Project_2: Bike-Sharing Analytics System Implementation

Professors: S. Sioutas, A. Komninos, G. Vonitsanos (Postdoc Researcher@CEID)

Coursework Specification

Overview

This coursework requires students to implement a real-time bike-sharing analytics system that demonstrates the practical application of decentralized data technologies. The system will integrate live data from multiple sources, process it using distributed computing frameworks, and implement machine learning predictions for bike utilization.

Learning Objectives

- Implement data ingestion from multiple REST APIs
- Design and develop data streaming pipelines using Apache Kafka
- Process real-time data using Apache Spark
- Develop predictive models using SparkML
- Apply software engineering best practices in a distributed system context

System Requirements

1. Data Sources Integration

The main data sources are open live data of bike-sharing systems using the GBFS specification. The specification documentation can be found here https://github.com/MobilityData/gbfs

A list of cities that publish data in the GBFS format can be found here https://github.com/MobilityData/gbfs/blob/master/systems.csv

As an example, students can use the New York CitiBike system
https://citibikenyc.com/system-data
https://gbfs.citibikenyc.com/gbfs/2.3/gbfs.json

However, you can use any city with a PHYSICAL bicycle docking service.

Weather data can be obtained from any source, but students are encouraged to use the OpenWeatherMap API https://openweathermap.org/api/one-call-3

Students must implement data collectors for two distinct APIs:

1.1. **Bike-Sharing System API (GBFS Format)**
- **Endpoint 1**: *Station Information (station_information.json)*
  - Provides static information about bike-sharing stations
  - Data includes station locations, names, and capacities
- **Endpoint 2**: *Station Status (station_status.json)*
  - Provides real-time information about bike availability
  - Data includes number of available bikes and docks

1.2. **Weather API (OpenWeatherMap)**
- Provides current weather conditions based on geographical coordinates
- Required parameters: latitude and longitude of the city
- Must collect relevant weather metrics (temperature, precipitation, wind speed, clouds)

2. Data Pipeline Implementation

2.1 Data Ingestion Layer

- Implement Python scripts to fetch data from both APIs
- Handle API rate limiting and error cases
- Implement proper data validation and error logging
- Ensure regular data updates (minimum every 5 minutes)
- Do any data pre-processing as required to remove unnecessary data

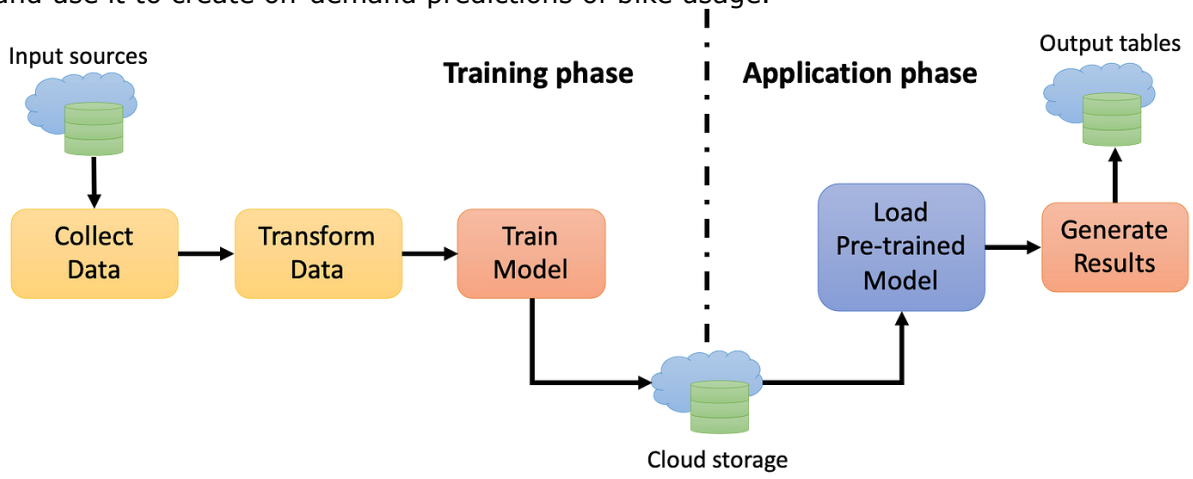
2.2 Message Queue Layer (Apache Kafka)
- Set up Kafka topics for:
  - Station information
  - Station status
  - Weather data
- Implement Kafka producers in Python
- Ensure proper message serialization/deserialization
- Implement error handling and recovery mechanisms

2.3 Data Processing Layer (Apache Spark)
- Implement SparkSQL queries to:
  - Join station information with status data
  - Calculate utilization rates per station and across the city
  - Correlate weather conditions with bike usage
  - Generate hourly usage summaries
- Required analytics:
  - Overall system utilization on a single dataframe, including weather conditions
- Data Storage:
  - Save the usage summaries (dataframe) to a database or other storage

Recommended dataframe fields:
- *timestamp*
- *city_name*
- *temperature*
- *wind_speed*
- *precipitation*
- *cloudiness*
- *average_docking_station_utilisation*
- *max_docking_station_utilisation*
- *min_docking_station_utilisation*
- *std_dev_docking_station_utilisation*

2.4 Machine Learning Component

Students will leave the system to collect and store data for a period of time (e.g. 7 days). With the saved data, students will train a simple regression ML model (e.g. RandomForest) and use it to create on-demand predictions of bike usage.



- Implement a SparkML pipeline to predict bike utilization
- Required input features:
  - Time (current time + 1H)
  - Forecasted Weather conditions at current time + 1H (temperature, precipitation, wind speed)
- Model requirements:
  - Predict utilization for the next hour (current time + 1H)
  - Include feature engineering steps
  - Implement model evaluation metrics

Do not worry if your model accuracy is not perfect as this is not part of the scoring. For this coursework, It is enough that you develop a basic model that produces some output, regardless of its accuracy.

Deliverables
1. Source code with complete implementation
2. Documentation including:
   - System architecture description
   - Installation and setup instructions
   - API integration details
   - Data processing pipeline description
   - ML model documentation
3. Test results and performance metrics
4. Brief report (max 2000 words) discussing:
   - Implementation challenges
   - System performance analysis
   - Potential improvements

Assessment Criteria
- Data Integration (20%)
  - Correct API integration
  - Error handling
  - Data validation
- Kafka Implementation (20%)
  - Message queue design
  - Producer/consumer implementation
  - Error handling
- Spark Processing (35%)
  - Data processing efficiency
  - Query implementation
  - Performance optimization
- Machine Learning (10%)
  - Model implementation
  - Feature engineering
  - Prediction pipeline
- Code Quality (15%)
  - Documentation
  - Testing
  - Best practices

Additional Notes
- Students must ensure their implementation handles API rate limiting appropriately
- The system should be scalable to handle increasing data volumes
- Students should document any assumptions made during implementation

Shared Data Repository

To collect enough data to build reasonable models, we provide a common repository accessible through API calls at http://usidas.ceid.upatras.gr/bikes

Students are encouraged to also send the data they collect and process through Spark, to this common repository.

/store_data.php [POST]

Accepts a JSON object as POST data, with the following schema
- timestamp : unix timestamp
- city_name : alphanumeric (50)
- temperature : float
- wind_speed : float
- precipitation : int (0-100)
- cloudiness : int (0-100)
- average_docking_station_utilisation : float
- max_docking_station_utilisation : float
- min_docking_station_utilisation : float
- std_dev_docking_station_utilisation : float

Example: https://reqbin.com/oweqsc5m

/get_data.php [GET]

Returns database records between the timestamps specified by the user. Call parameters are:

- from : unix timestamp
- to : unix timestamp

Example: https://reqbin.com/hwqpu8pv

(***) Deliverables: Zip or Rar file with executable files. Deadline: ~ 1st WEEK of February, 2024.