U.S. Census Data on Education, Finance, and Jobs

Team-20 Members:
Yan Zheng (zheng23001)
Fahimeh Gholami (fahimeh22000)
Yilin Lai (yilin23000)

Introduction

In this project, a large dataset from the U.S. Census was analyzed. The dataset contains information about **education**, **jobs**, and **income** over many years. The connection between education and income was explored, which can assist businesses and governments in making better decisions.
As the dataset is very large, **Apache Spark** was used to process it quickly, and **MongoDB** was used to store the data efficiently. **Python** was used for data analysis and creating predictions.
The project is significant because it involves a large amount of data that needs to be processed quickly. Spark and MongoDB were used to handle and analyze the data, enabling useful patterns to be identified.

Methodology

Dataset Details

Data from the U.S. Census was used, which contains:

- **Education**: Data about the highest level of education attained.
- **Finance**: Information about income levels.
- **Industry**: Data about job types.

The dataset contains millions of records across several years, requiring Big Data tools to process effectively.

Solution Architecture

A system was built using **Apache Spark** and **MongoDB**:

- **Apache Spark**: Spark was used to process the data quickly and efficiently.
- **MongoDB**: MongoDB was chosen to store the data due to its ability to manage large volumes of data.

To further improve the speed of processing:

- **Partitioning** was applied to the data by **year**, which sped up the queries.
- **Indexes** were created for fields like **Year** and **Education Level** to make data searches faster.

Implementation

Step-by-Step Execution

1.Data Ingestion:

The data was cleaned and organized and then stored in MongoDB.

```
# ------------------------------------
# Part 1: Data Ingestion via Pandas and MongoDB (using PyMongo)
# ------------------------------------
import pandas as pd
from pymongo import MongoClient

# Read CSV files
finance_df_pd = pd.read_csv('./Finance.csv')
industry_df_pd = pd.read_csv('./Industry.csv')
education_df_pd = pd.read_csv('./Educationv.csv')

# Check for missing values
print("Finance Missing Values:")
print(finance_df_pd.isnull().sum())
print("Industry Missing Values:")
print(industry_df_pd.isnull().sum())
print("Education Missing Values:")
print(education_df_pd.isnull().sum())

# Connect to MongoDB and select the target database and collections
client = MongoClient("mongodb://localhost:27017/")
db = client["regional_economy"]
finance_collection = db["finance"]
industry_collection = db["industry"]
education_collection = db["education"]

# Delete existing documents in collections
finance_collection.delete_many({})
industry_collection.delete_many({})
education_collection.delete_many({})

# Insert data into MongoDB collections
finance_data = finance_df_pd.to_dict("records")
finance_collection.insert_many(finance_data)

industry_data = industry_df_pd.to_dict("records")
industry_collection.insert_many(industry_data)

education_data = education_df_pd.to_dict("records")
education_collection.insert_many(education_data)

# Create indexes on frequently used fields (e.g., Year)
finance_collection.create_index([("Year", 1)])
industry_collection.create_index([("Year", 1)])
education_collection.create_index([("Year", 1)])
```

Using Apache Spark

After the data was inserted into MongoDB, **Apache Spark** was used to load and process the data. Spark transformations and queries were performed on the data to explore insights such as trends over time and income analysis.

```
# -------------------------------------
# Part 2: Spark Application with MongoDB Integration & Performance Optimization
# -------------------------------------
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, broadcast
import matplotlib.pyplot as plt
import time

# Create SparkSession with MongoDB connector configurations
spark = SparkSession.builder \
    .appName("RegionalEconomyAnalysis") \
    .config("spark.mongodb.input.uri", "mongodb://127.0.0.1/regional_economy.finance") \
    .config("spark.mongodb.output.uri", "mongodb://127.0.0.1/regional_economy.finance") \
    .config("spark.jars.packages", "org.mongodb.spark:mongo-spark-connector_2.12:2.4.2,org.mongodb:mongodb-drive
    .getOrCreate()

# Adjust the shuffle partitions; reducing from default 200 to 50 for better performance
spark.conf.set("spark.sql.shuffle.partitions", "50")

# -------------------------------
# Read MongoDB Data (original functionality maintained)
# -------------------------------
finance_df = spark.read.format("mongo") \
    .option("collection", "finance") \
    .load()

industry_df = spark.read.format("mongo") \
    .option("uri", "mongodb://127.0.0.1/regional_economy.industry") \
    .load()

education_df = spark.read.format("mongo") \
    .option("uri", "mongodb://127.0.0.1/regional_economy.education") \
    .load()

# Cache the finance DataFrame if used multiple times
finance_df.cache()
```

Optimizing Performance

Creating Indexes in MongoDB

After inserting the CSV data into MongoDB, we create indexes on frequently used fields (e.g., Year):

```
finance_collection.create_index([("Year", 1)])
industry_collection.create_index([("Year", 1)])
education_collection.create_index([("Year", 1)])
```

Since queries or aggregations commonly group, filter, or sort by the Year field, creating an index significantly reduces the amount of data to scan, improving lookup efficiency in MongoDB.

Caching frequently used DataFrames

```
# Cache the finance dataframe to keep it in memory
finance_df.cache()
```

For a DataFrame that will be used multiple times, caching it in memory (or in serialized form) prevents repeated computation and file reads, improving query speed.

Broadcast Join

When performing a join operation where one of the tables is significantly smaller (for example, `education_df`), we can use `broadcast(education_df)` to optimize the join:

```
from pyspark.sql.functions import broadcast

joined_df = finance_df.join(broadcast(education_df), on="Year", how="inner")
```

Broadcasting the smaller dataset to all executors avoids a full shuffle, significantly reducing the cost of the join operation.

Result and Insights

Key Findings

- **Education**: Higher education levels were found to be linked with higher incomes and better job opportunities. Some regions with higher levels of education showed better financial outcomes.
- **Industry and Jobs**: It was observed that certain jobs were associated with specific education levels, showing the impact of education on job types.

Performance Evaluation

The speed of query execution was tested, and performance improved after optimizations were made. The execution time was reduced by 30% through the use of fewer partitions and caching.

Visualizations

Charts were created to visualize trends, such as the relationship between education and income, and how GDP has changed over time.

Conclusion and Future Work

Challenges

- One challenge encountered was processing such a large dataset. It took some time to optimize the system for speed.
- Another difficulty was setting up the connection between MongoDB and Spark.

Possible Improvements

- **Sharding**: The performance could be further improved by using **sharding** in MongoDB, which divides the data across multiple servers.
- **Machine Learning**: Future work could involve using **machine learning** to predict future job trends based on education and income data.

Code Quality and Documentation

The code was organized and written clearly. A **README** file was included in the GitHub repository, explaining how to run the project and what each part of the code does.

GitHub Repository

GitHub Repository Link