Team Selene

Software Design Document

Web Visualization of High Dimensionality Spatial Data

February 9th, 2017

Project Sponsor: Dr. Jay Laura, USGS Astrogeology

Faculty Mentor: Dr. Palmer

Team Lead: Daniel Ohn

Team: Zowie Haugaard, Christopher Philabaum, Kelvin Rodriguez, Makayla Shepherd

1. Introduction

Planetary scientists study the planets, moons, and planetary systems of our universe, as well as their origins, formation, and processes of these bodies. Through the study of their composition, formation, and dynamics, these scientists use geophysics, atmospheric science, astrobiology, and other physical sciences to gain a better understanding of our solar system and celestial bodies, as well as our own planet Earth. Thanks to the relatively high number of observational spacecrafts currently exploring our solar system, planetary scientists have today access to a great deal of data collected from various planets and moons within our solar system, and the rate new discoveries today is very high. Yet there remain many unanswered questions within the discipline of planetary science, and in order to continue the progress of these discoveries these scientists require technologies that can assist in the analysis and exploration of often very large and complex data and problem sets.

The United States Geological Survey Astrogeology Science Center (USGS) in Flagstaff, AZ was founded to assist in the training of astronauts embarking on missions to Earth's moon, and to survey and map the lunar surface. Today they are doing important research into the geology and composition of the Moon, and employ a team of geologists, planetary scientists, volcanologists, software engineers, and others to further discoveries in this area. Recently, the USGS has been given access to a data set containing hyperspectral observation data collected by the Japanese Aerospace Exploration Agency through their Kaguya lunar orbiter, also known as the SELENE lunar orbiter. This data, which in entirety is 1.4 TB in size, can through its analysis provide novel insights into the geologic makeup and composition of Earth's moon, as well as its origins.

In order to analyze this data, planetary scientists use several tools in their study. To receive this hyperspectral data, scientists first query the database by providing the specific geographical coordinates of their predetermined region of interest. They must then download the resulting data in its entirety, and load it into a Geographical Information System (GIS) for plotting and analysis. Through this process, scientists are able to access and analyze this data and make new discoveries pertaining the geology of Earth's moon.

To assist the study of this dataset, Team Selene is designing and implementing a web application for the access, visualization, exploration, and analysis of the high dimensionality spatial data captured by the Kaguya lunar orbiter. This system will greatly improve and streamline the current methodologies used in the access and analysis of this dataset, and will provide key tools to assist planetary scientists in making discoveries. Specific key features this system will implement include:

- Generate a global plot of the distribution of observations
- Generate graph of observation data for each geographical point
- Allow users to explore the entire data set using tools such as pan and zoom

This document is intended to outline the approach of Team Selene in designing and implementing this application, including overviews of the system architecture and interfaces. We are confident that the solution we envision will meet the requirements of the USGS, and will be of assistance in the important work that planetary scientists are performing every day.

2. Implementation Overview

Team Selene's focus is on designing on a web application meant to visualize and explore Kaguya's large scale datasets. This web application will be based around the classic server-client pattern. Where part of the solution stems on, however, is a pipeline on the backend. By using a pipeline, Team Selene can take advantage of pre-existing technologies and frameworks with as little customization needed.

The server will be composed of several long-standing frameworks. One such framework, MongoDB[1], will be the database used to store the point and image data. Point queries will be made from and passed through to GeoServer. GeoServer[2] is an open-source web server that will help to serve and process geospatial data, in this case Moon-spatial data. Using RESTful commands, GeoServer will gather the hyperspectral and spatial data from the database.

The client, in turn, will be based on Web 2.0 technologies. For instance, HTML5 and ES2015+-compliant JavaScript are examples of modern technologies to be used. Relating to JavaScript frameworks, Leaflet will be used. Leaflet[3] is a framework that displays map data akin to Google Maps and Bing Maps; this will be the main interface for the client, allowing the user to appropriately zoom in/out, move, etc.

The large crux of solving the large dataset problem will be to customize these frameworks to selectively display only the necessary data dependent upon the right zoom level. At higher zoom levels, that is viewing a more zoomed out map, the GeoServer will serve rasterized map images utilizing the Python library Datashader[4] to generate raster images from points in real time. When zoomed further in, Team Selene's customized pipeline will instead serve the individualized point data as vectors as opposed to the higher level rasterized data. This will allow the client to retrieve only the data it needs to manipulate when a user is reasonably able to to interact with individual points on the map.

[1] MongoDB <https://www.mongodb.com/>
[2] GeoServer <http://geoserver.org/>
[3] Leaflet <http://leafletjs.com/>
[4] Datashader <https://github.com/bokeh/datashader>

3. Architectural Overview

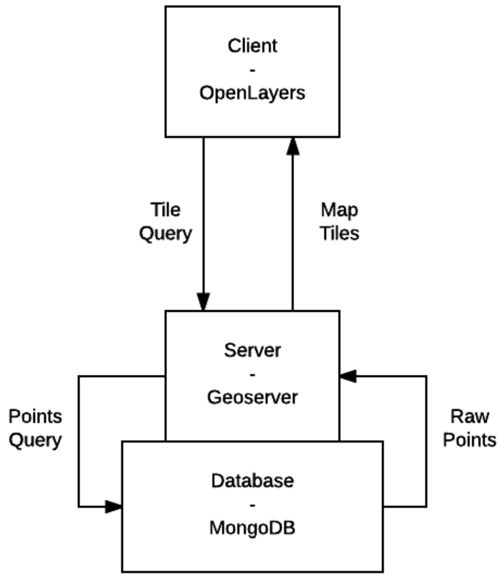


Fig 1 high level communication between the client-server architecture of Team Selene's system.

In order to produce a system that will satisfy the needs of our client, a carefully crafted system architecture must be implemented. As this application will be served as a web app, the overall architecture follows the client-server paradigm, as illustrated in Fig 1. The base of the architecture is a MongoDB system used to store and access the Kaguya observation data. Directly communicating with this database will be a GeoServer instance, which will act as the server in the client-server architecture. This server will be the intermediary between the client and the database, processing queries and serving the results. Finally, the client represents the software which will render the hyperspectral and spatial data provided by the server.

The client is a Web 2.0 HTML5 JavaScript frontend using Leaflet to read map tiles and interface with the map. The key responsibilities of the client are to provide a user interface that allows for panning and zooming, and to connect to GeoServer that is used for querying and serving raster data over the network.

GeoServer is a tile server that utilizes Web Map Service (WMS) to bridge the gap between the client and the database and serve the map data. GeoServer will receive and process tile requests from the client, translate the tile request into points request. Then GeoServer will query the database, rasterize the points that the database returns into tiles and serve them back to the client. The process of rasterizing the data to return to the client will take place in a micro pipe-and-filter architecture.

The database is a MongoDB database that will store the points and their respective metadata and serve the points when they are requested.

The communication protocol used between the OpenLayers client and GeoServer will be WMS as it is an industry standard format. In between the database and GeoServer Team Selene will use the GeoServer REST API. The overall architecture of the system is client-server, however GeoServer will contain a small pipe-and-filter system in order to transform the point data into tiles for the client.

4. Module and Interface Descriptions

4.1 Client

4.1.1 Overview

The client module represents the web page running on the user's browser. The client based software will render the geospatial and hyperspectral data, and allow for users to interact with the data set in the context of a map based application. Our map interface will be built using Leaflet, an open source library for the creation of dynamic and interactive maps using JavaScript. Leaflet is our choice for this project over other mapping libraries primarily due to its lightweight nature (37KB), ease of use, and flexibility. Our application will use the Leaflet map and layer objects to produce a highly interactive map to visualize the Kaguya observation data as produced by the GeoServer instance.

4.1.2 Map

The Leaflet map object is created via the definition of two parameters: target and options. The target parameter specifies the HTML element that the map object will be rendered within, while the options parameter accepts a dictionary containing a series of optional parameters and their defined values. For this application, the necessary options that must be defined are Coordinate Reference System (CRS), 'center', and 'zoom'. The CRS attribute defines the coordinate system the geospatial data will be mapped to, which in our case is EPSG:4326, or latitude and longitude. The center attribute defines the coordinates that the center of the map will be set to when the application is loaded, and zoom defines the zoom level of the map at load, with a zoom level of 1 being the most global view.

4.1.3 Layer

The layer object defines the visible map data which shall be rendered within the map object. The geospatial data shall be served as map tiles, and so a TileLayer will be used. The TileLayer requires a source attribute which defines the source of the tile service (the GeoServer instance) and a dictionary of optional parameters.

4.2 GeoServer

GeoServer acts the intermediary between the database and the front-end client. GeoServer enables use of the widely adopted OGC Web Mapping Service (WMS) and Web Feature Service (WFS) standards. Because these are widely accepted standards, it will allow us to serve data to the widest possible user base. Our browser based web client will be on top of GeoServer and receive or data using the standard protocols. Below, there is the geospatial database containing the observation data. Utilizing GeoServer Representational State Transfer (REST) API, we can dynamically query for points which in turn are used to generate tiles. Raster tiles and vector tiles are served through WMS and WFS respectively.

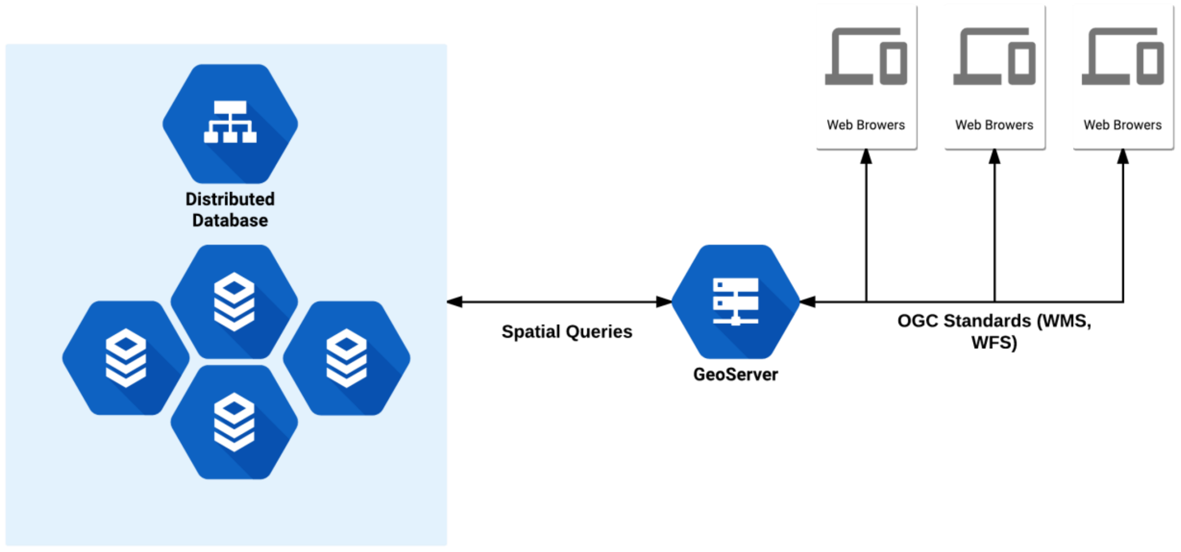


Fig 2 Diagram demonstrating the overall dataflow through GeoServer.

GeoServer will perform queries as the user interacts with the client while dynamically serving tiles using OGC standard protocols. Frequently accessed tiles are to be cached within the server to reduce the need for re-generating tiles. As shown in Fig 2, The GeoServer instance will act as the interface by which clients can access the services provided by the application. It acts as an intermediary, performing queries upon the data set within the distributed database and serving the resulting data to clients as map tiles across the WMS and WFS standards.

4.3 Database Design

4.3.1 Overview

A key component of the work is the development and deployment of a spatially enabled, point database containing the entirety of the available SP data, approximately 46 million individual spot observations. Two broad database solutions exist to efficiently store, query, and serve the spatially enabled data. These are traditional Relational Database Management System (RDBMS) that utilize the Structured Query Language (SQL) and more recently developed NoSQL databases that utilize a document based approach. Typically, a PostGreSQL + PostGIS solution is used for serving map data. Although originally a strong contender, for this project, we were looking for a solution that allows for rapid deployment and easy scalability. There is no expectation of additional observations as the mission has long since concluded, so any additional data is expected to be derived. By extension, the dataset needs to be scaled after initial deployment in order to accommodate the addition of derived data. Therefore, we decided upon the NoSQL DBMS MongoDB. MongoDB offers better horizontal scalability compared to most other DBMS based solutions and free failure protection through built in replica-set support. MongoDB is outperformed by PostgreSQL when it comes to complex join operations. This can be offset by implementing redundancy in the schema, sacrificing drive space for faster query execution time. As the data is already set, and derived data is not expected continuously grow, typically desirable database characteristics such as fast writes, ACID support or atomicity are not primary objectives. The key virtue behind our database design is highly responsive server which supports many concurrent reads.

4.3.2 Schema Design

The original data specification consists of two collections: *Images* and *Spots*. *Images* consists of image meta which encapsulate a ~100 element vector of *Spots* collected in lines. The *Spots* collection contains individual points representing all 46 million spot observations. Among the fields in *Spots* there exists four ~300 element parallel vectors for reflectance (see Fig 3), quality and two reference measures calibrated for dark and light areas on the moon at different wavelengths along with miscellaneous metadata.

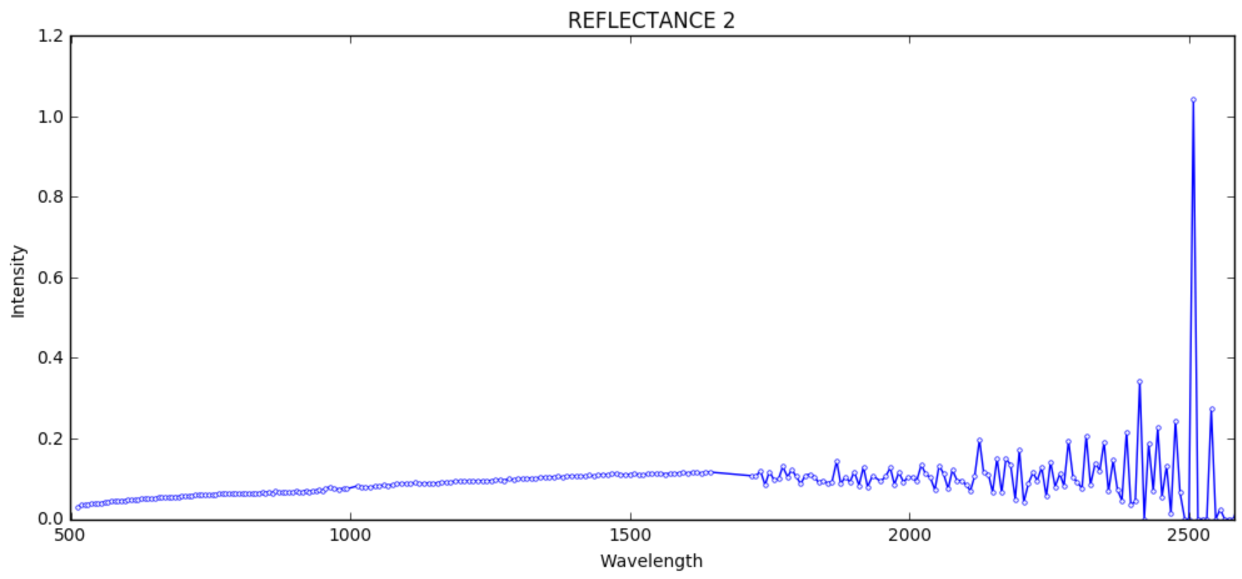


Fig 3 A graph of the ~300 element array for a single point. Wavelength is in nanometers and reflectance is almost always in the range [0, 1.0]. Quality goes down as wavelength goes up. You can notice an outlier near 2500 nanometers.

In order reduce the problem size, we encapsulated the points into Image collections. An image based can collection help aggregate ~100 points into a single document (exact number of points per image vary) which works as an intermediate structure and aids in reducing the problem size by a factor of about 100 (~460,000 documents). A spatially enabled index is still possible by indexing on centroids. Spot observations which make up the image are stored in an array structure. This array of spots contains the reflectance, radiance and quality arrays. As the wavelengths do not differ between points, wavelengths can be factored out and stored in a single document within the database. Two other important metrics are incident and emission angle for particular spots. To minimize query times, there are different representations optimized for different queries.

Spatial queries are on longitude/latitude location and angle range queries are on incidence angles and emission angles. MongoDB does not support multiple spatial indexes, so to optimize query time, these are split into their own collections: image_pos, image_angle that are keyed on (longitude, latitude) position and on (incidence angle, emission angle) respectively. By treating incidence angle and emission angle as 2D coordinates with added redundancy allows us to distribute the data more evenly over a cluster for the different documents while maintaining O(1) time complexity at the cost of using extra drive space.

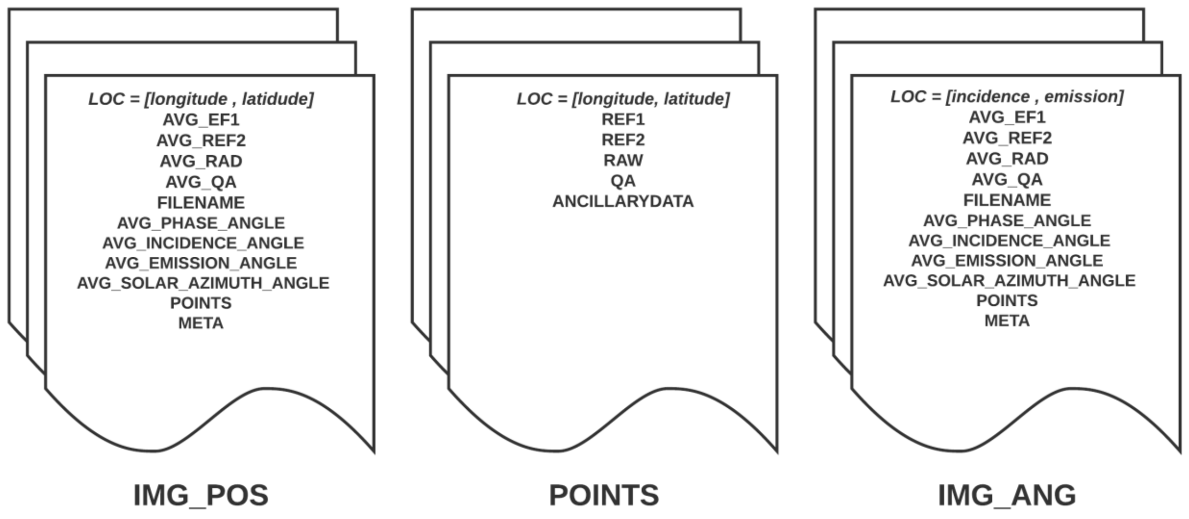


Fig 4 A diagram of the three collections and their fields.

As depicted in Fig 4, longitude and latitude values for images are the centroid of the line. Incidence and emission values are averages derived from the points contained in the image. The averages are used to perform queries on the intermediate image collections. The POINTS document represents data contained among points.

4.3.3 Sharding Strategy

Because of the large size of the dataset, all 46 million spot observations cannot be contained in a single machine but rather must be spread evenly across nodes composing a cluster. MongoDB aids us in this with built-in automatic sharding capabilities which gives us easy horizontal scaling distributing database shards over cluster nodes. The challenge is in choosing the correct shard key to ensure an even distribution of data among shards. A poor sharding strategy leads to uneven distribution of points which causes uneven loads across shards. Another challenge is guaranteeing that each shard contains contiguous data. MongoDB performs best when a query is isolated to a small number of shards and MongoDB can determine a priori which shards to route the query through if a shard has values in a known range. Range based sharding is the simplest form of sharding MongoDB offers. However, MongoDB does not support sharding on the 2D spatial indices. Therefore, a scalar must be derived that partitions the data into equal sized clusters and an insertion method must be used to guarantee contiguous ranged indices within a particular shard.

Our team has decided on using simple K-means clustering technique to divide images into clusters. K-means is one of the simplest algorithms that solve the well-known clustering problem.

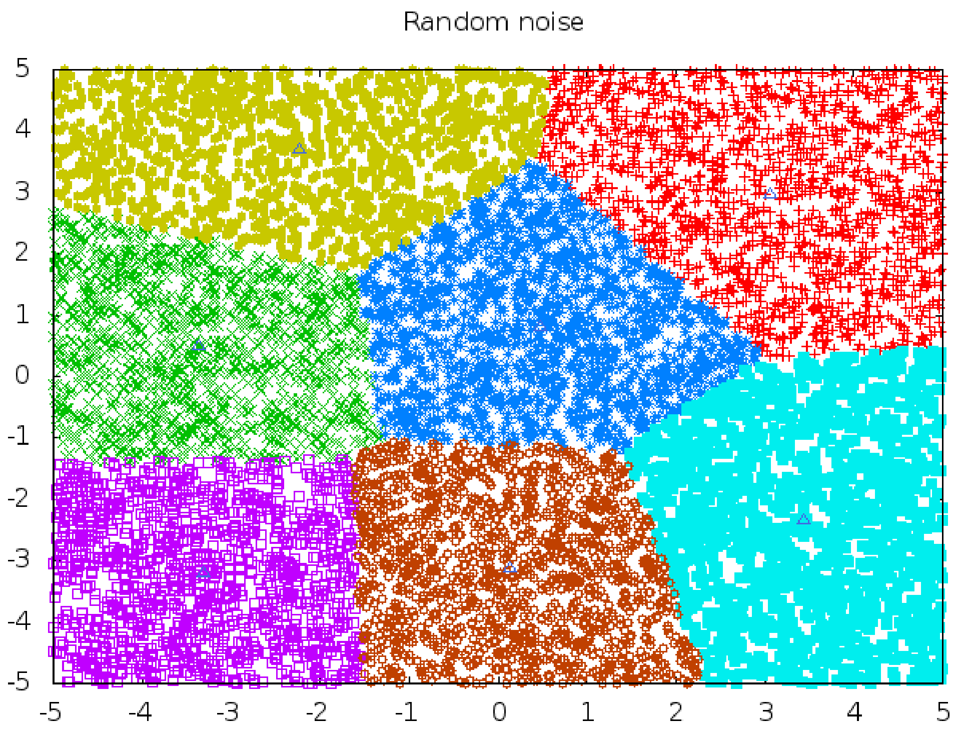


Fig 5 An example of k-means clustering on a randomly generated point distribution. This effectively lumps points into voronoi cells. Images are expected to divided in a similar fashion where each cluster represents a shard.

The objective of the K-means algorithm is to partition observations into k-clusters while minimizing within cluster sum of squares. That is, K-means attempts to find:

$$\underset{\mathbf{S}}{\arg\min} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \left\| \mathbf{x} - \boldsymbol{\mu}_i \right\|^2$$

Where || **x** - $\boldsymbol{\mu}_i$ ||^2 is the Euclidean distance between image centroid **x** and the centroid of all images in cluster S_i, $\boldsymbol{\mu}_i$.

With this method, we can easily classify images using k clusters fixed a priori and assign contiguous shard keys to images which belong to the same cluster. Images contained within one point cluster can be expected to be mapped to one shard on the MongoDB cluster. To maximize effectiveness of this strategy, the number of shards must be known a priori as well. Since the dataset is complete and no new images are expected to inserted after initial sharding, this won't pose a meaningful setback for our use case.

4.3.4 Database Hardware Layout

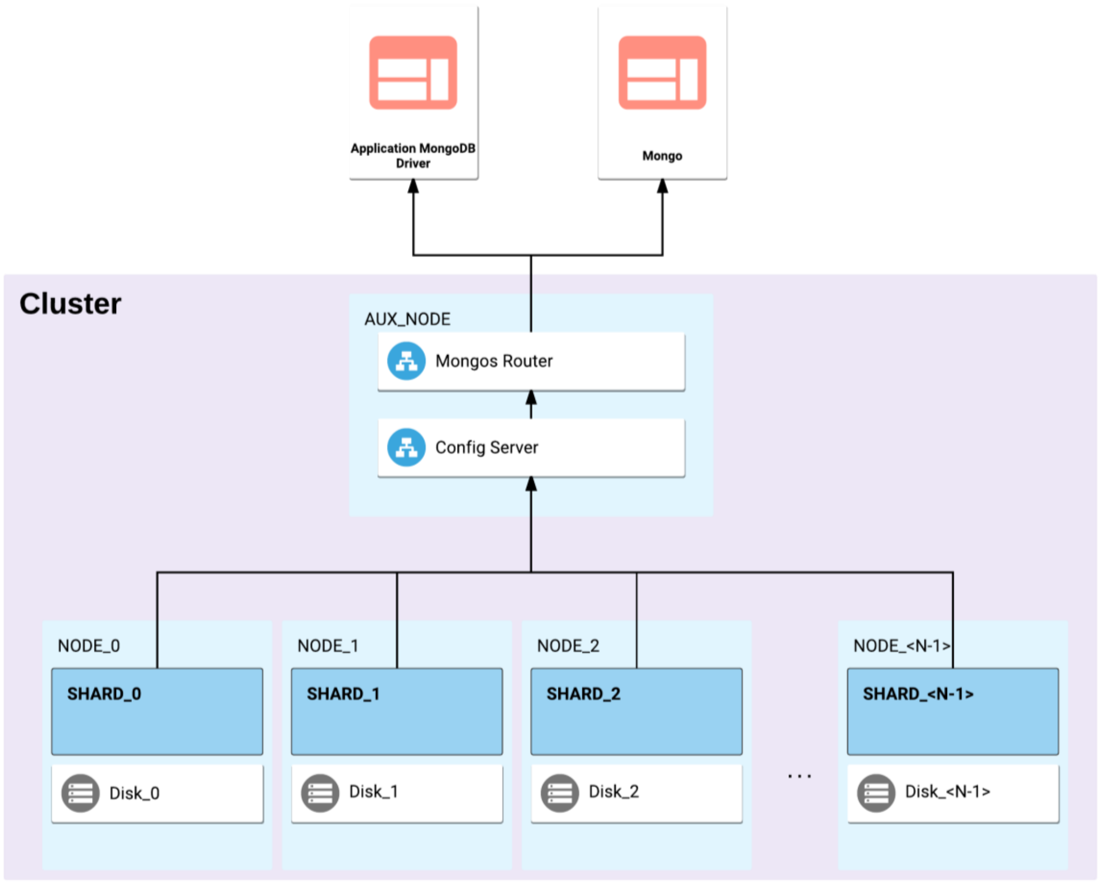


Fig 6 High level overview of cluster setup. Shards are contained in nodes and each have access to their own datastore. Each shard is assumed to be a replica set and each node can potentially contain multiple shards.

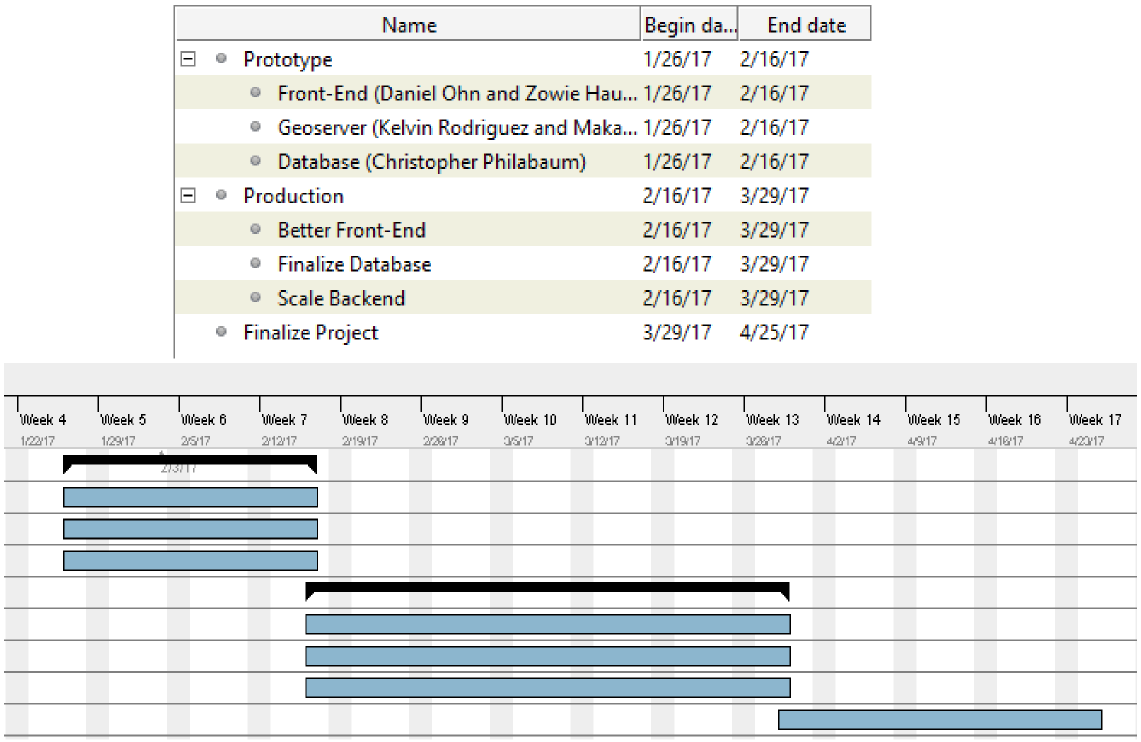
Taking advantage of resources made available to us by the USGS, we plan to set up the database on ASC's AstroVM cluster. AstroVM gives us access to about 2 dozen nodes with a few dozen CPUs, 128 GB of ram and several terabytes of parallel disk access for each node. We will utilize the mini VM service Docker to deploy MongoDB shards. Scheduling is provided via Slurm. Below is a listing of technologies and definitions that are used in this section.

- **Mongod** – the primary MongoDB server process/service that provides the database storage and indexing logic along with simple configuration capabilities for managing a single instance or a cluster. All shards are Mongod instances.
- **Mongos** – the routing and coordination process/service that abstracts individual cluster components from a client application. Mongos makes the distributed components appear as a single system.
- **Mongo** – administration client application for managing a large or small MongoDB deployment.

- **Config server** – Mongod instance which stores metadata for a sharded MongoDB database, used by Mongos for action guidance.
- **Docker** – Mini VM service, simplifies deployment of database services and allows for allocating resources from cluster nodes for each Mongod instance when deploying multiple shards on a single node.

With the database service dockerized, it will make it easy for us to schedule time on the cluster to run the database service. Each shard is dockerized individually and one more container is used to host both Mongos and the Config Server which will all communicate using Docker's DNS service. Mongos, GeoServer and other MongoDB database drivers will connect to this datastore via the Mongos instance which will have a public facing static IP.

5. Implementation Plan



Name	Begin da...	End date
⊟ ● Prototype	1/26/17	2/16/17
● Front-End (Daniel Ohn and Zowie Hau...	1/26/17	2/16/17
● Geoserver (Kelvin Rodriguez and Maka...	1/26/17	2/16/17
● Database (Christopher Philabaum)	1/26/17	2/16/17
⊟ ● Production	2/16/17	3/29/17
● Better Front-End	2/16/17	3/29/17
● Finalize Database	2/16/17	3/29/17
● Scale Backend	2/16/17	3/29/17
● Finalize Project	3/29/17	4/25/17

Fig 7 Gantt Chart of what the team will be working on for the semester.

Fig. 7 above displays how Team Selene will be spending its following semester working on its Capstone project. Currently, the team is working on the first section of Fig. 7 Prototype.

'Prototype' should be a very basic implementation of the project which covers the three major components: Front-End, GeoServer, and MongoDB. Front-end client will be the web design and user interface which will display the map and allow the user to interact with it by panning, zooming, etc. GeoServer will be the tile server that connects the Mongo database to the front-end client. Database will be designed during this section then implemented to store points and metadata, along with serving any points that are requested from the client.

The 'Production' portion of the project will take place immediately after the 'Prototype'. This portion will involve finalizing major decisions with each section of the project, along fixing any major bugs or adding important features to each section. Front-end client will include new features that involve iterating through the data with more attributes to filter. GeoServer will be placed on a cluster making it more scalable and more responsive. The design of the MongoDB will be finalized during this stage.

‘Finalizing Project’ within Fig. 7 will be where Team Selene should be finished with all the major portions of the capstone project and have the functionality working for each section. This portion will be fixing minor bugs within the code and adding any features that could improve the project.

6. Conclusion

Team Selene's design approach on serving such large scale spatial and spectral data places focus on previously established technologies (e.g. GeoServer) along with newly innovative technologies (like Datashader). Architecturally, the solution is based on a server-client pattern, where the server pipelines the database directly into Datashader and GeoServer. The server can then send the client a vector-based data set vs. a raster based on how zoomed in or out the user is. GeoServer can serve the tiles to Leaflet, allowing for an intuitive map interface not unlike Google Maps.

In database design, an aggregating structure is chosen. By collecting every roughly 100 points into an *Image* document, the 46 million observations can be reduced to around 460,000 documents: far more manageable for MongoDB to handle. Even then, a sharding strategy must be used to allow for better horizontal scalability. By using a K-means clustering algorithm, the shards are ensured a more reasonably even distribution amongst clusters of the 46 million data points.

By intelligently using individual points only when a user can effectively select it allows scalability to solve USGS's problem. Overall, not only will it grant the Astrogeology department to properly and reasonably serve the Kaguya's hyperspectral data of the Moon to likewise scientists, but other projects and departments may be able to use a similar design approach for other future large scale problems.