NoSQL End Sem Project Report Document

Group Number: 8
IMT2020128 Ujjwal Agarwal
IMT2020129 Deep Shashank Pani
IMT2020131 Amar Pratap Singh

May 16, 2023

Title of the project

Data Analysis on Indian Railways Dataset Using MongoDB

1 Abstract

In this project, we have tried to make fetching data user-friendly by providing a beautiful and neat User Interface for the user. This UI allows the user to upload large data and provides several choices to the user to process that data. This project has a wide future scope. Working on this project, we faced a lot of challenges (especially while configuring the backend to connect to the database). Meanwhile, we also learn to fix those bugs. We aimed at using both HIVE and MongoDB as databases and to compare the processing delay between them, but due to some backend configuration bugs and lack of community support for HIVE, we dropped the idea of using HIVE as our database and proceeded with MongoDB.

2 Problem Definition

Railways are one of the most used way of transport. This causes the Railway dataset to grow exponentially every day. If we are able to query on such large dataset, that too efficiently, it will be very helpful for regular users. It will help users to efficiently query for a train detail, view past train records, check punctuality of a train, check most suitable trains departing from some station in a preferable time slot, and so on. With diverse railway data, we can do much more.

This problem should be viewed through the lens of a NoSQL database, because the data need not be structured (considering its diversity). And our database should be able to scale up/out to cope with the increasing quantity of data. This is why we prefer using a NoSQL database which can accommodate both the above features.

We intend to provide a user-friendly interface to query on large dataset efficiently.

3 Dataset Used

You can Download and view the dataset here:

https://www.kaggle.com/datasets/arihantjain09/indian-railways-latest

There are 2 files in this dataset train_info and train_schedule. train_schedule has more than 186000 rows while train_info consists of 11114 rows.

4 Approach

We have built a website that can do the data ingestion and processing of data within a click of a button without involving user with the scripts and commands actually needed to perform those actions. This reduces transparency and helps user to focus on the work that matters!

4.1 What all did we use?

1. React: We have chosen Reactjs, a framework of Javascript, to build the frontend. The primary reason for choosing React as frontend language is it's compatibility with Node and easier to re-use it's components as it is based on component-based architecture.

2. Node: We have used Nodejs as backend language to write our server which can then communicate with database for fetching, posting data, etc. The reason for choosing Node is that it is easily configurable to connect with different databases.

3. MongoDB: We have chosen MongoDB as our database. MongoDB is a NoSQL database that can store large volume of structured or unstructured data and is highly scalable.

4. HIVE: We tried using Hive. HIVE is a data warehouse infrastructure built on top of Hadoop, designed to provide a high-level, SQL-like interface for querying and analyzing large datasets. It is highly scalable. But due to some problems in configuring Node for HIVE, we had to drop it.

5. Postman: We have used Postman to fire request to API endpoints to check if it is working as expected. Postman is an easy to use GUI tool that makes it possible to call API endpoint without even writing a single line of code for the frontend part of your application.

6. Axios: We have used a famous JavaScript library 'axios' to make HTTP requests from frontend to the backend. Axios provides a simple and intuitive API for making HTTP requests. It supports all major HTTP methods (GET, POST, PUT, DELETE, etc.) and provides a consistent interface for handling request and response data.

7. Mongoose: Mongoose is an Object-Data Mapping (ODM) library for MongoDB and Node.js. It makes it really simple and convenient to interact with MongoDB databases by defining schemas, models, and performing database operations using JavaScript objects.

4.2 Frontend

We have built a beautiful, neat, and user-friendly single page user interface using react. It can be splitted into following components:-

1. **Navbar** Contains "Home" and "Contact Us" anchor tags.

2. **UploadDataset Section** Contains a button "Upload Data", which when clicked calls a corresponding API Endpoint which uploads the dataset "Indian Railways Dataset".

3. **Query Section** We provided user with 3 queries. At a time, only one query can be run. As soon as the result of the query is available from the backend, **Output** section of the frontend is displayed just above the "Contact Us" section.

4. **Output Section** Output Section has different structure corresponding to different queries. All of the output of users' queries are displayed over here. One feature that should be added to this section is to limit the output to atmost 20 rows and make a "Download Output" button to download the output in a suitable format.

4.3 Backend

We have created a "models" directory to define all the models in one place. It's important to note here that before a user populates a collection, the collection's schema should be ready beforehand, this is how MongoDB works. To define a collection, you need to define schema of the collection.

In the main server file, **app.js**, we imported all the dependencies and connected to MongoDB Cloud service which provides user to create scalable database and create collections within the database. We spin our backend server on localhost with PORT 8080. We did not host our website as of now.

1. **Data Ingestion Pipeline**: To Upload Data to MongoDB Database, we defined an API Endpoint **/api/uploadData**, which when called by the user in the frontend (by clicking on "Upload Dataset" button), does the following:

   It reads a CSV file using the **fs** module and parse it using the **csv-parse** library. It then inserts the parsed data into a MongoDB collection using Mongoose, a MongoDB object modeling tool for Nodejs.

   It pipes the read stream through the **csv.parse()** function.

2. **Data Processing Pipeline**: For each query options provided to the user in the User Interface, we have a defined a separate API Endpoint for it. It helps make the codebase manageable and easy to debug in case of any errors. This can be easily extended by creating more API Endpoints to provide more queries options to the users.

   As of now, we have defined only 3 API Endpoints for 'POST' request.

   (a) */api/ TrainDetailsSourceAndDestinationStation* It requires source and destination station name as input. These inputs are something which is given by the user in the frontend. Based on source and destination station name, this query returns the list of all the trains which starts at source station and ends at destination station.

   (b) */api/TrainDetailsTrainNumber* This API Endpoint expects a "Train Number" input. It then run a query to fetch details of the train like train number, station code, station name,

arrival time, departure time, etc with that train number.

(c) */api/TrainDetailsMoreConstrained*
This is particularly helpful for coolies and daily wage workers at railway stations. This API Endpoint expects a time slot and station code as input. With these inputs, it runs a query to return the list of all the trains which arrives in between the time slot on the station as given by the user.

5 Evaluation and Results

The runtime of each query depends upon the input values given by the user. It cannot be determined that the efficiency/runtime of query with A set of parameters is better than the same query with B set of parameters.

However, we have seen some problem while Uploading large dataset to the MongoDB database. While smaller datasets with 1,00,000 rows are uploaded to MongoDB quite fast, larger datasets with 10,00,00,000 rows are throwing "Javascript Heap Out of Memory" error. We tried exporting an environment variable that specifies the amount of virtual memory allocated to Node.js using the command:

'*export NODE_OPTIONS=–max-old-space-size=sizeInMB*'

Increasing the virtual memory allocated to the Node.js made the whole program crash, so we stick to a relatively smaller dataset. But, on a system with more storage, even the larger dataset will work seamlessly.

6 References

We used the following references for our project:-

- https://www.kaggle.com/datasets/arihantjain09/indian railways-latest
- https://www.makeuseof.com/javascript-heap-out-of-memory-error-fix/
- https://medium.com/analytics-vidhya/import-csv-file-into-mongodb-9b9b86582f34https://medium.com/analytics-vidhya/import-csv-file-into-mongodb-9b9b86582f34
- https://medium.com/@phas0ruk/how-to-bulk-upload-a-csv-to-mongodb-via-a-node-script-7ab1f8f9e8edhttps://medium.com/@phas0ruk/how-to-bulk-upload-a-csv-to-mongodb-via-a-node-script-7ab1f8f9e8ed

7 Future Scope

As we mentioned already, this project is just a newbie project and many more features focusing on queries can be added to this project. With more powerful machines, larger dataset can also be processed very efficiently.