Proximity vs. Diversity in Heterogeneous Datasets

DESIGN DOCUMENT

Group sdmay21-31
Client Advisor: Dr. Goce Trajcevski

https://sdmay21-31.sd.ece.iastate.edu/
sdmay21-31@iastate.edu

Revised April 25, 2021

Thomas Beckler
Bradley Gannon
Benjamin Huinker
Gabriel Huinker
Koushhik Kumar
Cristina Marquez
Jacob Spooner

Executive Summary

In this document, we will show our design and implementation for group sdmay21-31 and the process we took to create a web application to help astrophysicists view cluster patterns in heterogeneous datasets.

Development Standards and Practices Used

Engineering Standards:

- IEEE 802.11-2020
- IEEE 802.3-2018
  - Web application uses wireless ethernet network
- P2989
  - Login authentication required to access database
- IEEE 23026-2015
  - Maintenance required for the website

Team Agile Processes:

- Agile software development practices
- Two week sprints
- Sprint planning meetings
- Daily standups
- Sprint review meetings
- Estimation of the difficulty of tasks as a team
- Allocation of responsibilities among team members

Version Control Standards:

- Group review and approval
- Store code in repository
- Git practices

Coding and Testing Standards:

- Linters
- UI development standards

Summary of Requirements

- Intuitive and responsive website to interact with a heterogeneous dataset
- Secure data storage
- Efficient and accurate data lookup
- Users can upload new datasets
- Users can easily input attributes/parameters and select proximity/diversity
- Interface is easy to control and read

Applicable Courses from Iowa State University Curriculum

- Com S 309: Agile, Git techniques
- Com S 311: Analyzing and designing efficient algorithms
- Com S 319: UML Diagrams
- Com S 329: Risk management
- Com S 339: Architecture design introduction
- Com S 363: Database Systems
- Cpr E 414: Big data analysis
- Cpr E 419: Batch & Real-time large scale data analytics
- DS 301: Machine learning
- Stat 330: Data algorithm design

New Skills/Knowledge acquired that was not taught in courses

- Web Development
- K-means, DBSCAN clustering
- Asynchronous server processes
- Remote development and team work
- Dynamic database implementations
- Different methods of time interpolation and time series clustering

Contents

List of Figures

List of Tables

1 Introduction

1.1 Acknowledgement

We are grateful for Dr. Goce Trajcevski, who meets with us every week, provides us with fundamental astrophysics knowledge, and walks us through the complicated topics our project addresses. We would also like to thank our class professor and guest speakers for taking time out of their day to teach us new skills.

1.2 Problem and Project Statement

Researchers have difficulty clustering and visualizing large and multivariate (heterogeneous) datasets whilst combining weighted impacts on both proximity and diversity. The problem is made more difficult by the amount of clustering algorithms and parameters that could be used for application.

The solution is to create a web application that allows customizable algorithms to relativise the data. Similarly, the application will allow users to upload various datasets and analyze the datasets with given proximity and diversity parameters.

1.3 Operational Environment

For this project, we are not concerned with the operational environment since the application will be hosted on university servers.

1.4 Requirements

This table contains a list of requirements with their descriptions.

ID	Requirement	Description
R1	A website to interact with stored data	The website must have tools to input constraints, which will be sent to the server to search for binary stars.
R2	Secure data storage	Only privileged users must be able to edit data.
R3	Displays relevant data in a timely manner	Users must be able to view the data that they want clustered shortly after they make a call to the database.
R4	Users determine which attributes require proximity/diversity	Users must be able to assign weights to certain data points to simulate proximity and diversity within the database.
R5	Allow users to query heterogeneous datasets	Backend must extract rows matching certain user-inputted measures and analyze the data.
R6	Data entry interface is intuitive to all users	Users must be able to understand how to input data without any additional instructions.
R7	Privileged dataset upload	Privileged users must be able to upload datasets with a composite primary key and attributes.
R8	Privileged attribute selection	Privileged users must be able to select a subset of attributes to be used from new and existing datasets.
R9	Privileged dataset deletion	Users that upload datasets must be able to delete these datasets.

Table 1: Requirements

Functional Requirements

- A website to interact with stored data.
- Secure data storage.
- Displays relevant data in a timely manner.
- Users determine which attributes require proximity/diversity.
- Users can change the weight of attributes.
- Allow users to query heterogeneous datasets.
- Allow users to upload and manage datasets.

Non-Functional Requirements

- Public access
- Easy to navigate
- Simple and intuitive design

UI Requirements

- The UI works for all general client devices
- Data entry interface is intuitive to the average user

1.5 Intended Users and Uses

The intended end users are astrophysicists who will use the web application to view clustering patterns in data. Although this application will be made specifically for researchers in this field, it will be accessible to the general public.

1.6 Assumptions and Limitations

Assumptions

- Dataset will be over 5 GB
- Primary users will be knowledgeable about the data they are viewing
- Data will be imported through CSV (Comma Separated Value) files, or will be changed in an admin interface
- An Ubuntu single-core server will be provided

Limitations

- Budget: We are not given money for resources.
- Time: We have two semesters to implement a web application.
- COVID-19: Physical interactions between group members are discouraged during these semesters.
- Computing Environment: The service must run on a university-provided virtual machine.
- Developer Experience: Field experience of developers can limit feature implementations.
- Intuitiveness: Data entry interface must be intuitive to the average user.

1.7 Expected End Product and Deliverables

- Web application fulfilling all requirements set up on an Iowa State Server
  - The web application shall be set up and ready to be used by the clients. It must have the given Binary Stellar Trajectories dataset ready to be processed.
- A repository for the code of the application
  - Access or the transfer of the GitHub repository to the clients.
- Operation Manual
  - The manual must guide inexperienced users through the process of using the web application. This includes the uploading, editing, and deleting of datasets, and the selection of attributes to process.

2 Project Plan

In this section we will lay out our schedule, split up tasks, delegate responsibilities, and plan our milestones for the upcoming semester.

2.1 Task Decomposition

The following table shows all the tasks that will be required to complete this project:

ID	Title	Description
T0	Design Infrastructure	Have frameworks and application structure formalized.
T1	Create local environment	A local environment implementing the structure from T0 is created and shared among developers.
T2	Initialize database	An SQL database is initialized with given data.
T3	Design UI	Design a web frontend that allows user selection for proximity and diversity at different weights for data attributes.
T4	Implement UI Design	Implement and develop the UI as designed in T3.
T5	Create queries	Have all the necessary queries for each function finalized.
T6	Implement backend business logic	All algorithms for parsing and representing data from T5 are implemented in the backend.
T7	Add UI Functionality	The user is able to functionally request and view data.
T8	Create production environment	Have a publicly accessible and usable environment created, and code deployed.
T9	Data visualization	Add data visualization to the frontend UI
T10	Help Documentation	Finalize help documentation for how to use the web application
T11	Presentation	Finish presentation and document report for project.

Table 2: Task decomposition

2.2 Risks And Risk Management/Mitigation

The following table assigns an estimated risk to each of the tasks described in section 2.1. If the estimated risk is 0.5 or greater, a possible mitigation strategy is described. The described mitigation strategy aims to provide solutions to possible issues that may arise during the product's development.

ID	Title	Risk	Reason	Mitigation Strategy
T0	Design Infrastructure	0.1	Planning Stage	None
T1	Create local environment	0.5	Security Risk	To ensure a secure production environment, we will use environment variables laid out by the Django 12 Factor application pattern.
T2	Initialize database	0.4	Initialization Stage	None
T3	Design UI	0.1	Planning Stage	None
T4	Implement UI Design	0.1	Initialization Stage	None
T5	Create queries	0.7	Performance and Security risk	Make sure that queries meet performance levels. Given the large amount of data, timely queries are very important. Use Django QuerySets to retrieve data from the database.
T6	Implement backend business logic	0.9	Performance and Security Risk	Algorithms need to be thoroughly tested to make sure they meet performance criteria. We will use Django forms for input sanitization.
T7	Add UI Functionality	0.4	Performance and Security Risk	None
T8	Create production environment	0.9	Security Risk	The production environment must ensure no breaches or unauthorized access through any form of protocol and attack. We will go through server hardening protocols to mitigate the risk.
T9	Data visualization	0.5	Performance Risk	Possible issues with performance. Mitigation may require different tools, increased testing, and possibly no implementation.
T10	Help Documentation	0.1	Documentation	None
T11	Presentation	0.1	Documentation	None

Table 3: Risks and risk management/mitigation

2.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

The following milestones are the major goals and dates for our project.

Milestone 1: Architecture Design (Oct. 25)

We will have a completed diagram that shows how frameworks for frontend and backend work together to create a web application. This diagram should be flexible and maintainable to allow future changes to be adopted to the infrastructure without serious change or overhaul.

Milestone 2: Finalize Design Document V1 (Nov. 15)

The design document was comprehensive and contained all information necessary for the development of the software. The document contained all design plans for the software, how the team was planning to work together to develop the software, and testing plans to maintain software quality.

Milestone 3: Implement UI (Nov. 23)

The UI was an implementation of an intuitive and pleasant interface design. It allowed them to select attributes for proximity and diversity and input a specific time instance and number of clusters.

Milestone 4: Finalize Database Design (Mar. 25)

With the added requirement of the uploading new datasets to the project, the database schema had to be rewritten. This new database design allows for new and generic datasets to be uploaded by users.

Milestone 5: Finalize Algorithm Solution (Apr. 4)

After many months the algorithm solution was finalized with both k-means clustering, weighted euclidean distance, and diversity threshold selection.

Milestone 6: Complete Testing (Apr. 20)

The final set of tests for our project consisted of both unit and integrated tests. Unit tests covered algorithms and their respective needs such as relativised node attributes, business logic ensuring all nine endpoints return proper results, and database queries function as planned. Integration and acceptance testing mainly consisted of validating endpoints and their wanted results, and making sure the endpoints met the clients needs and expectations.

Milestone 7: Finalize Software Version (Apr. 24)

The final version of our application will be on our production server. The software will meet all of the requirements specified in the design document, and the software will also be tested thoroughly to ensure that it contains no major bugs. The help documentation will be created for the final version of the software, and the intended users will be able to use the software for performing work. The software will be in its finished state, and no further changes will be required.

Milestone 8: Finalize Design Document (Apr. 25)

The final version of our design document, which contains the details of how the project was implemented in the end. The document contains details about the new requirements of the project, the technologies that were actually used in the final project, and other content that changed during the development of the project.

2.4 Project Timeline/Schedule

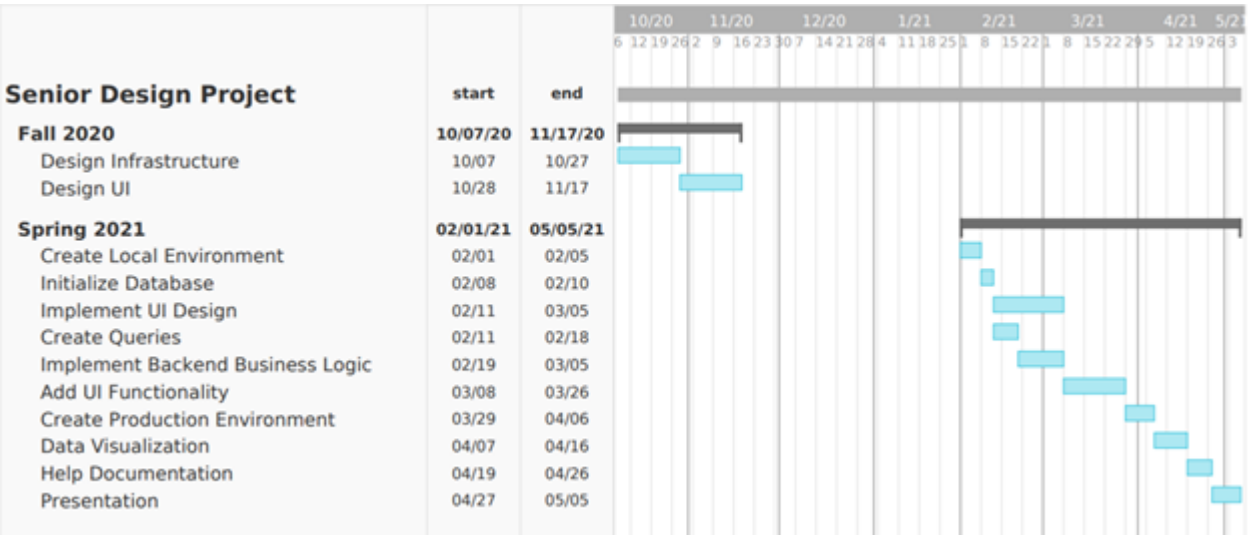

Figure 1: Preliminary Gantt chart

Above is our project's preliminary Gantt chart. The following table will elaborate on the tasks in the Gantt chart, and the dates in which these tasks are to be completed.

ID	Task	Start Date	End Date
T0	Design Infrastructure	10/7/20	10/27/20
T3	Design UI	10/28/20	11/17/20
T1	Create Local Environment	2/1/21	2/5/21
T2	Initialize Database	2/8/21	2/10/21
T4	Implement UI Design	2/11/21	3/5/21
T5	Create Queries	2/11/21	2/18/21
T6	Implement Backend Logic	2/19/21	3/5/21
T7	Add UI Functionality	3/15/21	4/2/21
T8	Create Production Environment	4/5/21	4/13/21
T9	Data Visualization	4/7/21	4/16/21
T10	Help Documentation	4/19/21	4/26/21
T11	Presentation	4/27/21	5/5/21

Table 4: Project schedule

2.5 Project Tracking Procedures

Our group will track progress through the use of GitHub, Slack, and Google Drive. We will use GitHub to store the codebase. We will also use a GitHub project board as a Kanban board for members to create, assign, review, and complete tasks. The GitHub project board will be our primary way to track progress throughout the development of our project. Slack will be used as a message board for group members to ask and answer questions. Finally, all of our documentation will be located in Google Drive. This will grant all group members access to the documentation and allow all members to work on the documentation synchronously. Together, these tools will assist our team in the completion of the project by May 2021. To ensure progress is made each week, we will hold a meeting once a week to discuss the progress we have made and the current tasks we are working on.

2.6 Personnel Effort Requirements

The following table outlines the estimated hours required of each task in the project. Additionally, a short explanation of each task is provided in the rightmost column. In total, this project will require approximately 160 hours to complete.

ID	Task	Hours	Explanation
T0	Design Infrastructure	10	The frameworks and application structure form the foundation on which application is built.
T1	Create local environment	5	A local environment is necessary to develop in and verify proper functionality of application before deployment.
T2	Initialize database	15	A SQL database with the given data will be utilized for proximity and diversity algorithms in the backend.
T3	Design UI	1	Creating a visual design guides development of the UI.
T4	Implement UI Design	30	The UI forms a large portion of this project. This is the medium in which the users of the app will interact with the data.
T5	Create queries	10	The queries used with the SQL database will be somewhat complex to account for proximity and diversity weighting.
T6	Implement backend business logic	25	Implementation of the SQL queries into the backend is necessary before interaction with the frontend.
T7	Add UI functionality	20	Fully functional ability of frontend to communicate with backend and data requests.
T8	Create production environment	20	The app is now functional and can be deployed to the public user.
T9	Data Visualization	4	Utilization of data visualization techniques in the frontend would add a nice feature to the app.
T10	Help Documentation	10	A guide on how to use the application will be helpful for the user.
T11	Presentation	10	An overview of the project in written and oral form.

Table 5: Personnel effort

2.7 Other Resource Requirements

This project necessitates the use of computing and storage resources, which the university is providing.

During the development and testing stages, our team will generally require one server and a database instance. If the purpose of the tests is to test something like basic functionality, a local server and database can be used. In the case of testing a high number of users or large amounts of data being continuously sent, we may need more or higher powered servers. To account for multiple test cases, the server should have at least 1 vCPU, 4 GiB of memory, and 50 GiB SSD.

An ideal production architecture will include several servers and a load balancer and queue for handling multiple concurrent requests. However, for the purposes of this project and developing what resembles a

proof-of-concept, that level of complexity may not be necessary. Since we are using ISU servers, we may have to develop these tools ourselves.

A more applicable production architecture will include between one and three servers and a database with multiple read replicas. Since the application will not be creating, updating, or deleting data, our focus will not have to include dealing with concurrent writes to the database.

2.8 Financial Requirements

Our project will be hosted on university machines; therefore, there will be no cost associated with this project.

3 Design

In this section we will discuss the design of our project and how the individual components of our design will interact together.

3.1 Previous Work And Literature

In order to accurately design and implement the clients vision we had to research existing implementations of clustering and algorithms that manage similar data.

3.1.1 Relevant Background/Literature

Our two main clustering algorithms we found were K-means and DBSCAN. These two algorithms will provide a crucial foundation to the algorithms for our data.

K-means clustering is a clustering algorithm that aims to partition data into k clusters in a way that data points in the same cluster are similar and data points in different clusters are farther apart. This will allow us to group data together based on a defined distance function(Euclidean distance) [1].

DBSCAN Clustering stands for Density-Based Spatial Clustering of Applications with Noise. This type of clustering is able to find arbitrary shaped clusters and clusters with noise. The main limiter of DBSCAN is that a point belongs to a cluster if it is close to many points from that cluster. There are 2 key parameters of DBSCAN: eps and mindPts. Eps is defined as the distance that specifies neighbors. If the distance between 2 points is less than or equal to eps then they are classified as neighbors. MinPts is the minimum number of data points that define a cluster [2,3].

3.1.2 Describe what other products exist in the market

The other products that exist in the market are similar databases with different types of visualizations. Since our goal isn't visualization the existing products are useful for getting a better grasp of the database queries we will use. In these tools all of them have some type of "query database" section that shows the database tables based on some user query. They also all have some type of visualization that may help the user better understand the information in the tables [4,5,6].

3.1.3 Differentiate your project from what is available

Our project is taking our database and performing certain clustering algorithms based on user input and returning a graphical representation of the determined user inputs. This is quite different from the links provided above since they chose to visualize their data within the scope of the universe. We will visualize our data using specific clustering algorithms in a graphical representation.

3.2 Design Thinking

In designing our project, we have identified and defined needs which will be addressed in our final product. Our users need a way to query and visualize data regarding binary stellar trajectories, in order to facilitate their research. The astrophysicists for which this product is intended need to identify patterns in the given heterogeneous data sets. These needs drive our design, and inform our choices. We know that our solution will involve a web application, which further limits our possible solutions. In the following figure we illustrate our high level architecture, based on our ideation of our product.

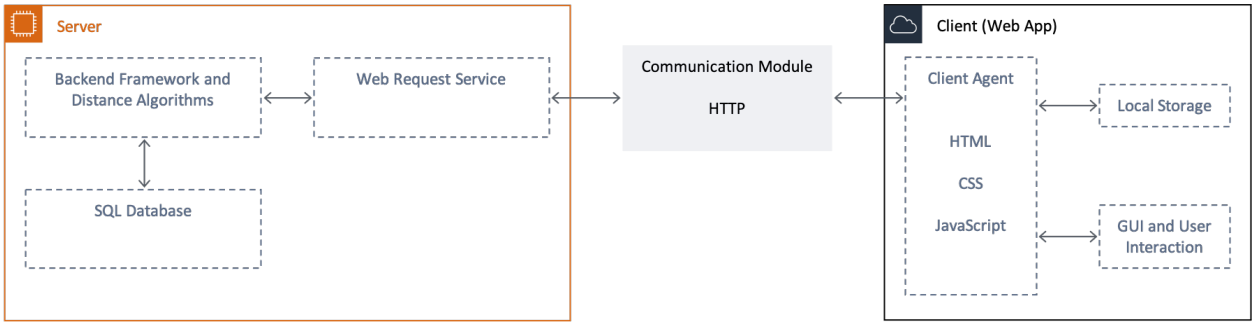

Figure 2: Broad overview of architecture

Web Server and Algorithms (Backend framework):
In order to properly and efficiently parse and format data we will need a server to run algorithms on large bits of data. For this we will need a backend framework and subsequent language specific packages to help create the algorithms.

Store Data (SQL Database):
Our data sets are very large, and in order to efficiently query and run our algorithms, we need a SQL database or distributed database.

User Interface (Client web browser):
We need to implement a simple, intuitive user interface that facilitates use of our web application. To facilitate this implementation, we intend to use a client-side framework.

3.3 Technology Considerations

In order to complete all the components mentioned in the previous section, we had to consider 4 main areas of technologies to fulfill our design: Backend, Algorithms, Database, and User Interface. Our considered frameworks are shown below in the following table, with an explanation in each relevant subsection.

Backend	Packages/Languages	Database	User Interface
Django	Python	MySQL	HTML/CSS/JS
Flask	PHP	PostgreSQL	Tailwind CSS
Node.js	Java	HDFS	Bootstrap
Laravel	JavaScript		SCSS
Springboot	scikit-learn		ReactJS
	NumPy		Vue.js
	pandas		AngularJS
	Django REST		D3.js
	Express.js		Chart.js
			Vega
			three.js
			P5.js

Table 6: All considered technologies

3.3.1 Backend

Our first, and most important, module we had to consider was our Backend framework and language. Because 6 out of 7 team members had developed in Python, we narrowed the selection down to Python, and Django or Flask. From these two frameworks, we decided on Django over Flask, because Django has a larger community, more built-in tools, and a gradual learning curve.

3.3.2 Algorithms

To analyze the data extracted from the database, we need to perform some data analytics prior to the server sending a response to the client. We plan to use Python and some of its libraries because it is the most widely-used language most used for data analytics. In addition, writing PySpark code in future iterations for distributed data analytics will interface well with the Django framework we plan to use for a server.

As described in Section 2.7, we plan to use PySpark in future iterations to speed up transaction processing.

3.3.3 Database

Because our data was given in SQL, the experience of our team, and because Django supports these frameworks out of the box we had to decide between MySQL and PostgreSQL. From the advice of our advisor Goce, and because one of our datasets was given as a PostgreSQL database, we chose PostgreSQL.

With the anticipation of processing large amounts of data, in future iterations, if we use Apache Spark, we also plan to use the Hadoop Distributed File System (HDFS) to store the data in a distributed file system.

3.3.4 User Interface

Since our goal is to create a website to act as an interface to the backend, we will use the fundamental HTML/CSS/JS stack along with some frameworks. After designing an initial UI, shown in the following section, we decided this project did not need a frontend Javascript framework like ReactJS, Vue.js, or AngularJS. For quicker development and more appealing aesthetics we had to choose between the CSS frameworks Bootstrap and Tailwind CSS. Bootstrap is a commonly used CSS framework that comes with a plethora of UI components and utility classes. On the other hand Tailwind CSS is a newer, lighter framework focused solely on its versatile and easy-to-use classes based on its utility first principles. Since our UI is designed with a focus on simplicity, Tailwind CSS's lighter size will make our project easier to develop and faster to render.

3.4 Proposed Design

From our chosen frameworks in the previous section and the higher level design in section 3.2, the following lays out a detailed map of our proposed design. First we show how each of our frameworks interact with each other, and then our initial User Interface design.

3.4.1 Web Application Architecture

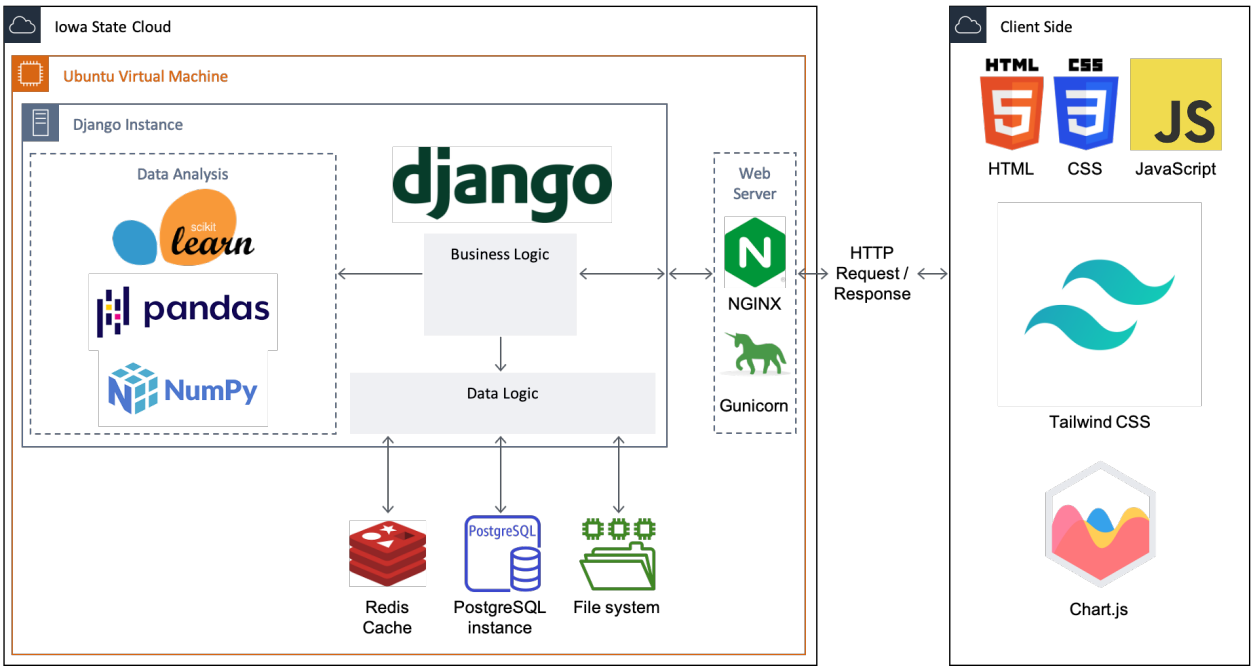

Figure 3: Preliminary detailed architecture design

The following sections will show how these components interact with each other to create a cohesive, scalable, and maintainable web application.

Client Side:
Our client side consists of a website that allows users to set proximity attribute weights and diversity attribute thresholds. A chart is rendered that allows users to easily visualize the data and clusters.

Virtual Machine:
On our virtual machine our primary systems are nginx, Gunicorn, PostgreSQL, and Memcached. nginx is our primary web server, and is our initial interaction to incoming requests on port 80. nginx spawns new processes that tell Gunicorn to run a WSGI server for the Django instance. The Django instance also interacts with a PostgreSQL database for data storage, as well as a Memcached cache for caching of data to speed up response times.

Django Instance:
The Django instance houses the logic and functionality of our project, as well as the algorithms that interpret the data.

3.4.2 User Interface

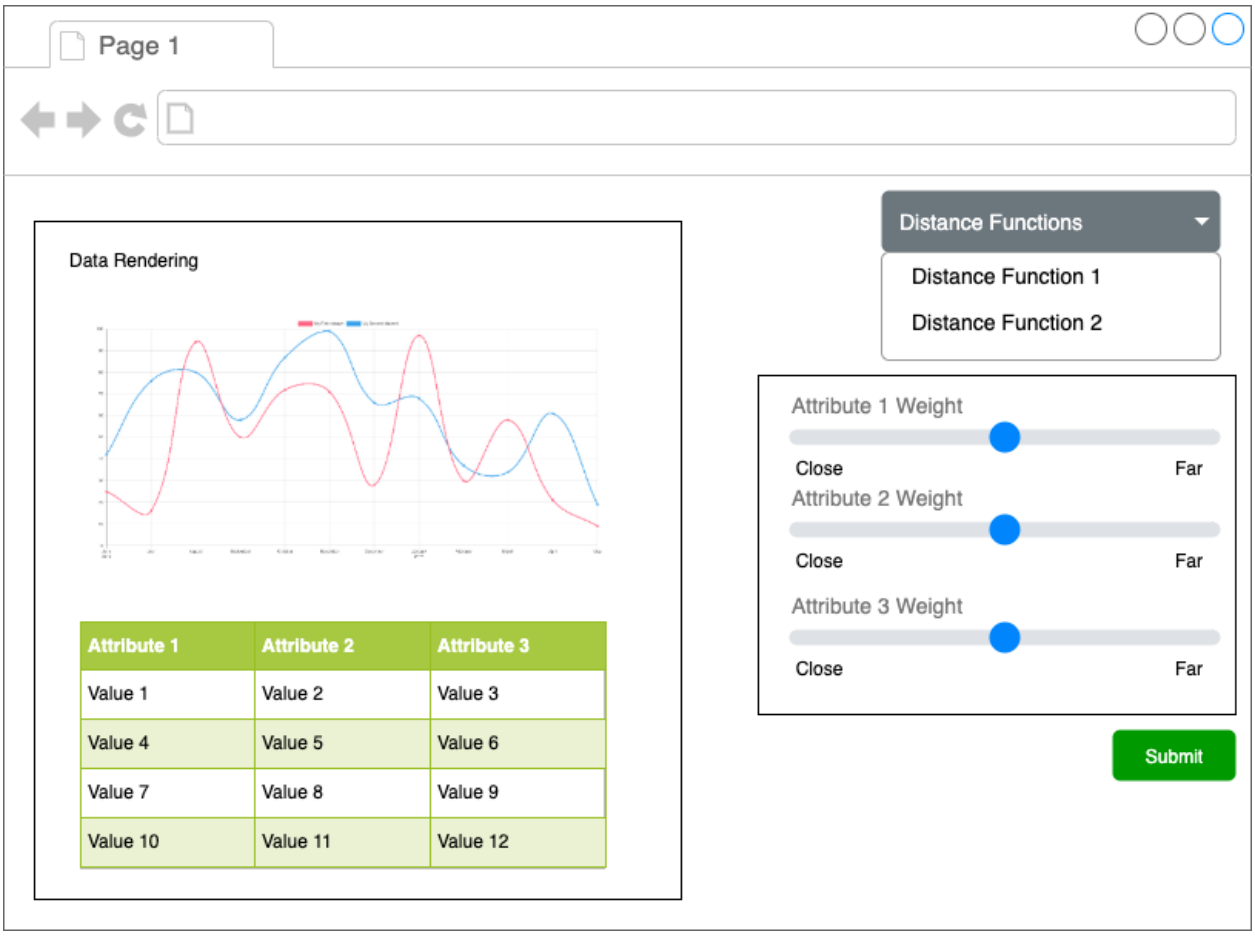

Figure 4: User interface wireframe

Using the frontend tools we chose in the previous section, our UI will be built with three main parts: a data rendering section, a distance function selection, and an attributes weight section. The data rendering will contain a graph and data table showing users their results. From the distance drop down a user can change their preferred distance function. Finally, users can adjust the weights of attributes in the final section. When a user would like to see their resulting data they can click the submit button to render the new data.

3.5 Design Analysis

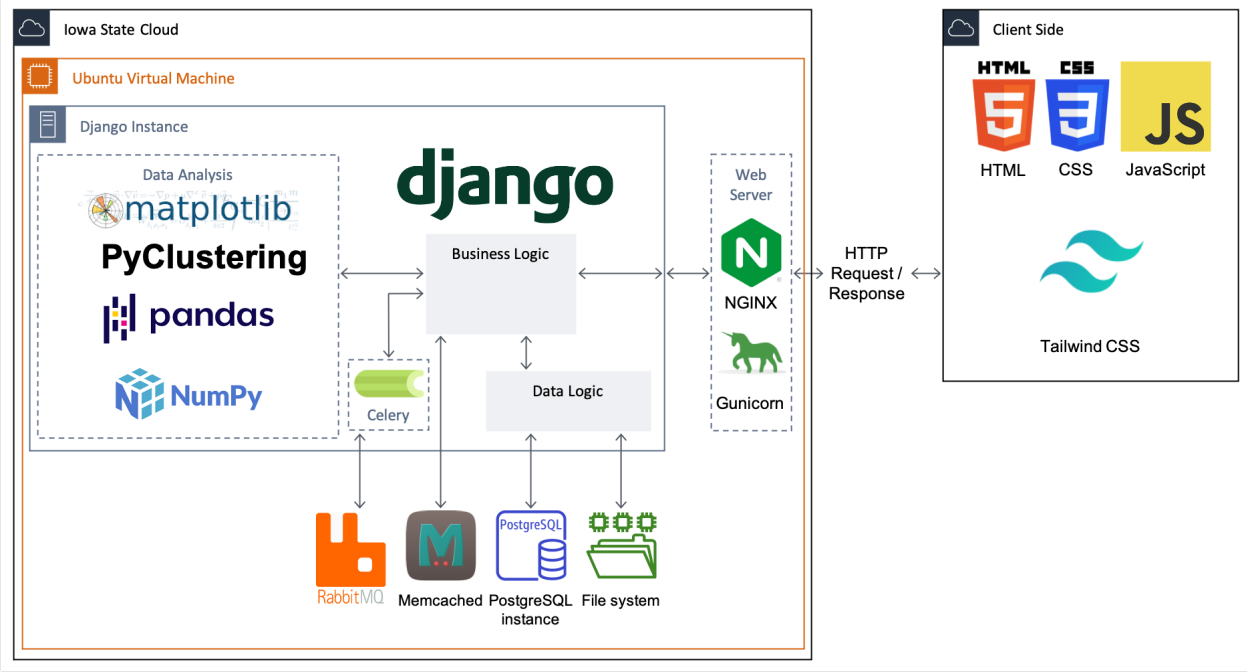

Figure 5: Final Architecture Design

For the most part, the proposed design was implemented, but a few key features had to be added to accommodate the large data size and to finish the algorithm. The noticeable changes are:

- Pyclustering Replaced scikit-learn
  - While implementing the algorithm the clustering algorithm required an adjustable metric for the distance function. scikit-learn did not have this ability, but the pyclustering package did, along with the implementation of other clustering algorithms.
- Django Celery and RabbitMQ
  - In order to provide a better user experience with the large files for datasets, the asynchronous package Celery was used, along with it's Asynchronous service provider RabbitMQ.
- Memcached replaced Redis
  - Memcached is the recommended cache provider for Django, is easy to implement, and provides more efficient data storage.
- Matplotlib replaced chart.js
  - The job of rendering the chart shifted from the client to the backend to reduce data transmission.

This current design should be maintainable and stable for a high number of concurrent users. Possible areas of growth to this design could be:

- Multiple servers
- Separate database and cache servers
- Load balancer

3.6 Development Process

For the development process, we will be following an Agile methodology with a close implementation of Scrum. Given that the project size is relatively small, the waterfall model could have been considered. However, this methodology would quickly fail due to constant updates of the project and the reception of new information that changes aspects of the design. Agile and Scrum allow for quick adaptation to these changes in a way that will allow for success.

3.7 Design Plan

Our design plan is not a perfect reflection of the structure of our architecture. Rather, it is an abstraction of how we are breaking down our project into manageable components/teams.

The next part shows the finished work and the plan relative to the use cases and requirements for which each component is responsible.

Algorithms

What we have done:

- Research on how to find proximity and diversity of data
- Research on possible algorithms to find proximity and diversity
- Decided on which existing algorithms to use in our algorithm
- Decided how to implement the overall algorithm
- Implemented the algorithm in program code

Use-case / Requirement fulfillment:

- R3: The algorithm displays results in a timely manner
- R4: Users determine which attributes require proximity/diversity
  - Users can select different weights for attributes

Backend

What we have done:

- Set up server
- Set up initial infrastructure and database
- Created the viewpoints
- Created the interface between algorithm and frontend
- Created the Matplotlib graph

Use-case / Requirement fulfillment:

- Web application
- Publicly available
- Users need to visualize data
- R2: Secure data storage
- R6: Data entry

Frontend

What we have done:

- Created the design mockup
- Set up and styled the wireframe to make the user interface visually appealing
- Set up Django forms and JS-controlled forms

Use-case / Requirement fulfillment:

- Users need to input desired query parameters

Our Design Plan is shown in Figure 5 with the 5 main steps: Empathize, Define, Ideate, Prototype, and Test. After we talked with our clients we gathered information on the project and defined our requirements as mentioned in Section 1.4. With these requirements we created our solution to the infrastructure and design as shown in Section 3.4. In our second semester we will use an Agile process to Prototype and Test our application.

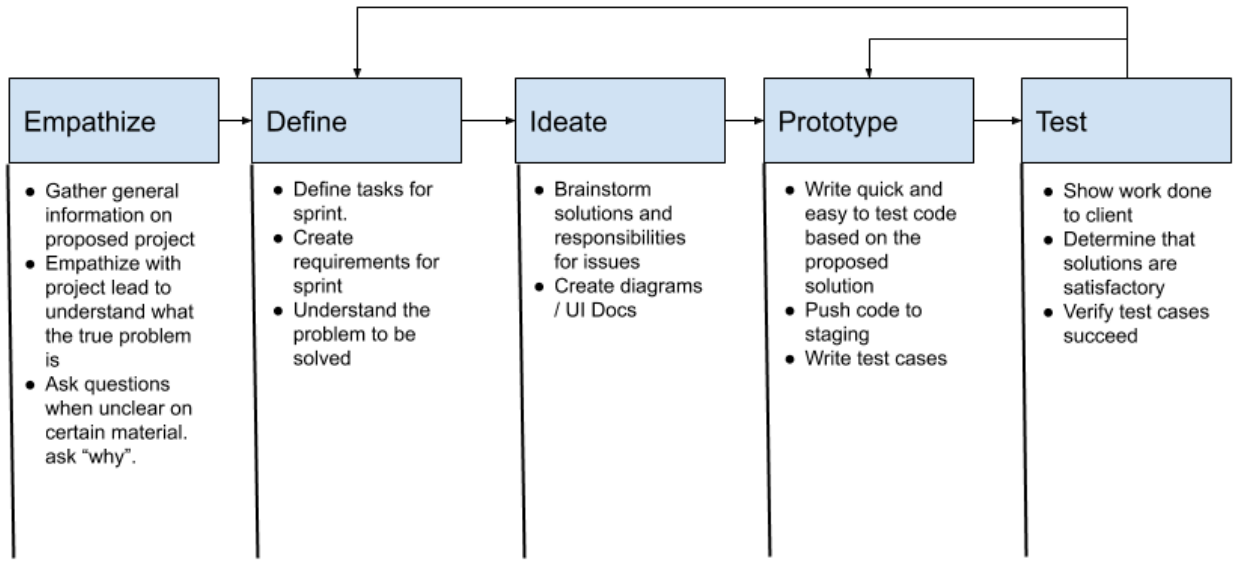

Figure 6: Design plan

4 Testing and Implementation

To make sure the project fulfills all requirements, and allows for better maintainability, testing will be an important part of development.

4.1 Unit Testing

Unit testing is the smallest form of testing that will make sure individual components of our project run correctly.

Algorithms
For the k-means algorithm, we had to ensure:

- Euclidean algorithm returns correct results
- Weighted Euclidean algorithm returns correct results
- Node attributes are properly relativised

Business Logic
The majority of the project utilizes Django's built-in functionality, so testing for business logic encompassed ensuring all 9 endpoints returned proper results. **Database queries**
The majority of our queries uses Django built-in QuerySet builder, but our primary query did involve a few unit tests:

- TimeFrame Query:
  - Return only one node per simulation
  - Return the correct index based on the time_percentage given

4.2 Interface Testing

Interface testing involved manually checking each page for correctness and functionality. The functionality would make sure that each page returns the complete workflow from ui to backend to algorithm and data.

4.3 Acceptance Testing

Acceptance testing is an important part of development, for both Agile and for making sure we are developing the product the client wants. Before interacting with the clients we will make sure that all requirements listed in Section 1.4 are fulfilled.

While asking for feedback from the client, we will have them use the web application to evaluate the following areas for correctness and intuitiveness:

- Distance functions
- User Interface
- Visualizations
- Assigning weights of attributes
- Results returned in timely manner

4.4 Results

We have explored our designs with our client to verify the intuitive approach we will use for our user interface. From these results we added in data visualization in the form of a Matplotlib graph that will allow users to more easily perceive and understand the data.

5 Implementation:

The majority of the implementation for our project followed our original plan. We had to make some changes to accommodate for the size of the datasets we were working with on the server and the complexity and intricacies of the algorithm.

Our initial assumption on 5 core tasks core tasks was simplified to 3: algorithms, frontend, and backend. This allowed us to have our teams work on more succinct missions. The implementation of frontend and backend went smoothly with weekly progression and consistent work. Algorithms had difficulty making any progress as requirements, implementations, and technology stacks would change on a bi-weekly basis. These changes happened because of misunderstanding, and lack of knowledge and pre-existing packages for this specific task.

Algorithms:
The core of our project is our algorithms, and how we parse our data. Since most of our team was new to the idea of clustering algorithms, this aspect of the project took the majority of the time to understand and implement. We originally believed that we would utilize scikit-learn to make this problem more feasible; however, we switched to using Pyclustering as they allow for customizable distance metrics.

Frontend:
The frontend's primary focus is the style and display of our user interface. We used a combination of Tailwind CSS and Vanilla CSS to quickly style the frontend and create an interface that is both intuitive and pleasant for users and maintainable for developers.

To create user inputs, we used both Django's built in Form class and controlled HTML inputs with JavaScript. Using the Django Form class makes user input simple because it itself handles input verification validation and form building. Using HTML inputs controlled by JavaScript simplifies the custom input creation, validation, and error displaying.

For this project, data visualization took the form of a 2D/3D scatterplot – with cluster centroids – which contained the data corresponding to the user-specified parameters. We used Matplotlib to create the scatterplot, and we exported the scatterplot as an image, which was sent to the client and rendered for the user.

Backend:
The backend was made simple by using Django's core functionality to quickly facilitate the connection between frontend and the algorithms and data.

6 Closing Material

6.1 Conclusion

Our goal was to create an easy way for researchers to visualize large multivariate datasets while being able to cluster the data using weighted proximity and diversity. Our team was able to accomplish this and our functional requirements in time for the final demo. The web application also succeeds in our non-functional requirements by returning processed data in a decent amount of time.

There are some features that we would have liked to add with more time available, things like higher dimensional analysis, multiple distance metric algorithm choices, and options for linear interpolation scaling. These features are not requirements but would add to the functionality of the web application.

Overall, the team was able to follow our original plan with a few hiccups along the way. The algorithm posed far more challenges than we originally thought, but after many months of toiling, refactoring, and researching, the algorithm was implemented and the project was able to be completed on time.

6.2 References

[1] Gavo.mpa-garching.mpg.de. 2020. Millennium Simulations - Databases. [Online]. Available at: http://gavo.mpa-garching.mpg.de/Millennium/, [Accessed: 25 October 2020].

[2] J. Gao, "Clustering Lecture 4: Density-based Methods," University of Buffalo, Buffalo, Colorado, United States [PowerPointslides]. Available: https://cse.buffalo.edu/ jing/cse601/fa12/materials/clustering_density.pdf, [Accessed: Oct 25, 2020].

[3] S. Yıldırım, "K-Means Clustering - Explained," Medium, 03-Mar-2020. [Online]. Available: https://towardsdatascience.com/k-means-clustering-explained-4528df86a120. [Accessed: 25-Oct-2020].

[4] S. Yıldırım, "DBSCAN Clustering - Explained," Medium, 22-Apr-2020. [Online]. Available: https://towardsdatascience.com/dbscan-clustering-explained-97556a2ad556. [Accessed: 25-Oct-2020].

[5] "Sloan Digital Sky Survey," SDSS SkyServer DR12, 1999. [Online]. Available: http://skyserver.sdss.org/dr12/en/tools/search/sql.aspx. [Accessed: 25-Oct-2020].

[6] US Space Agency, "gaia archive," Gaia Archive. [Online]. Available: https://gea.esac.esa.int/archive/. [Accessed: 25-Oct-2020].

6.3 Appendices

6.3.1 Code Repository

Github URL: https://github.com/sdmay21-31/ProximityVsDiversity

6.3.2 Technologies and Frameworks Used

Django - https://www.djangoproject.com/
Python - https://www.python.org/
Tailwind CSS - https://tailwindcss.com/
PostgreSQL - https://www.postgresql.org/
Celery - https://docs.celeryproject.org/en/stable/
Memcached - https://memcached.org/
Gunicorn - https://gunicorn.org/
nginx - https://www.nginx.com/
Matplotlib - https://matplotlib.org/
pandas - https://pandas.pydata.org/
NumPy - https://numpy.org/

7 Appendix I - Operation Manual

7.1 Homepage

The user will be sent to the homepage by default when accessing the website. The page is split into different sections that provide different functionality. The functionality of each section is described in the following sections.

7.2 Processing Data

Upon selecting a dataset, the user is directed to the page below where they can select up to three attributes and define their values, the time instance, and the number of clusters. After selecting data, the user clicks the process button. After a short time, the page is updated with a graph based on the input data.

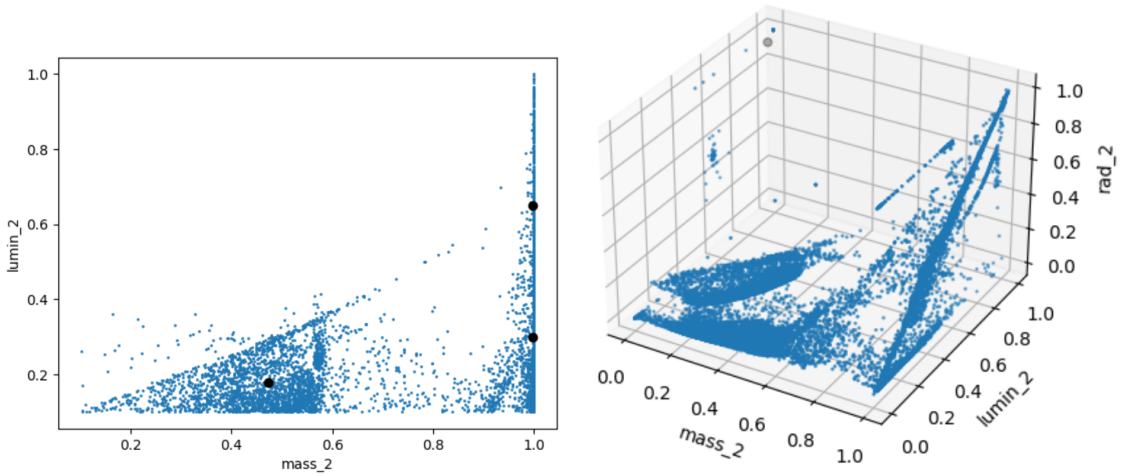

Figure 7: Generated graphs

7.3 Proximity and Diversity

On the data selection screen, there are two tables: Proximity Attributes and Diversity Thresholds. The Proximity Attributes selection allows users to add weighted values to the distance function. Proximity increases the weight of attributes that are more similar, whereas diversity limits the number of nodes that meet the diversity threshold for that attribute. Accordingly, when a user selects a diversity attribute, the user is essentially asking the algorithm to only show nodes that are x% more diverse than all other selected attributes, where x is the input value.

7.4 Adding a User

After logging in as admin, additional users can be added from the admin page by clicking on the “+ Add” button under authentication and authorization. Here, the admin types the name and password of the user to be added. After doing so, the new user can be saved and added.

7.5 Uploading a File

The user can process any dataset with the websites’ algorithms after uploading a file.

Figure 8: Add a dataset

The user selects a file with the “Choose File” button, and then the user types the desired name into the “Name” field. After adding a file and clicking “Submit”, the user is redirected to a new page to select Simulation Identifiers and Attributes.

7.6 Selecting a File to Process

To edit an already uploaded file, the user selects the target file under the “Select a file to process” heading on the left side of the above page. From there, the user can change the name, simulation identifiers, and attributes.

7.7 Simulation Fields and Attributes

Simulation Fields: The fields that differentiate each simulation.
Attributes: Fields that are stored as desired attributes for simulations.