MINI RESTAURANT EXPLORER - REPORT

Group 14

1. PROJECT AND VIDEO LINK

Project Link
`https://github.com/TejaJanakiRam/NoSQL_restaurant`
Video Link
`https://www.youtube.com/watch?v=3U8_s3AcxNU`

2. ABSTRACT

This is a project done as part of NoSQL course. As part of it, we are supposed to use a NoSQL technology taught in the course and bring out a practical application of it. In our project, we tried on a restaurant dataset to understand the concept of pipelines and how they efficiently reduce the time to compute queries and also make it usable in real life by having queries that find the restaurants according to our specifications. Note The queries we have here are of a basic level, Upon the implementation we performed we can create more complex queries for driving them towards greater real-world usability. The results we provide here are mostly referring to the time consumed to perform a particular query. A detailed description of the queries and their results would be discussed later.

3. PROBLEM DEFINITION

Here we took a restaurant dataset which consists of two collections - restaurants and reviews. The main attributes in the restaurants collection are name, restaurant id, address and cuisine. The main attributes in the reviews collection are restaurant id, user id and text. The above-mentioned attributes are the ones we use majorly for our project. Here we created a web application that looks up different restaurants that satisfy the conditions we make. We have 3 major queries related to that, one query is to search for a restaurant with a given input string by the user as part of the name of the restaurant and gives a list of restaurants satisfying this condition. Another query is to check the list of restaurants in a location and we perform that by using the location's pincode which is unique to that location. So Restaurants that have that pincode as part of their address structure will be displayed on the screen. The last query is to get the list of restaurants that has a particular cuisine. For example, we might want to look for restaurants that provide food of Asian or American or French cuisine. In addition to this, we also have customer reviews of the restaurants. We gave the user access to read these reviews after selecting a restaurant and if he wants to add a review or modify a review that was previously written by him then he should log in with his credentials and can perform the desired operation on that review as per his/her wish. As discussed earlier, we have 2 tables in our database - restaurants and reviews. To get the reviews of a particular restaurant we need to compute a join operation and get the results.

The application we developed requires NoSQL technology in its database as,

- The search through the database is faster with these when compared to the traditional databases as in the presence of large data the NoSQL queries compute queries faster due to their usage of the divide and conquer approach, unlike a traditional database.
- The usage of a distributed system makes life easier by having an added advantage for huge data processing, and its high availability.
- NoSQL databases can easily integrate with other services, such as online ordering platforms, delivery services, and payment gateways. We can integrate this application with maps to have a navigation system for the desired restaurant. So NoSQL helps in for a further development purposes too.

Here the main insights we intend to obtain are how quickly the queries are running with and without a pipeline on the restaurant dataset. Where the pipeline is majorly used to speed up the join restaurant and reviews tables to give results for the last query among all set of queries we discussed above

4. APPROACH

In this project, we have used MongoDB Atlas as our database, Reactjs for our frontend, and Expressjs and Nodejs for our backend. We have used the sample restaurants dataset which is available on the MongoDB Atlas.
We start by creating our backend. We start a server using Expressjs and define various routes through which we can access the database for different operations. We create separate

data access classes corresponding to each of the collections present in our database. These files contain the code for accessing the database using MongoDB queries and commands. We also create separate controller classes corresponding to each of the collections in our database. These files act as an API which extract necessary information from the requests to the server and respond with the appropriate data. The functions in the controller classes internally call the functions in the data access classes for accessing the database. This is the basic structure of our backend.
Most of our MongoDB queries as straightforward and do not require pipelining. However, the join operations in our dataset require pipelining as they require lookup operation with a condition.
After creating our backend and testing it using API tool like Postman, we start creating our frontend. We use axios for making http requests to our backend server. We create a service class which is used to send different types of request(GET, POST, PUT, DELETE etc.) to our server. The functions in this service class are called by our react components for sending requests to the server. The frontend consists of various components which include search bars for searching specific restaurants. The top 20 restaurants are displayed as cards which contain the name, cuisine and address of the restaurant. This is the basic approach we use for developing our frontend.
Here we have chosen MongoDB as part of our database as we are majorly focussing on pipelines and when going through various articles we read that Pipelining in MongoDB allows for faster and more efficient data processing compared to Hive and Pig for several reasons:

- Native support for pipelining: MongoDB is designed with a native support for pipelining. It has a flexible document data model that makes it easy to pipeline data operations, and the MongoDB query language supports chaining multiple operations together. This means that data processing can be done entirely within MongoDB without having to move data to another processing engine.

- No data serialization and deserialization: Hive and Pig require data to be serialized and deserialized in order to move data between different stages of processing. This can be time-consuming and resource-intensive, especially for large datasets. In MongoDB, data can be processed without the need for serialization and deserialization, leading to faster processing times

- Distributed processing: MongoDB has a distributed architecture that allows for parallel processing across multiple nodes. This means that data can be processed more quickly and efficiently than in Hive and Pig, which rely on a centralized processing engine.

- Integration with other tools: MongoDB can be easily integrated with other tools, such as Apache Kafka, Spark, and Hadoop, allowing for data to be moved between different processing engines as needed. This means that MongoDB can be used as part of a larger data processing pipeline that includes other tools and technologies. Although this last point is unnecessary for our implementation, we are just mentioning it as part of the advantages.

In addition, we have selected Reactjs, Nodejs and Expressjs tools for the following reasons:

- JavaScript-based stack: React, Node.js, and Express are all built on JavaScript, which makes it easier to develop web applications using a unified language and toolset. This reduces the overhead of learning multiple languages and frameworks, making it easier to build and maintain complex applications.

- High performance: Node.js and Express are both designed to be highly performant, which makes them well-suited for building scalable and efficient web applications. This is particularly important when working with MongoDB, as it allows for high throughput and low-latency database operations.

- Asynchronous programming: Node.js and Express both support asynchronous programming, which allows for non-blocking I/O operations. This means that web applications can handle many concurrent requests without being slowed down by I/O operations.

- Modular architecture: Node.js and Express both have a modular architecture that makes it easy to build complex web applications in a modular and reusable way. This is particularly useful when building applications that involve multiple components and data sources.

- Easy integration with MongoDB: MongoDB is a popular NoSQL database that works well with Node.js and Express. There are many MongoDB drivers available for Node.js, which makes it easy to connect to MongoDB and perform database operations.

Our approach for this project is simple and reusable as this approach can be used to solve other problems as well. Suppose we have a movies dataset which has two collections - one of movies objects and other of reviews of the movies. We can use a similar approach for filtering out documents from such a dataset depending on imdb rating, title, year. We can also use the same pipeline with minor changes for performing join operation between movies collection and reviews collection.
We have used the modified version of sample restaurants dataset which is available on MongoDB Atlas. It has two collections - restaurants and reviews, which are both suitable for our project.

5. EVALUATION AND RESULTS

We have used MongoDB pipelining in our project. Using pipelines instead of non-pipelined queries has many advantages. It optimises the query performance, especially for a large dataset as stages can be used to narrow down the dataset early. Moreover, it allows us to perform complex data transformations in one go using various stages and operators like filter, group, sort, lookup, project. In our project, we had to perform join operation, for which we have used pipelining as it was a complex data transformation. The size of the reviews collection in our dataset is small as we create it on the go. Therefore there will not be a notable difference in the execution times of join queries depending on whether pipelining is used or not. However, as the size of reviews collection increases, the join operation would take a long time because of which we would have to use pipeline to optimise the query time to some extent.

6. REFERENCES

1. `https://www.mongodb.com/docs/atlas/sample-data/sample-restaurants/`
2. `https://hevodata.com/learn/mongodb-atlas-nodejs/`
3. `https://expressjs.com/en/starter/hello-world.html`
4. `https://nodejs.org/api/http.html`
5. `https://devdocs.io/react/`
6. `https://getbootstrap.com/`
7. `https://nodejs.org/en/docs`