SPARK STRUCTURED STREAMING - FILE-TO-FILE REAL-TIME STREAMING (3/3)

CSV FILE TO JSON FILE REAL TIME STREAMING EXAMPLE

JUNE 28, 2018

PAVAN KULKARNI

6 MINUTE READ

In this post we will see how to build a simple application to process file to file real time processing.

Most of the clients I have worked with so far still rely on files - either CSV, TSV or JSON. These applications are

batch processes to some kind of streaming process that can provide realtime data processing.

We will see in this post how to process data from a CSV file to JSON file in realtime.

Set up to this is similar to all our previous Spark Examples.

LET'S BUILD A USE CASE

We have 2 directories,

- `src/main/resources/input/cutomer_info` which contains a static file with Customer information
- `src/main/resources/input/orders` in which CSV files with order details are dropped periodically. This directory is partitioned date-wise as shown below

```
Pavans-MacBook-Pro:Spark_Streaming_Examples
Pavans-MacBook-Pro:input pavanpkulkarni$ ls
cutomer_info:
total 8
-rw-r--r--  1 pavanpkulkarni  staff  58 Jun

orders:
total 0
drwxr-xr-x  3 pavanpkulkarni  staff  96 Jun
drwxr-xr-x  3 pavanpkulkarni  staff  96 Jun
drwxr-xr-x  3 pavanpkulkarni  staff  96 Jun
drwxr-xr-x  3 pavanpkulkarni  staff  96 Jun
```

Objective here is to join the order details from CSV file with the customer information file, and write the resulting data to JSON file as output in real-time.

LET'S TALK SCALA !

1. We have customer information is as follows

```
Pavans-MacBook-Pro:input pavanpkulkarni$ ca
id,name,location
1,kash,VT
2,pavan,IL
3,john,CA
4,jane,NJ
Pavans-MacBook-Pro:input pavanpkulkarni$
```

2. Sample CSV data with order information is :

```
Pavans-MacBook-Pro:input pavanpkulkarni$ ca
id,pid,pname,date
1,011,p11,2018-06-01
2,012,p12,2018-06-01
1,012,p12,2018-06-01
2,023,p23,2018-06-01
2,034,p34,2018-06-01
3,034,p34,2018-06-01
```

3. Now we begin by initializing Spark context

```
    //initialize the spark session
        val spark = SparkSession
```

```
                .appName("File_Streaming")
                .getOrCreate()
```

4. In order to stream data from CSV file, we need to define a schema for the data. Spark will not allow streaming of CSV data, unless the schema is defined.

```
val schema = StructType(
            Array(StructField("customer_id",
                StructField("pid", StringTyp
                StructField("pname", StringT
                StructField("date", StringTy

//stream the orders from the csv files.
val ordersStreamDF = spark
            .readStream
            .option("header", "true")
            .schema(schema)
            .csv(args(0))
```

5. Read the customer information from the static file and store it in a static dataset

```
case class Customer(customer_id : String, cu

import spark.implicits._

//get the static customer data
val customerDS = spark.read
            .format("csv")
            .option("header", true)
            .load("src/main/resources/input/
            .map(x => Customer(x.getString(0
```

field.

```
val finalResult = ordersStreamDF.join(custor
```

The resultant dataframe is now a streaming dataframe containing the resultant aggregation.

7. Using the above stremaing dataframe we can write data to any source supported by Spark

```
//write the joined stream to json/parquet ou
val query = finalResult
            .writeStream
            .queryName("count_customer")
            //.format("console")
            .outputMode("append")
            .format("json")
            .partitionBy("date")
            .option("path", "src/main/resou
            .option("checkpointLocation", "
            .start()
```

Here,

- ***format("console")*** : can be used for debugging purpose by printing the values on console.
- ***outputMode("append")***.format("json") : Write the output in append mode to JSON files.
- ***partitionBy("date")*** : The output is partitioned date-wise. Partitioning data is one of the good strategies to adopt for performance improvement.
- ***option("path", "src/main/resources/output/")*** : Specify output path to dump the data as JSON files.

- ***option("checkpointLocation", "src/main/resources/chkpoint_dir")*** : Specify a path for checkpoint directory for fault tolerance.

The full code can be found in my Github Repo

LET'S GET STREAMING STARTED !

1. Run the project as Scala project in IDE.
2. Once the streaming job starts, you will see the `chkpoint_dir` and `output` directories created.

```
Pavans-MacBook-Pro:resources pavanpkulkarni$
total 0
drwxr-xr-x  4 pavanpkulkarni  staff  128 Jul
drwxr-xr-x  8 pavanpkulkarni  staff  256 Jul
drwxr-xr-x  7 pavanpkulkarni  staff  224 Jul
```

3. The `output` directory will now contain partitioned subdirectories based off of the `date` field in schema.

```
Pavans-MacBook-Pro:resources pavanpkulka
total 0
drwxr-xr-x  4 pavanpkulkarni  staff  128
drwxr-xr-x  4 pavanpkulkarni  staff  128
drwxr-xr-x  4 pavanpkulkarni  staff  128
drwxr-xr-x  4 pavanpkulkarni  staff  128
drwxr-xr-x  4 pavanpkulkarni  staff  128
drwxr-xr-x  3 pavanpkulkarni  staff   96
```

> ***/path/to/input/key=value*** *so that we achieve desired partitioning. Here key is the field name from schema (date in our demo) and value will be the values of that column.*

4. Let's look at the data for partition `date=2018-06-01` .

```
Pavans-MacBook-Pro:resources pavanpkulkarni$
total 16
-rw-r--r--  1 pavanpkulkarni  staff   16 Jul
-rw-r--r--  1 pavanpkulkarni  staff  567 Jul
drwxr-xr-x  8 pavanpkulkarni  staff  256 Jul
drwxr-xr-x  4 pavanpkulkarni  staff  128 Jul

Pavans-MacBook-Pro:resources pavanpkulkarni$
{"customer_id":"1","pid":"011","pname":"p11"
{"customer_id":"2","pid":"012","pname":"p12"
{"customer_id":"1","pid":"012","pname":"p12"
{"customer_id":"2","pid":"023","pname":"p23"
{"customer_id":"2","pid":"034","pname":"p34"
{"customer_id":"3","pid":"034","pname":"p34"
```

As seen here, the JSON file is aggregated value of both the static customer information and the orders information.

BONUS ADVANTAGES OF THIS APPLICATION

The application does not stop here. It keeps getting awesome !!

data loss. Let's go ahead add new data file in the `input/orders/date\=2018-06-01` directory.

```
Pavans-MacBook-Pro:resources pavanpkulkarni
total 16
-rw-r--r--  1 pavanpkulkarni  staff  144 Ju
-rw-r--r--  1 pavanpkulkarni  staff   79 Ju

Pavans-MacBook-Pro:resources pavanpkulkarni
id,pid,pname,date
2,012,p34,2018-06-01
3,003,p3,2018-06-01
4,004,p4,2018-06-01
```

As soon as the new file is detected by the Spark engine, the streaming job is initiated and we can see the JSON file almost immediately. The most awesome part is that, a new JSON file will be created in the same partition.

```
Pavans-MacBook-Pro:resources pavanpkulkarni$ ls
total 32
-rw-r--r--  1 pavanpkulkarni  staff   16 Jul  6
-rw-r--r--  1 pavanpkulkarni  staff  567 Jul  6
-rw-r--r--  1 pavanpkulkarni  staff   12 Jul  6
-rw-r--r--  1 pavanpkulkarni  staff  281 Jul  6
drwxr-xr-x  8 pavanpkulkarni  staff  256 Jul  6
drwxr-xr-x  6 pavanpkulkarni  staff  192 Jul  6
```

Checkpoint directory maintains the state of the engine and processes the new files from there on. By doing so, we can avoid re-running if the job for every new file or late file arrival. Another advantage is that the Spark engine will stay idle until the data arrives. Thus saving

All the files can be viewed in my GitHub Repo

REFERENCES:

1. https://spark.apache.org/docs/latest/streaming-programming-guide.html
2. https://spark.apache.org/docs/latest/rdd-programming-guide.html

DATA ENGINEERING

BLOG