Case study 4

Optimised Real-Time Data Processing

- with Caching, Persisting, and Broadcasting

1. Aim:

To design and implement a high-performance data pipeline for processing, aggregating, and storing Walmart sales data in real-time using Apache Spark, Kafka, and Google Cloud Storage (GCS). The focus is on optimising performance through caching, persisting, and broadcasting techniques while ensuring data validation, enrichment, and efficient storage.

2. Objectives:

- **Real-Time Data Ingestion:** Implement a streaming pipeline to consume weekly sales data from Kafka.
- **Data Validation:** Clean the incoming sales data by removing negative values and handling missing or invalid values.
- **Data Enrichment:** Enhance the dataset by joining it with additional metadata from features.csv and stores.csv.
- **Aggregation:** Calculate store-level and department-level sales metrics.
- **Optimised Storage:** Store the processed data efficiently in GCS using Parquet (for structured data) and JSON (for aggregated metrics).
- **Performance Optimisation:** Use caching for intermediate datasets, broadcasting for small datasets, and persisting for fault tolerance.
- **Real-Time Simulation:** Simulate continuous real-time data processing and aggregation.
- **Visualisation:** Visualise key metrics and data at various stages.

3. Datasets Involved:

- **Sales Data (train.csv)**: Contains weekly sales data for various stores and departments.
- **Features Data (features.csv)**: Provides metadata like holiday flags and regional data.
- **Stores Data (stores.csv)**: Includes metadata for stores, such as region and type.

4. Tools & Technologies:

- **Apache Spark** (Scala and Spark SQL for data processing)
- **Apache Kafka** (for real-time data streaming)
- **Google Cloud Storage (GCS)** (for storing processed data)
- **Parquet** (for efficient data storage)
- **JSON** (for storing aggregated data)
- **Spark Streaming** (for real-time data processing)

5. Procedure:

Step 1: Environment Setup

- Set up the Spark session for local execution or cluster execution.
- Configure the connection to Kafka for real-time data consumption.
- Configure the connection to Google Cloud Storage for storing results.

Step 2: Data Loading

- Load the features.csv, stores.csv, and the streaming data from Kafka into Spark DataFrames.

```
val featuresDF = spark.read.option("header", "true").csv(featuresPath)
val storesDF = spark.read.option("header", "true").csv(storesPath)
val kafkaStreamDF = spark.readStream.format("kafka").option("subscribe",
                    topic).load()
```

Step 3: Data Validation and Enrichment

- Filter out negative sales values and handle missing data.
- Enrich the sales data with features.csv and stores.csv using left joins.
- Cache the featuresDF and storesDF DataFrames to optimize repeated usage during joins.

```
val validDF = parsedDF.filter("Weekly_Sales >= 0")
val enrichedDF = validDF.join(F.broadcast(featuresDF), "Store")
                        .join(F.broadcast(storesDF), "Store")
enrichedDF.cache()  // Cache enriched data for future use
```

Step 4: Data Aggregation

- Calculate store-level and department-level aggregation metrics (e.g., total sales, average sales).

```
val storeMetricsDF = enrichedDF.groupBy("Store")
                                .agg(
                               sum("Weekly_Sales").alias("Total_Weekly_Sales"),
                               avg(“Weekly_Sales").alias("Avg_Weekly_Sales"),
                               max("Weekly_Sales").alias("Top_Store_Sales")
                                )

val departmentMetricsDF = enrichedDF.groupBy("Store", "Department")
                               .agg(
                               sum("Weekly_Sales").alias("Total_Sales"),
                               avg("Weekly_Sales").alias("Avg_Sales")
                                )
```

Step 5: Data Storage

- Write the processed store-level and department-level metrics to GCS in Parquet format for efficient storage.

```
storeMetricsDF.writeStream
        .outputMode("append")
        .format("parquet")
        .option("checkpointLocation", checkpointLocation)
        .option("path", storeLevelPath)
        .trigger(Trigger.ProcessingTime("10 seconds"))
        .start()

departmentMetricsDF.writeStream
        .outputMode("append")
        .format("parquet")
        .option("checkpointLocation", checkpointLocation)
        .option("path", departmentLevelPath)
        .trigger(Trigger.ProcessingTime("10 seconds"))
        .start()
```

6. Output Visualization:

- **Real-Time Ingestion Output:** Display Kafka-provided data (weekly sales) from the kafkaStreamDF.
- **Aggregated Output:** Show results of store-level and department-level metrics using .show().
- **Time-Based Windows:** If windowed aggregations are implemented, visualise the output for each batch processed.

7. Data Persistence:

- Store enriched data and aggregated metrics in Parquet format for long-term storage and analysis.
- Store aggregated metrics in JSON format for efficient, lightweight storage.

8. Performance Metrics:

- Monitor data ingestion speed and processing throughput in real time.
- Display metrics related to Kafka offsets, batch processing times, and state management during the execution of the streaming job.

Here are the screenshots of the project deliverables:

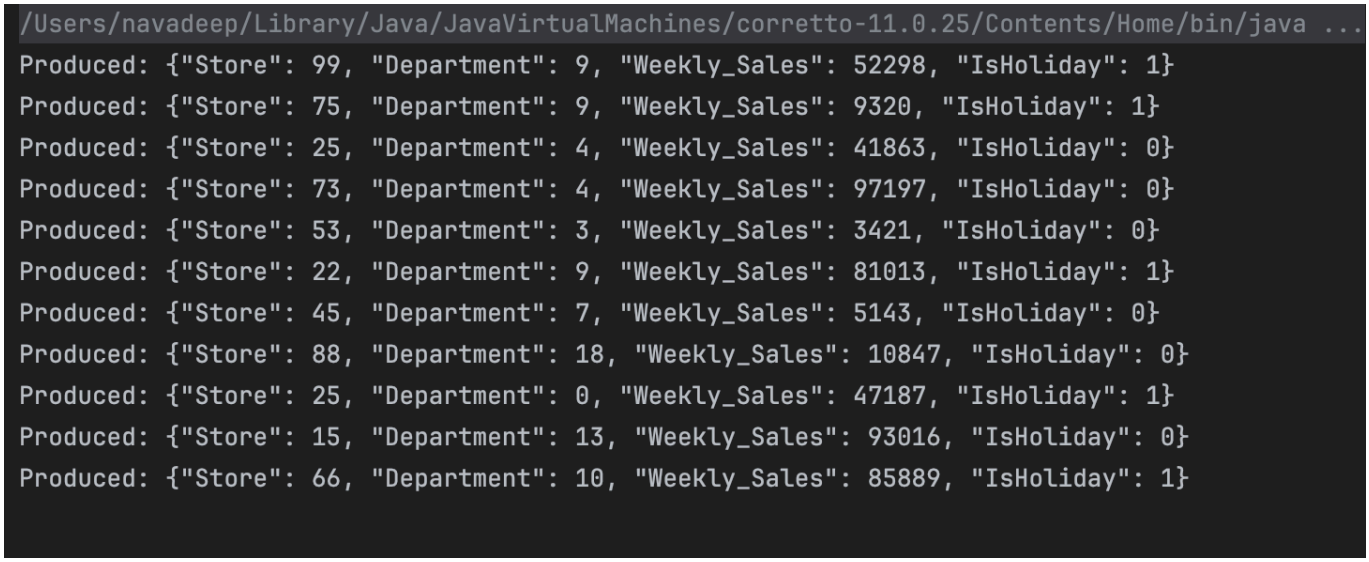

Fig: Data from producer being produced.

```
+------------------------------------------------------------------------+
|json_string                                                             |
+------------------------------------------------------------------------+
|{"Store": 20, "Department": 0, "Weekly_Sales": 908, "IsHoliday": 0}    |
|{"Store": 59, "Department": 12, "Weekly_Sales": 49147, "IsHoliday": 0}|
|{"Store": 18, "Department": 1, "Weekly_Sales": 6489, "IsHoliday": 1}  |
|{"Store": 4, "Department": 5, "Weekly_Sales": 65223, "IsHoliday": 0}  |
|{"Store": 89, "Department": 17, "Weekly_Sales": 81438, "IsHoliday": 0}|
|{"Store": 72, "Department": 7, "Weekly_Sales": 15564, "IsHoliday": 1} |
|{"Store": 23, "Department": 11, "Weekly_Sales": 7484, "IsHoliday": 0} |
|{"Store": 88, "Department": 14, "Weekly_Sales": 96948, "IsHoliday": 1}|
|{"Store": 14, "Department": 15, "Weekly_Sales": 93048, "IsHoliday": 1}|
|{"Store": 78, "Department": 18, "Weekly_Sales": 2860, "IsHoliday": 1} |
|{"Store": 48, "Department": 18, "Weekly_Sales": 29238, "IsHoliday": 1}|
|{"Store": 82, "Department": 19, "Weekly_Sales": 16642, "IsHoliday": 1}|
|{"Store": 7, "Department": 17, "Weekly_Sales": 71999, "IsHoliday": 1} |
|{"Store": 2, "Department": 16, "Weekly_Sales": 34726, "IsHoliday": 0} |
|{"Store": 11, "Department": 12, "Weekly_Sales": 66733, "IsHoliday": 1}|
|{"Store": 95, "Department": 3, "Weekly_Sales": 20296, "IsHoliday": 1} |
|{"Store": 72, "Department": 18, "Weekly_Sales": 5776, "IsHoliday": 0} |
|{"Store": 50, "Department": 0, "Weekly_Sales": 87388, "IsHoliday": 1} |
|{"Store": 12, "Department": 7, "Weekly_Sales": 84760, "IsHoliday": 0} |
|{"Store": 44, "Department": 15, "Weekly_Sales": 43714, "IsHoliday": 0}|
+------------------------------------------------------------------------+
only showing top 20 rows
```

```
-------------------------------------------
Batch: 26
-------------------------------------------
+-----+----------+------------+---------------+----+-----------------+-----------+---------------------+
|Store|Department|Weekly_Sales|Kafka_IsHoliday|Type|Features_IsHoliday|Stream_Date|Stream_Date_Timestamp|
+-----+----------+------------+---------------+----+-----------------+-----------+---------------------+
|29   |11        |98872       |1              |B   |false            |NULL       |NULL                 |
|29   |11        |98872       |1              |B   |false            |NULL       |NULL                 |
|29   |11        |98872       |1              |B   |false            |NULL       |NULL                 |
|29   |11        |98872       |1              |B   |false            |NULL       |NULL                 |
|29   |11        |98872       |1              |B   |false            |NULL       |NULL                 |
|29   |11        |98872       |1              |B   |false            |NULL       |NULL                 |
|29   |11        |98872       |1              |B   |false            |NULL       |NULL                 |
|29   |11        |98872       |1              |B   |false            |NULL       |NULL                 |
|29   |11        |98872       |1              |B   |false            |NULL       |NULL                 |
|29   |11        |98872       |1              |B   |false            |NULL       |NULL                 |
|29   |11        |98872       |1              |B   |false            |NULL       |NULL                 |
|29   |11        |98872       |1              |B   |false            |NULL       |NULL                 |
|29   |11        |98872       |1              |B   |false            |NULL       |NULL                 |
|29   |11        |98872       |1              |B   |false            |NULL       |NULL                 |
|29   |11        |98872       |1              |B   |false            |NULL       |NULL                 |
|29   |11        |98872       |1              |B   |false            |NULL       |NULL                 |
|29   |11        |98872       |1              |B   |false            |NULL       |NULL                 |
|29   |11        |98872       |1              |B   |false            |NULL       |NULL                 |
|29   |11        |98872       |1              |B   |false            |NULL       |NULL                 |
|29   |11        |98872       |1              |B   |false            |NULL       |NULL                 |
+-----+----------+------------+---------------+----+-----------------+-----------+---------------------+
only showing top 20 rows
```

Fig: Data from consumer