STA323

Big Data Analysis Software and Application (Hadoop or Spark)

Report on Assignment 3

12112627 李乐平

Question 1. Use the data of /shareddata/data/flights/departuredelays.csv . Complete the following task using Spark RDD API.

(1). [2 point] Partition the data according to the origin columns. Put the rows where origin is ATL into one partition, while the rest rows are randomly partitioned into other three partitions.

Answer:

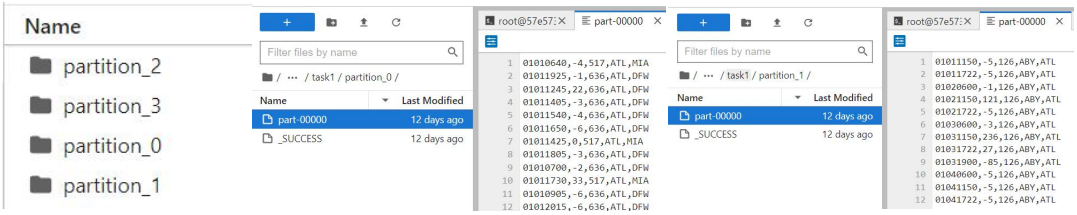
首先读进.csv 文件，根据 origin 列是否为"ATL"划分类别。如果 origin=="ATL"则划为 0 类，其余按 hash(row) % 3 + 1 划为 1-3 类。最后保存划分即可。

```
sc = SparkContext("local", "partitioning")
data_rdd = sc.textFile("./data/task1/departuredelays.csv")
header = data_rdd.first()
data_rdd = data_rdd.filter(lambda line: line != header)

def custom_partition(row):
    origin = row.split(',')[3]
    if origin == 'ATL':
        return 0
    else:
        return hash(origin) % 3 + 1

partitioned_rdd = data_rdd.map(lambda line: (custom_partition(line), line))

for i in range(4):
    partitioned_rdd.filter(lambda x: x[0] == i) \
        .values().saveAsTextFile(f"./output/task1/partition_{i}")
```



Question 2. Use the data in /shareddata/data/activity-data/ path. Complete the following tasks using Spark Streaming API.

(1). [2 points] Read the streaming data from json files. Set the partitions of the streaming data as 3 with Spark configuration. Count the streaming data grouped by user with an event-time (**Creation_Time**) window of 6 minutes. The overlap of adjacent windows is **3 minutes**. Write the streaming data output into **memory** with **update** mode. Name the query as **activity_query** and configure the checkpoint location path. Display the first 3 query results of **activity_query**.

Answer:

```
streaming_df = spark.readStream \
    .format("json") \
    .option("path", "./data/task2/activity-data/") \
    .schema("Arrival_Time long, Creation_Time long, Device string, Index long, Model string, User string,
gt string, x double, y double, z double") \
    .load()
window_query = streaming_df.withColumn("Creation_Time_seconds", col("Creation_Time") / 1000000000) \
    .withColumn("Creation_Time_timestamp", from_unixtime(col("Creation_Time_seconds"))) \
```

```
    .groupBy("User", window("Creation_Time_timestamp", "6 minute", "3 minute")).count() \
    .writeStream.queryName("activity_query").format("memory").outputMode("update").trigger(processing
Time="2 seconds") \
    .option("checkpointLocation", "./checkpoint") \
    .start()
```

注意 Creation_Time 到时间戳的处理即可。

```
+----+-----------------------------------------+-----+
|User|window                                   |count|
+----+-----------------------------------------+-----+
|b   |{2015-02-24 14:21:00, 2015-02-24 14:27:00}|64907|
|e   |{2015-02-24 15:15:00, 2015-02-24 15:21:00}|61783|
|c   |{2015-02-23 13:06:00, 2015-02-23 13:12:00}|55610|
+----+-----------------------------------------+-----+
only showing top 3 rows
```

(2). [1 point] For the above task, update your code to write the query results simultaneously into **memory** and **parquet** files with **append** mode.

Answer:

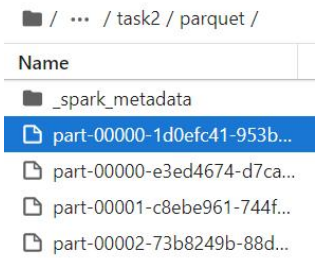
在上一题的基础上，修改模式为 append。为保证顺序一致性，使用 foreachBatch 处理，在其中，将数据输出到 parquet 中，并复制到内存中，使之能够访问。

```
df_updated = None

def write_to_memory_and_parquet(df, epoch_id):
    global df_updated
    if df_updated is None:
        df_updated = df.toDF("User", "window", "count")
    else:
        df_updated = df_updated.union(df.toDF("User", "window", "count"))
    df.persist()
    df.write.format("parquet").mode("append").save("./output/task2/parquet")

window_query_updated = streaming_df.withColumn("Creation_Time_seconds", col("Creation_Time") /
1000000000) \
    .withColumn("Creation_Time_timestamp", from_unixtime(col("Creation_Time_seconds"))) \
    .groupBy("User", window("Creation_Time_timestamp", "6 minute", "3 minute")).count() \
    .writeStream \
    .queryName("activity_query") \
    .outputMode("update") \
    .foreachBatch(write_to_memory_and_parquet) \
    .trigger(processingTime='2 seconds') \
    .option("checkpointLocation", "./ckpt_parquet/") \
    .start()
```

```
+----+-----------------------------------------+-----+
|User|window                                   |count|
+----+-----------------------------------------+-----+
|g   |{2015-02-23 10:45:00, 2015-02-23 10:51:00}|62201|
|g   |{2015-02-23 10:15:00, 2015-02-23 10:21:00}|18846|
|g   |{2015-02-23 10:21:00, 2015-02-23 10:27:00}|57165|
```



Question 3. Data is in /shareddata/data/shopping_data folder. This is a user log that records the actions of users in online shopping. In the user log,

The 'action' column is the user action

0 represents clicking the page, 1 represents adding the item to the cart, 2 is to buy the item, 3 represents adding to the wish list.

The 'gender' column is the gender of the user

0 represents female, 1 represents male.

(1). [3 points] Build a Kafka pipeline to generate the streaming data from the files /shareddata/data/shopping_data/user_log.csv. The kafka topic is named as **'q3'**, which contains the action and gender of each message in the log. The message sending interval is set as **0.5s**. Use Spark Streaming to read the kafka streaming data and extract the data whose **action** value is **2**. Count the number of transactions of both female and male with a windows size of 5 seconds (the sliding interval of the window is 0). The watermark delay is 10 seconds. Write the results into the memory with the view **queryName=transaction_count** with complete mode. Display the first 20 rows.

Answer:

首先使用 bash 脚本启动 kafka 以及 producer 脚本。

```
echo "Now start the services needed for spark + kafka. "

## Step 1: open a tmux window to run the zookeeper service
workloc="/opt/module/kafka/kafka_2.13-3.7.0"
$workloc/bin/zookeeper-server-start.sh $workloc/config/zookeeper.properties

## Step 2: open another tmux window to run the kafka service
workloc="/opt/module/kafka/kafka_2.13-3.7.0"
$workloc/bin/kafka-server-start.sh $workloc/config/server.properties
# $workloc/bin/kafka-topics.sh --list --bootstrap-server localhost:9092
```

在 q3_producer.py 中，.csv 文件被读入，并每隔约 0.5 秒发送 1000 行至 Kafka 流中。

```
import csv
import time
from kafka import KafkaProducer

broker = "localhost:9092"
producer = KafkaProducer(bootstrap_servers = broker, retries = 5)

csv_file = "./data/task3/shopping_data/user_log.csv"
with open(csv_file, "r", encoding = "utf-8") as file:
    reader = csv.reader(file, delimiter = ",")
    cnt = 0
    for row in reader:
        producer.send("q3", (",".join(row)).encode("utf-8"))
        producer.flush()
        cnt += 1
        if cnt % 1000 == 0:
            time.sleep(0.5)

producer.close()
```

然后通过 readStream 从消息队列中读取并处理数据即可。

```
q3_df = spark.readStream.format("kafka")\
    .option("kafka.bootstrap.servers", "localhost:9092")\
    .option("subscribe", "q3") \
    .option("startingOffsets", "earliest") \
    .load()

streaming_df = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "localhost:9092") \
    .option("subscribe", "q3") \
    .option("failOnDataLoss", "false") \
```

```
    .load()
streaming_df = streaming_df.selectExpr("CAST(value AS STRING)")

parsed_df = streaming_df.selectExpr("split(value, ',') as columns") \
    .selectExpr("columns[7] as action", "columns[9] as gender")

filtered_df = parsed_df.filter("action = '2' and (gender = '0' or gender = '1')")

watermark_delay = "10 seconds"

windowed_df = filtered_df \
    .withColumn("timestamp", expr("current_timestamp()")) \
    .withWatermark("timestamp", watermark_delay) \
    .groupBy(window("timestamp", "5 seconds"), "gender") \
    .count()

query = windowed_df.writeStream \
    .format("memory") \
    .outputMode("complete") \
    .queryName("transaction_count") \
    .option("checkpointLocation", "./q3_checkpoint") \
    .start()
```

```
+-----------------------------------------+------+-----+
|window                                   |gender|count|
+-----------------------------------------+------+-----+
|{2024-05-05 03:10:00, 2024-05-05 03:10:05}|0     |161  |
|{2024-05-05 03:14:30, 2024-05-05 03:14:35}|0     |119  |
|{2024-05-05 03:18:00, 2024-05-05 03:18:05}|0     |133  |
|{2024-05-05 03:17:45, 2024-05-05 03:17:50}|1     |150  |
|{2024-05-05 03:12:40, 2024-05-05 03:12:45}|0     |104  |
|{2024-05-05 03:12:30, 2024-05-05 03:12:35}|1     |117  |
|{2024-05-05 03:13:55, 2024-05-05 03:14:00}|1     |107  |
|{2024-05-05 03:11:20, 2024-05-05 03:11:25}|0     |150  |
|{2024-05-05 03:10:25, 2024-05-05 03:10:30}|0     |165  |
|{2024-05-05 03:11:35, 2024-05-05 03:11:40}|0     |153  |
|{2024-05-05 03:11:50, 2024-05-05 03:11:55}|1     |137  |
|{2024-05-05 03:13:00, 2024-05-05 03:13:05}|0     |137  |
|{2024-05-05 03:09:15, 2024-05-05 03:09:20}|0     |117  |
|{2024-05-05 03:14:00, 2024-05-05 03:14:05}|1     |149  |
|{2024-05-05 03:17:50, 2024-05-05 03:17:55}|0     |126  |
|{2024-05-05 03:13:40, 2024-05-05 03:13:45}|0     |149  |
|{2024-05-05 03:13:25, 2024-05-05 03:13:30}|1     |129  |
|{2024-05-05 03:12:45, 2024-05-05 03:12:50}|0     |130  |
|{2024-05-05 03:09:30, 2024-05-05 03:09:35}|0     |149  |
|{2024-05-05 03:09:40, 2024-05-05 03:09:45}|1     |153  |
+-----------------------------------------+------+-----+
```

(2). [1 point] Explain the logging mechanism and implementation of Kafka.

Answer:

Kafka 中的日志机制是一种持久化存储消息的方式，用于处理流式数据。每个主题的日志由多个分区组成，每个分区对应一个目录，其中包含了存储消息的数据文件。日志文件的格式是一系列「日志条目」，每个日志条目由一个存储消息长度的 4 字节整数 N 和紧随其后的长度为 N 的消息字节组成。每条消息都由一个 64 位整数偏移量唯一标识，该偏移量指示了该消息在该分区上所有发送到该主题的消息流中的字节位置。日志文件的命名以其包含的第一条消息的偏移量命名。日志支持顺序追加写入，当日志文件达到可配置的大小上限时会自动滚动到一个新文件。写入操作可以配置强制刷新到磁盘的条件，从而提供了一定程度的数据持久性保证。

对于读取操作，可以通过给定消息的 64 位逻辑偏移量和最大字节块大小来进行。读取会返回一个迭代器，遍历返回在指定字节缓冲区中的消息。如果消息过大，可以进行多次重

试，每次将缓冲区大小加倍，直到成功读取消息为止。此外，可以指定最大消息和缓冲区大小，以拒绝过大的消息，并为客户端提供读取完整消息所需的最大缓冲区大小。

日志提供了获取最近写入消息的能力，允许客户端从「现在」开始订阅消息。此外，日志还提供了删除数据的机制，可以根据时间和大小两种策略进行删除。日志还提供了一些保证措施，如通过配置参数控制最大写入消息数量以及在启动时运行日志恢复过程来验证消息的有效性，并在检测到数据损坏时进行修复。