Báo cáo Bài tập lớn Môn Lưu trữ và xử lý dữ liệu lớn
Phân tích dữ liệu các chuyến bay

I. Đặt vấn đề

1. Đặt vấn đề

Vận tải hàng không là một trong những ngành công nghiệp quan trọng, góp phần thúc đẩy sự phát triển kinh tế và kết nối toàn cầu. Tuy nhiên, các vấn đề liên quan đến chậm trễ và hủy chuyến bay gây ra không ít bất tiện cho hành khách, thiệt hại tài chính cho các hãng hàng không, và ảnh hưởng lớn đến hiệu quả vận hành của ngành.

Những yếu tố như thời tiết, kỹ thuật, hoạt động sân bay, và quản lý không lưu có thể làm tăng khả năng chậm trễ hoặc hủy chuyến. Vì vậy, việc phân tích dữ liệu chuyến bay, xác định các xu hướng và nguyên nhân, cũng như dự đoán các sự cố tiềm tàng là rất quan trọng để cải thiện trải nghiệm người dùng và tối ưu hóa hoạt động của ngành hàng không.

2. Mục tiêu

Xây dựng một hệ thống tích hợp trên kiến trúc Lambda, kết hợp xử lý dữ liệu theo lô (batch) và thời gian thực (streaming), nhằm:

- Cung cấp các phân tích toàn diện về lịch sử chuyến bay (2009–2020).
- Dự đoán khả năng xảy ra các sự cố như chậm trễ hoặc hủy chuyến dựa trên dữ liệu lịch sử.
- Giám sát và hiển thị thông tin thời gian thực, hỗ trợ ra quyết định nhanh chóng và hiệu quả trong quản lý vận tải hàng không.

3. Dữ liệu

Bộ dữ liệu sử dụng trong dự án này là Airline Delay Analysis Dataset chứa thông tin hàng ngày về các chuyến bay nội địa được cung cấp bởi các hãng hàng không lớn nhất tại Hoa Kỳ. Dữ liệu được thu thập bởi Cục Thống kê Vận tải của Bộ Giao thông Vận tải Hoa Kỳ, với trọng tâm là phân tích tình trạng chậm trễ chuyến bay. Bộ dữ liệu bao gồm giai đoạn từ 2009 đến 2020, trong đó mỗi năm được lưu trữ dưới dạng một tệp CSV riêng biệt. Tổng dung lượng của toàn bộ bộ dữ liệu là khoảng 9GB và có thể được truy cập công khai trên Kaggle.

Dữ liệu gồm các thuộc tính:

Cột	Mô tả
FL_DATE	Ngày thực hiện chuyến bay (định dạng yy/mm/dd).
OP_CARRIER	Mã định danh duy nhất của hãng hàng không.
OP_CARRIER_FL_NUM	Số hiệu chuyến bay của hãng hàng không.
ORIGIN	Mã duy nhất của sân bay khởi hành.
DEST	Mã duy nhất của sân bay đến.
CRS_DEP_TIME	Thời gian khởi hành dự kiến.
DEP_TIME	Thời gian khởi hành thực tế.
DEP_DELAY	Tổng thời gian chậm trễ khi khởi hành (tính bằng phút).
TAXI_OUT	Thời gian di chuyển từ sân bay khởi hành đến khi máy bay rời khỏi mặt đất.
WHEELS_OFF	Thời điểm bánh máy bay rời khỏi mặt đất.
WHEELS_ON	Thời điểm bánh máy bay chạm đất.
TAXI_IN	Thời gian di chuyển từ khi bánh chạm đất đến khi đến sân bay đích.
CRS_ARR_TIME	Thời gian đến dự kiến.
ARR_TIME	Thời gian đến thực tế.
ARR_DELAY	Tổng thời gian chậm trễ khi đến nơi (âm nếu đến sớm), tính bằng phút.
CANCELLED	Chuyến bay bị hủy (1 = hủy).
CANCELLATION_CODE	Lý do hủy chuyến: A - Hãng hàng không; B - Thời tiết; C - Hệ thống quốc gia; D - An ninh.
DIVERTED	Máy bay hạ cánh tại sân bay khác so với sân bay đích (1 = có).
CRS_ELAPSED_TIME	Thời gian bay dự kiến.
ACTUAL_ELAPSED_TIME	Thời gian bay thực tế (AIR_TIME + TAXI_IN + TAXI_OUT).
AIR_TIME	Thời gian máy bay ở trên không (từ WHEELS_OFF đến WHEELS_ON).
DISTANCE	Khoảng cách giữa sân bay khởi hành và sân bay đích.
CARRIER_DELAY	Thời gian chậm trễ do hãng hàng không (tính bằng phút).
WEATHER_DELAY	Thời gian chậm trễ do điều kiện thời tiết.
NAS_DELAY	Thời gian chậm trễ do Hệ thống Hàng không Quốc gia.
SECURITY_DELAY	Thời gian chậm trễ do kiểm tra an ninh.
LATE_AIRCRAFT_DELAY	Thời gian chậm trễ do ↓ bay đến muộn.

II. Kiến trúc và thiết kế

1. Kiến trúc tổng thể

Dự án này áp dụng kiến trúc Lambda, một mô hình thiết kế xử lý dữ liệu mạnh mẽ, tích hợp cả xử lý theo lô (batch processing) và xử lý thời gian thực (stream processing). Kiến trúc này phù hợp với các ứng dụng yêu cầu phân tích dữ liệu thời

gian thực, đồng thời đảm bảo tính chính xác và đầy đủ trong phân tích dữ liệu lịch sử.

Ưu điểm của kiến trúc lambda trong dự án này:

- Khả năng mở rộng (Scalability): Xử lý hiệu quả bộ dữ liệu chuyến bay khổng lồ (~9GB) và sẵn sàng mở rộng cho cả dữ liệu thời gian thực và lịch sử.
- Khả năng chịu lỗi (Fault Tolerance): Đảm bảo tính dự phòng thông qua các tầng xử lý độc lập, giúp hệ thống hoạt động ổn định trước các lỗi tiềm ẩn.
- Độ trễ thấp và chính xác cao (Low Latency and Accuracy): Cân bằng giữa việc cung cấp thông tin thời gian thực và đảm bảo phân tích dữ liệu lịch sử chính xác.

=> Sự kết hợp của các thành phần này tạo nên một hệ thống toàn diện, vừa có khả năng dự đoán chính xác vừa cung cấp thông tin chi tiết kịp thời cho các quyết định quan trọng.

2. Phần Batch layer

Dữ liệu thô được làm sạch và lưu trữ trong HDFS để sẵn sàng cho phân tích.

Sử dụng Apache Spark để xử lý dữ liệu phân tán, đảm bảo khả năng mở rộng khi xử lý tập dữ liệu khổng lồ.

Thực hiện các mô hình dự đoán, như dự đoán chậm trễ và hủy chuyến bay, thông qua thư viện Spark MLlib.

Kết quả phân tích được lưu trong Cassandra cluster để phục vụ truy vấn và phân tích sau này.

3. Phần Speed layer

Sử dụng Kafka cluster để tiếp nhận và quản lý luồng dữ liệu thời gian thực từ nhiều nguồn khác nhau.

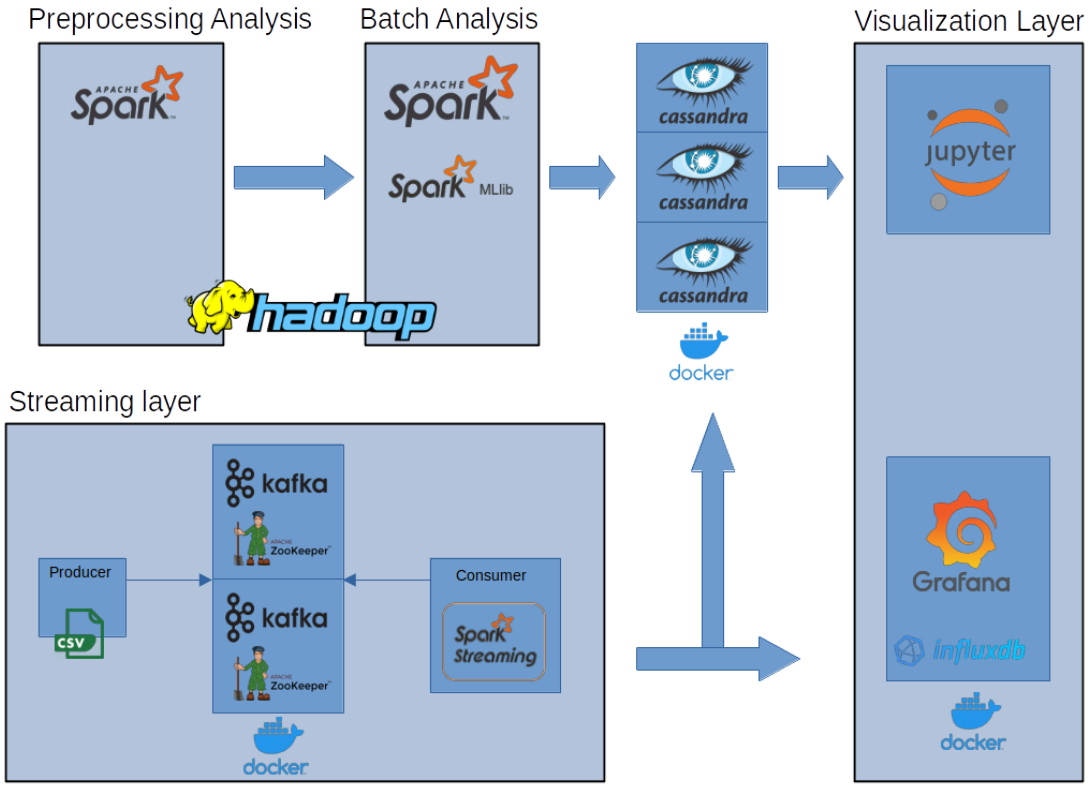
Spark Streaming xử lý các luồng dữ liệu này, thực hiện các phép tính và phân tích ngay lập tức để cung cấp thông tin chi tiết kịp thời.

4. Phần serving layer

Jupyter Notebook kết nối với Cassandra qua cassandra-driver connector để truy xuất dữ liệu và trực quan hóa bằng các biểu đồ, hỗ trợ phân tích tùy chỉnh.

Grafana được sử dụng để tạo các bảng điều khiển thời gian thực, trực quan và dễ sử dụng, giúp giám sát và khám phá dữ liệu một cách hiệu quả

5. Data flow và component interaction diagrams



III. Chi tiết triển khai

1.Triển khai Batch layer

1.1. Triển khai cassandra

Tạo 2 Keyspace trong cassandra (StreamingKeySpace và BatchKeySpace) để lưu dữ liệu sau khi xử lý của batch layer và steaming layer. Mỗi bản sao của dữ liệu sẽ được lưu trên 3 node trong cụm.

Ví dụ tạo keyspace:

```
//Create keyspace
CREATE KEYSPACE IF NOT EXISTS StreamingKeySpace WITH replication = {'class':'SimpleStrategy', 'replication_factor':3};
CREATE KEYSPACE IF NOT EXISTS BatchKeySpace WITH replication = {'class':'SimpleStrategy', 'replication_factor':3};
```

Tạo các bảng trong StreamingKeySpace:

- Bảng Flight_Data lưu dữ liệu thô các chuyến bay
- Bảng Delay_Data Lưu dữ liệu tổng hợp thời gian thực (delay theo tháng và hãng bay).

Tạo các bảng rong BatchKeySpace:

- Bảng `delay_total`: Trung bình delay theo hãng bay\
- Bảng `delay_total_src_dest`: Delay trung bình theo sân bay đi và đến
- Bảng `cancellation_diverted_total`: Thống kê tỷ lệ và số lượng hủy chuyến/chuyển hướng theo hãng bay.
- Bảng `dist_total`: Tổng khoảng cách bay theo hãng bay.
- Bảng `max_consec_delay_year`: Số ngày delay liên tiếp lớn nhất theo năm và hãng bay.
- Bảng `src_dest_canc_code`: Thống kê số lượng chuyến bay bị hủy theo lý do.

Ví dụ mã tạo bảng Delay_Data:

```
CREATE TABLE IF NOT EXISTS StreamingKeySpace.Delay_Data ("OP_CARRIER" text, "MONTH" bigint, "DEP_DELAY" float, "ARR_DELAY" float, primary key ("OP_CARRIER", "MONTH"));
```

1.2. Tiền xử lý dữ liệu

Tạo SparkSession và kết nối với cassandra (qua cổng localhost:9042')
Ví dụ tạo SparkSession và cấu hình:

```
#Tạo SparkSession và kết nối với cassandra
cluster_seeds = ['localhost:9042']

spark = SparkSession \
    .builder \
    .appName("Flight Batch Analysis") \
    .config("spark.cassandra.connection.host", ','.join(cluster_seeds)) \
    .config("spark.cassandra.auth.username", "cassandra") \
    .config("spark.cassandra.auth.password", "cassandra") \
    .getOrCreate()
```

Đọc dữ liệu các file csv chứa thông tin các chuyến bay qua các năm (từ 2009-2019) từ hdfs, loại bỏ các cột không có thông tin và đưa vào cache.
Ví dụ đọc dữ liệu từ hdfs:

```
df_2009 = spark.read.option("header", True).csv("hdfs://localhost:9000/input/2009.csv")
```

Gộp lại thành 1 file tổng hợp làm dữ liệu cho batch layer và file 2019 dùng làm dữ liệu cho streaming layer. Xóa các bản ghi trùng lặp và lưu vào hdfs.
Ví dụ gộp và lưu file:

```
df = df_2009.union(df_2010) \
    .unionByName(df_2011, allowMissingColumns=False) \
    .unionByName(df_2012, allowMissingColumns=False) \
    .unionByName(df_2013, allowMissingColumns=False) \
    .unionByName(df_2014, allowMissingColumns=False) \
    .unionByName(df_2015, allowMissingColumns=False) \
    .unionByName(df_2016, allowMissingColumns=False) \
    .unionByName(df_2017, allowMissingColumns=False) \
    .unionByName(df_2018, allowMissingColumns=False) \
    .unionByName(df_2019, allowMissingColumns=False)

df = df.dropna(how="all").dropDuplicates()
df.coalesce(1).write.mode('overwrite').option('header', 'true').csv('hdfs://localhost:9000/input/flight_data.csv')
```

1.3. Triển khai xử lý Batch data

Tạo SparkSession và kết nối với cassandra (qua 2 cổng localhost:9042)
Đọc dữ liệu từ file dữ liệu tổng hợp (ở phần tiền xử lý) từ hdfs.
Ví dụ:

```
df = spark.read.option("header", True).csv("hdfs://localhost:9000/input/flight_data.csv").cache()
```

Xử lý các batch data bằng cách dùng các transformation để trích xuất các thông tin:

- Trung bình độ trễ theo hãng bay và theo thời gian (ngày, tháng, năm)
- Trung bình độ trễ theo điểm đi và đến và theo thời gian
- Tỉ lệ hủy chuyến theo hãng bay và theo thời gian
- Tỉ lệ hủy chuyến theo điểm đi và đến và theo thời gian

- Trung bình khoảng cách bay theo hãng bay và theo thời gian
- Số ngày bị hoãn liên tiếp lớn nhất của từng chuyến bay
- Số lượng chuyến bay bị hủy theo từng tuyến bay

Ví dụ tính trung bình độ trễ theo hãng bay:

```
delay_total_df = df.select("OP_CARRIER", "DEP_DELAY", "ARR_DELAY") \
    .groupBy("OP_CARRIER") \
    .agg(avg("DEP_DELAY").alias("AVG_DEP_DELAY"), avg("ARR_DELAY").alias("AVG_ARR_DELAY")) \
    .select("OP_CARRIER", "AVG_DEP_DELAY", "AVG_ARR_DELAY")
```

Sau đó ghi dữ liệu vào các bảng tương ứng trên cassandra.
Ví dụ ghi dữ liệu độ trễ chuyến bay theo hãng bay:

```
delay_total_df.write \
    .format("org.apache.spark.sql.cassandra") \
    .option("keyspace", "batchkeyspace") \
    .option("table", "delay_total") \
    .mode("append") \
    .save()
```

2. Triển khai Speed layer

2.1.Triển khai kafka:

Dự án sử dụng cụm Kafka với hai broker và hai Zookeeper để quản lý dịch vụ Kafka, đảm bảo khả năng truyền dữ liệu từ producer đến các topic Kafka. Cụ thể:

- Hai zookeeper hoạt động trên các cổng 22181 và 32181 để quản lý cụm Kafka.
- Hai broker Kafka sử dụng các cổng 29092 và 39092 để kết nối với producer.

Các bước truyền dữ liệu:

- Tạo producer kafka và cầu hình chuyển đổi dữ liệu thành định dạng JSON và mã hóa nó sang UTF-8 trước khi gửi đến Kafka.
- Đọc file 2019.csv và chuyển từng dòng thành một dictionary, với các cột làm khóa
- Gửi dữ liệu từng dòng đến topic live-data của kafka và điều chỉnh tốc độ bằng cách mỗi khi gửi 20 dòng, chương trình sẽ dừng 10 giây.

Ví dụ:

```python
from kafka import KafkaProducer
import json, csv, time
# tạo producer để gửi dữ liệu
producer = KafkaProducer(
    bootstrap_servers='localhost:29092',
    value_serializer=lambda v: json.dumps(v).encode('utf-8')
)
# đọc file và gửi dữ liệu cho kafka mỗi 10s.
with open('../dataset/2019.csv') as file:
    reader = csv.DictReader(file, delimiter=",")
    index = 0
    for row in reader:
        print("sending data...")
        print(row)
        producer.send(topic='live-data', value=row)
        producer.flush()
        index += 1
        if (index % 20) == 0:
            time.sleep(10)
```

2.2. Triển khai xử lý streaming data

Tạo SparkSession kết nối địa chỉ node cassandra (node ở cổng 9042) và thiết lập việc đọc dữ liệu streaming (topic=live-data) từ Kafka thông qua địa chỉ của kafka broker (localhost:29092)

Ví dụ:

```python
cluster_seeds = ['localhost:9042']
# Khởi tạo sparksession để thực hiện biến đổi dữ liệu

spark = SparkSession \
    .builder \
    .appName("Flight Streaming Analysis") \
    .config("spark.cassandra.connection.host", ','.join(cluster_seeds)) \
    .config("spark.cassandra.auth.username", "cassandra") \
    .config("spark.cassandra.auth.password", "cassandra") \
    .getOrCreate()

# Đọc dữ liệu topic live-data từ kafka
df_kafka = spark \
    .readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "localhost:29092") \
    .option("subscribe", "live-data") \
    .option("startingOffsets", "earliest") \
    .option("failOnDataLoss", "false") \
    .load()
```

Đọc dữ liệu từ kafka theo các khóa của file json và trích xuất và tính các thông tin thời gian thực xuất phát, thời gian thực đến,thực hiện tính thời gian trễ trung bình theo tháng của từng hãng bay.

Ví dụ tính thời gian trễ theo tháng của từng hãng:

```
delay_df = df.select("OP_CARRIER", "FL_DATE", "DEP_DELAY", "ARR_DELAY") \
    .drop_duplicates() \
    .withColumn("MONTH", month("FL_DATE")) \
    .groupBy("OP_CARRIER", "MONTH") \
    .agg(avg("DEP_DELAY").alias("DEP_DELAY"), avg("ARR_DELAY").alias("ARR_DELAY")) \
    .select("OP_CARRIER", "MONTH", "DELAY")
```

Tạo `InfluxDB client` để kết nối đến cơ sở dữ liệu InfluxDB và viết các hàm lưu data ghi dữ liệu streaming vào InfluxDB để phục vụ tầng serving layers.
Ví dụ hàm lưu graph data:

```
def saveToInfluxGraph(row):
    timestamp = datetime.now()
    json_body = [
        {
            "measurement": "edges_graph",
            "tags": {
                "AIR_TIME": row["AIR_TIME"]
            },
            "time": timestamp,
            "fields": {
                "id": int(float(timestamp.timestamp()) * 1000000),
                "source": row["ORIGIN"],
                "target": row["DEST"],
                "mainStat": row["AIR_TIME"]
            }
        }
    ]
    client.write_points(json_body)
    json_body = [
        {
            "measurement": "nodes_graph",
            "tags": {
                "AIR_TIME": row["AIR_TIME"]
            },
            "time": timestamp,
            "fields": {
                "id": row["ORIGIN"],
            }
        }
    ]
    client.write_points(json_body)
    json_body = [
        {
            "measurement": "nodes_graph",
            "tags": {
                "AIR_TIME": row["AIR_TIME"]
            },
            "time": timestamp,
            "fields": {
                "id": row["DEST"],
            }
        }
    ]
    client.write_points(json_body)
```

Lưu dữ liệu trễ chuyến bay đã xử lý vào Cassandra để làm batch data.Thực hiện lưu theo từng batch nhỏ với thời gian xử lý là 10 giây.

Ví dụ:

```
delay_query = delay_df.writeStream \
    .trigger(processingTime="10 seconds") \
    .foreachBatch(writing) \
    .outputMode("update") \
    .start()
```

```
influx_query = df.writeStream \
    .foreach(saveToInflux) \
    .start()

graph_query = graph_df.writeStream \
    .foreach(saveToInfluxGraph) \
    .outputMode("update") \
    .start()

delay_query.awaitTermination()
influx_query.awaitTermination()
graph_query.awaitTermination()
```

3. Triển khai serving layer

3.1. Triển khai Grafana và InfluxDB bằng Docker Compose

Tệp cấu hình Docker Compose (“visualization/docker-compose.yml”) chứa các thông số để triển khai Grafana và InfluxDB.

Trong tệp này, phần khai báo volumes cần được tùy chỉnh để đường dẫn đến thư mục cấu hình Grafana phù hợp với máy chủ cục bộ.

Ví dụ cấu hình mã như sau:

```
volumes:
  - /home/luca/Progetti/ProgettoBigData/visualization/grafana/provisioning/:/etc/grafana/provisioning/
  - /etc/localtime:/etc/localtime:ro # Đồng bộ múi giờ hệ thống.
  - /etc/timezone:/etc/timezone:ro # Đồng bộ thông tin timezone.
```

Sau khi tùy chỉnh xong, chạy lệnh sau để khởi động Grafana và InfluxDB:

```
docker-compose up -d
```

3.2. Cấu hình Datasource và Dashboard

Datasource: Grafana được kết nối với InfluxDB thông qua tệp cấu hình `datasource.yml`.

Thông số truy cập (token, URL) của InfluxDB phải được cập nhật phù hợp khi triển khai thực tế. Ví dụ:

```yaml
apiVersion: 1  # Phiên bản API cấu hình datasource.
datasources:
- name: InfluxDB  # Tên của datasource.
  type: influxdb  # Loại datasource, ở đây là InfluxDB.
  access: proxy  # Chế độ truy cập qua proxy.
  orgId: 1  # ID tổ chức. Nếu không chỉ định, mặc định là 1.
  url: http://influx:8086  # URL của InfluxDB.
  user: user  # Tên người dùng cơ sở dữ liệu (nếu cần).
  password: pass  # Mật khẩu cơ sở dữ liệu.
  basicAuth: true  # Kích hoạt xác thực cơ bản.
  basicAuthUser: user  # Tên người dùng xác thực cơ bản.
  basicAuthPassword: pass  # Mật khẩu xác thực cơ bản.
  database: flight  # Tên cơ sở dữ liệu trong InfluxDB.
  jsonData:
    timeInterval: "10s"  # Khoảng thời gian mặc định cho dữ liệu.
  secureJsonData:
    password: grafana  # Mật khẩu được bảo mật cho datasource.

```

Dashboard: Tệp `flight_dashboard.json` định nghĩa bố cục biểu đồ, bao gồm:

- Dữ liệu chuyến bay.
- Xu hướng theo thời gian.
- Các chỉ số tổng hợp.

```yaml
apiVersion: 1  # Phiên bản API của cấu hình. Đảm bảo tương thích với Grafana.
providers:
- name: 'default'  # Tên provider mặc định.
  orgId: 1  # ID tổ chức. Nếu không chỉ định, giá trị mặc định là 1.
  folder: ''  # Tên thư mục trong Grafana. Để trống để dùng thư mục gốc.
  folderUid: ''  # UID thư mục. Sẽ tự tạo nếu không được chỉ định.
  type: file  # Provider loại 'file', tức là bảng điều khiển được lưu từ tệp YAML.
  disableDeletion: false  # Nếu true, không cho phép xóa bảng điều khiển.
  editable: true  # Nếu true, cho phép chỉnh sửa bảng điều khiển từ giao diện Grafana.
  updateIntervalSeconds: 300  # Khoảng thời gian (giây) Grafana quét thư mục để cập nhật.
  options:
    path: /etc/grafana/provisioning/dashboards  # Đường dẫn tới thư mục chứa bảng điều khiển.

```

3.3. Phân tích và Truy vấn dữ liệu bằng Jupyter Notebook

Tệp Notebook (“batch-analysis.ipynb”) cung cấp các bước:

1. Kết nối đến Cassandra bằng thư viện cassandra-driver.

```python
from cassandra.cluster import Cluster, PlainTextAuthProvider  # Import các công cụ kết nối với Cassandra
import pandas as pd  # Import thư viện pandas để xử lý dữ liệu dạng bảng
import matplotlib.pyplot as plt  # Import công cụ vẽ đồ thị cơ bản
import seaborn as sns  # Import thư viện seaborn để trực quan hóa dữ liệu

sns.set(rc={'figure.figsize':(20,10)})  # Cấu hình kích thước mặc định cho biểu đồ (20x10 inch)

# Xác thực thông tin đăng nhập Cassandra
auth_provider = PlainTextAuthProvider(username='cassandra', password='cassandra')

# Kết nối tới cụm Cassandra tại localhost qua cổng 9042
cluster = Cluster(["localhost"], port=9042, auth_provider=auth_provider)
```

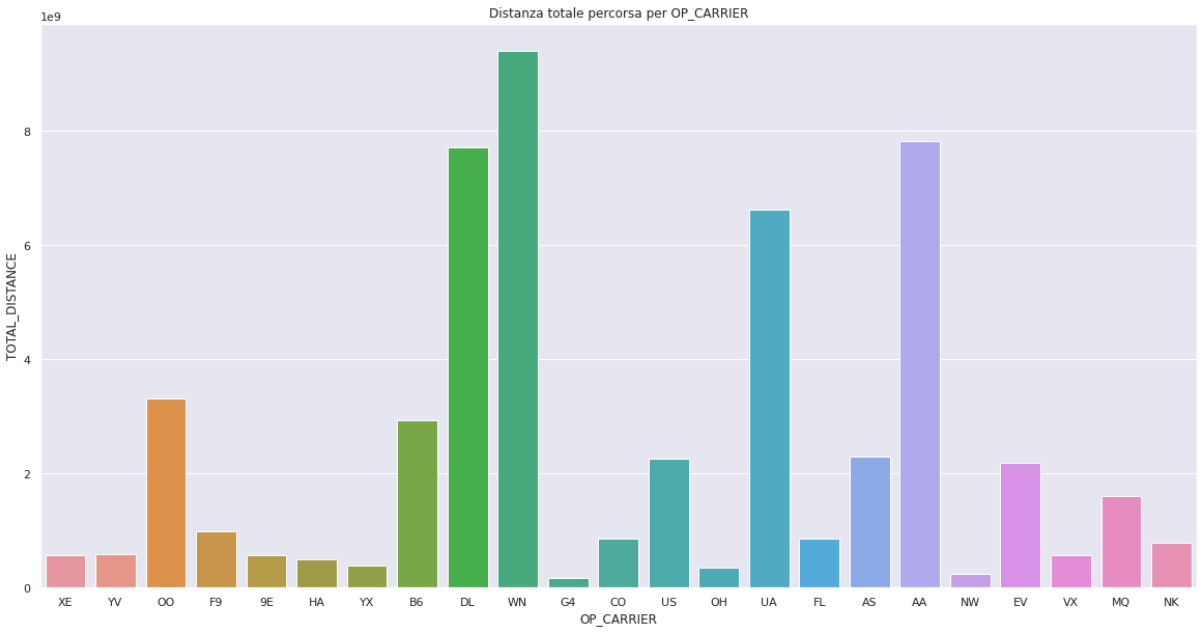
2. Truy vấn dữ liệu trong các bảng (đánh giá chuyến bay, thời gian bay, v.v.).

```
# Truy vấn dữ liệu về số lần chậm trễ liên tiếp lớn nhất trong mỗi năm
rows = session.execute("SELECT * FROM max_consec_delay_year")
max_consec_delay_year = rows._current_rows  # Lưu kết quả vào DataFrame

# Truy vấn dữ liệu về số lần chậm trễ liên tiếp lớn nhất theo nguồn và đích
rows = session.execute("SELECT * FROM max_consec_delay_year_src_dest")
max_consec_delay_year_src_dest = rows._current_rows  # Lưu kết quả vào DataFrame
```

3. Phân tích kết quả và tạo biểu đồ dùng pandas và matplotlib.

```
rows = session.execute("SELECT * FROM dist_total")
results = rows._current_rows
sns.barplot(data=results, x="OP_CARRIER", y="TOTAL_DISTANCE")
plt.title("Distanza totale percorsa per OP_CARRIER")
plt.show()
```



3.4. Truy cập và sử dụng Grafana

Truy cập Grafana tại địa chỉ: localhost:3000.

Tài khoản mặc định:

- User: `admin`
- Password: `admin123`

Trong giao diện, bạn có thể theo dõi biểu đồ cấu hình sẵn và thêm các phân tích khác theo nhu cầu.

IV. Kinh nghiệm

Kinh nghiệm xử lý dữ liệu thiếu và trùng lặp

Mô tả vấn đề

Context và background: Trong quá trình xử lý dữ liệu lớn, một số bản ghi không có đầy đủ thông tin cần thiết (missing data), và một số bản ghi bị trùng lặp hoàn toàn (duplicate data).

Thách thức gặp phải:

- Xác định cách xử lý dữ liệu thiếu mà không làm mất đi các thông tin hữu dụng.
- Loại bỏ các bản ghi trùng lặp để đảm bảo tính chính xác và hiệu quả của phân

Giải pháp đã thử

Approach 1: Xóa tất cả các bản ghi bị thiếu dữ liệu và dữ liệu trùng lặp:

- Ưu điểm: Đơn giản, đảm bảo dữ liệu sau xử lý gọn gàng và không bị lỗi.
- Nhược điểm: Dễ làm mất đi các thông tin quan trọng trong các bản ghi thiếu một phần dữ liệu.

Giải pháp cuối cùng

Chỉ xóa các bản ghi trùng lặp và giữ lại các bản ghi bị thiếu dữ liệu. Lý do chọn giải pháp:

- Dữ liệu trùng lặp gây ảnh hưởng tiêu cực đến kết quả phân tích, cần được loại bỏ.
- Các bản ghi bị thiếu dữ liệu có thể vẫn chứa thông tin quan trọng cho các trường không bị thiếu, cần được giữ lại.