dataset.

Our goal is to tackle all these problems and provide a better user experience when accessing these datasets. We built a foundation layer which will work with the most common data sets and be flexible to accept newer types of data, process it and present data in a format readable by both machines and users. Some users would like access to raw data while others want to ask questions about the data to get specific answers. We want to accommodate both types of requests. To do this we created a big data database with modern techniques including web service tools to allow querying of this data and present the data in a format which would be useful to the maize community.

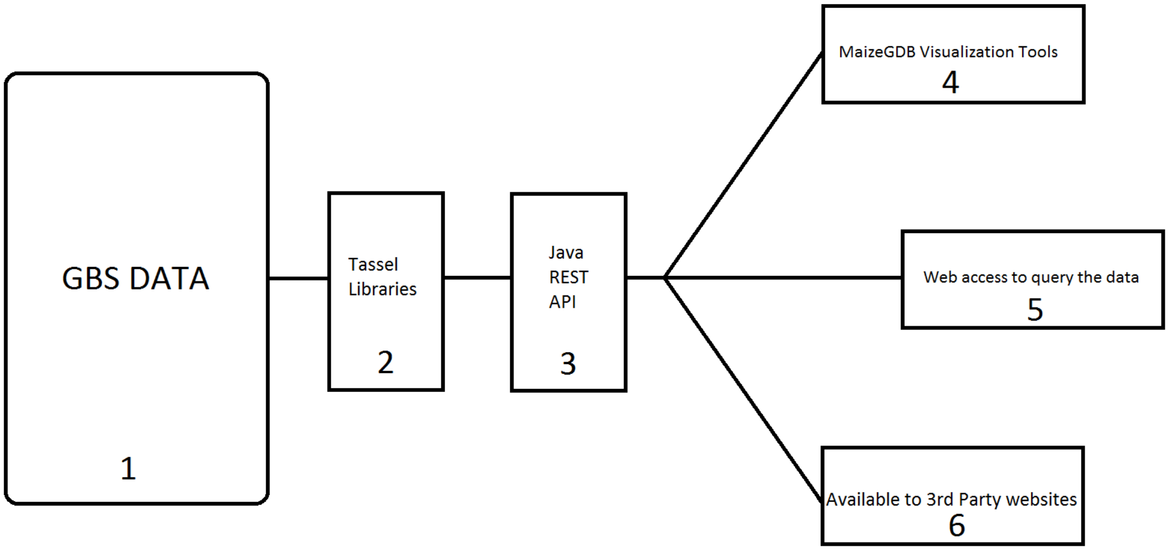

GBS DATA
1
Tassel
Libraries
2
Java
REST
API
3
MaizeGDB Visualization Tools
4
Web access to query the data
5
Available to 3rd Party websites
6

Figure 1. *High level architecture.*

The high level architecture of our software solution is shown in Figure 1. Block 1 is the GBS data which has been sequenced by various research groups. Block 2 is the layer above the raw data. It is an open source library package which provides various functions to access and query the data. The one used is called Tassel [5], it is under active development by the Buckler Lab at Cornell University. The library package is written in Java and handles HDF5 [6] data format very efficiently. HDF5 is a hierarchal data format to store biological big data efficiently.

Block 3 is the solution I developed. It is a Java wrapper around the Tassel libraries. It utilizes the functions Tassel provides and uses them to answer the queries asked by the users. This block is written in Java to ensure compatibility with the Tassel libraries. However, interaction with this solution is language independent since we implement the communication using the REST [7] API with JSON [8] objects. REST, which stands for representational state transfer is a one of the best software architecture style for creating scalable web services. JSON, which stands for JavaScript Object Notation is an open standard format that uses human readable text to transmit data objects between a server and web application. REST and JSON together makes it very easy to interact with other services and remain language independent. This block takes JSON input from web applications, processes the request and returns the output in a JSON format for the web applications to display.

Block 4 is the set of tools which will be developed by the MaizeGDB [9] group which will utilize the data and represent it in a visually appealing way to the users. I worked closely with the team to make sure they got the data in their required format for visual representation. Block 5 is a web application layer developed to demonstrate the functionality of the solution. It is a simple form which would take input from the user, convert it into JSON object and query the Java API. It gets a JSON reply from the API which it uses to display the result. To show language independence, I developed this block in PHP.

Block 6 represents the future possibilities and how any 3rd party web applications can use the data. Anyone can query the Java API and get results. They can further process the results and provide more insight into the data. They can also do multiple queries and collate them to form a better picture.

Next, I summarize my contributions to the project. I created the web service platform which will accept JSON data requests. The tool uses the requests to identify the set of Tassel functions which need to be executed. It then runs the Tassel functions, which will search the database and return the requested data. The data is used to answer the question which was asked in the request. The web service will create an answer in JSON format for the request and send it back to the requesting client.

The remainder of this thesis builds upon the core ideas introduced in this chapter. Chapter II consists of related work done and how they try to solve the data accessibility problems. Chapter III provides details on how we developed the solution. Chapter IV consists of the implementation details. Chapter V summarizes our accomplishments and discusses the scope of future work.

CHAPTER II

RELATED WORK

This chapter discusses the terminology used in the thesis, background of this work along with related work. It also describes how our approach is different from others. A lot of related work consists of research groups using the genotyping by sequencing (GBS) [11] technique to sequence data or creating tools to make the data more accessible.

Panzea [10] is a NSF-funded project called “Biology of Rare Alleles in Maize and its Wild Relatives”. They are investigating the connection between phenotype and genotype of complex traits in maize and its wild relative – teosinte. They study how rare genetic variations contribute to overall plant functionality. These studies enrich our knowledge of evolution, sustainable agriculture, and genetic diversity and conservation. The project also developed mapping resources for the maize community including the association panel and the maize nested association mapping (NAM) [12] panel, which has been used extensively by the maize community for mapping and sampling global diversity. The resources created via the genetic mapping studies have been used to help unravel flowering time, height, leaf architecture and planting density, pro-vitamin A content, and disease resistance. Their biggest contribution has been over 100 scientists trained by the project. They are leaders in crop genetics and breeding globally. Their current project focus is on looking at recent mutations in a wide range of germ plasm, and trying to make predictions at to its effect based on its genomic context and our understanding of regulation and heterosis.

Panzea is one of the biggest data resource for maize diversity data. They host genotypic and phenotypic data sets. Their latest public “flat file” versions of prepackaged, genotypic data sets are available for download [13]. These genotypes were obtained by whole genome sequencing (“HapMap”) [14], GBS [11] or the maize SNP50 chip [15] or traditional SNP assays. The data sets are differentiated by the process which is used to obtain them. Their latest HapMapV3 dataset consists of 3.7 Terabytes of whole genome sequence data from 916 maize lines for more than 60 million SNPs. Their latest GBS data set is called ZeaGBSv2.7, it contains genotypes for 955,690 SNPs at more than 60,000 lines. Only 17,280 lines of 60,000 lines are public. They provide raw and partially imputed genotypes as HDF5 files with AGPv2 coordinates, HapMap and VCF formats in AGPv3 coordinates. HDF5 [6] files store big data in an efficient way. HapMap [14] is an internationally accepted format for representing the genomic data. VCF [16] is a Variant Call Format that is a text format which stores data in a compressed manner. AGP [17] is a specification standard which identifies the position of the markers.

Panzea supports two different genotype searches, one for GBS data and the other for HapMap data. The GBS search tool first asks for the dataset we want to query. It will give you the basic details of the dataset chosen, list of all the lines (also called taxa) which are available in the selected dataset. The list is shown as text file in the browser. Users have to provide their email id, to which the results of the query will be emailed. Panzea supports HDF5, HapMap, VCF formats which are widely used.

From the user’s perspective, they have to go through the list of lines available copy the lines which they want to query and then paste them in another window which will ask for more details regarding the query. The output formats are machine readable but not user friendly. The entire process seems to be cumbersome if a user wants just one position in the required region of a line.

Genbank [18] is the NIH genetic sequence database, an annotated collection of all publicly available DNA sequences. Genbank stores all the raw sequence data in their required format and the data is publicly available. It is designed to provide and encourage access within the scientific community to the most up to date and comprehensive DNA sequence information. NCBI places no restriction the on the use or distribution of the Genbank data.

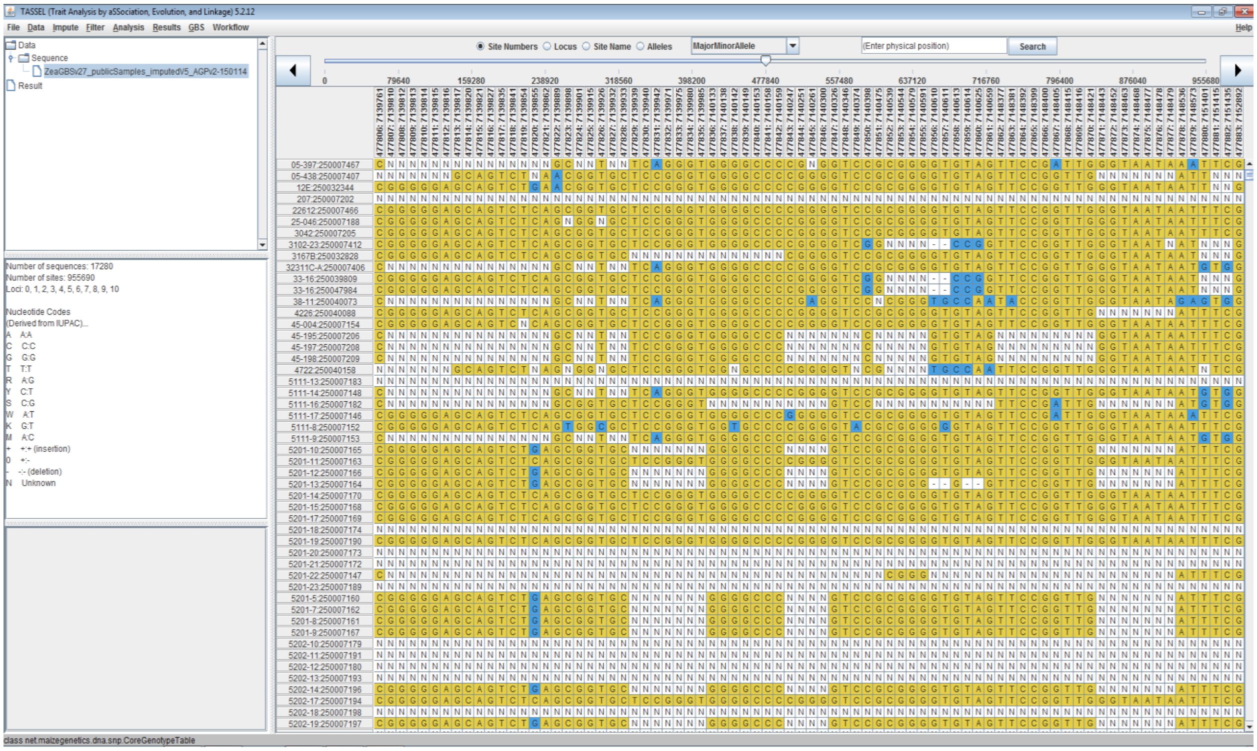
The International Maize and Wheat Improvement Center [19] also works on challenges of growing more maize sustainably. They collaborate with national agricultural research institutions, non-government organizations, community-based organizations, seed sector organizations, regional research networks, private companies and advanced research institutions to tackle the problem on a global scale. They provide diverse, high-yielding maize varieties that withstand infertile soils, drought, pests and diseases. They have a large data set which is not publicly available. They have 700,000 SNPs at more than 27,000 lines.

“Comprehensive genotyping of the USA national maize inbred seed bank” is a paper published by Romay *et al.* in 2013. They use the GBS technique on 2,815 maize inbred accessions, preserved mostly at the National Plant Germplasm System [29] in the USA. This collection includes inbred lines from breeding programs all over the world. They published their finding which included about 681,000 SNPs across the entire genome, with an ability to detect rare alleles at high confidence levels. More than half the SNPs in the collection are rare. The data published in the paper is available to be exploited by researchers to answer the problems faced in creating crops for sustainable agriculture. This data set is highly regarded by the maize community.

The Buckler Lab for Maize Genetics and Diversity [24] uses functional genomic approaches to dissect complex traits in maize, cassava and grapes. They exploit the natural diversity of these plant genomes to identify the individual nucleotides responsible for complex variation and then apply it to breeding. Buckler Lab created a tool called Tassel, it is designed for the optimized analysis of crop genomic diversity. Tassel stands for Trait Analysis by Association, Evolution and Linkage. Tassel is a software package used to evaluate genotype and traits associations with the characteristics of population and quantitative genetics. Tassel can handle datasets which are commonly encountered in the plant community like trails, inbred lines and complex structured pedigrees. It is open source and Buckler lab encourages people to build tools on their Tassel platform. Tassel provides various complex functions which are generally used on maize genomic data. They also have an active discussion group [20] in google groups which answers any question, technical or scientific. Tassel is designed to work with the HDF5 data sets which are available from Panzea.

Tassel consists of two parts, the Tassel standalone desktop application and the Tassel Java library package. The Tassel library package is the Java software which interfaces the standalone desktop client and the HDF5 data. This is what they call the Tassel platform. It is under active development and they keep adding new functions to fully utilize its potential. The entire code is open sourced and available in their Bitbucket repository. The Java functions designed are very efficient and quite often use bit calculations to improve the speed at which queries run. Any developer can use their library package and develop their own implementation of representing the data set. Buckler Lab also conducts hackathons [21] to encourage development.

The Tassel standalone application is a GUI tool which is used to browse the HDF5 data sets. Tassel is currently on Version 5.0 and uses Java Version 1.8. It supports Windows, Mac and UNIX operating systems. It shows all the lines in the data set as rows and all the markers as columns. It also provides various filter options, imputing options, data analysis options and various formats of showing results. A screenshot of the Tassel desktop application is shown in Figure 2.



Number of sequences: 17280
Number of sites: 955690
Loci: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Nucleotide Codes
(Derived from IUPAC)...

A	A:A
C	C:C
G	G:G
T	T:T
R	A:G
Y	C:T
S	C:G
W	A:T
K	G:T
M	A:C
+	+:+ (insertion)
0	+:-
-	-:- (deletion)
N	Unknown

class net.maizegenetics.dna.snp.CoreGenotypeTable

Figure 2. *Tassel desktop application screenshot.*

Tassel also supports HapMap, VCF file formats, though the performance is best when the file is in HDF5 format. It has a fairly steep learning curve. A lot of users are will not be able to download such huge data. Tassel standalone application is very memory intensive. A few processes like distance matrix would require us to either increase the memory used by the application or reduce the input query size so that our desktop computers can handle it. The desktop computer which I ran the tests the runs on needed more memory for some operations. The desktop computer I used had latest available hardware.

Our solutions will address the shortcomings of the best tools available and provide a better way to access the data. The next chapter will talk about how we designed our solutions after carefully analyzing the problems with the existing software and data.

CHAPTER III

MODEL

In 2011, MaizeGDB [9] foresaw the rate at which large scale diversity data was being sequenced. They wanted to be prepared to handle the flood of diversity data and make it available to the user community. MaizeGDB formed an open collaboration with iPlant Collaborative [22], SOL Genomics Network (SGN) [23] and the Bucker Lab [24] at Cornell University. The goal of the collaboration is to solve the problems of efficiently storing, querying and integrating the data. They also wanted to provide visualization tools for the public users at MaizeGDB.

To determine the best way to represent the maize data, MaizeGDB conducted a survey. They asked the users how they would like to use the data and requested a use case example. They also asked what kind of maize lines the users would be interested in seeing and how much time the user would be willing to wait for a response.

MaizeGDB predicted that data to be 1.6 million SNPs for a few dozen lines in 2011. They projected it would increase to over 10 million SNPs for 100 lines in 2012. They foresaw 100,000 lines for over 100 million SNPs coming from several labs in the next few years [25].

MaizeGDB proposed a tiered approach. The first tier started by making all the tools and services available for the B73 reference genome [3]. The second tier would add a set of tools and services to support the NAM lines [12]. The third and fourth tiers would only offer limited tools and support for up to 1000 lines and eventually the entire diversity data. Supporting all the tools for all the lines would take years. They wanted to limit the tools for the higher tiers to enable quickly adding support to more lines. The project was halted in mid-2012 when MaizeGDB had to allocate funds to higher priority projects. This projected seemed very interesting to me. In 2014, I took up the project.

A traditional relational database cannot handle the data required for this project. If the data was stored in a table it would have one million rows and tens of thousands of columns. This was unfeasible for our data. We need to use big data format to handle the data this large. The best data structure for this data is HDF5 [6]. It is a hierarchal format which uses an abstract data model which maps the storage model to different storage mechanisms. The HDF5 library provides a programming interface to a concrete implementation of the abstract models. This storage format is very efficient to store biological data. It is also the most preferred storage model for big data in various research labs [10].

Two systems appeared to be useful for developing our system, Apache Hadoop [26] and Tassel [5]. Both systems are open source, but Tassel (developed by the Bucker Lab) was developed to solve the same kinds of issues we needed to deal with. In particular the Tassel developers had already developed some of the functions that would have to be implemented for the Apache Hadoop environment. We decided to use Tassel and build on top of it.

Figure 3 shows the internal design of our web platform. It consists of two parts: the Java wrapper and a PHP web service. The Java wrapper uses Tassel’s functionality to query the data and process it. The PHP web service is the communication hub. It receives requests from clients and interfaces with the Java wrapper, it gathers the result and sends it to the client.

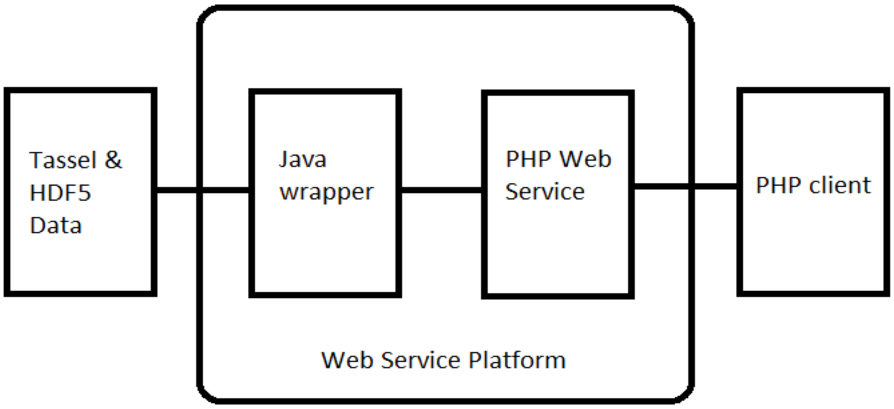


Figure 3. *Web Service Platform Internal Design.*

We decided to use PHP web service implementing REST [7] API for communication and JSON [8] data format. REST is a stateless software architecture. It demands the use of hypertext, which scales very well since the client and server are loosely coupled. The server is free to change the resource at will. The client needs to only know the initial URI to perform actions. It allows for rapid evolution of servers and allows an astronomical number of applications to interact freely on an ad hoc basis [27]. JSON (JavaScript Object Notation) is a lightweight format that is used for data interchanging. JSON is the de facto standard for transferring data from webserver to the client [28]. It has a simple structure which is a collection of name value pairs. JavaScript also recognizes JSON as an object which has major client side advantages when we want to display the data on the web page.

These choices of technology for the project allow it to be scalable and modular. Scalable to make sure we can increase the hardware of the server to efficiently handle all the incoming requests even under a huge load. Modularity has its own advantage. The genomic data which we are working with is still being sequenced. It is bound to change in the future. The data may become more detailed or more diverse data will come in fast. The tool we are going to develop should be able to handle these changes without a major rewrite of the code. If the architecture is modular we can simply update the piece which is outdated and the tool will seamlessly work with the new data. It is also true for the web services, if a new advanced web service is discovered tomorrow it will not be a big hassle to update the project to support the new service. With scalability, modularity and speed we came up with a design as illustrated in Figure 3.

The data set we are using currently is from Panzea [13]. It is called ZeaGBSv2.7. This is their current ZeaGBS build, containing 955,690 SNPs at 17,280 lines which are publically available. For these samples, both raw and partially imputed data sets are available in HDF5 files. Imputation is the statistical process of replacing missing data with substituted values. These files are directly compatible with Tassel.

We had to decide what exactly we want to do with data, how will we show the data to the users, and what datasets should we choose. Should we dump all the data in one large data set which will take longer time to query or should we have small sets of grouped data which users can select which will run faster? Our group decided that we should use the survey results. These are the most important of the list of use cases from the survey results:

Looking at the use cases, each one would require a different input and different processing. We decided to select a simple one, extend it to other use cases if possible then chose another use case and implement it. The first one we wanted to develop was a variation of use case number 4. *Given a genomic position or range and lines of interest, retrieve all the SNPs in that region.*

Table 1. *Important use cases from the survey.*

1. Between two lines, give me all the amino acid changes based on high quality SNPs
2. Between two lines, give me all present/absent/variations.
3. Given a gene of interest, retrieve all alleles for that gene.
4. Given a genomic position/range, retrieve all the SNPs in that region.
5. Given a maize line, retrieve all SNPs compared to a reference strain (B73).
6. Use the information to mine the best allele for breeding or functional studies.

The second and third questions in the survey were about lines of maize which the users wants. The second question was 'how many lines they want the use case to run against'. The third question was which subset of maize lines the user is most interested in. The answers varied from only NAM [12] lines to all the lines available. There was no clear answer. Each user works with different lines and we want to accommodate all of them. Tassel provides the option of creating data subsets. So we created different subsets of data according to user interest and the also have an option to query the entire data set.

A critical question was how long the user would be willing to wait to get a response. The options suggest in the survey were 10 seconds, 1 minute, 10 minutes, 1 hour, 1 day and 1 week. The answers were quite diverse. The users with use cases which wanted to query all the lines were willing to wait for 1 day or even 1 week. Other users who have smaller subsets of data were willing to wait for 1 minute up to 1 hour. During initial testing of Tassel, sample query of the user case against the entire data set to return the data in a specific position it took less than 1 minute. We should remember that this is a simple query without much data processing. If we had to process the data further it would require more queries, hence more time. Upon further testing, we were confident that we could do most of the use cases in under 1 minute.

The next chapter will talk about how we implemented the design, the problems faced and how we solved them and documentation of the code.

CHAPTER IV

IMPLEMENTATION

This chapter will discuss the implementation of the web service platform. The high level architecture is illustrated in Figure 4. Implementation of blocks 1, 2, 3, and 5 are discussed in the following subsections. Implementation started with setting up the development environment. Eclipse IDE was used for coding in Java, gedit for coding in PHP, Bitbucket for code repository, virtual machine client VMware for hosting the server, Vsphere client for remotely accessing the server, CentOS as the operating system for the server. Two versions of GBS [11] data from ZeaGBSv2.7 from Panzea [13], raw and imputed. The raw data is 12 Gigabytes and imputed data is 5.5 Gigabytes. Both the data sets contains 955,690 SNPs at 17,280 lines in HDF5 format [6]. Qi Sun from Cornell University supplied a sample wrapper class which he developed on the Tassel platform. It was used as a starting point during development.

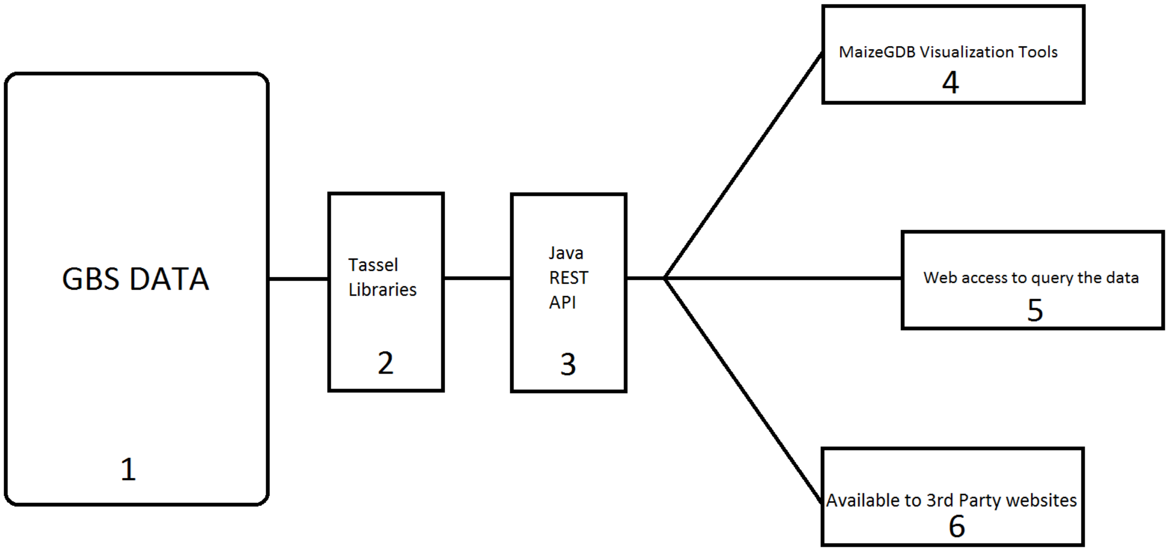

GBS DATA
1
Tassel
Libraries
2
Java
REST
API
3
MaizeGDB Visualization Tools
4
Web access to query the data
5
Available to 3rd Party websites
6

Figure 4. *High level Architecture.*

GBS Data

The GBS [11] data used is the ZeaGBS build from Panzea [10]. The current version is ZeaGBSv2.7. Both raw and imputed data is available. It contains 955,690 SNPs at 17,280 lines in HDF5 format [6]. In the implementation, the data sets are loosely coupled with the rest of the system. Every time the wrapper class is run, the data set needs to be specified as a command line parameter. This enables us to use multiple data sets. Some users prefer not to work with the entire data set, they want to work with lines related to their field of work. They can choose a subset of the data when querying. The wrapper class provides the functionality required to create the subset. The entire data set is loaded into an object, then it is filtered according to the required lines. It creates a smaller HDF5 file with the specified file name. This HDF5 file generated is the subset of the data. It can be used with the wrapper to query the smaller data set. Implementation of this function is discussed in detail in the Java wrapper subsection of this chapter. Another advantage of loosely coupled data is it can be updated very easily. When a new version of the data set is released, it can be supported with no change in the code. The new data set needs to be loaded on the server in the required directory and it is ready to work with our platform. This is an advantage of our modular design.

Tassel Library

The Tassel [5] library package was developed by The Buckler Lab [24] at Cornell University. It is a layer on top of the HDF5 data. Tassel has developed functions for handling the data. It is easier to interact with the data using Tassel then interacting directly with the data. Tassel is open source and its code repository is present in

Bitbucket [30]. Using Tassel was straight forward. There were a couple of issues with Tassel during the implementation of the use cases. They were resolved by the Tassel developers immediately.

During the implementation of the first use case, the position filtering was not working. Tassel was ignoring the start and end positions. After a lot of debugging, an error in the sample wrapper class surfaced. If the end position or start position is not mentioned in the command line arguments they default to zero. The end position value always defaulted to zero, even when the value was non-zero. Whenever one of the start or end position values are zero, Tassel considers the entire range of that chromosome. The variable “EndPhysicalPosition” which takes the value from the command line argument was overridden by the start position value due to the mistake in the code. We informed Qi Sun about this bug. He acknowledged it and corrected the sample wrapper class code. This is one of those very hard to find bugs which is a result of a typo.

All the Tassel searches are in one format. The entire data set is loaded and then Tassel filters the data by lines and then filters it by positions until the desired region is determined. The second and the third use case had a request which is quite the opposite of this format. Using the given region, list of lines with the same allele value in the same region. There was no function in the Tassel library which supports this. We tried a brute force approach, but it was very inefficient. Ultimately we came up with logic which could be implemented in the Tassel library, which was more efficient. It is a matching function. It matches the byte value of the given region with the array of byte values and returns a binary array, 1 representing match and 0 otherwise. The array of byte values are the allele values present in the same region for the entire data set. I emailed our solution to the

Tassel software architect, Terry Casstevens. After a discussion regarding handling of unknown data, he added the function to the Tassel library. This function was immensely useful in implementing the last two use cases. The next layer is the web service platform. It consists of the Java wrapper class and the web service.

Java Wrapper

The Java wrapper code is available at https://bitbucket.org/abhinav1/tassel-wrapper. The wrapper is packaged into a JAR file and it needs command line arguments to process the queries. The file Tassel_gt_server.java handles all the requests. It takes all the command line arguments and calls the required function depending upon the use case. The command line arguments are listed in Table 2.

Table 2. *Command line arguments for wrapper.*

-bf	Build the file. If skip, only give the dimensions
-sf	source file name
-st	source file type (hdf5, vcf, or hapmap)
-df	destination file name
-dt	destination file type (hdf5, vcf, hapmap or plink)
-tf	taxa file name
-ch	chromosome name
-start	physical start position on the chromosome
-end	physical end position on the chromosome

The three functions which implement the use cases are slice(), sliceNuke(), sliceRangeNuke(). The slice() function implements the first use case. The sliceNuke() function implements the second function. The sliceRangeNuke() function implements the third use case.

The slice() function accepts all the command line arguments from the main function. Processes all the input data into variables then creates a “GenotypeTable” object from the source file. Filters the required taxa list and then filters the required positions. It uses the "ExportUtils” class to export the filtered object to the user requested format. This function can be used in two ways. When the output format is selected as HDF5, the “ExportUtils” class has a function which converts the contents of the “GenotypeTable” object into a HDF5 file. As the slice() function already filters the contents of the “GenotypeTable” object this function can be used to generate subsets of the data. These HDF5 files created can again be used to query the data. When the output format is any other format (HapMap, VCF), the slice() function creates the output file in the required format using the “ExportUtils” class. It writes the output file into the output destination selected and exits.

The sliceNuke() function accepts all the command line arguments from the main function. It processes all the input data into variables and then creates a “GenotypeTable” object from the source file. It filters the required taxa and then filters for the required position. The object now contains only one value. The single value contained in the object is the required allele. It then retrieves the unique site number of the allele. It creates a byte array of all the allele values at the same site for all the taxa lines using the “genotypeAllTaxa” function. The byte array of all the allele values in the data set for the specific position and the byte value of the single allele value in the same position are compared by the “calcBitPresenceOfDiploidValueFromGenotype” function. This function returns a “BitSet” object, it is a binary array which represents the result of the comparison. A positive match is represented by 1 and a negative match is represented by 0. Then the function “getIndicesOfSetBits” function of the “BitSet” object is used to return an integer array. The integer array contains list of all the positions of the binary array which is a 1. The integer array is used to iterate through the list of lines in the entire data set, which is an unfiltered “GenotypeTable” object. “BufferedWriter” object is used to create the custom output format which lists all the lines which have the same allele value at the required position. It writes the output file into the output destination selected and exits.

The sliceRangeNuke() function accepts all the command line arguments from the main function. It processes all the input data into variables then creates a “GenotypeTable” object from the source file. It filters the required taxa and then filters the required range of positions. Then it retrieves the range of unique site numbers for the corresponding range of positions and stores them in an integer ArrayList. For each site we extract the byte value for the allele and create the byte array of all the allele values in the data set for the specific position and compare using “calcBitPresenceOfDiploidValueFromGenotype” function. Each comparison returns a “BitSet” object, it is stored in a “BitSet” ArrayList. All the individual “BitSet” objects which are stored in the ArrayList are combined to create one object. The bitwise AND operation is used to combine them. The combined “BitSet” object represents the lines which have the same sequence of alleles in the given range of positions. Then the function “getIndicesOfSetBits” function of the “BitSet” object is used to return an integer array. The integer array contains list of all the positions of the binary array which is a 1. The integer array is used to iterate through the list of lines in the entire data set, which is an unfiltered “GenotypeTable” object. The “BufferedWriter” object is used to create the custom output format which lists all the lines which have the same sequence of alleles in the given range of positions. It writes the output file into the output destination selected and exits. When the output file is created in the destination and the wrapper class exits, the web service uses the output file and communicates it with the client.

Web Service

The PHP web service code for is available at https://bitbucket.org/abhinav1/datadiversity. The server is on a virtual machine based on the CentOS operating system. The virtual machine allows the hardware configuration of the server to be changed as needed. Memory can be increased to allow more memory intense operations. The web service is built using REST [7] and JSON [8].

When the web service receives a request from a client, it decodes the JSON request and stores the input request details in local variables. It assigns a unique code to each incoming request based on the micro second at which the request was received. This code is used to identify every request uniquely. According to the input request, the set of command line arguments are generated and the Java wrapper JAR function is executed with the command line arguments. As soon as the output file is generated, a JSON response is created using the directory of the output file. The response is sent to the client.

There are three different instances of the web services to accept the requests for three different use cases. The web service URIs are listed in Table 3. The three web services have slightly different APIs, because the use cases require different input. The API specifications are easier to understand with an example. The examples are illustrated in the next section.

Table 3. *Web service URIs.*

Use case 1	http://abhinav2.gdcb.iastate.edu/Diversity/service.php
Use case 2	http://abhinav2.gdcb.iastate.edu/Diversity/milestone_service.php
Use case 3	http://abhinav2.gdcb.iastate.edu/Diversity/milestone2_service.php

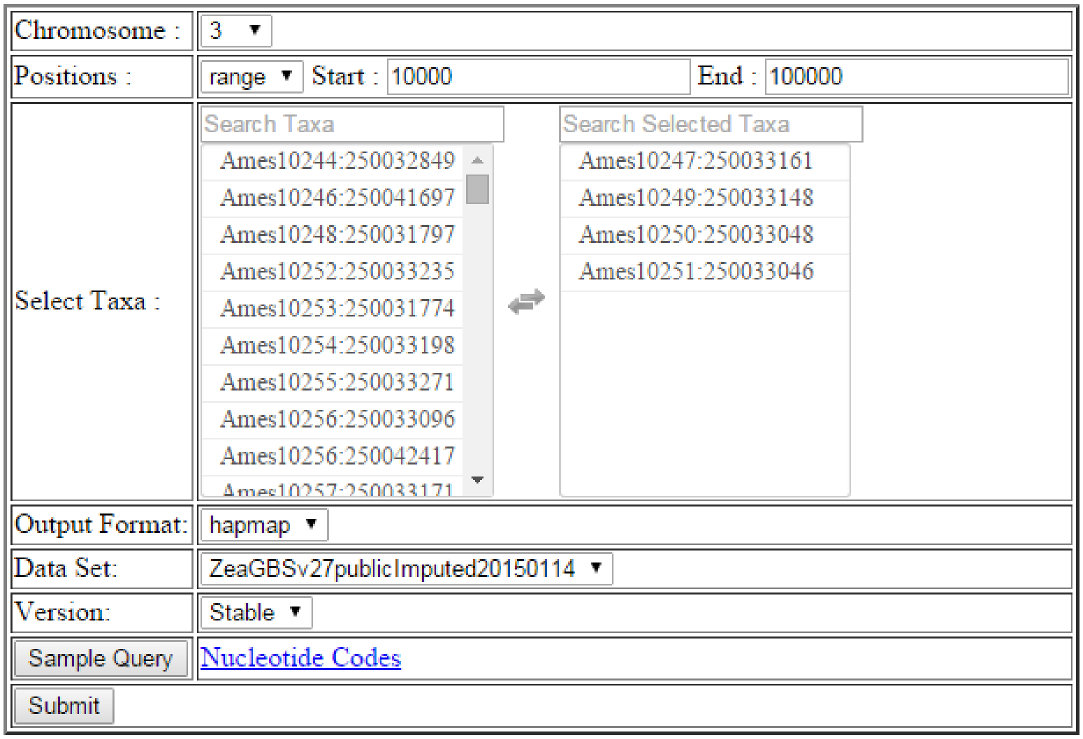
Web Client

A PHP client was developed to demonstrate how of our web service platform. It consists of three different forms for the three use cases. When a form is submitted, the client takes all the input details entered in the form and creates a JSON format of the data. It then sends the request to the corresponding URI with the JSON data. The server processes the request and send the JSON response. The client receives the JSON response and open the output file link in the browser. The URIs of the three forms are listed in Table 4.

Table 4. *Web client URIs.*

Use case 1	http://abhinav2.gdcb.iastate.edu/Diversity/home.php
Use case 2	http://abhinav2.gdcb.iastate.edu/Diversity/milestone.php
Use case 3	http://abhinav2.gdcb.iastate.edu/Diversity/milestone2.php

The three use cases are accessible from the URIs in Table 4. They will open a form in the web browser. The use cases are better illustrated with the examples. The form for the first use case is in Figure 5. It is hosted in home.php. The first use case is: *Given a genomic position or range and lines of interest, retrieve all the SNPs in that region.*



Chromosome : 3

Positions : range Start : 10000 End : 100000

Select Taxa :

Search Taxa

Ames10244:250032849
Ames10246:250041697
Ames10248:250031797
Ames10252:250033235
Ames10253:250031774
Ames10254:250033198
Ames10255:250033271
Ames10256:250033096
Ames10256:250042417
Ames10257:250033171

Search Selected Taxa

Ames10247:250033161
Ames10249:250033148
Ames10250:250033048
Ames10251:250033046

Output Format: hapmap

Data Set: ZeaGBSv27publicImputed20150114

Version: Stable

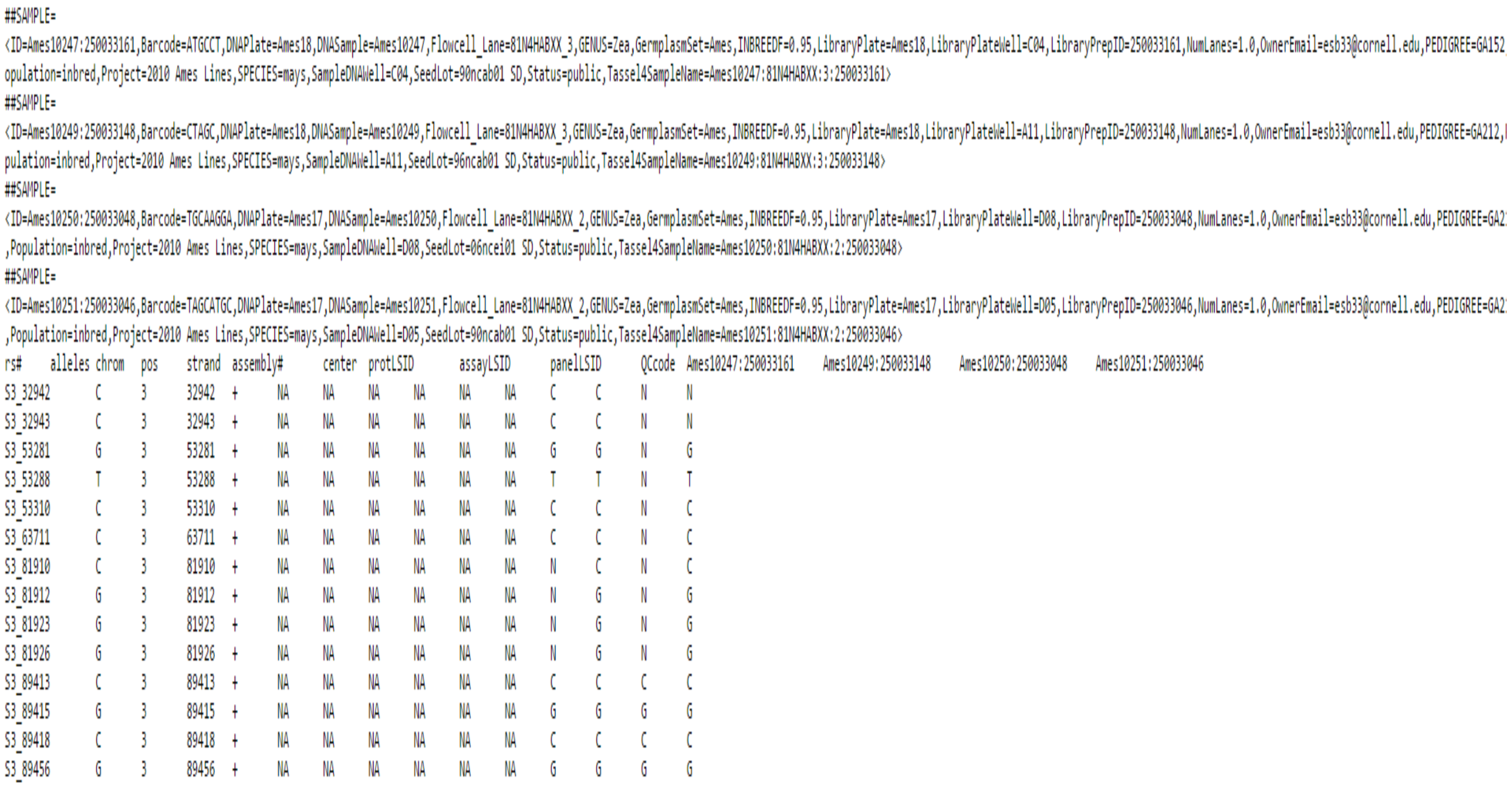
Figure 5. *Form to demonstrate the functionality of service.php – home.php.*

```
JSON input format example - service.php

{
"chromosome":"3",
"positions":"range",
"startPosition":"10000",
"endPosition":"100000",
"taxaArray":["Ames10247:250033161","Ames10249:250033148","Ames10250:250033048","Ames10251:250033046"],
"outputFormat":"hapmap",
"dataSet":"ZeaGBSv27publicImputed20150114",
"version":"stable"
}
```

```
JSON output format example - service.php

{
"uri":"http://abhinav2.gdcb.iastate.edu/Diversity/tassel/output/014369172300385.txt"
}
```

Figure 6. *JSON API specification for service.php.*

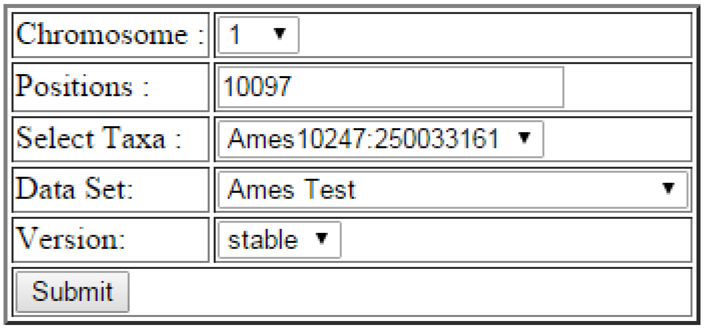
The API Specification for home.php is represented in Figure 6. It requires the chromosome, positions, start position, end position, taxa array (list of lines), output format, dataset and version of the wrapper class. It returns only the URI of the output file. Figure 7 shows the output file generated for this request.



```
##SAMPLE=
<ID=Ames10247:250033161,Barcode=ATGCCT,DNAPlate=Ames18,DNASample=Ames10247,Flowcell_Lane=81N4HABXX_3,GENUS=Zea,GermplasmSet=Ames,INBREEDF=0.95,LibraryPlate=Ames18,LibraryPlateWell=C04,LibraryPrepID=250033161,NumLanes=1.0,OwnerEmail=esb33@cornell.edu,PEDIGREE=GA152
opulation=inbred,Project=2010 Ames Lines,SPECIES=mays,SampleDNAWell=C04,SeedLot=90ncab01 SD,Status=public,Tassel4SampleName=Ames10247:81N4HABXX:3:250033161>
##SAMPLE=
<ID=Ames10249:250033148,Barcode=CTAGC,DNAPlate=Ames18,DNASample=Ames10249,Flowcell_Lane=81N4HABXX_3,GENUS=Zea,GermplasmSet=Ames,INBREEDF=0.95,LibraryPlate=Ames18,LibraryPlateWell=A11,LibraryPrepID=250033148,NumLanes=1.0,OwnerEmail=esb33@cornell.edu,PEDIGREE=GA212,
pulation=inbred,Project=2010 Ames Lines,SPECIES=mays,SampleDNAWell=A11,SeedLot=96ncab01 SD,Status=public,Tassel4SampleName=Ames10249:81N4HABXX:3:250033148>
##SAMPLE=
<ID=Ames10250:250033048,Barcode=TGCAAGGA,DNAPlate=Ames17,DNASample=Ames10250,Flowcell_Lane=81N4HABXX_2,GENUS=Zea,GermplasmSet=Ames,INBREEDF=0.95,LibraryPlate=Ames17,LibraryPlateWell=D08,LibraryPrepID=250033048,NumLanes=1.0,OwnerEmail=esb33@cornell.edu,PEDIGREE=GA2
,Population=inbred,Project=2010 Ames Lines,SPECIES=mays,SampleDNAWell=D08,SeedLot=06ncei01 SD,Status=public,Tassel4SampleName=Ames10250:81N4HABXX:2:250033048>
##SAMPLE=
<ID=Ames10251:250033046,Barcode=TAGCATGC,DNAPlate=Ames17,DNASample=Ames10251,Flowcell_Lane=81N4HABXX_2,GENUS=Zea,GermplasmSet=Ames,INBREEDF=0.95,LibraryPlate=Ames17,LibraryPlateWell=D05,LibraryPrepID=250033046,NumLanes=1.0,OwnerEmail=esb33@cornell.edu,PEDIGREE=GA2
,Population=inbred,Project=2010 Ames Lines,SPECIES=mays,SampleDNAWell=D05,SeedLot=90ncab01 SD,Status=public,Tassel4SampleName=Ames10251:81N4HABXX:2:250033046>
rs#     alleles chrom   pos     strand  assembly#       center  protLSID        assayLSID       panelLSID       QCcode  Ames10247:250033161     Ames10249:250033148     Ames10250:250033048     Ames10251:250033046
S3_32942        C       3       32942   +       NA      NA      NA      NA      NA      NA      C       C       N       N
S3_32943        C       3       32943   +       NA      NA      NA      NA      NA      NA      C       C       N       N
S3_53281        G       3       53281   +       NA      NA      NA      NA      NA      NA      G       G       N       G
S3_53288        T       3       53288   +       NA      NA      NA      NA      NA      NA      T       T       N       T
S3_53310        C       3       53310   +       NA      NA      NA      NA      NA      NA      C       C       N       C
S3_63711        C       3       63711   +       NA      NA      NA      NA      NA      NA      C       C       N       C
S3_81910        C       3       81910   +       NA      NA      NA      NA      NA      NA      N       C       N       C
S3_81912        G       3       81912   +       NA      NA      NA      NA      NA      NA      N       G       N       G
S3_81923        G       3       81923   +       NA      NA      NA      NA      NA      NA      N       G       N       G
S3_81926        G       3       81926   +       NA      NA      NA      NA      NA      NA      N       G       N       G
S3_89413        C       3       89413   +       NA      NA      NA      NA      NA      NA      C       C       C       C
S3_89415        G       3       89415   +       NA      NA      NA      NA      NA      NA      G       G       G       G
S3_89418        C       3       89418   +       NA      NA      NA      NA      NA      NA      C       C       C       C
S3_89456        G       3       89456   +       NA      NA      NA      NA      NA      NA      G       G       G       G
```

Figure 7. *Example output file from service.php in HapMap format.*

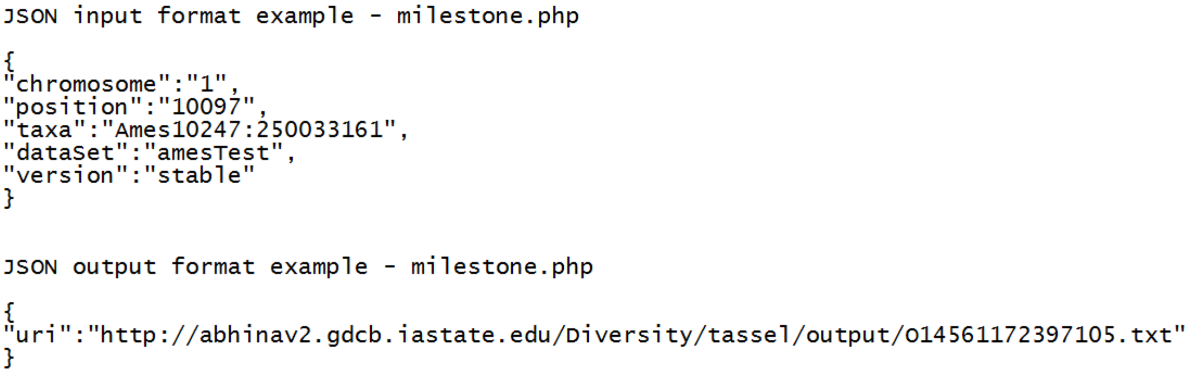
The second use case is: *Given a specific genomic position and a specific line what is allele present? List all the lines which have the same allele in the same genomic position.* . The form for the second use case is in Figure 8.



Chromosome :	1 ▼
Positions :	10097
Select Taxa :	Ames10247:250033161 ▼
Data Set:	Ames Test ▼
Version:	stable ▼

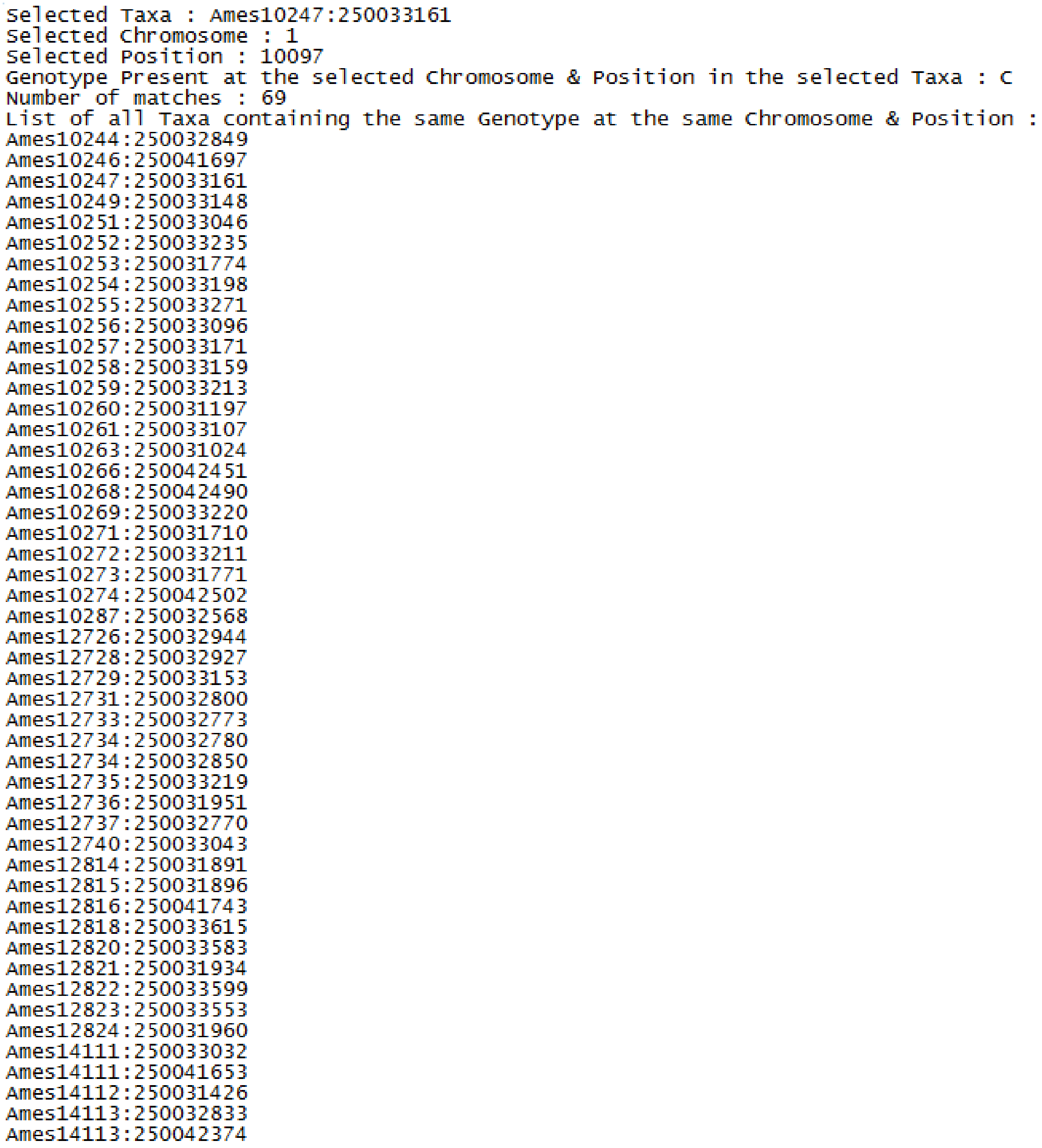
Figure 8. *Form to demonstrate the functionality of milestone_service.php – milestone.php.*

The API Specification for milestone.php is represented in Figure 9. It requires the chromosome, position, taxa, dataset and version of the wrapper class. It returns only the URI of the output file. Figure 10 shows the output file generated for this request.



```
JSON input format example - milestone.php

{
"chromosome":"1",
"position":"10097",
"taxa":"Ames10247:250033161",
"dataSet":"amesTest",
"version":"stable"
}

JSON output format example - milestone.php

{
"uri":"http://abhinav2.gdcb.iastate.edu/Diversity/tassel/output/014561172397105.txt"
}
```

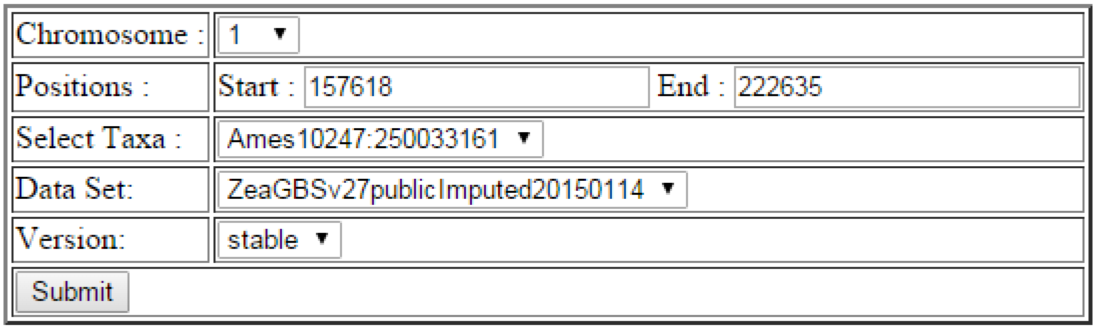
Figure 9. *JSON API specification for milestone_service.php.*



```
Selected Taxa : Ames10247:250033161
Selected Chromosome : 1
Selected Position : 10097
Genotype Present at the selected Chromosome & Position in the selected Taxa : C
Number of matches : 69
List of all Taxa containing the same Genotype at the same Chromosome & Position :
Ames10244:250032849
Ames10246:250041697
Ames10247:250033161
Ames10249:250033148
Ames10251:250033046
Ames10252:250033235
Ames10253:250031774
Ames10254:250033198
Ames10255:250033271
Ames10256:250033096
Ames10257:250033171
Ames10258:250033159
Ames10259:250033213
Ames10260:250031197
Ames10261:250033107
Ames10263:250031024
Ames10266:250042451
Ames10268:250042490
Ames10269:250033220
Ames10271:250031710
Ames10272:250033211
Ames10273:250031771
Ames10274:250042502
Ames10287:250032568
Ames12726:250032944
Ames12728:250032927
Ames12729:250033153
Ames12731:250032800
Ames12733:250032773
Ames12734:250032780
Ames12734:250032850
Ames12735:250033219
Ames12736:250031951
Ames12737:250032770
Ames12740:250033043
Ames12814:250031891
Ames12815:250031896
Ames12816:250041743
Ames12818:250033615
Ames12820:250033583
Ames12821:250031934
Ames12822:250033599
Ames12823:250033553
Ames12824:250031960
Ames14111:250033032
Ames14111:250041653
Ames14112:250031426
Ames14113:250032833
Ames14113:250042374
```

Figure 10. *Example output file from milestone_service.php in text format.*

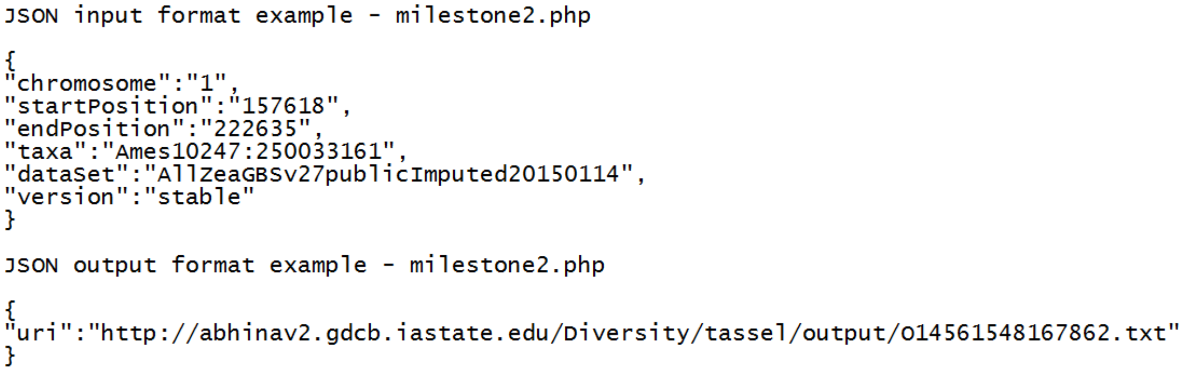
The third use case is: *Given a range of genomic positions and a specific line what is sequence of alleles present? List all the lines which have the same sequence in the same range of genomic positions.* The form for the third use case is in Figure 11.



Chromosome :	1 ▼
Positions :	Start : 157618 End : 222635
Select Taxa :	Ames10247:250033161 ▼
Data Set:	ZeaGBSv27publicImputed20150114 ▼
Version:	stable ▼
Submit	

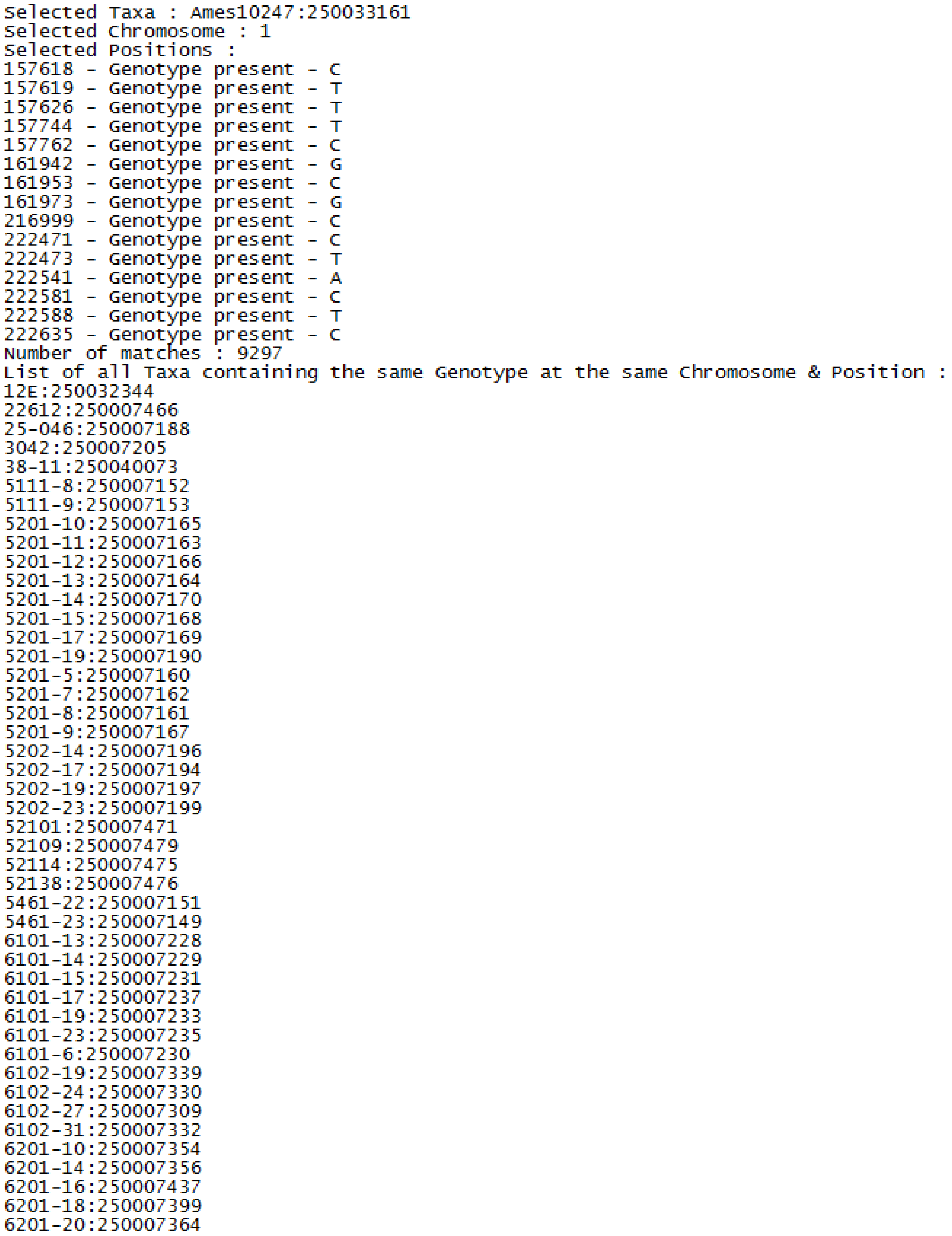
Figure 11. *Form to demonstrate the functionality of milestone2_service.php – milestone2.php.*

The API Specification for milestone2.php is represented in Figure 12. It requires the chromosome, start position, end position, taxa, dataset and version of the wrapper class. It returns only the URI of the output file. Figure 13 shows the output file generated for this request.



```
JSON input format example - milestone2.php

{
"chromosome":"1",
"startPosition":"157618",
"endPosition":"222635",
"taxa":"Ames10247:250033161",
"dataSet":"AllZeaGBSv27publicImputed20150114",
"version":"stable"
}

JSON output format example - milestone2.php

{
"uri":"http://abhinav2.gdcb.iastate.edu/Diversity/tassel/output/014561548167862.txt"
}
```

Figure 12. *JSON API specification for milestone2_service.php.*



```
Selected Taxa : Ames10247:250033161
Selected Chromosome : 1
Selected Positions :
157618 - Genotype present - C
157619 - Genotype present - T
157626 - Genotype present - T
157744 - Genotype present - T
157762 - Genotype present - C
161942 - Genotype present - G
161953 - Genotype present - C
161973 - Genotype present - G
216999 - Genotype present - C
222471 - Genotype present - C
222473 - Genotype present - T
222541 - Genotype present - A
222581 - Genotype present - C
222588 - Genotype present - T
222635 - Genotype present - C
Number of matches : 9297
List of all Taxa containing the same Genotype at the same Chromosome & Position :
12E:250032344
22612:250007466
25-046:250007188
3042:250007205
38-11:250040073
5111-8:250007152
5111-9:250007153
5201-10:250007165
5201-11:250007163
5201-12:250007166
5201-13:250007164
5201-14:250007170
5201-15:250007168
5201-17:250007169
5201-19:250007190
5201-5:250007160
5201-7:250007162
5201-8:250007161
5201-9:250007167
5202-14:250007196
5202-17:250007194
5202-19:250007197
5202-23:250007199
52101:250007471
52109:250007479
52114:250007475
52138:250007476
5461-22:250007151
5461-23:250007149
6101-13:250007228
6101-14:250007229
6101-15:250007231
6101-17:250007237
6101-19:250007233
6101-23:250007235
6101-6:250007230
6102-19:250007339
6102-24:250007330
6102-27:250007309
6102-31:250007332
6201-10:250007354
6201-14:250007356
6201-16:250007437
6201-18:250007399
6201-20:250007364
```

Figure 13. *Example output file from milestone_service.php in text format.*

CHAPTER V

CONCLUSION & FUTURE WORK

Conclusion

Our work focuses on creating a platform on which users will be able to access the maize diversity data seamlessly. It is a difficult problem to solve. There is always new data being sequenced with more accuracy and detail. There are complications to run huge analysis on the big data which run quickly. We tackled some of the issues in our solution.

We created a web platform with the latest technologies. It’s based on REST with JSON which is widely used by client applications and web servers. It can give you the raw data or it can answer meaningful questions (use cases). We implemented three of those questions. We provide an easy way to implement more questions. We provided various output formats and the platform is flexible to add more. Formats useful for other applications (HapMap and VCF) and formats which are readable (Text) by the user are supported. We developed a PHP client to demonstrate the use of the web platform. Users are given a choice of which data set they want to query. We also have a process in place to create sub data sets on request by the users. Our API is documented, it can be used by other applications which can build on top of our output. The entire source code for both the web service and the wrapper class are available online and up to date. We believe we have made an easier way for users to access diversity data.

Future work

We have just finished the platform, there is a lot of scope for future work. The most exciting work will be building tools on this platform. MaizeGDB plans on building a set of visualization tools which take uses the output from the web service and represent the diversity data in a visually appealing way. I am currently working to finalize the format which is required for their input. We can support various 3rd party tools which can use our API and build tools on the data.

The platform can also be improved in various ways. We can implement more use cases which will add more functionality. Adding a new case is as simple as writing a function with the logic. The connection with the data set, reading the data into objects and creating the output format are already defined and can be reused for the new use cases. Interesting use cases will be to identify haplotypes. Haplotypes are blocks of regions which are similar. Similarity in haplotypes does not have a proper definition as of now. Implementing percentage based matching is also an interesting idea. Some of the users may not be looking for a 100% match across a wide range of positions. Giving an option where a user can choose what percentage they want will be a good feature. For example given a genomic region and a specific line will return the list of lines which have the same alleles present in the same region with 10% tolerance. It will result in 90% matching. Another interesting use case is identifying genetic diversity between two or more lines. MaizeGDB is conducting another survey which will give more insight into how the users want to utilize the data and ideas for more use cases.

REFERENCES

[1] “The Importance of Corn In The American Economy” - http://www.offthegridnews.com/off-grid-foods/the-importance-of-corn-in-the-american-economy/

[2] “The Maize Genome” poster by Anne W. Sylvester, Patrick S. Schnable, and Rob Martienssen

[3] “The B73 Maize genome: complexity, diversity and dynamics” - http://www.ncbi.nlm.nih.gov/pubmed/19965430

[4] “Maize: The Genome Sequence Itself” - http://www.jamesandthegiantcorn.com/2009/11/20/maize-the-genome-sequence-itself/

[5] “TASSEL-GBS: A High Capacity Genotyping by Sequencing Analysis Pipeline” - http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0090346

[6] “What is HDF5?” - https://www.hdfgroup.org/HDF5/whatishdf5.html

[7] “RESTful Web Services” - https://books.google.com/books?id=XUaErakHsoAC&hl=en

[8] “The JSON Data Interchange Format” - http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

[9] “MaizeGDB becomes ‘sequence-centric’” - http://database.oxfordjournals.org/content/2009/bap020.full?ijkey=NtoVnRZ51irh47m&keytype=ref

[10] “The Maize Diversity Project” - http://www.panzea.org/

[11] “A robust, simple Genotyping-by-Sequencing (GBS) approach for high diversity species.” - Elshire RJ, Glaubitz JC, Sun Q, Poland JA, Kawamoto K, Buckler ES, Mitchell SE - http://www.ncbi.nlm.nih.gov/pubmed/21573248?dopt=Abstract&holding=f1000,f1000m,isrctn

[12] “Genetic Design and Statistical Power of Nested Association Mapping in Maize” - http://www.genetics.org/content/178/1/539.abstract

[13] “Genotype data download” - http://www.panzea.org/#!genotypes/cctl

[14] “The International HapMap Project” - http://hapmap.ncbi.nlm.nih.gov/downloads/nature02168.pdf

[15] “A powerful tool for genome analysis in maize: development and evaluation of the high density 600 k SNP genotyping array” - http://www.biomedcentral.com/1471-2164/15/823

[16] “The Variant Call Format” - https://samtools.github.io/hts-specs/VCFv4.2.pdf

[17] “AGP Specifications” - http://www.ncbi.nlm.nih.gov/projects/genome/assembly/agp/AGP_Specification.shtml

[18] “GenBank” - http://www.ncbi.nlm.nih.gov/pubmed/23193287

[19] “CIMMYT” - http://www.cimmyt.org/en/who-we-are

[20] Tassel Google Group - https://groups.google.com/forum/#!forum/tassel

[21] Tassel Hackathon - https://groups.google.com/forum/#!searchin/tassel/hackaton/tassel/FbFjhqqmjDM/OaB-de46UsUJ

[22] iPlant Collaborative - http://www.iplantcollaborative.org/

[23] SOL Genomics Network - http://solgenomics.net/

[24] Buckler Lab for Maize Genetics and Diversity - http://www.maizegenetics.net/

[25] Waves of diversity for maize - http://survey.maizegdb.org/diversity/

[26] Apache Hadoop - https://hadoop.apache.org/

[27] Scalability of REST API - http://stackoverflow.com/a/5321470

[28] JSON de facto Standard - http://fossil.wanderinghorse.net/repos/cson/index.cgi/wiki?name=JSON

[29] National Germplasm Resource - http://www.ars.usda.gov/main/site_main.htm?modecode=80-42-05-45

[30] Tassel code repository - https://bitbucket.org/tasseladmin/tassel-5-source/wiki/Home

APPENDIX

CODE

The Java code for the wrapper class is included below. It is from the tassel_gt_server.java file.

```
package tassel_gt_server;

/**
 *
 * @author Abhinav
 *
 */
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Map;

import net.maizegenetics.dna.WHICH_ALLELE;
import net.maizegenetics.dna.map.Chromosome;
import net.maizegenetics.dna.map.Position;
import net.maizegenetics.dna.map.PositionList;
import net.maizegenetics.dna.snp.ExportUtils;
import net.maizegenetics.dna.snp.FilterGenotypeTable;
import net.maizegenetics.dna.snp.GenotypeTable;
import net.maizegenetics.dna.snp.GenotypeTableBuilder;
import net.maizegenetics.dna.snp.GenotypeTableUtils;
import net.maizegenetics.dna.snp.ImportUtils;
import net.maizegenetics.taxa.TaxaList;
import net.maizegenetics.taxa.TaxaListBuilder;
import net.maizegenetics.taxa.Taxon;
import net.maizegenetics.util.ArgsEngine;
import net.maizegenetics.util.BitSet;
import net.maizegenetics.util.ExceptionUtils;
import net.maizegenetics.util.Utils;



public class Tassel_gt_server {

    /**
     * @param args the command line arguments
     * create_result_file T or F
     *
     */
    public static void main(String[] args) {
        if (args.length == 0) {
            System.out.print("No method is specified!\n");
            printUsage("main");
        }
        String method = args[0];
        String[] newargs = new String[args.length-1];
        System.arraycopy(args, 1, newargs, 0, args.length-1);
        if (method.equals("slice"))
        {
            String results = slice(newargs);
            System.out.print(results);
        }
        else if (method.equals("sliceNuke"))
        {
            String results = sliceNuke(newargs);
            System.out.print(results);
        }
        else if (method.equals("sliceRangeNuke"))
        {
            String results = sliceRangeNuke(newargs);
            System.out.print(results);
        }
        else if (method.equals("dbinfo"))
        {
            String results = dbinfo(newargs);
            System.out.print(results);
        }
        else if (method.equals("get_taxa_list"))
        {
            String[] results = get_taxa_list(newargs);
            for (String t:results)
            {
                System.out.println(t);
            }
        }
        else if (method.equals("chr_marker_info"))
        {
            String[] results = chr_marker_info(newargs);
            for (String t:results)
            {
                System.out.println(t);
            }
        }
        else
        {
           System.out.print("The method '"+ method + "' is not
recognized!\n");
           printUsage("main");
        }
    }

    private static String sliceNuke(String[] args)
    {

        //get parameters
        ArgsEngine  myArgsEngine = new ArgsEngine();
        myArgsEngine.add("-sf", "--sourcefile", true);
        myArgsEngine.add("-st", "--sourcefile-type", true);
        myArgsEngine.add("-df", "--destinationfile", true);
        myArgsEngine.add("-dt", "--destinationfile-type", true);
        myArgsEngine.add("-tf", "--taxa-file", true);
        myArgsEngine.add("-tl", "--taxa-list", true);
        myArgsEngine.add("-ch", "--chromosome", true);
        myArgsEngine.add("-start", "--chr-start", true);
        myArgsEngine.add("-end", "--chr-end", true);
        myArgsEngine.add("-bf", "--build-file", false);
        myArgsEngine.parse(args);

       boolean buildfile = false;
       String source_file = null;
       String source_file_type = null;
       String dest_file = null;
       String dest_file_type=null;
       String TaxaListFile=null;
       ArrayList<String> TaxaArrayList = new ArrayList<String>();
       String ChromosomeStr=null;
       int StartPhysicalPosition = 0;
       int EndPhysicalPosition = 0;

       if (myArgsEngine.getBoolean("-bf"))
       {
            buildfile = true;
       }

      if (myArgsEngine.getBoolean("-sf")) {
          source_file = myArgsEngine.getString("-sf");
      } else {
          printUsage("slice");
          throw new IllegalArgumentException("Please specify a source
file (option -sf).");
      }

      if (myArgsEngine.getBoolean("-st")) {
          source_file_type = myArgsEngine.getString("-st");
      } else {
          printUsage("slice");
          throw new IllegalArgumentException("Please specify a source
file type (option -st).");
      }

      if (buildfile)
      {
          if (myArgsEngine.getBoolean("-df")) {
              dest_file = myArgsEngine.getString("-df");
          } else {
              printUsage("slice");
              throw new IllegalArgumentException("Please specify a
destination file (option -df).");
          }

          if (myArgsEngine.getBoolean("-dt")) {
              dest_file_type = myArgsEngine.getString("-dt");
          } else {
              printUsage("slice");
              throw new IllegalArgumentException("Please specify a
destination file type (option -dt).");
          }
      }

      if (myArgsEngine.getBoolean("-tf"))
      {
          TaxaListFile = myArgsEngine.getString("-tf");
          File outDirectory = new File(TaxaListFile);
          if (!outDirectory.isFile()) {
              printUsage("slice");
              throw new IllegalArgumentException("The taxa file you
supplied (option -tf) is not a file: " + TaxaListFile);
          }
          //verify and create sub-taxalist
          //create taxa filtered genotype table

         try {
              BufferedReader br = new BufferedReader(new
FileReader(TaxaListFile), 65536);

              String temp;
              int currLine = 0;
              while (((temp = br.readLine()) != null)) {
                  if (!temp.trim().isEmpty())
                  {
                      TaxaArrayList.add(temp.trim());
                      currLine++;
                  }

              }
          } catch (Exception e) {
              System.out.println("Couldn't open taxa file to read taxa
list: " + e);
          }
      }
      else if (myArgsEngine.getBoolean("-tl"))
      {
          String TaxaListStr = myArgsEngine.getString("-tf");
          if (TaxaListStr.equalsIgnoreCase("all"))
          {
              TaxaArrayList.add("all");
          }
          else
          {
              String[] wordList = TaxaListStr.split(";");
              TaxaArrayList.addAll(Arrays.asList(wordList));
          }
      }
      else {
          printUsage("slice");
          throw new IllegalArgumentException("Please specify the file
with taxa list (option -tf).");
      }
      if (myArgsEngine.getBoolean("-ch")) {
          ChromosomeStr = myArgsEngine.getString("-ch");
      } else {
          if ((myArgsEngine.getBoolean("-start")) ||
(myArgsEngine.getBoolean("-end")))
          {
              printUsage("slice");
              throw new IllegalArgumentException("Please specify a
chromosome name (option -ch).");
          }
          else
          {
              ChromosomeStr = "all";
          }
      }

      if (myArgsEngine.getBoolean("-start")) {
          StartPhysicalPosition =
Integer.parseInt(myArgsEngine.getString("-start"));
      }

      if (myArgsEngine.getBoolean("-end")) {
          EndPhysicalPosition =
Integer.parseInt(myArgsEngine.getString("-end"));
      }

       //create a genotypetable from source
      GenotypeTable source_gt_table = null;
      if (source_file_type.equalsIgnoreCase("hdf5"))
      {
          source_gt_table =
GenotypeTableBuilder.getInstance(source_file);
      }
      else if (source_file_type.equalsIgnoreCase("hapmap"))
      {
          source_gt_table = ImportUtils.readFromHapmap(source_file);
      }
      else if (source_file_type.equalsIgnoreCase("vcf"))
      {
          source_gt_table = ImportUtils.readFromVCF(source_file, null,
false);
      }




      GenotypeTable gt_taxa_filtered = null ;



      //if TaxaListString=all or not specified, all taxa included
      if ((TaxaArrayList.get(0).equalsIgnoreCase("all")) ||
(TaxaArrayList.size()==0))
      {
          gt_taxa_filtered = source_gt_table;
      }
      //if TaxaListString specified, create subset
      else
      {
          TaxaList source_taxa_list = source_gt_table.taxa();
          TaxaList dest_taxa_list=null;
          TaxaListBuilder dest_taxa_list_builder = new
TaxaListBuilder();

          ArrayList<String> unknow_taxa = new ArrayList<String>();
          for (int i =0; i<TaxaArrayList.size(); i++)
          {
             if (TaxaArrayList.get(i).matches(".*\\w.*"))
             {
                  int itaxa_index =
source_taxa_list.indexOf(TaxaArrayList.get(i));
                  if (itaxa_index>=0)
                  {

dest_taxa_list_builder.add(source_taxa_list.get(itaxa_index));
                  }
                  else
                  {
                      unknow_taxa.add(TaxaArrayList.get(i));
                  }
             }
          }
          dest_taxa_list = dest_taxa_list_builder.build();
          if (unknow_taxa.size()>0)
          {
              String errmsg = "-1 -1\nUnknown taxa names:\n";
              for (int i=0; i<unknow_taxa.size(); i++)
              {
                  errmsg += unknow_taxa.get(i) + "\n";
              }
              return errmsg;
          }
          gt_taxa_filtered =
FilterGenotypeTable.getInstance(source_gt_table, dest_taxa_list);
      }
      //create sites filtered genotype table
      GenotypeTable gt_taxa_sites_filtered = null;

      //create position filtered genotype table
      if (ChromosomeStr.toLowerCase().equalsIgnoreCase("all"))
      {
          gt_taxa_sites_filtered = gt_taxa_filtered;
      }
      else
      {
          Chromosome ch = source_gt_table.chromosome(ChromosomeStr);
          if (ch == null)
          {
              return "-1 -1\n" + "The chromosome name '" +
ChromosomeStr +  "' is not recognized\n";
          }
          if ((StartPhysicalPosition==0) || (EndPhysicalPosition==0))
          {
              gt_taxa_sites_filtered =
FilterGenotypeTable.getInstance(gt_taxa_filtered, ch);
          }
          else
          {
              gt_taxa_sites_filtered =
FilterGenotypeTable.getInstance(gt_taxa_filtered, ch,
StartPhysicalPosition, EndPhysicalPosition);
          }

      }
        BufferedWriter writer = null;
        try {

                       File outFile = new File(dest_file + ".txt");
                       writer = new BufferedWriter(new
FileWriter(outFile));
                    System.out.println(outFile.getCanonicalPath());

                       writer.write("Selected Taxa : " +
gt_taxa_sites_filtered.taxaName(0));
                       writer.write("\nSelected Chromosome : " +
gt_taxa_sites_filtered.chromosome(ChromosomeStr).toString());
                       writer.write("\nSelected Position : " +
StartPhysicalPosition);

               // Get the Unique site number for the specific physical
position and specific Chromosome.
               int site =
source_gt_table.siteOfPhysicalPosition(StartPhysicalPosition,
gt_taxa_sites_filtered.chromosome(ChromosomeStr));

               writer.write("\nSite Value for the selected Chromosome &
Position : " + site);
               writer.write("\nGenotype Present at the selected
Chromosome & Position in the selected Taxa : " +
gt_taxa_sites_filtered.genotypeAsString(0,0));
               writer.write("\nGenotype byte value : " +
gt_taxa_sites_filtered.genotype(0,0));

               byte[] allGenotypeDiploidValues =
source_gt_table.genotypeAllTaxa(site);
               byte genotypeDiploidValueAtSpecificPosition =
gt_taxa_sites_filtered.genotype(0,0);
               BitSet matchedTaxa =
GenotypeTableUtils.calcBitPresenceOfDiploidValueFromGenotype(allGenotyp
eDiploidValues, genotypeDiploidValueAtSpecificPosition);
               int[] matchedPositions =
matchedTaxa.getIndicesOfSetBits();
               writer.write("\nNumber of matches : " +
matchedPositions.length);
               writer.write("\nList of all Taxa containing the same
Genotype at the same Chromosome & Position : ");

               for (int i : matchedPositions)
                       {
                       writer.write("\n" + source_gt_table.taxaName(i));
```

```
                   }
    } catch (IOException e) {
                   // TODO Auto-generated catch block
                   e.printStackTrace();
            }
    finally
    {
           try {
                           writer.close();
                   } catch (IOException e) {
                           // TODO Auto-generated catch block
                           e.printStackTrace();
                   }
    }
    System.out.println("Done");
  return "";

}

private static String sliceRangeNuke(String[] args)
{

    //get parameters
    ArgsEngine  myArgsEngine = new ArgsEngine();
    myArgsEngine.add("-sf", "--sourcefile", true);
    myArgsEngine.add("-st", "--sourcefile-type", true);
    myArgsEngine.add("-df", "--destinationfile", true);
    myArgsEngine.add("-dt", "--destinationfile-type", true);
    myArgsEngine.add("-tf", "--taxa-file", true);
    myArgsEngine.add("-tl", "--taxa-list", true);
    myArgsEngine.add("-ch", "--chromosome", true);
    myArgsEngine.add("-start", "--chr-start", true);
    myArgsEngine.add("-end", "--chr-end", true);
    myArgsEngine.add("-bf", "--build-file", false);
    myArgsEngine.parse(args);

   boolean buildfile = false;
   String source_file = null;
   String source_file_type = null;
   String dest_file = null;
   String dest_file_type=null;
   String TaxaListFile=null;
   ArrayList<String> TaxaArrayList = new ArrayList<String>();
   String ChromosomeStr=null;
   int StartPhysicalPosition = 0;
   int EndPhysicalPosition = 0;

   if (myArgsEngine.getBoolean("-bf"))
   {
        buildfile = true;
   }

  if (myArgsEngine.getBoolean("-sf")) {
      source_file = myArgsEngine.getString("-sf");
  } else {
      printUsage("slice");
```

```
          throw new IllegalArgumentException("Please specify a source
file (option -sf).");
      }

      if (myArgsEngine.getBoolean("-st")) {
          source_file_type = myArgsEngine.getString("-st");
      } else {
          printUsage("slice");
          throw new IllegalArgumentException("Please specify a source
file type (option -st).");
      }

      if (buildfile)
      {
          if (myArgsEngine.getBoolean("-df")) {
              dest_file = myArgsEngine.getString("-df");
          } else {
              printUsage("slice");
              throw new IllegalArgumentException("Please specify a
destination file (option -df).");
          }

          if (myArgsEngine.getBoolean("-dt")) {
              dest_file_type = myArgsEngine.getString("-dt");
          } else {
              printUsage("slice");
              throw new IllegalArgumentException("Please specify a
destination file type (option -dt).");
          }
      }

      if (myArgsEngine.getBoolean("-tf"))
      {
          TaxaListFile = myArgsEngine.getString("-tf");
          File outDirectory = new File(TaxaListFile);
          if (!outDirectory.isFile()) {
              printUsage("slice");
              throw new IllegalArgumentException("The taxa file you
supplied (option -tf) is not a file: " + TaxaListFile);
          }
          //verify and create sub-taxalist
          //create taxa filtered genotype table

         try {
              BufferedReader br = new BufferedReader(new
FileReader(TaxaListFile), 65536);

              String temp;
              int currLine = 0;
              while (((temp = br.readLine()) != null)) {
                  if (!temp.trim().isEmpty())
                  {
                      TaxaArrayList.add(temp.trim());
                      currLine++;
                  }

              }
          } catch (Exception e) {
              System.out.println("Couldn't open taxa file to read taxa
list: " + e);
          }
      }
      else if (myArgsEngine.getBoolean("-tl"))
      {
          String TaxaListStr = myArgsEngine.getString("-tf");
          if (TaxaListStr.equalsIgnoreCase("all"))
          {
              TaxaArrayList.add("all");
          }
          else
          {
              String[] wordList = TaxaListStr.split(";");
              TaxaArrayList.addAll(Arrays.asList(wordList));
          }
      }
      else {
          printUsage("slice");
          throw new IllegalArgumentException("Please specify the file
with taxa list (option -tf).");
      }

      if (myArgsEngine.getBoolean("-ch")) {
          ChromosomeStr = myArgsEngine.getString("-ch");
      } else {
          if ((myArgsEngine.getBoolean("-start")) ||
(myArgsEngine.getBoolean("-end")))
          {
              printUsage("slice");
              throw new IllegalArgumentException("Please specify a
chromosome name (option -ch).");
          }
          else
          {
              ChromosomeStr = "all";
          }
      }

      if (myArgsEngine.getBoolean("-start")) {
          StartPhysicalPosition =
Integer.parseInt(myArgsEngine.getString("-start"));
      }

      if (myArgsEngine.getBoolean("-end")) {
          EndPhysicalPosition =
Integer.parseInt(myArgsEngine.getString("-end"));
      }

       //create a genotypetable from source
      GenotypeTable source_gt_table = null;
      if (source_file_type.equalsIgnoreCase("hdf5"))
      {
          source_gt_table =
GenotypeTableBuilder.getInstance(source_file);
      }
      else if (source_file_type.equalsIgnoreCase("hapmap"))
      {
          source_gt_table = ImportUtils.readFromHapmap(source_file);
      }
      else if (source_file_type.equalsIgnoreCase("vcf"))
      {
          source_gt_table = ImportUtils.readFromVCF(source_file, null,
false);
      }




      GenotypeTable gt_taxa_filtered = null ;



      //if TaxaListString=all or not specified, all taxa included
      if ((TaxaArrayList.get(0).equalsIgnoreCase("all")) ||
(TaxaArrayList.size()==0))
      {
          gt_taxa_filtered = source_gt_table;
      }
      //if TaxaListString specified, create subset
      else
      {
          TaxaList source_taxa_list = source_gt_table.taxa();
          TaxaList dest_taxa_list=null;
          TaxaListBuilder dest_taxa_list_builder = new
TaxaListBuilder();

          ArrayList<String> unknow_taxa = new ArrayList<String>();
          for (int i =0; i<TaxaArrayList.size(); i++)
          {
             if (TaxaArrayList.get(i).matches(".*\\w.*"))
             {
                  int itaxa_index =
source_taxa_list.indexOf(TaxaArrayList.get(i));
                  if (itaxa_index>=0)
                  {

dest_taxa_list_builder.add(source_taxa_list.get(itaxa_index));
                  }
                  else
                  {
                      unknow_taxa.add(TaxaArrayList.get(i));
                  }
             }
          }
          dest_taxa_list = dest_taxa_list_builder.build();
          if (unknow_taxa.size()>0)
          {
              String errmsg = "-1 -1\nUnknown taxa names:\n";
              for (int i=0; i<unknow_taxa.size(); i++)
              {
                  errmsg += unknow_taxa.get(i) + "\n";
              }
              return errmsg;
          }
          gt_taxa_filtered =
FilterGenotypeTable.getInstance(source_gt_table, dest_taxa_list);
      }
      //create sites filtered genotype table
      GenotypeTable gt_taxa_sites_filtered = null;

      //create position filtered genotype table
      if (ChromosomeStr.toLowerCase().equalsIgnoreCase("all"))
      {
          gt_taxa_sites_filtered = gt_taxa_filtered;
      }
      else
      {
          Chromosome ch = source_gt_table.chromosome(ChromosomeStr);
          if (ch == null)
          {
              return "-1 -1\n" + "The chromosome name '" +
ChromosomeStr +  "' is not recognized\n";
          }
          if ((StartPhysicalPosition==0) || (EndPhysicalPosition==0))
          {
              gt_taxa_sites_filtered =
FilterGenotypeTable.getInstance(gt_taxa_filtered, ch);
          }
          else
          {
              gt_taxa_sites_filtered =
FilterGenotypeTable.getInstance(gt_taxa_filtered, ch,
StartPhysicalPosition, EndPhysicalPosition);
          }

      }
        BufferedWriter writer = null;
        try {

                       File outFile = new File(dest_file + ".txt");
                       writer = new BufferedWriter(new
FileWriter(outFile));
                    System.out.println(outFile.getCanonicalPath());

                       writer.write("Selected Taxa : " +
gt_taxa_sites_filtered.taxaName(0));
                       writer.write("\nSelected Chromosome : " +
gt_taxa_sites_filtered.chromosome(ChromosomeStr).toString());
                       writer.write("\nSelected Positions : ");
                       ArrayList sites = new ArrayList<Integer>();
                       int c=0;
                       for (int i :
gt_taxa_sites_filtered.physicalPositions())
                       {

        sites.add(source_gt_table.siteOfPhysicalPosition(i,
gt_taxa_sites_filtered.chromosome(ChromosomeStr)));
                               writer.write("\n" + i + " - Site Value -
" + sites.get(c) +" - Genotype present - " +
gt_taxa_sites_filtered.genotypeAsString(0,c) + " - Byte Value - " +
gt_taxa_sites_filtered.genotype(0,c));
                                c++;
                        }

                        ArrayList<BitSet> allMatchs = new
ArrayList<BitSet>();

                        for (int i = 0; i<c; i++)
                        {
                                byte
genotypeDiploidValueAtSpecificPosition =
gt_taxa_sites_filtered.genotype(0,i);
                                if
(genotypeDiploidValueAtSpecificPosition == -1) break;
                                byte[] allGenotypeDiploidValues =
source_gt_table.genotypeAllTaxa((int) sites.get(i));
                                BitSet matchedTaxa =
GenotypeTableUtils.calcBitPresenceOfDiploidValueFromGenotype(allGenotyp
eDiploidValues, genotypeDiploidValueAtSpecificPosition);
                                allMatchs.add(matchedTaxa);
                        }

                        BitSet globalMatch = allMatchs.get(0);
                        for (int i=1; i<allMatchs.size(); i++)
                        {
                                globalMatch.and(allMatchs.get(i));
                        }

                int[] matchedPositions =
globalMatch.getIndicesOfSetBits();
                writer.write("\nNumber of matches : " +
matchedPositions.length);
                writer.write("\nList of all Taxa containing the same
Genotype at the same Chromosome & Position : ");

                for (int i : matchedPositions)
                        {
                        writer.write("\n" + source_gt_table.taxaName(i));
                        }
        } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
        finally
        {
                try {
                                writer.close();
                        } catch (IOException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
        }
        System.out.println("Done");
      return "";

    }
    private static String dbinfo (String[] args)
    {
          //get parameters
          ArgsEngine  myArgsEngine = new ArgsEngine();
          myArgsEngine.add("-sf", "--sourcefile", true);
          myArgsEngine.add("-st", "--sourcefile-type", true);
          myArgsEngine.add("-dbname", "--dbname", true);
          myArgsEngine.add("-o", "--out", true);
          myArgsEngine.parse(args);
          String source_file ="";
          String source_file_type = "";
          String dbname = "";
          String outdir = "";
         if (myArgsEngine.getBoolean("-sf")) {
            source_file = myArgsEngine.getString("-sf");
          } else {
            printUsage("dbinfo");
            throw new IllegalArgumentException("Please specify a source
file (option -sf).");
        }

        if (myArgsEngine.getBoolean("-st")) {
            source_file_type = myArgsEngine.getString("-st");
        } else {
            printUsage("dbinfo");
            throw new IllegalArgumentException("Please specify a source
file type (option -st).");
        }

        if (myArgsEngine.getBoolean("-dbname")) {
            dbname = myArgsEngine.getString("-dbname");
        } else {
            printUsage("dbinfo");
            throw new IllegalArgumentException("Please specify a
database name (option -dbname ).");
        }

        if (myArgsEngine.getBoolean("-o")) {
            outdir = myArgsEngine.getString("-o");
        } else {
            outdir = ".";
        }

        //create a genotypetable from source
        GenotypeTable source_gt_table = null;
        if (source_file_type.equalsIgnoreCase("hdf5"))
        {
            source_gt_table =
GenotypeTableBuilder.getInstance(source_file);
        }
        else if (source_file_type.equalsIgnoreCase("hapmap"))
        {
            source_gt_table = ImportUtils.readFromHapmap(source_file);
        }
        else if (source_file_type.equalsIgnoreCase("vcf"))
        {
            source_gt_table = ImportUtils.readFromVCF(source_file, 
null, false);
        }


        BufferedWriter bw = null;
        //write the taxa file
        try {
            bw = Utils.getBufferedWriter(outdir + "/" + dbname + 
".taxainfo");
            TaxaList taxalist  = source_gt_table.taxa();
            for (int i=0; i<taxalist.size(); i++)
            {
                Taxon t = taxalist.get(i);
                Map.Entry<String,String>[] tannotations = 
t.getAnnotation().getAllAnnotationEntries(); // 
t.getAllAnnotationEntries();
                if (tannotations.length>0)
                {
                    for (Map.Entry<String,String> ta:tannotations)
                    {
                          bw.write(dbname + "\t" + t.getName() + "\t" + 
ta.getKey() + "\t" + ta.getValue() + "\n");
                    }
                }
                else
                {
                    bw.write(dbname + "\t" + t.getName() + "\t" + "" + 
"\t" + "" + "\n");
                }
            }
            bw.flush();
            bw.close();
        } catch (Exception e) {
            e.printStackTrace();
            throw new IllegalArgumentException("Error writing dbinfo 
taxainfo file: " + outdir + "/" + dbname + ".taxainfo " + ": " + 
ExceptionUtils.getExceptionCauses(e));
        } finally {
            try {
                bw.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        //write chromosome file
        try {
            bw = Utils.getBufferedWriter(outdir + "/" + dbname + 
".chrinfo");
            Chromosome[] chrlist  = source_gt_table.chromosomes();
            for (int i=0; i<chrlist.length; i++)
            {
                Chromosome c = chrlist[i];
                int sitecount = source_gt_table.chromosomeSiteCount(c);
                //int[] firstlast = 
source_gt_table.firstLastSiteOfChromosome(c);
```

```
                bw.write("\t" + dbname + "\t" + c.getName() + "\t" +
c.getLength() + "\t" + sitecount  + "\n");
            }
            bw.flush();
            bw.close();
        } catch (Exception e) {
            e.printStackTrace();
            throw new IllegalArgumentException("Error writing chrinfo
file: " + outdir + "/" + dbname + ".chrinfo " + ": " +
ExceptionUtils.getExceptionCauses(e));
        } finally {
            try {
                bw.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        //write dbinfo
        try {
            bw = Utils.getBufferedWriter(outdir + "/" + dbname +
".dbinfo");
            String genomeversion = source_gt_table.genomeVersion();
            int hasdepth =0;
            if (source_gt_table.hasDepth())
            {
                hasdepth=1;
            }
            String date_created = "";
            String tassel_version = "";
            String description = "DB description unavailable.";
            bw.write("\t" + dbname + "\t" + description + "\t" +
genomeversion + "\t" + source_file + "\t" + source_file_type + "\t" +
date_created + "\t" + hasdepth + "\t" + tassel_version + "\n" );
            bw.flush();
            bw.close();
        } catch (Exception e) {
            e.printStackTrace();
            throw new IllegalArgumentException("Error writing dbinfo
dbinfo file: " + outdir + "/" + dbname + "dbinfo " + ": " +
ExceptionUtils.getExceptionCauses(e));
        } finally {
            try {
                bw.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        return "done";

    }
    private static String  slice (String[] args)
    {
          //get parameters
          ArgsEngine  myArgsEngine = new ArgsEngine();
          myArgsEngine.add("-sf", "--sourcefile", true);
          myArgsEngine.add("-st", "--sourcefile-type", true);
          myArgsEngine.add("-df", "--destinationfile", true);
          myArgsEngine.add("-dt", "--destinationfile-type", true);
          myArgsEngine.add("-tf", "--taxa-file", true);
          myArgsEngine.add("-tl", "--taxa-list", true);
          myArgsEngine.add("-ch", "--chromosome", true);
          myArgsEngine.add("-start", "--chr-start", true);
          myArgsEngine.add("-end", "--chr-end", true);
          myArgsEngine.add("-bf", "--build-file", false);
          myArgsEngine.parse(args);

         boolean buildfile = false;
         String source_file = null;
         String source_file_type = null;
         String dest_file = null;
         String dest_file_type=null;
         String TaxaListFile=null;
         ArrayList<String> TaxaArrayList = new ArrayList<String>();
         String ChromosomeStr=null;
         int StartPhysicalPosition = 0;
         int EndPhysicalPosition = 0;

         if (myArgsEngine.getBoolean("-bf"))
         {
              buildfile = true;
         }

        if (myArgsEngine.getBoolean("-sf")) {
            source_file = myArgsEngine.getString("-sf");
        } else {
            printUsage("slice");
            throw new IllegalArgumentException("Please specify a source
file (option -sf).");
        }

        if (myArgsEngine.getBoolean("-st")) {
            source_file_type = myArgsEngine.getString("-st");
        } else {
            printUsage("slice");
            throw new IllegalArgumentException("Please specify a source
file type (option -st).");
        }

        if (buildfile)
        {
            if (myArgsEngine.getBoolean("-df")) {
                dest_file = myArgsEngine.getString("-df");
            } else {
                printUsage("slice");
                throw new IllegalArgumentException("Please specify a
destination file (option -df).");
            }

            if (myArgsEngine.getBoolean("-dt")) {
                dest_file_type = myArgsEngine.getString("-dt");
            } else {
                printUsage("slice");
                throw new IllegalArgumentException("Please specify a
destination file type (option -dt).");
            }
        }

        if (myArgsEngine.getBoolean("-tf"))
        {
            TaxaListFile = myArgsEngine.getString("-tf");
            File outDirectory = new File(TaxaListFile);
            if (!outDirectory.isFile()) {
                printUsage("slice");
                throw new IllegalArgumentException("The taxa file you
supplied (option -tf) is not a file: " + TaxaListFile);
            }
            //verify and create sub-taxalist
            //create taxa filtered genotype table

           try {
                BufferedReader br = new BufferedReader(new
FileReader(TaxaListFile), 65536);

                String temp;
                int currLine = 0;
                while (((temp = br.readLine()) != null)) {
                    if (!temp.trim().isEmpty())
                    {
                        TaxaArrayList.add(temp.trim());
                        currLine++;
                    }

                }
            } catch (Exception e) {
                System.out.println("Couldn't open taxa file to read
taxa list: " + e);
            }
        }
        else if (myArgsEngine.getBoolean("-tl"))
        {
            String TaxaListStr = myArgsEngine.getString("-tf");
            if (TaxaListStr.equalsIgnoreCase("all"))
            {
                TaxaArrayList.add("all");
            }
            else
            {
                String[] wordList = TaxaListStr.split(";");
                TaxaArrayList.addAll(Arrays.asList(wordList));
            }
        }
        else {
            printUsage("slice");
            throw new IllegalArgumentException("Please specify the file
with taxa list (option -tf).");
        }

        if (myArgsEngine.getBoolean("-ch")) {
            ChromosomeStr = myArgsEngine.getString("-ch");
        } else {
            if ((myArgsEngine.getBoolean("-start")) ||
(myArgsEngine.getBoolean("-end")))
            {
                printUsage("slice");
                throw new IllegalArgumentException("Please specify a
chromosome name (option -ch).");
            }
            else
            {
                ChromosomeStr = "all";
            }
        }

        if (myArgsEngine.getBoolean("-start")) {
            StartPhysicalPosition =
Integer.parseInt(myArgsEngine.getString("-start"));
        }

        if (myArgsEngine.getBoolean("-end")) {
            EndPhysicalPosition =
Integer.parseInt(myArgsEngine.getString("-end"));
        }

         //create a genotypetable from source
        GenotypeTable source_gt_table = null;
        if (source_file_type.equalsIgnoreCase("hdf5"))
        {
            source_gt_table =
GenotypeTableBuilder.getInstance(source_file);
        }
        else if (source_file_type.equalsIgnoreCase("hapmap"))
        {
            source_gt_table = ImportUtils.readFromHapmap(source_file);
        }
        else if (source_file_type.equalsIgnoreCase("vcf"))
        {
            source_gt_table = ImportUtils.readFromVCF(source_file,
null, false);
        }



        GenotypeTable gt_taxa_filtered = null ;



        //if TaxaListString=all or not specified, all taxa included
        if ((TaxaArrayList.get(0).equalsIgnoreCase("all")) ||
(TaxaArrayList.size()==0))
        {
            gt_taxa_filtered = source_gt_table;
        }
        //if TaxaListString specified, create subset
        else
        {
            TaxaList source_taxa_list = source_gt_table.taxa();
            TaxaList dest_taxa_list=null;
            TaxaListBuilder dest_taxa_list_builder = new
TaxaListBuilder();

            ArrayList<String> unknow_taxa = new ArrayList<String>();
            for (int i =0; i<TaxaArrayList.size(); i++)
            {
               if (TaxaArrayList.get(i).matches(".*\\w.*"))
               {
                    int itaxa_index =
source_taxa_list.indexOf(TaxaArrayList.get(i));
                    if (itaxa_index>=0)
                    {

dest_taxa_list_builder.add(source_taxa_list.get(itaxa_index));
                    }
                    else
                    {
                        unknow_taxa.add(TaxaArrayList.get(i));
                    }
               }
            }
            dest_taxa_list = dest_taxa_list_builder.build();
            if (unknow_taxa.size()>0)
            {
                String errmsg = "-1 -1\nUnknown taxa names:\n";
                for (int i=0; i<unknow_taxa.size(); i++)
                {
                    errmsg += unknow_taxa.get(i) + "\n";
                }
                return errmsg;
            }
            gt_taxa_filtered =
FilterGenotypeTable.getInstance(source_gt_table, dest_taxa_list);
        }


        //create sites filtered genotype table
        GenotypeTable gt_taxa_sites_filtered = null;

        //create position filtered genotype table
        if (ChromosomeStr.toLowerCase().equalsIgnoreCase("all"))
        {
            gt_taxa_sites_filtered = gt_taxa_filtered;
        }
        else
        {
            Chromosome ch = source_gt_table.chromosome(ChromosomeStr);
            if (ch == null)
            {
                return "-1 -1\n" + "The chromosome name '" +
ChromosomeStr +  "' is not recognized\n";
            }
            if ((StartPhysicalPosition==0) || (EndPhysicalPosition==0))
            {
                gt_taxa_sites_filtered =
FilterGenotypeTable.getInstance(gt_taxa_filtered, ch);
            }
            else
            {
                gt_taxa_sites_filtered =
FilterGenotypeTable.getInstance(gt_taxa_filtered, ch,
StartPhysicalPosition, EndPhysicalPosition);
            }

        }
        String return_Values = "";
        if (gt_taxa_sites_filtered == null)
        {
            return "-1 -1\nNo data after filtering!\n";
        }
        else
        {
            return_Values= gt_taxa_sites_filtered.taxa().size() + " " +
gt_taxa_sites_filtered.positions().size();
        }
        if (buildfile)
        {
            if (dest_file_type.equalsIgnoreCase("hdf5"))
            {
                ExportUtils.writeGenotypeHDF5(gt_taxa_sites_filtered,
dest_file, source_gt_table.hasDepth());
            }
            else if (dest_file_type.equalsIgnoreCase("hapmap"))
            {
                ExportUtils.writeToHapmap(gt_taxa_sites_filtered,
dest_file);
            }
            else if (dest_file_type.equalsIgnoreCase("vcf"))
            {
                ExportUtils.writeToVCF(gt_taxa_sites_filtered,
dest_file, source_gt_table.hasDepth());
            }
            else if (dest_file_type.equalsIgnoreCase("plink"))
            {
                ExportUtils.writeToPlink(gt_taxa_sites_filtered,
dest_file, ' ');
            }
            else
            {
                //System.err.println ("File format '" + dest_file_type
+ "' not recognized!");
                return "-1 -1\n" + "Output file format '" +
dest_file_type + "' not recognized!";
            }
        }



        return return_Values;
    }
    private static String[]  get_taxa_list (String[] args)
    {
        ArgsEngine  myArgsEngine = new ArgsEngine();
          myArgsEngine.add("-sf", "--sourcefile", true);
          myArgsEngine.add("-st", "--sourcefile-type", true);
          myArgsEngine.parse(args);

         String source_file = null;
         String source_file_type = null;

        if (myArgsEngine.getBoolean("-sf")) {
            source_file = myArgsEngine.getString("-sf");
        } else {
            printUsage("get_taxa_list");
            throw new IllegalArgumentException("Please specify a source
file (option -sf).");
        }

        if (myArgsEngine.getBoolean("-st")) {
            source_file_type = myArgsEngine.getString("-st");
        } else {
            printUsage("get_taxa_list");
            throw new IllegalArgumentException("Please specify a source
file type (option -st).");
        }

        //create a genotypetable from source
        GenotypeTable source_gt_table = null;
        if (source_file_type.equalsIgnoreCase("hdf5"))
        {
            source_gt_table =
GenotypeTableBuilder.getInstance(source_file);
        }
        else if (source_file_type.equalsIgnoreCase("hapmap"))
        {
            source_gt_table = ImportUtils.readFromHapmap(source_file);
        }
        else if (source_file_type.equalsIgnoreCase("vcf"))
        {
            source_gt_table = ImportUtils.readFromVCF(source_file,
null, false);
        }

        TaxaList taxalist =source_gt_table.taxa();
        int taxaCount = taxalist.size();
        String[] taxa_array = new String[taxaCount];
        for (int i=0; i<taxaCount; i++)
        {
            taxa_array[i] = taxalist.taxaName(i);
        }
        return taxa_array;
    }

    private static String[]  chr_marker_info (String[] args)
    {
        ArgsEngine  myArgsEngine = new ArgsEngine();
```

```
          myArgsEngine.add("-sf", "--sourcefile", true);
          myArgsEngine.add("-st", "--sourcefile-type", true);
          myArgsEngine.parse(args);

         String source_file = null;
         String source_file_type = null;

        if (myArgsEngine.getBoolean("-sf")) {
            source_file = myArgsEngine.getString("-sf");
        } else {
            printUsage("chr_marker_info");
            throw new IllegalArgumentException("Please specify a source
file (option -sf).");
        }

        if (myArgsEngine.getBoolean("-st")) {
            source_file_type = myArgsEngine.getString("-st");
        } else {
            printUsage("chr_marker_info");
            throw new IllegalArgumentException("Please specify a source
file type (option -st).");
        }

                 //create a genotypetable from source
        GenotypeTable source_gt_table = null;
        if (source_file_type.equalsIgnoreCase("hdf5"))
        {
            source_gt_table =
GenotypeTableBuilder.getInstance(source_file);
        }
        else if (source_file_type.equalsIgnoreCase("hapmap"))
        {
            source_gt_table = ImportUtils.readFromHapmap(source_file);
        }
        else if (source_file_type.equalsIgnoreCase("vcf"))
        {
            source_gt_table = ImportUtils.readFromVCF(source_file,
null, false);
        }

        Chromosome[] chrlist =source_gt_table.chromosomes();
        int chrCount = chrlist.length;
        String[] chr_array = new String[chrCount];
        for (int i=0; i<chrCount; i++)
        {
            int[] firstlastpos =
source_gt_table.firstLastSiteOfChromosome(chrlist[i]);
            int firstposition =
source_gt_table.chromosomalPosition(firstlastpos[0]);
            int lastposition =
source_gt_table.chromosomalPosition(firstlastpos[1]);
            chr_array[i] = chrlist[i].getName() + "\t" +
source_gt_table.chromosomeSiteCount(chrlist[i]) +  "\t" + firstposition
+ "\t" + lastposition;
        }
        return chr_array;
    }
    private static void printUsage (String menuName)
    {
        if (menuName.equals("main"))
        {
            System.out.print("Usage:\tjava -jar Tassel_gt_server.jar
[options]\n\n"
                    +"Command:\tslice\tcreate a slice from the genotype
file\n"
                    +"\t\tdbinfo\twrite the database information into
files\n");
        }
        else if (menuName.equals("slice"))
        {
            System.out.print("Usage:\tjava -jar Tassel_gt_server.jar
slice [options]\n\n"
                    +"Options\t-bf\tbuild the file. If skip, only give
the dimensions\n"
                    +"\t\t-sf\tsource file name\n"
                    +"\t\t-st\tsource file type (hdf5, vcf, or
hapmap)\n"
                    +"\t\t-df\tdestination file name\n"
                    +"\t\t-dt\tdestination file type (hdf5, vcf, hapmap
or plink)\n"
                    +"\t\t-tf\ttaxa file name\n"
                    +"\t\t-ch\tchromosome name\n"
                    +"\t\t-start\tphysical start position on the
chromosome\n"
                   +"\t\t-end\tphysical end position on the
chromosome\n"
                    );
        }
        else if (menuName.equals("get_taxa_list"))
        {
            System.out.print("Usage:\tjava -jar Tassel_gt_server.jar
get_taxa_list [options]\n\n"
                    +"Options\t-bf\tbuild the file. If skip, only give
the dimensions\n"
                    +"\t\t-sf\tsource file name\n"
                    +"\t\t-st\tsource file type (hdf5, vcf, or
hapmap)\n"
                    );
        }
        else if (menuName.equals("dbinfo"))
        {
            System.out.print("Usage:\tjava -jar Tassel_gt_server.jar
dbinfo [options]\n\n"
                    +"Options\t-bf\tbuild the file. If skip, only give
the dimensions\n"
                    +"\t\t-sf\tsource file name\n"
                    +"\t\t-st\tsource file type (hdf5, vcf, or
hapmap)\n"
                    +"\t\t-dbname\tdatabase name in one word\n"
                    +"\t\t-o\toutput directory\n"
                    );
        }
        else if (menuName.equals("chr_marker_info"))
        {
            System.out.print("Usage:\tjava -jar Tassel_gt_server.jar
chr_marker_info [options]\n\n"
                    +"Options\t-bf\tbuild the file. If skip, only give
the dimensions\n"
                    +"\t\t-sf\tsource file name\n"
                    +"\t\t-st\tsource file type (hdf5, vcf, or
hapmap)\n"
                    );
        }
        else
        {
            printUsage("main");
        }
    }

}
```