MPCS 53014 Big Data Architecture Final Project

Shichen Sun

shichen@uchicago.edu

1. Overview

This project provides a platform for analyzing taxi trip data. It uses a Lambda architecture consisting of a **Batch Layer**, a **Speed Layer** and a **Serving Layer** to process and analyze large-scale taxi trip data in near real-time. The platform supports querying aggregated statistics (e.g., average fare, trip count, total fare) for specific taxi companies and displays results in a web-based interface.

The system leverages **Hive** and **HBase** for data storage, **Kafka** for streaming data ingestion, and **Node.js** for the web application layer. Users can input a taxi company name via the web interface and retrieve detailed statistics.

Data source is Kaggle: https://www.kaggle.com/datasets/aungdev/chicago-transportation-2020?resource=download

2. Architecture

2.1 Batch Layer

The batch layer is responsible for precomputing aggregated statistics for taxi companies. These statistics are stored in a dedicated HBase table (`shichen_taxi_trips_by_company`) and include:

- Total number of trips (`trip_count`)
- Total fare (`total_fare`)
- Total trip duration in seconds (`total_trip_seconds`)
- Most frequent pickup and dropoff areas (`top_pickup_area`, `top_dropoff_area`)

Workflow:

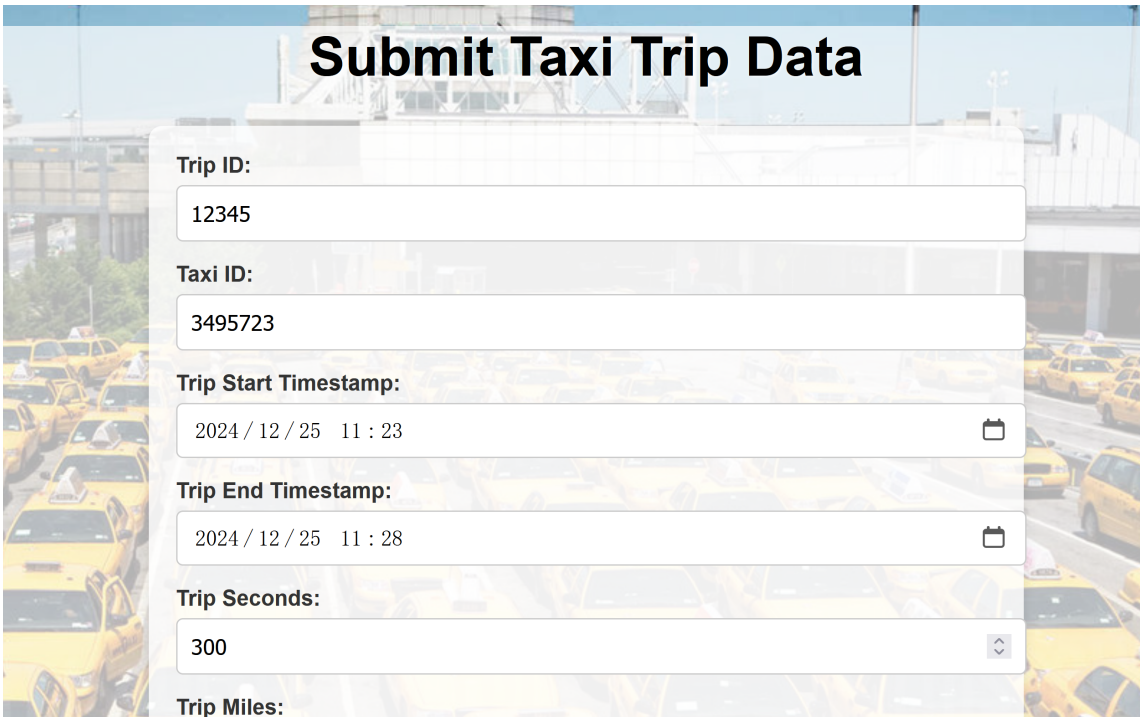
1. Raw data is processed in **Hive** and aggregated by company.
2. The results are written to the `shichen_taxi_trips_by_company` table in HBase.

2.2 Speed Layer

The speed layer processes incoming taxi trip events in near real-time. It uses Kafka to stream data, and a Spark job consumes the data, calculates statistics, and writes them into an HBase table (`shichen_taxi_trips_speed`).

Workflow:

1. In the webpage, user can submit new taxi data and the data would be fed into my Kafka topic `shichen-taxi_trip-events`.



2. A Spark Streaming application consumes Kafka events, computes statistics for the latest data, and writes the results to HBase.

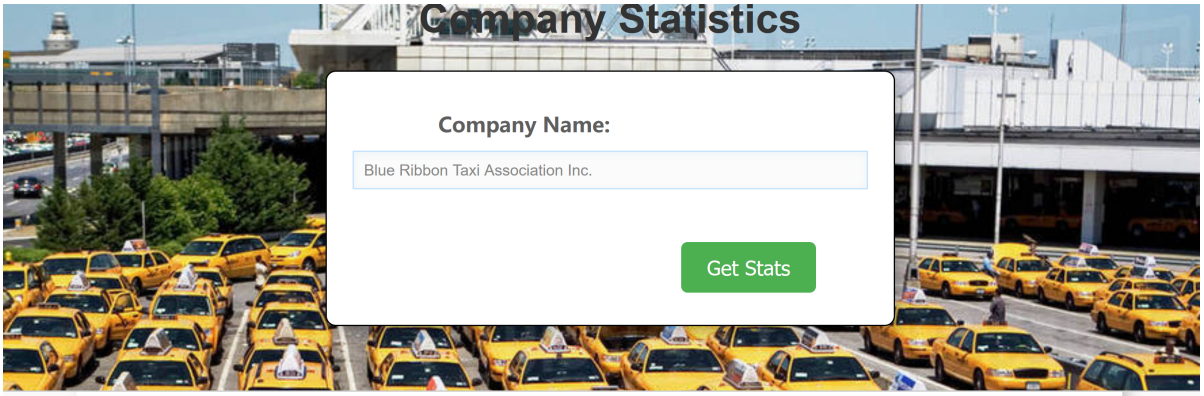
2.3 Serving Layer

Extracted several hbase tables for quick queries:

```
create 'shichen_taxi_trips_by_id', 'trip_info'

create 'shichen_taxi_trips_by_location', 'trip_info'

create 'shichen_taxi_trips_by_pickup', 'trip_info'

create 'shichen_taxi_trips_by_dropoff', 'trip_info'

create 'shichen_taxi_trips_by_date', 'trip_info'

create 'shichen_taxi_trips_by_company', 'trip_info'
```

```
create 'shichen_taxi_trips_by_fare', 'trip_info'

create 'shichen_taxi_trips_by_geo', 'trip_info'
```

And every time a query reaches, it would query both batch layer and speed layer, and combine the data together to show the final results:



Statistics for Blue Ribbon Taxi Association Inc.

Metric	Value
Total Trips	324586
Total Fare	$3743013.26
Total Trip Seconds	239585400.00
Average Fare	$11.53

3. Components

3.1 Hive Tables

`shichen_taxi_trips`

`shichen_taxi_trips_by_company_hive`

`shichen_taxi_trips_by_pickup_hive`

`shichen_taxi_trips_by_dropoff_hive`

`shichen_taxi_trips_company_aggregated`

`shichen_taxi_trips_by_fare_hive`

Used to preprocess and aggregate historical taxi trip data for the Batch Layer.

3.2 HBase Tables

- `shichen_taxi_trips_by_company`:

  Stores pre-aggregated statistics for each taxi company (Batch Layer).

- `shichen_taxi_trips_speed`:

  Stores trip-level data for real-time updates (Speed Layer).

3.3 Kafka

- **Topic**: `shichen-taxi-trip-events`
- Streams raw taxi trip events to the Speed Layer.

3.4 Node.js Application

Queries the Batch Layer (`shichen_taxi_trips_by_company`) for precomputed statistics.

Queries the Speed Layer (`shichen_taxi_trips_speed`) for real-time statistics.

Merges results from both layers.

4. Setup

1. **Start Spark Streaming Job**:

   ```
   spark-submit --master local[2] --driver-java-options "-
   Dlog4j.configuration=file:///home/hadoop/ss.log4j.properties" --class
   org.example.StreamTaxiTrips uber-speed_layer-1.0-SNAPSHOT.jar wn0-
   kafka.m0ucnnwuiqae3jdorci214t2mf.bx.internal.cloudapp.net:9092,wn1-
   kafka.m0ucnnwuiqae3jdorci214t2mf.bx.internal.cloudapp.net:9092
   ```

2. **Start the Node.js Application**:

   ```
   node app.js 3174 http://10.0.0.26:8090 wn0-kafka:9092
   ```

3. **Access the Web Interface**:

   After configuring the proxy following the instructions on the slides,

   Submit the data at this page: `http://10.0.0.38:3174/submit-taxi.html`

   Query the statistic at this page: `http://10.0.0.38:3174/`

5. Videos

Check this out to learn how my project works: https://drive.google.com/file/d/1CDOqh3xqAWUVDoZerBMe4ZfjN_0k1TE_/view?usp=sharing