Design of an Apache Spark Structured Streaming application for near real time processing of invoice data for RetailCorp Inc.

Atharv Darekar

12th August 2022

Contents

1 Reading Data From Apache Kafka Topic

1.1 Analyze Incoming Data

The data read from the Kafka topic is in JSON format. A sample data looks as given below:


```
{
        "items":
                [
                        {
                                "SKU": "21485",
                                "title": "RETROSPOT HEART HOT WATER BOTTLE",
                                "unit_price": 4.95,
                                "quantity": 6
                        },
                        {
                                "SKU": "23499",
                                "title": "SET 12 VINTAGE DOILY CHALK",
                                "unit_price": 0.42,
                                "quantity": 2
                        }
                ],
        "type": "ORDER",
        "country": "United Kingdom",
        "invoice_no": 154132541653705,
        "timestamp": "2020-09-18 10:55:23"
}
```


1.2 Define Schema

The schema is defined for the given data as follows:

```
14 invoice_schema = StructType(
15     [
16         StructField(name="items", dataType=StringType(),
           ↪  nullable=True),
17         StructField(name="type", dataType=StringType(),
           ↪  nullable=True),
18         StructField(name="country", dataType=StringType(),
           ↪  nullable=True),
19         StructField(name="invoice_no", dataType=LongType(),
           ↪  nullable=True),
20         StructField(name="timestamp", dataType=TimestampType(),
           ↪  nullable=True)
21     ]
22 )
```

1.3 Create A Streaming DataFrame

Using the schema, the data is flatten and stored in a dataframe.

```
invoices = spark \
    .readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "18.211.252.152:9092") \
    .option("subscribe", "real-time-project") \
    .load().selectExpr("cast(value as string)") \
    .select(from_json(col("value").cast("string"),
    ↪  invoice_schema).alias("invoice")) \
    .select(col("invoice.*")) \
    .select("invoice_no", "country", "timestamp", "type",
    ↪  "items")
```

2 Enrichment Of Invoice Data

2.1 Approach

As in Apache Spark version 2.1.2, multiple streaming aggregations are not yet supported on streaming dataframe and datasets. Considering the limitations, various user define functions are utilized to calculate metrics necessary for further processing of data.

2.2 Define User Defined Functions

2.2.1 Total Cost

The total cost of an order is sum of product of unit price and quantity:

```
def calculate_total_cost(orders):
    orders = literal_eval(orders)
    total_cost = 0
    for order in orders:
        total_cost += order["unit_price"] * order["quantity"]
    return total_cost
```

2.2.2 Total Items

The total units in an order are calculated as sum of quantity of each item in the order:

```
def sum_items(orders):
    orders = literal_eval(orders)
    total_orders = 0
    for order in orders:
        total_orders += order["quantity"]
    return total_orders
```

2.2.3 Order Type

Based on weather the order is a new order or a return order, the invoice is flagged separately for each type:

```
51 def is_order(order_type):
52     if order_type == "ORDER":
53         return 1
54     return 0
55
56
57 def is_return(order_type):
58     if order_type == "RETURN":
59         return 1
60     return 0
```

2.3 Create PySpark User Defined Functions

```
63 calculate_total_cost_udf = udf(lambda orders:
   ↪  calculate_total_cost(orders))
64 sum_items_udf = udf(lambda orders: sum_items(orders))
65 is_order_udf = udf(lambda order_type: is_order(order_type))
66 is_return_udf = udf(lambda order_type: is_return(order_type))
```

2.4 Enrich Invoice Data

A new order is credited where as a return order is debited from the total cost of an order.

```
68 invoices_enriched = invoices \
69     .withColumn("is_order", is_order_udf("type")) \
70     .withColumn("is_return", is_return_udf("type")) \
71     .withColumn("total_cost",
72                 when(col("is_return") == 1, -1 *
                    ↪  calculate_total_cost_udf("items"))
73                 .otherwise(calculate_total_cost_udf("items"))
74                 ) \
75     .withColumn("total_items", sum_items_udf("items")) \
76     .select("invoice_no", "country", "timestamp", "total_cost",
       ↪  "total_items", "is_order", "is_return")
```

2.5 Write The Enriched Invoice Data To Console

The invoice data is processed for every one minute interval and the stream is written to the console in append mode.

```
97 write_invoices_enriched = invoices_enriched \
98     .writeStream \
99     .outputMode("append") \
100    .format("console") \
101    .option("truncate", "false") \
102    .trigger(processingTime="1 minute") \
103    .start()
```

3 Calculate Key Performance Indicators

3.1 Total Volume of Sales

The total volume of sales in a time interval can be calculated as the summation of the transaction values of all the orders in that time interval.

$$\sum (quantity \times unitprice) \tag{1}$$

3.2 Orders Per Minute

Orders per minute (OPM) is the total number of orders received in a minute.

$$\sum \text{orders per minute} \tag{2}$$

3.3 Rate Of Return

The rate of return is calculated against the total number of invoices using the following equation:

$$\frac{\sum \text{return orders}}{\sum orders} \tag{3}$$

3.4 Average Transaction Size

This can be calculated using the following equation:

$$\frac{\text{total volume of sales}}{\sum orders} \tag{4}$$

3.5 Calculate Time Based KPIs

The above four KPIs are calculated on a tumbling window of one minute on orders across the globe.

```
87 time_based_kpis = invoices_enriched \
88     .withWatermark("timestamp", "1 minute") \
89     .groupBy(window("timestamp", "1 minute")) \
90     .agg(pyspark_round(pyspark_sum("total_cost"),
       ↪  2).alias("total_sales_volume"),
91          count("invoice_no").alias("OPM"),
92          pyspark_round(
93              pyspark_sum("is_return") / (pyspark_sum("is_return")
                ↪  + pyspark_sum("is_order")),
94              2).alias("rate_of_return"),
95          pyspark_round(pyspark_sum("total_cost") / "OPM",
            ↪  2).alias("average_transaction_size"))
```

3.6 Calculate Time And Country Based KPIs

Except the average transaction size, the rest three KPIs are calculated on a tumbling window of one minute and per country basis.

```
time_country_based_kpis = invoices_enriched \
    .withWatermark("timestamp", "1 minute") \
    .groupBy(window("timestamp", "1 minute"), "country") \
    .agg(pyspark_round(pyspark_sum("total_cost"),
    ↪  2).alias("total_sales_volume"),
         count("invoice_no").alias("OPM"),
         pyspark_round(
             pyspark_sum("is_return") / (pyspark_sum("is_return")
             ↪  + pyspark_sum("is_order")),
             2).alias("rate_of_return"))
```

3.7 Write The KPIs to JSON Files

The calculated KPIs are written to JSON files for a one minute window.

```
write_time_based_kpis = time_based_kpis \
    .writeStream \
    .outputMode("append") \
    .format("json") \
    .option("truncate", "false") \
    .option("path", "/user/livy/calculate_kpis/time_based_kpis")
    ↪  \
    .option("checkpointLocation",
    ↪  "hdfs:///user/livy/calculate_kpis/time_based_kpis/checkpoints")
    ↪  \
    .trigger(processingTime="1 minute") \
    .start()

write_time_country_based_kpis = time_country_based_kpis \
    .writeStream \
    .outputMode("append") \
    .format("json") \
    .option("truncate", "false") \
    .option("path",
    ↪  "/user/livy/calculate_kpis/time_country_based_kpis") \
    .option("checkpointLocation",
    ↪  "hdfs:///user/livy/calculate_kpis/time_country_based_kpis/checkpoints")
    ↪  \
    .trigger(processingTime="1 minute") \
    .start()
```

4 Run The Spark Streaming Application

To prepare the environment and run the Spark Streaming Application, following bash script is to be used:

```
#!/usr/bin/env bash

# create directories to save KPI data and checkpoints in Apache Hadoop Distributed File System

hdfs dfs -mkdir -p /user/livy/calculate_kpis/time_based_kpis/checkpoints
hdfs dfs -mkdir -p /user/livy/calculate_kpis/time_country_based_kpis/checkpoints

# run the Spark Streaming application
# write or overwrite the standard output stream to console-output.txt

spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.11:2.4.5 spark-streaming.py | tee Console-output.txt
```

5 Copy The KPI Data From Apache HDFS To Local Path

To copy and compress the various data generated by the application, following bash script is to be used:

```
#!/usr/bin/env bash

# create directories to copy KPI data from Apache Hadoop Distributed File System (HDFS)

mkdir -p {~/calculate_kpis/time_based_kpis,~/calculate_kpis/time_country_based_kpis}

# copy data from HDFS to local directories

hdfs dfs -copyToLocal /user/livy/calculate_kpis/time_based_kpis/*.json ~/calculate_kpis/time_based_kpis
hdfs dfs -copyToLocal /user/livy/calculate_kpis/time_country_based_kpis/*.json ~/calculate_kpis/time_country_based_kpis

# pack the data into a Output.zip file

zip ~/Output.zip ./console-output.txt -r ./calculate_kpis
```