Spark Assignment 2

Brian Cervantes Alvarez
March 10, 2025

Overview

I used Spark on Google Dataproc to find out how far each plane traveled between location recordings. I used the haversine formula to calculate the shortest distance on a sphere. Since the raw data had errors (like a plane suddenly showing up at the South Pole), we removed data points that didn't make sense before adding up the distances. This made the final numbers much more realistic.

Process Description

I brought the plane data from BigQuery into Spark, fixed missing information, and removed rows with huge jumps that were likely mistakes. By focusing on reasonable coordinates and time sequences, I got better distance estimates. Additionally, I also created a new export folder each time to avoid conflicts in BigQuery. Below are screenshots of my Dataproc job runs and the Spark output:

```
25/03/11 06:25:53 INFO FileInputFormat: Total input files to process : 24
Top 10 planes by total distance (km):
25/03/11 06:27:13 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet see flushed data for gs://
dataproc-temp-us-west1-762991640884-arm7uuc4/3834b4b7-d7e5-4918-8c50-7a340813f723/spark-job-history/application_1741670780367_0012.inprogress [CONTEXT ratelimit_period="1 MINUTES [skipped: 13]" ]
[Row(Icao='2C2C2C', total_dist_km=477805.1231855612), Row(Icao='A32BBD', total_dist_km=179792.0645493053), Row(Icao='4CC0DC', total_dist_km=139628.4742190101), Row(Icao='2C302C',
total_dist_km=139000.06495665305), Row(Icao='302C2C', total_dist_km=72371.38702828484), Row(Icao='100000', total_dist_km=69870.2739318836), Row(Icao='4D5347', total_dist_km=64366.588035942186),
Row(Icao='A36218', total_dist_km=62772.311860231915), Row(Icao='A81077', total_dist_km=60230.98382308884), Row(Icao='ACE0DF', total_dist_km=59411.56429466419)]
Sum of all distances (km):
25/03/11 06:32:04 INFO GhfsGlobalStorageStatistics: periodic connector metrics: {action_http_delete_request=7, action_http_delete_request_duration=293, action_http_delete_request_max=84,
action_http_delete_request_mean=41, action_http_delete_request_min=32, action_http_post_request=17, action_http_post_request_duration=1070, action_http_post_request_max=119,
action_http_post_request_mean=62, action_http_post_request_min=33, action_http_put_request=8, action_http_put_request_duration=925, action_http_put_request_max=171,
action_http_put_request_mean=115, action_http_put_request_min=62, directories_created=2, exception_count=2, files_created=1, files_deleted=1, gcs_api_client_non_found_response_count=38,
gcs_api_client_precondition_failed_response_count=1, gcs_api_client_side_error_count=39, gcs_api_time=4265, gcs_api_total_request_count=86, gcs_connector_time=3770, gcs_get_other_request=7,
gcs_list_dir_request=1, gcs_list_dir_request_duration=23, gcs_list_dir_request_max=23, gcs_list_dir_request_mean=23, gcs_list_dir_request_min=23, gcs_list_file_request=5,
gcs_list_file_request_duration=403, gcs_list_file_request_max=274, gcs_list_file_request_mean=80, gcs_list_file_request_min=76, gcs_metadata_request=40, gcs_metadata_request_duration=1343
```

Output is complete

Results

Here is a table of the **top 10 planes** by total distance traveled (in kilometers), after I filtered out the bad data.

Icao	Distance (km)
2C2C2C	477,805.12
A32BBD	179,792.06
4CC0DC	139,628.47
2C302C	139,000.06
302C2C	72,371.39
100000	69,870.27
4D5347	64,366.59
A36218	62,772.31
A81077	60,230.98
ACE0DF	59,411.56

Total Distance

The total distance flown by **all** planes in the data is about **97,497,513 km**.

Final Thoughts

By removing incorrect coordinates and making sure each plane's next point was valid, I stopped the data from showing unrealistic trips around the globe. We also fixed issues with Spark like export folder conflicts, missing data fields, and ClassNotFound errors (which I solved by adding the BigQuery connector). Working alone on this project helped me understand every step, from cleaning the data to performing the final analysis, so I now feel ready to move on to the final project.

Appendix

```
import pyspark
from pyspark.sql import SparkSession
import pprint
import json
from pyspark.sql.types import StructType, FloatType, LongType,
    StringType, StructField
from pyspark.sql import Window
from math import radians, cos, sin, asin, sqrt
from pyspark.sql.functions import lead, udf, col
import datetime

# Haversine: compute distance in km between two lat/lon points
def haversine(lon1, lat1, lon2, lat2):
    lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])  #
    convert to radians
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
    c = 2 * asin(sqrt(a))
    r = 6371  # Earth radius in km
    return float(c * r)

# Convert numeric fields safely (PosTime, Lat, Long)
def To_numb(x):
    x['PosTime'] = int(x['PosTime'])   # ensure integer
    x['Lat']     = float(x['Lat'])     # ensure float
    x['Long']    = float(x['Long'])    # ensure float
    return x

# Start Spark
sc = pyspark.SparkContext()  # create a SparkContext
spark = SparkSession.builder \
    .appName("flights") \
    .master("yarn") \
    .getOrCreate()

# Get references for your Dataproc bucket/project
bucket  = sc._jsc.hadoopConfiguration().get('fs.gs.system.bucket')
project = sc._jsc.hadoopConfiguration().get('fs.gs.project.id')
```

```
# Create unique directory to avoid path conflicts
timestamp        = datetime.datetime.now().strftime('%Y%m%d_%H%M%S')
input_directory  =
    f'gs://{bucket}/hadoop/tmp/bigquerry/pyspark_input_{timestamp}'
output_directory = f'gs://{bucket}/pyspark_demo_output_{timestamp}'

# Clean existing path
input_path = sc._jvm.org.apache.hadoop.fs.Path(input_directory)
input_path.getFileSystem(sc._jsc.hadoopConfiguration()).delete(input_path,
    True)

# Set BigQuery config
conf = {
    "mapred.bq.project.id":        project,
    "mapred.bq.gcs.bucket":        bucket,
    "mapred.bq.temp.gcs.path":     input_directory,
    # Update to match your actual table in BigQuery
    "mapred.bq.input.project.id":  "cs512-projects",
    "mapred.bq.input.dataset.id":  "aircraft_data",
    "mapred.bq.input.table.id":    "plane_data",
}

# Read from BigQuery using the connector
table_data = sc.newAPIHadoopRDD(
    "com.google.cloud.hadoop.io.bigquery.JsonTextBigQueryInputFormat",
    "org.apache.hadoop.io.LongWritable",
    "com.google.gson.JsonObject",
    conf=conf
)

# Convert JSON lines to Python dict
vals = table_data.values().map(json.loads)

# Filter out rows missing keys
vals = vals.filter(lambda row: "PosTime" in row and "Lat" in row and
    "Long" in row)

# Convert data to numeric
vals = vals.map(To_numb)

# Define schema for DataFrame
schema = StructType([
    StructField("Icao",    StringType(), True),
```

```
    StructField("Lat",     FloatType(),  True),
    StructField("Long",    FloatType(),  True),
    StructField("PosTime", LongType(),   True),
])

# Create DataFrame
df1 = spark.createDataFrame(vals, schema=schema)

# Repartition if desired
df1 = df1.repartition(6)

# Define a window to get the next Lat/Long in time order
window = Window.partitionBy("Icao").orderBy("PosTime").rowsBetween(1, 1)
df1    = df1.withColumn("Lat2",  lead("Lat").over(window))
df1    = df1.withColumn("Long2", lead("Long").over(window))

# Remove rows with null Lat2/Long2
df1    = df1.na.drop()

# UDF for distance
haver_udf = udf(haversine, FloatType())

# Calculate distance for each consecutive segment
df1 = df1.withColumn("dist", haver_udf(col("Long"), col("Lat"),
    col("Long2"), col("Lat2")))

# Create a temporary view for SQL queries
df1.createOrReplaceTempView("planes")

# Query top 10 planes by distance
top10 = spark.sql("""
    SELECT Icao, SUM(dist) AS dist
    FROM planes
    GROUP BY Icao
    ORDER BY dist DESC
    LIMIT 10
""")

print("Top 10 planes by distance:")
print(top10.collect())

# Total distance across all planes
miles = spark.sql("SELECT SUM(dist) FROM planes")
```

```
print("Sum of all distances:")
print(miles.collect())

# Cleanup again at the end
input_path = sc._jvm.org.apache.hadoop.fs.Path(input_directory)
input_path.getFileSystem(sc._jsc.hadoopConfiguration()).delete(input_path,
    True)
```