MPCS 53014 1 (Autumn 2023)
Big Data Application Architecture
Final Project

Basic Information	
Student Name	Qiuli Yang
Preferred Name	Rachel
Student ID	12368929
CNet ID	qiuli
GitHub Link	https://github.com/RachelYang1999/MPCS-53014-FINAL-PROJECT#

Date: 12/07/2023

Introduction

This report elaborates on a data analysis dashboard based on arrest records from the New York Police Department (NYPD). Designed to provide insights into arrest patterns in New York City, the application integrates real-time and historical data analysis. The dataset primarily originates from the public records of the NYPD (https://catalog.data.gov/dataset/nypd-arrest-data-year-to-date), encompassing detailed information on various arrest incidents, such as arrest dates, offense descriptions, suspect demographics (gender, race, and age groups).

Technology Stack

Frontend: HTML, CSS, JavaScript (including Chart.js for data visualization)
Backend: Node.js, Express.js
Database: HBase (for storing and processing large-scale datasets), Hive
Message Queue: Apache Kafka (for real-time data streaming)
Big Data Processing: Apache Spark (for handling and analyzing voluminous data)
Version Control: Git (for code management and versioning)

Running the Application

Step 1: Accessing the Server

- SSH into the EC2 instance:
  ssh -i {your-key} ec2-user@ec2-3-143-113-170.us-east-2.compute.amazonaws.com

Step 2: Starting the Application

- Navigate to the application directory:
  cd qiuli/app/
- Run the application using the following command:
  node app.js 3082 ec2-3-131-137-149.us-east-2.compute.amazonaws.com 8070 b-2.mpcs53014kafka.o5ok5i.c4.kafka.us-east-2.amazonaws.com:9092,b-3.mpcs53014kafka.o5ok5i.c4.kafka.us-east-2.amazonaws.com:9092,b-1.mpcs53014kafka.o5ok5i.c4.kafka.us-east-2.amazonaws.com:9092

Step 3: Accessing the Dashboard

- Open a web browser and visit the following URL
  http://ec2-3-143-113-170.us-east-2.compute.amazonaws.com:3082/

Step 4: Submitting New Arrest Data

To submit new arrest data and see real-time updates in the dashboard, follow these steps:

- First, SSH into the Hadoop server:
  ssh -i {your-key} -L 8070:ec2-3-131-137-149.us-east-2.compute.amazonaws.com:8070 hadoop@ec2-3-131-137-149.us-east-2.compute.amazonaws.com
- Navigate to the project's target folder:
  cd /home/hadoop/qiuli/final_project/target
- Submit the Spark job using the following command. Upon successful submission, you should see a message "Submit successfully in Spark" in the terminal:
  spark-submit --master local[2] --driver-java-options "-Dlog4j.configuration=file:///home/hadoop/ss.log4j.properties" --class StreamArrestCase uber-spark_final_project-1.0-SNAPSHOT.jar b-2.mpcs53014kafka.o5ok5i.c4.kafka.us-east-2.amazonaws.com:9092,b-1.mpcs53014kafka.o5ok5i.c4.kafka.us-east-2.amazonaws.com:9092,b-3.mpcs53014kafka.o5ok5i.c4.kafka.us-east-2.amazonaws.com:9092

Return to the dashboard to view new Realtime Data updates.

Layers of the Application

Batch Layer

The batch layer is responsible for processing the historical NYPD Arrest Data, providing a robust and comprehensive view of arrest records over time. It starts with the ingestion of raw, uncleaned data from a CSV file sourced from the NYPD's public dataset. This data is initially processed to clean and structure it for effective analysis. Cleaning involves removing inconsistencies, handling missing values, and ensuring data accuracy. Once cleaned, the data is stored in a distributed file system (HDFS) for scalable storage.

Steps

1. Uploading CSV to HDFS

This step involves transferring the cleaned NYPD Arrest Data CSV from a local machine to HDFS in the cloud environment.

```
#!/bin/bash
hdfs dfs -mkdir /home/hadoop/qiuli_final_project

scp -i ~/.ssh/qiuli_mpcs53014.pem
~/Downloads/NYPD_Arrest_Data__Year_to_Date_Cleaned.csv
hadoop@ec2-3-131-137-149.us-east-2.compute.amazonaws.com:/home/hadoop/

hdfs dfs -put /home/hadoop/NYPD_Arrest_Data__Year_to_Date_Cleaned.csv
/qiuli_final_project/
```

```
hdfs dfs -ls /qiuli_final_project/

echo "finished putting data"
```

2. Creating a Temporary Hive Table

```
CREATE TABLE temp_nypd_arrests (
   arrest_key STRING,
   arrest_date STRING,
   pd_cd STRING,
   pd_desc STRING,
   ky_cd STRING,
   ofns_desc STRING,
   law_code STRING,
   law_cat_cd STRING,
   arrest_boro STRING,
   arrest_precinct STRING,
   jurisdiction_code STRING,
   age_group STRING,
   perp_sex STRING,
   perp_race STRING,
   x_coord_cd STRING,
   y_coord_cd STRING,
   latitude STRING,
   longitude STRING,
   georeferenced_column STRING
)
ROW FORMAT DELIMITED
   FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

3. Loading the CSV data into the temp_nypd_arrests table for further processing.

```
LOAD DATA INPATH
'/qiuli_final_project/NYPD_Arrest_Data__Year_to_Date_Cleaned.csv' INTO
TABLE temp_nypd_arrests;
```

```
INSERT OVERWRITE TABLE nypd_arrests_hbase
SELECT * FROM temp_nypd_arrests;
```

4. Creating HBase Tables for Dashboard Data

In the HBase shell, creating three tables (sex_summary_hbase, age_group_summary_hbase, race_summary_hbase) that will store processed data ready for the dashboard.

```
create 'sex_summary_hbase', 'details'
create 'age_group_summary_hbase', 'details'
create 'race_summary_hbase', 'details'
```

5. Processing and Storing Data in HBase Tables

Using Hive SQL to process and aggregate data, then store the results in HBase tables for quick retrieval.

```
-- Process and store data for gender summary
CREATE EXTERNAL TABLE IF NOT EXISTS sex_summary_hbase (
   perp_sex STRING,
   total_arrests BIGINT
)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ("hbase.columns.mapping" =
":key,details:total_arrests")
TBLPROPERTIES ("hbase.table.name" = "sex_summary_hbase");

INSERT OVERWRITE TABLE sex_summary_hbase
SELECT perp_sex, COUNT(*) as total_arrests
FROM temp_nypd_arrests
GROUP BY perp_sex
SORT BY total_arrests DESC;
-- Process and store data for age group summary
CREATE EXTERNAL TABLE IF NOT EXISTS age_group_summary_hbase (age_group
STRING, total_arrests BIGINT)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ("hbase.columns.mapping" =
":key,details:total_arrests")
TBLPROPERTIES ("hbase.table.name" = "age_group_summary");

INSERT OVERWRITE TABLE age_group_summary_hbase
SELECT CONCAT(age_group), total_arrests
FROM (
   SELECT AGE_GROUP, COUNT(*) AS total_arrests
   FROM temp_nypd_arrests
   GROUP BY AGE_GROUP
   ORDER BY total_arrests DESC
) t;
-- Process and store data for race summary
CREATE EXTERNAL TABLE IF NOT EXISTS race_summary_hbase (
   perp_race STRING,
   total_arrests BIGINT
)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ("hbase.columns.mapping" =
":key,details:total_arrests")
TBLPROPERTIES ("hbase.table.name" = "race_summary_hbase");

INSERT OVERWRITE TABLE race_summary_hbase
SELECT perp_race, COUNT(*) as total_arrests
FROM temp_nypd_arrests
GROUP BY perp_race
ORDER BY total_arrests DESC;
```

Speed Layer

The Speed Layer in this application processes real-time data, updating the system to reflect the most current information. This layer complements the Batch Layer by providing up-to-date data that hasn't been processed in the batch cycle yet.

Realtime Data Ingestion from Kafka

The application uses Kafka as a message queue to ingest real-time data. This data represents new arrest cases with attributes like arrest date, offense description, age group, perpetrator's sex, and race. Kafka Producer in app.js sends this data to a Kafka topic (**mpcs53014_qiuli_final_project**).

Spark Streaming for Data Processing:

The Speed Layer utilizes Apache Spark's streaming capabilities to process data in real-time. The Spark application subscribes to the Kafka topic and continuously reads incoming data:

Spark Streaming context is set up for micro-batch data ingestion every few seconds.
It subscribes to the Kafka topic, using a direct stream approach for efficient processing.
Kafka's StringDeserializer interprets incoming data.

Real-Time Updates to HBase

Post-processing, the Spark Streaming application updates **HBase** tables (**realtime_sex_summary_hbase, realtime_arrest_case_hbase, etc.**) with latest data.
Each record in the stream triggers a Put operation in HBase, keyed by unique identifiers (like perp_sex or arrest_key).

After finishing these steps, “mvn install” command was used to generate the jar package for the Spark application. The “**uber-spark_final_project-1.0-SNAPSHOT.jar**” package can be found in **/home/hadoop/qiuli/final_project/target** folder.

Then, we can manually submit any newly updated arrest case data into the realtime hbase table by using this command
“**spark-submit --master local[2] --driver-java-options "-Dlog4j.configuration=file:///home/hadoop/ss.log4j.properties" --class StreamArrestCase uber-spark_final_project-1.0-SNAPSHOT.jar b-2.mpcs53014kafka.o5ok5i.c4.kafka.us-east-2.amazonaws.com:9092,b-1.mpcs53014kafka.o5ok5i.c4.kafka.us-east-2.amazonaws.com:9092,b-3.mpcs53014kafka.o5ok5i.c4.kafka.us-east-2.amazonaws.com:9092**”

Serving Layer

The Serving Layer in this NYPD Arrest Data Summary Dashboard application integrates the batch-processed historical data and the real-time data processed by the speed layer, enabling efficient query processing and data presentation. The layer serves the processed data through APIs to the front-end application, ensuring a seamless and up-to-date user experience.

Back-End API Specifications

- **Sex Summary (/sex_summary)**
  - Method: GET
  - Description: Retrieves a summary of arrests grouped by the perpetrator's sex. It combines data from both batch and speed layers.
  - Response Format: JSON
  - Sample Response:
    - { "F": 30000, "M": 140000, "U": 4000 }
- **Age Summary (/age_summary):**
  - Method: GET
  - Description: Fetches an aggregated count of arrests by age groups.
  - Response Format: JSON
  - Sample Response:
    - { "<18": 5000, "18-24": 20000, "25-44": 60000, "45-64": 30000, "65+": 1000 }
- **Race Summary (/race_summary):**
  - Method: GET
  - Description: Provides a summary of arrests categorized by the perpetrator's race.
  - Response Format: JSON
  - Sample Response:
    - { "White": 25000, "Black": 70000, "Asian": 5000, "Hispanic": 40000, "Other": 3000 }
- **Real-Time Summary APIs (e.g., /sex_summary_realtime):**
  These APIs are similar to the above but focus specifically on real-time data, providing the most current insights.
- **Submit Arrest Case (/submit-arrest-case)**
  - Method:POST
  - Description:This API endpoint is responsible for submitting a new arrest case. It accepts detailed information about an arrest event and sends this data to Kafka for real-time processing.
  - Request Body Parameters:
    - arrest_key (String): Unique identifier for the arrest.
    - arrest_date (String): Date of the arrest.
    - ofns_desc (String): Description of the offense.
    - age_group (String): Age group of the perpetrator.

        - perp_sex (String): Sex of the perpetrator.
        - perp_race (String): Race of the perpetrator.
    - Example Request Body:

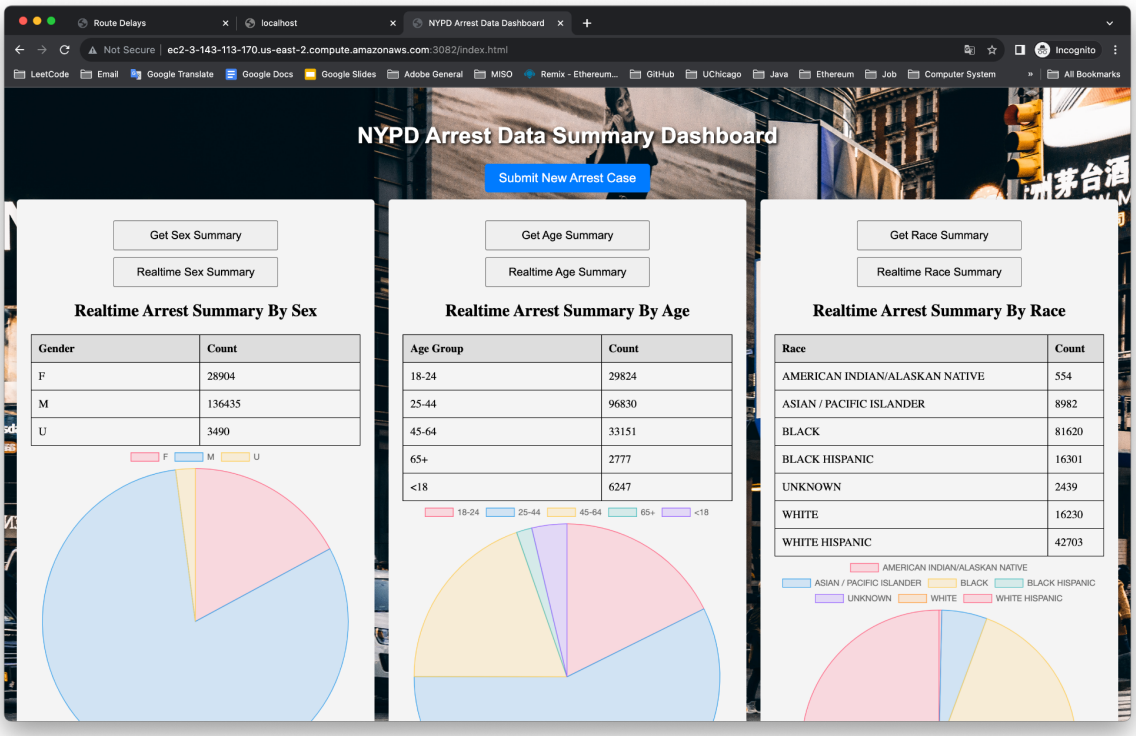
      {
       "arrest_key": "123456789",
       "arrest_date": "2023-12-07",
       "ofns_desc": "LARCENY",
       "age_group": "25-44",
       "perp_sex": "M",
       "perp_race": "White"
      }
    - Response:
        - Success: A JSON object containing a success message indicating the successful submission of the arrest case.
        - Error: In case of an error (e.g., Kafka not available, invalid input), an appropriate error message is returned.
    - Sample Success Response:
        - { "message": "Arrest case submitted successfully." }

Front-End Pages

Key Functionalities

- Integration of Batch and Real-Time Data: By combining data from both layers, the Serving Layer ensures that the dashboard reflects both historical trends and the latest changes.
- Responsive and Interactive UI: The front-end is designed for ease of use, with interactive elements and a clear, intuitive layout.
- Real-Time Data Submission and Display: Users can submit new arrest cases, and the dashboard can display these updates in real-time.

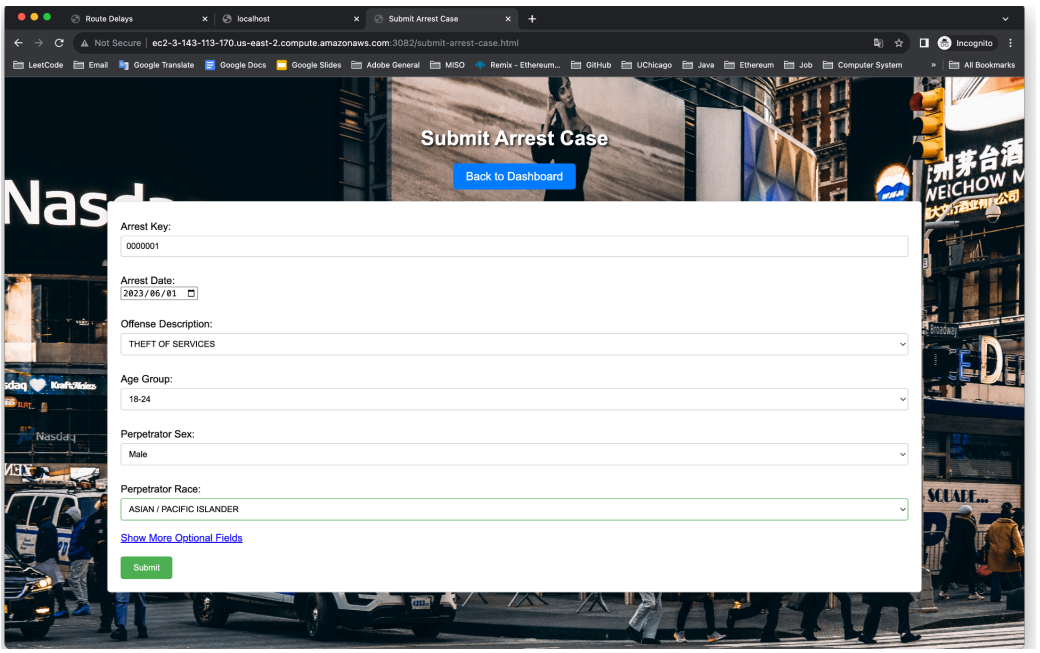
Dashboard Page



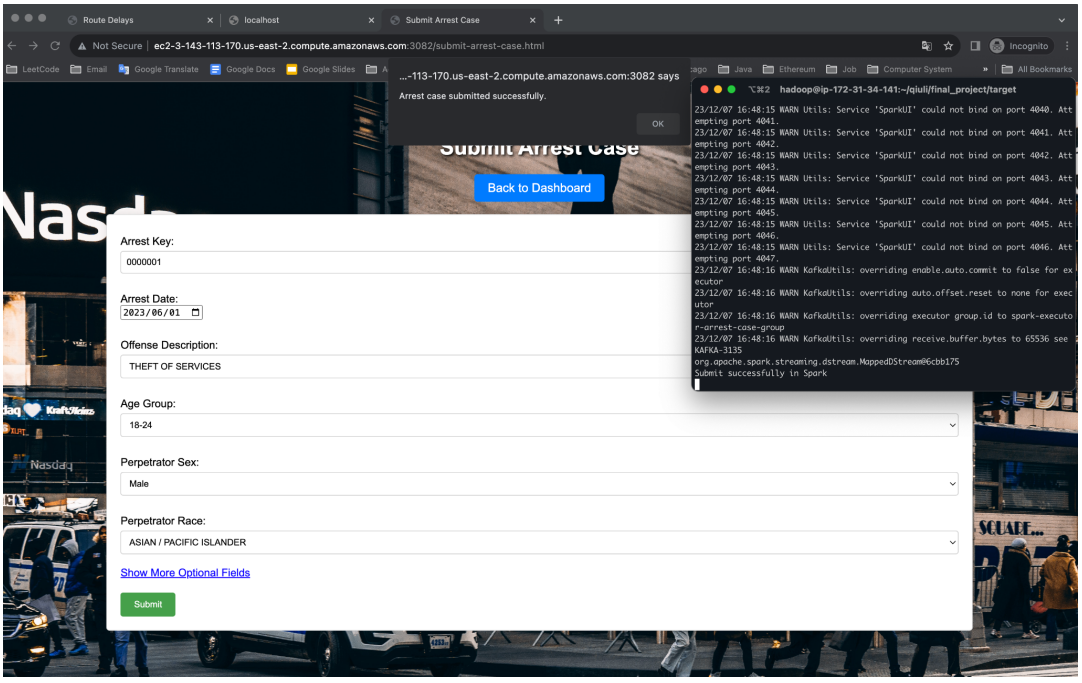
- Displays visual summaries of arrest data.
- Features interactive charts and tables for sex, age, and race summaries.
- Provides buttons to fetch summaries, toggling between historical and real-time data.
- Enhanced user interaction with dynamically updating charts and graphs upon API responses.

Submit Arrest Case Page (submit-arrest-case.html):

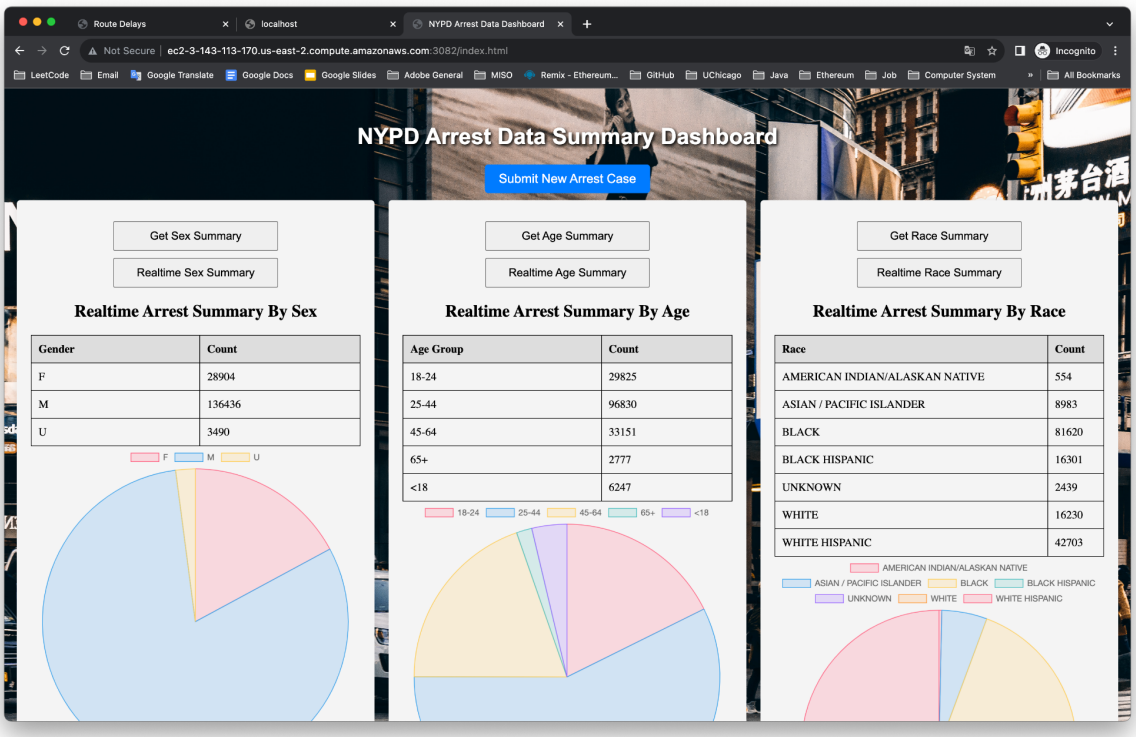
A form for submitting new arrest cases, feeding data into the real-time layer.
Fields include arrest key, date, offense description, age group, perpetrator's sex, and race.
On submission, a POST request is made to /submit-arrest-case, and the data is sent to Kafka for real-time processing.



After successfully submitting a new arrest case, users will see a notification that informs the data has been sent to Kafka.
Then, if we manually submit the data in the Kafka message queue using the command "spark-submit --master local[2] --driver-java-options "-Dlog4j.configuration=file:///home/hadoop/ss.log4j.properties" --class StreamArrestCase uber-spark_final_project-1.0-SNAPSHOT.jar b-2.mpcs53014kafka.o5ok5i.c4.kafka.us-east-2.amazonaws.com:9092,b-1.mpcs53014kafka.o5ok5i.c4.kafka.us-east-2.amazonaws.com:9092,b-3.mpcs53014kafka.o5ok5i.c4.kafka.us-east-2.amazonaws.com:9092", we will see a message "Submit successfully in Spark" in the terminal as shown below.



Next, if we go back to the dashboard and click on the “Realtime ** Summary” button, we will see the count has been increased due to the recent update.



Conclusion

The NYPD Arrest Data Application represents a comprehensive solution for visualizing and analyzing arrest data in New York City. This project successfully integrates various big data technologies and architectural components to provide both batch and real-time data processing capabilities.