assignment08_StreamingData_MeyerJake

May 7, 2023

0.1 Assignment 8 Streaming Data Feed Notebook

0.2 DSC 650

0.3 Professor Iranitalab

0.4 Jake Meyer

0.5 05/06/2023

0.5.1 Code help from the following two examples provided this week:

Example Kafka Consumer Example Kafka Producer

0.5.2 Import Essential Packages/Modules/Libraries

[1]:
```
## The following libraries/packages/modules are useful for completing the␣
 ↪assignment.
import numpy as np
import pandas as pd
import pyarrow as pa
import pyarrow.parquet as parq
import json
from json import dumps
import uuid
import threading
import decimal
import shutil
import os
import glob
import time
import sys
from time import sleep
from kafka import KafkaProducer, KafkaAdminClient, KafkaConsumer, TopicPartition
from kafka.errors import TopicAlreadyExistsError, KafkaError
from kafka.admin.new_topic import NewTopic
import dask.dataframe as dd
from pathlib import Path
from datetime import datetime, timedelta
from pyspark.sql.types import *
```

```
from pyspark.sql import SparkSession
```

[2]:
```
## Create Spark Session as previously outlined in assignment 4. Use Spark Session Builder.
spark = SparkSession.builder \
    .master('local') \
    .appName('parquetFile') \
    .config('spark.executor.memory', '5gb') \
    .config('spark.cores.max', '6') \
    .getOrCreate()
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
setLogLevel(newLevel).
23/05/07 04:19:17 WARN NativeCodeLoader: Unable to load native-hadoop library
for your platform… using builtin-java classes where applicable
23/05/07 04:19:18 WARN Utils: Service 'SparkUI' could not bind on port 4040.
Attempting port 4041.
23/05/07 04:19:18 WARN Utils: Service 'SparkUI' could not bind on port 4041.
Attempting port 4042.
```

[3]:
```
## Configure Spark Session to resolve errors that were surfacing.
spark.conf.set("spark.sql.execution.arrow.pyspark.enabled", "true")
spark.conf.set("spark.rapids.sql.format.parquet.read.enabled", "true")
spark.conf.set("spark.rapids.sql.format.parquet.write.enabled", "true")
spark.conf.set("spark.rapids.sql.format.parquet.reader.type=MULTITHREADED", "true")
```

0.5.3 Define Directories

[4]:
```
cwd = os.getcwd()
cwd
```

[4]:
```
'/home/jovyan/dsc650/dsc650/assignments/assignment08'
```

[5]:
```
current_dir = Path('/home/jovyan/dsc650/dsc650/assignments/assignment08/')
results_dir = current_dir.joinpath('results')
```

[6]:
```
## Setup begin datetime, time, and interval.
begin_datetime = datetime.now()
begin_time = time.time()
interval = 0.1
```

0.5.4 Load Parquet

[7]:
```
'''
Argument accepted is results_dir.
Loaded data will be moved to partitioned folders within the results section.
'''

def loadParquet(results_dir):
    p_file = results_dir.joinpath('routes.parquet')

    stream_dir = results_dir.joinpath('stream')
    stream_dir.mkdir(parents=True, exist_ok=True)

    input_dir = stream_dir.joinpath('input')
    input_dir.mkdir(parents=True, exist_ok=True)

    acceleration_input_parquet_dir = input_dir.joinpath('accelerations')
    acceleration_input_parquet_dir.mkdir(parents=True, exist_ok=True)

    ## Clear the Input Folders
    try:
        shutil.rmtree(acceleration_input_parquet_dir)
    except OSError as e:
        print("Error: %s : %s" % (file_path, e.strerror))
    acceleration_input_parquet_dir.mkdir(parents=True, exist_ok=True)

    output_dir = stream_dir.joinpath('output')
    output_dir.mkdir(parents=True, exist_ok=True)

    staging_dir = stream_dir.joinpath('staging')

    ## Use CSV for troubleshooting...
    csv_file = results_dir.joinpath('routes_parquet.csv')
    data_dir = Path('/home/jovyan/dsc650/dsc650/assignments/assignment08//').
 ↪joinpath('Data')

    # Read using spark
    pq = pd.read_parquet(p_file, engine='fastparquet')

    # Reduce df.
    ## Create seconds with time partitions.
    pq['group_1'] = np.arange(len(pq)) // 600
    pq['group_10'] = pq['group_1'].astype(str)
    pq['group_2'] = pq['group_10'].apply(lambda x: x.zfill(3))
    pq['group_3'] = np.random.randint(0, 10, pq.shape[0])
    pq['t'] = pq['group_2'] + "." + pq['group_3'].astype(str)
    pq['tid'] = pq['t']
```

```
pq['airline_alias'] = pq['airline.alias']
pq['airline_callsign'] = pq['airline.callsign']
pq['airline_country'] = pq['airline.country']
pq['airline_iata'] = pq['airline.iata']
pq['airline_icao'] = pq['airline.icao']
pq['airline_name'] = pq['airline.name']
pq['dst_airport_name'] = pq['dst_airport.name']
pq['src_airport_name'] = pq['src_airport.name']

## Cleanup df with drop().
pq = pq.drop('group_1', axis = 1)
pq = pq.drop('group_10', axis = 1)
pq = pq.drop('group_2', axis = 1)
pq = pq.drop('group_3', axis = 1)
pq = pq.drop('airline.active', axis = 1)
pq = pq.drop('airline.airline_id', axis = 1)
pq = pq.drop('airline.alias', axis = 1)
pq = pq.drop('airline.callsign', axis = 1)
pq = pq.drop('airline.country', axis = 1)
pq = pq.drop('airline.iata', axis = 1)
pq = pq.drop('airline.icao', axis = 1)
pq = pq.drop('airline.name', axis = 1)
pq = pq.drop('codeshare', axis = 1)
pq = pq.drop('equipment', axis = 1)
pq = pq.drop('src_airport.airport_id', axis = 1)
pq = pq.drop('src_airport.altitude', axis = 1)
pq = pq.drop('src_airport.city', axis = 1)
pq = pq.drop('src_airport.country', axis = 1)
pq = pq.drop('src_airport.dst', axis = 1)
pq = pq.drop('src_airport.iata', axis = 1)
pq = pq.drop('src_airport.icao', axis = 1)
pq = pq.drop('src_airport.latitude', axis = 1)
pq = pq.drop('src_airport.longitude', axis = 1)
pq = pq.drop('src_airport.name', axis = 1)
pq = pq.drop('src_airport.source', axis = 1)
pq = pq.drop('src_airport.timezone', axis = 1)
pq = pq.drop('src_airport.type', axis = 1)
pq = pq.drop('src_airport.tz_id', axis = 1)
pq = pq.drop('dst_airport.airport_id', axis = 1)
pq = pq.drop('dst_airport.altitude', axis = 1)
pq = pq.drop('dst_airport.city', axis = 1)
pq = pq.drop('dst_airport.country', axis = 1)
pq = pq.drop('dst_airport.dst', axis = 1)
pq = pq.drop('dst_airport.iata', axis = 1)
pq = pq.drop('dst_airport.icao', axis = 1)
pq = pq.drop('dst_airport.latitude', axis = 1)
pq = pq.drop('dst_airport.longitude', axis = 1)
```

```
    pq = pq.drop('dst_airport.name', axis = 1)
    pq = pq.drop('dst_airport.source', axis = 1)
    pq = pq.drop('dst_airport.timezone', axis = 1)
    pq = pq.drop('dst_airport.type', axis = 1)
    pq = pq.drop('dst_airport.tz_id', axis = 1)

    ## Remove rows with missing values.
    pq = pq.dropna(how='any')
    pq = pq.drop(pq[pq.airline_alias == '\\N'].index)
    pq = pq.drop(pq[pq.airline_alias == 'nan'].index)

    ## Created For troubleshooting dataframe contents.
    pq.to_csv(csv_file, sep='\t')
    table = pa.Table.from_pandas(pq)

    ## Use write_to_dataset for new partitioned parquet dataset.
    parq.write_to_dataset(
        table,
        root_path=acceleration_input_parquet_dir,
        partition_cols=['t'],
    )

    ## Clear staging folder.
    staging_dir.mkdir(parents=True, exist_ok=True)
    try:
        shutil.rmtree(staging_dir)
    except OSError as e:
        print("Error: %s : %s" % (file_path, e.strerror))
```

[8]:
```
def beginParquetStreamUpdateLoop(results_dir):
    ## Write the updated parquet with spark
    ## Define stream, input, and acceleration input parquet directories.
    stream_dir = results_dir.joinpath('stream')
    input_dir = stream_dir.joinpath('input')
    acceleration_input_parquet_dir = input_dir.joinpath('accelerations')

    ## Define staging directory.
    staging_dir = stream_dir.joinpath('staging')
    staging_dir.mkdir(parents=True, exist_ok=True)

    ## Staging directories for acceleration and location.
    acceleration_staging_parquet_dir = staging_dir.joinpath('accelerations')
    acceleration_staging_parquet_dir.mkdir(parents=True, exist_ok=True)
    locations_staging_parquet_dir = staging_dir.joinpath('locations')
    locations_staging_parquet_dir.mkdir(parents=True, exist_ok=True)
    std = (time.time() - begin_time)
    before, after = str(std).split('.')
```

```
   time_value_parquet = "t=" + before.zfill(3) + "." + after[:1]

   ## Setup read directory for acceleration input.
   acceleration_input_parquet_read_dir = acceleration_input_parquet_dir.
↪joinpath(time_value_parquet)

   ## Code to check availability and convert dask df to pandas df.
   checkavail = os.path.exists(acceleration_input_parquet_read_dir)
   if checkavail == True:
       print("Processing partition: " + time_value_parquet + "from: ")
       print(acceleration_input_parquet_read_dir)
       updated_p_file = acceleration_staging_parquet_dir.
↪joinpath(time_value_parquet)

       dataset = dd.read_parquet(acceleration_input_parquet_read_dir)
       print("Before offset column add")
       print(type(dataset))

       ## Convert dask df to pandas df.
       dataset = dataset.compute()
       print(type(dataset))

       dataset['t']=dataset['tid']
       dataset = dataset.drop('tid', axis = 1)

       print(dataset)
       print(dataset.shape)
       print(dataset.ndim)
       print(dataset.dtypes)
       print(dataset.head())

       ## Schema
       print(dataset.info())

       ## Create DateTimeOffset dataframe column
       secval= float(before +"."+ after)
       print("These Seconds")
       print(secval)
       dataset["DateTimeOffset"]=begin_datetime + timedelta(seconds=secval)

       print("After offset column add")
       print(dataset.head())


       print("Writing to staging parquet folder")

       ## Create Spark Dataframe and 1 partition parquet
```

```
        mySchema = StructType([StructField('airline_alias', StringType(), True)
                               , StructField('airline_callsign', StringType(),␣
↪True)
                               , StructField('airline_country', StringType(),␣
↪True)
                               , StructField('airline_iata', StringType(), True)
                               , StructField('airline_icao', StringType(), True)
                               , StructField('airline_name', StringType(), True)
                               , StructField('dst_airport_name', StringType(),␣
↪True)
                               , StructField('src_airport_name', StringType(),␣
↪True)
                               , StructField('t', StringType(), True)
                               , StructField('DateTimeOffset', TimestampType(),␣
↪True)
                                ])
        print(mySchema)
        dataset = spark.createDataFrame(dataset, schema=mySchema)
        dataset.show()
        dataset.printSchema()

        print(updated_p_file)
        print(updated_p_file)
        print(acceleration_staging_parquet_dir)

        parquetwritepath = '/home/jovyan/dsc650/dsc650/assignments/assignment08/
↪results/stream/staging/accelerations/' + time_value_parquet
        dataset.repartition(1).write.parquet(parquetwritepath)
        print("Spark Written Parquet Complete")

        ## Move from the staging to the input folder
        print("Moving from Staging Folder to Input folder")


        ## Get name of the parquet file to swap with part.0.parquet
        source_dir = updated_p_file
        target_dir = acceleration_input_parquet_read_dir
        targetparqfilenames = os.listdir(target_dir)
        for file_name in targetparqfilenames:
            targetparqfilename=file_name
        print(targetparqfilename)
        print(updated_p_file)


        ## Get the parquet file name using glob
```

```
        globpath = '/home/jovyan/dsc650/dsc650/assignments/assignment08/results/
↪stream/staging/accelerations/' + time_value_parquet+'/*.parquet'
        for fname in glob.glob(globpath):
            source_filename = fname
        print(source_filename)

        ## Move the parquet file
        shutil.move(os.path.join(source_dir, source_filename), os.path.
↪join(target_dir, targetparqfilename))

        ## Move the rest of the files.
        file_names = os.listdir(source_dir)
        for file_name in file_names:
            shutil.move(os.path.join(source_dir, file_name), target_dir)

    ## Return if the input folder was not available.
        retval = 0
    else:
        retval = 1
        print("Load Partition Does Not Exist: ",␣
↪acceleration_input_parquet_read_dir)
    return retval
```

[9]:
```
def beginTimer(results_dir):
    ## Create a loop with time
    print("Call Function Here")
    retval = beginParquetStreamUpdateLoop(results_dir)

    ## Stop if time is over and there are no more partitions.
    if ((time.time() - begin_time) < 70 and retval == 0):
        t = threading.Timer(interval, startTimer(results_dir))
```

[10]:
```
## Run the loadParquet() function.
loadParquet(results_dir)
```

[11]:
```
## Run the beginTimer() function.
beginTimer(results_dir)
```

```
Call Function Here
Load Partition Does Not Exist:  /home/jovyan/dsc650/dsc650/assignments/assignmen
t08/results/stream/input/accelerations/t=011.1
```