Assignment 9.3Sutow Brett

July 19, 2021

0.1 Assignment 9.3

[1]:
```
import os
import shutil
import json
from pathlib import Path

import pandas as pd

from kafka import KafkaProducer, KafkaAdminClient
from kafka.admin.new_topic import NewTopic
from kafka.errors import TopicAlreadyExistsError

from pyspark.sql import SparkSession
from pyspark.streaming import StreamingContext
from pyspark import SparkConf
from pyspark.sql.functions import window, from_json, col, expr, to_json,␣
 ↪struct, when
from pyspark.sql.types import StringType, TimestampType, DoubleType,␣
 ↪StructField, StructType
from pyspark.sql.functions import udf

current_dir = Path(os.getcwd()).absolute()
checkpoint_dir = current_dir.joinpath('checkpoints')
joined_checkpoint_dir = checkpoint_dir.joinpath('joined')

if joined_checkpoint_dir.exists():
    shutil.rmtree(joined_checkpoint_dir)

joined_checkpoint_dir.mkdir(parents=True, exist_ok=True)
```

0.1.1 Configuration Parameters

TODO: Change the configuration prameters to the appropriate values for your setup.

[2]:
```
config = dict(
    bootstrap_servers=['kafka.kafka.svc.cluster.local:9092'],
    first_name='Brett',
```

```
    last_name='Sutow'
)

config['client_id'] = '{}{}'.format(
    config['last_name'],
    config['first_name']
)
config['topic_prefix'] = '{}{}'.format(
    config['last_name'],
    config['first_name']
)

config['locations_topic'] = '{}-locations'.format(config['topic_prefix'])
config['accelerations_topic'] = '{}-accelerations'.
 ↪format(config['topic_prefix'])
config['joined_topic'] = '{}-joined'.format(config['topic_prefix'])

config
```

[2]:
```
{'bootstrap_servers': ['kafka.kafka.svc.cluster.local:9092'],
 'first_name': 'Brett',
 'last_name': 'Sutow',
 'client_id': 'SutowBrett',
 'topic_prefix': 'SutowBrett',
 'locations_topic': 'SutowBrett-locations',
 'accelerations_topic': 'SutowBrett-accelerations',
 'joined_topic': 'SutowBrett-joined'}
```

0.1.2 Create Topic Utility Function

The `create_kafka_topic` helps create a Kafka topic based on your configuration settings. For instance, if your first name is *John* and your last name is *Doe*, `create_kafka_topic('locations')` will create a topic with the name `DoeJohn-locations`. The function will not create the topic if it already exists.

[3]:
```
def create_kafka_topic(topic_name, config=config, num_partitions=1,␣
 ↪replication_factor=1):
    bootstrap_servers = config['bootstrap_servers']
    client_id = config['client_id']
    topic_prefix = config['topic_prefix']
    name = '{}-{}'.format(topic_prefix, topic_name)

    admin_client = KafkaAdminClient(
        bootstrap_servers=bootstrap_servers,
        client_id=client_id
    )
```

```
    topic = NewTopic(
        name=name,
        num_partitions=num_partitions,
        replication_factor=replication_factor
    )

    topic_list = [topic]
    try:
        admin_client.create_topics(new_topics=topic_list)
        print('Created topic "{}"'.format(name))
    except TopicAlreadyExistsError as e:
        print('Topic "{}" already exists'.format(name))

create_kafka_topic('joined')
```

```
Topic "SutowBrett-joined" already exists
```

TODO: This code is identical to the code used in 9.1 to publish acceleration and location data to the `LastnameFirstname-simple` topic. You will need to add in the code you used to create the `df_accelerations` dataframe. In order to read data from this topic, make sure that you are running the notebook you created in assignment 8 that publishes acceleration and location data to the LastnameFirstname-simple topic.

[5]:
```
spark = SparkSession\
    .builder\
    .appName("Assignment09")\
    .getOrCreate()

df_locations = spark \
  .readStream \
  .format("kafka") \
  .option("kafka.bootstrap.servers", "kafka.kafka.svc.cluster.local:9092") \
  .option("subscribe", config['locations_topic']) \
  .load()

## TODO: Add code to create the df_accelerations dataframe
df_accelerations = spark \
  .readStream \
  .format("org.apache.spark.sql.kafka010.KafkaSourceProvider") \
  .option("kafka.bootstrap.servers", "localhost:9092") \
  .option("subscribe", config['accelerations_topic']) \
  .load()
```

The following code defines a Spark schema for location and acceleration data as well as a user-defined function (UDF) for parsing the location and acceleration JSON data.

[6]:
```
location_schema = StructType([
    StructField('offset', DoubleType(), nullable=True),
```

```
    StructField('id', StringType(), nullable=True),
    StructField('ride_id', StringType(), nullable=True),
    StructField('uuid', StringType(), nullable=True),
    StructField('course', DoubleType(), nullable=True),
    StructField('latitude', DoubleType(), nullable=True),
    StructField('longitude', DoubleType(), nullable=True),
    StructField('geohash', StringType(), nullable=True),
    StructField('speed', DoubleType(), nullable=True),
    StructField('accuracy', DoubleType(), nullable=True),
])

acceleration_schema = StructType([
    StructField('offset', DoubleType(), nullable=True),
    StructField('id', StringType(), nullable=True),
    StructField('ride_id', StringType(), nullable=True),
    StructField('uuid', StringType(), nullable=True),
    StructField('x', DoubleType(), nullable=True),
    StructField('y', DoubleType(), nullable=True),
    StructField('z', DoubleType(), nullable=True),
])

udf_parse_acceleration = udf(lambda x: json.loads(x.decode('utf-8')),␣
 ↪acceleration_schema)
udf_parse_location = udf(lambda x: json.loads(x.decode('utf-8')),␣
 ↪location_schema)
```

TODO:

- Complete the code to create the `accelerationsWithWatermark` dataframe.
  - Select the `timestamp` field with the alias `acceleration_timestamp`
  - Use the `udf_parse_acceleration` UDF to parse the JSON values
  - Select the `ride_id` as `acceleration_ride_id`
  - Select the `x`, `y`, and `z` columns
  - Use the same watermark timespan used in the `locationsWithWatermark` dataframe

[7]:
```
locationsWithWatermark = df_locations \
  .select(
    col('timestamp').alias('location_timestamp'),
    udf_parse_location(df_locations['value']).alias('json_value')
   ) \
  .select(
    col('location_timestamp'),
    col('json_value.ride_id').alias('location_ride_id'),
    col('json_value.speed').alias('speed'),
    col('json_value.latitude').alias('latitude'),
    col('json_value.longitude').alias('longitude'),
    col('json_value.geohash').alias('geohash'),
    col('json_value.accuracy').alias('accuracy')
```

```
  ) \
 .withWatermark('location_timestamp', "2 seconds")

accelerationsWithWatermark = df_accelerations \
  .select(
    col('timestamp').alias('acceleration_timestamp'),
    udf_parse_acceleration(df_accelerations['value']).alias('json_value')
   ) \
  .select(
    col('acceleration_timestamp'),
    col('json_value.ride_id').alias('acceleration_ride_id'),
    col('json_value.x').alias('x'),
    col('json_value.y').alias('y'),
    col('json_value.z').alias('z')
  ) \
 .withWatermark('acceleration_timestamp', "2 seconds")
```

TODO:

- Complete the code to create the `df_joined` dataframe. See http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#stream-stream-joins for additional information.

[8]:
```
df_joined = locationsWithWatermark.join(
    accelerationsWithWatermark,
    expr("""
        location_ride_id = acceleration_ride_id
        """)
)
df_joined
```

[8]:
```
DataFrame[location_timestamp: timestamp, location_ride_id: string, speed:
double, latitude: double, longitude: double, geohash: string, accuracy: double,
acceleration_timestamp: timestamp, acceleration_ride_id: string, x: double, y:
double, z: double]
```

If you correctly created the `df_joined` dataframe, you should be able to use the following code to create a streaming query that outputs results to the `LastnameFirstname-joined` topic.

[9]:
```
ds_joined = df_joined \
  .withColumn(
    'value',
    to_json(
        struct(
            'ride_id', 'location_timestamp', 'speed',
            'latitude', 'longitude', 'geohash', 'accuracy',
            'acceleration_timestamp', 'x', 'y', 'z'
        )
```

```
    )
    ).withColumn(
     'key', col('ride_id')
    ) \
  .selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)") \
  .writeStream \
  .format("kafka") \
  .option("kafka.bootstrap.servers", "kafka.kafka.svc.cluster.local:9092") \
  .option("topic", config['joined_topic']) \
  .option("checkpointLocation", str(joined_checkpoint_dir)) \
  .start()

try:
    ds_joined.awaitTermination()
except KeyboardInterrupt:
    print("STOPPING STREAMING DATA")
```

```
---------------------------------------------------------------------------
AnalysisException                         Traceback (most recent call last)
<ipython-input-9-72756735dc4c> in <module>
----> 1 ds_joined = df_joined \
      2   .withColumn(
      3     'value',
      4     to_json(
      5         struct(

/opt/conda/lib/python3.8/site-packages/pyspark/sql/dataframe.py in␣
↪withColumn(self, colName, col)
   2453         """
   2454         assert isinstance(col, Column), "col should be Column"
-> 2455         return DataFrame(self._jdf.withColumn(colName, col._jc), self.
↪sql_ctx)
   2456
   2457     def withColumnRenamed(self, existing, new):

/opt/conda/lib/python3.8/site-packages/py4j/java_gateway.py in __call__(self,␣
↪*args)
   1302
   1303         answer = self.gateway_client.send_command(command)
-> 1304         return_value = get_return_value(
   1305             answer, self.gateway_client, self.target_id, self.name)
   1306

/opt/conda/lib/python3.8/site-packages/pyspark/sql/utils.py in deco(*a, **kw)
    115                 # Hide where the exception came from that shows a␣
↪non-Pythonic
```

```
    116                 # JVM exception message.
--> 117                 raise converted from None
    118             else:
    119                 raise

AnalysisException: cannot resolve '`ride_id`' given input columns:␣
↪[acceleration_ride_id, acceleration_timestamp, accuracy, geohash, latitude,␣
↪location_ride_id, location_timestamp, longitude, speed, x, y, z];
'Project [location_timestamp#63-T2000ms, location_ride_id#68, speed#69,␣
↪latitude#70, longitude#71, geohash#72, accuracy#73,␣
↪acceleration_timestamp#87-T2000ms, acceleration_ride_id#92, x#93, y#94, z#95,␣
↪to_json(struct(NamePlaceholder, 'ride_id, location_timestamp,␣
↪location_timestamp#63-T2000ms, speed, speed#69, latitude, latitude#70,␣
↪longitude, longitude#71, geohash, geohash#72, accuracy, accuracy#73,␣
↪acceleration_timestamp, acceleration_timestamp#87-T2000ms, x, x#93, y, y#94,␣
↪z, z#95), Some(Etc/UTC)) AS value#129]
+- Join Inner, (location_ride_id#68 = acceleration_ride_id#92)
   :- EventTimeWatermark location_timestamp#63: timestamp, 2 seconds
   :  +- Project [location_timestamp#63, json_value#65.ride_id AS␣
↪location_ride_id#68, json_value#65.speed AS speed#69, json_value#65.latitude␣
↪AS latitude#70, json_value#65.longitude AS longitude#71, json_value#65.geohas␣
↪AS geohash#72, json_value#65.accuracy AS accuracy#73]
   :     +- Project [timestamp#33 AS location_timestamp#63, <lambda>(value#29)␣
↪AS json_value#65]
   :        +- StreamingRelationV2 org.apache.spark.sql.kafka010.
↪KafkaSourceProvider@180d1642, kafka, org.apache.spark.sql.kafka010.
↪KafkaSourceProvider$KafkaTable@70feb780, [kafka.bootstrap.servers=kafka.kafka
↪svc.cluster.local:9092, subscribe=SutowBrett-locations], [key#28, value#29,␣
↪topic#30, partition#31, offset#32L, timestamp#33, timestampType#34],␣
↪StreamingRelation DataSource(org.apache.spark.sql.
↪SparkSession@64adba0c,kafka,List(),None,List(),None,Map(kafka.bootstrap.
↪servers -> kafka.kafka.svc.cluster.local:9092, subscribe ->␣
↪SutowBrett-locations),None), kafka, [key#21, value#22, topic#23, partition#24␣
↪offset#25L, timestamp#26, timestampType#27]
   +- EventTimeWatermark acceleration_timestamp#87: timestamp, 2 seconds
      +- Project [acceleration_timestamp#87, json_value#89.ride_id AS␣
↪acceleration_ride_id#92, json_value#89.x AS x#93, json_value#89.y AS y#94,␣
↪json_value#89.z AS z#95]
         +- Project [timestamp#54 AS acceleration_timestamp#87,␣
↪<lambda>(value#50) AS json_value#89]
            +- StreamingRelationV2 org.apache.spark.sql.kafka010.
↪KafkaSourceProvider@5879a813, org.apache.spark.sql.kafka010.
↪KafkaSourceProvider, org.apache.spark.sql.kafka010.
↪KafkaSourceProvider$KafkaTable@28360a58, [kafka.bootstrap.servers=localhost:
↪9092, subscribe=SutowBrett-accelerations], [key#49, value#50, topic#51,␣
↪partition#52, offset#53L, timestamp#54, timestampType#55], StreamingRelation␣
↪DataSource(org.apache.spark.sql.SparkSession@64adba0c,org.apache.spark.sql.
↪kafka010.KafkaSourceProvider,List(),None,List(),None,Map(kafka.bootstrap.
↪servers -> localhost:9092, subscribe -> SutowBrett-accelerations),None),␣
↪kafka, [key#42, value#43, topic#44, partition#45, offset#46L, timestamp#47,␣
↪timestampType#48]
```

[ ]: