Assignment 9.2

By Kurt Stoneburner

Pyspark Time/Session Windows: https://towardsdatascience.com/spark-3-2-session-windowing-feature-for-streaming-data-e404d92e267 (https://towardsdatascience.com/spark-3-2-session-windowing-feature-for-streaming-data-e404d92e267)

In [1]:

```
import os
import shutil
import json
from pathlib import Path
from IPython.display import clear_output
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
from pyspark.sql.types import StringType, TimestampType, DoubleType,
from pyspark.sql.functions import udf

current_dir = Path(os.getcwd()).absolute()
checkpoint_dir = current_dir.joinpath('checkpoints')
locations_windowed_checkpoint_dir = checkpoint_dir.joinpath('location

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

```

Configuration Parameters

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

In [2]:

```
config = dict(
    bootstrap_servers=['kafka.kafka.svc.cluster.local:9092'],
    first_name='Kurt',
    last_name='Stoneburner'
)

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_prefi
config['accelerations_topic'] = '{}-accelerations'.format(config['top
config['windowed_topic'] = '{}-windowed'.format(config['topic_prefix'

```

Out[2]:

```
{'bootstrap_servers': ['kafka.kafka.svc.cluster.local:9092'],
 'first_name': 'Kurt',
 'last_name': 'Stoneburner',
 'client_id': 'StoneburnerKurt',
 'topic_prefix': 'StoneburnerKurt',
 'locations_topic': 'StoneburnerKurt-locations',
 'accelerations_topic': 'StoneburnerKurt-accelerations',
 'windowed_topic': 'StoneburnerKurt-windowed'}
```

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.

In [3]:

```
def create_kafka_topic(topic_name, config=config, num_partitions=1, r
    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('windowed')
```

```
Topic "StoneburnerKurt-windowed" 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.

In [4]:

```
spark = SparkSession\
    .builder\
    .appName("Assignment09")\
    .getOrCreate()

df_locations = spark \
  .readStream \
  .format("kafka") \
  .option("kafka.bootstrap.servers", config['bootstrap_servers'][0])
  .option("subscribe", config['locations_topic']) \
  .load()

df_accelerations = spark\
    .readStream.format("kafka")\
    .option("kafka.bootstrap.servers", config['bootstrap_servers'][0]
    .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.

In [5]:

```
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')),
udf_parse_location = udf(lambda x: json.loads(x.decode('utf-8')), loc
```

See http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#window-

operations-on-event-time (http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#window-operations-on-event-time) for details on how to implement windowed operations.

The following code selects the `timestamp` column from the `df_locations` dataframe that reads from the `LastnameFirstname-locations` topic and parses the binary value using the `udf_parse_location` UDF and defines the result to the `json_value` column.

```
df_locations \
  .select(
    col('timestamp'),
    udf_parse_location(df_locations['value']).alias('json_value')
  )
```

From here, you can select data from the `json_value` column using the `select` method. For instance, if you saved the results of the previous code snippet to `df_locations_parsed` you could select columns from the `json_value` field and assign them aliases using the following code.

```
df_locations_parsed.select(
    col('timestamp'),
    col('json_value.ride_id').alias('ride_id'),
    col('json_value.uuid').alias('uuid'),
    col('json_value.speed').alias('speed')
  )
```

Next, you will want to add a watermark and group by `ride_id` and `speed` using a window duration of *30 seconds* and a slide duration of *15 seconds*. Use the `withWatermark` method in conjunction with the `groupBy` method. The Spark streaming documentation (http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#window-operations-on-event-time) should provide examples of how to do this.

Next use the `mean` aggregation method to compute the average values and rename the column `avg(speed)` to `value` and the column `ride_id` to `key`. The reason you are renaming these values is that the PySpark Kafka API expects `key` and `value` as inputs. In a production example, you would setup serialization that would handle these details for you.

When you are finished, you should have a streaming query with `key` and `value` as columns.

In [6]:

```
df_locations_parsed = df_locations \
  .select(
    col('timestamp'),
    udf_parse_location(df_locations['value']).alias('json_value')
  )

df_locations_parsed.select(
    col('timestamp'),
    col('json_value.ride_id').alias('ride_id'),
    col('json_value.uuid').alias('uuid'),
    col('json_value.speed').alias('speed')
  )
```

```
root
 |-- timestamp: timestamp (nullable = true)
 |-- json_value: struct (nullable = true)
 |    |-- offset: double (nullable = true)
 |    |-- id: string (nullable = true)
 |    |-- ride_id: string (nullable = true)
 |    |-- uuid: string (nullable = true)
 |    |-- course: double (nullable = true)
 |    |-- latitude: double (nullable = true)
 |    |-- longitude: double (nullable = true)
 |    |-- geohash: string (nullable = true)
 |    |-- speed: double (nullable = true)
 |    |-- accuracy: double (nullable = true)
```

In [7]:

```
"""
# Group the data by session window and userId, and compute the count

    sessionizedCounts = events \
    .withWatermark("timestamp", "10 minutes") \
    .groupBy(
        session_window(events.timestamp, "5 minutes"),
        events.userId) \
    .count()

slidingWindows = windowing_df.withWatermark("timeReceived", "10 minut
.groupBy("eventId", window("timeReceived", "10 minutes", "5 minutes")
```

Out[7]:

```
'\n# Group the data by session window and userId, and compute the coun
t of each group\n\n    sessionizedCounts = events     .withWatermark("
timestamp", "10 minutes")     .groupBy(\n        session_window(event
s.timestamp, "5 minutes"),\n        events.userId)     .count()\n\nsli
dingWindows = windowing_df.withWatermark("timeReceived", "10 minute
s")\n.groupBy("eventId", window("timeReceived", "10 minutes", "5 minut
es")).count()slidingWindows.show(truncate = False)\n\n'
```

In [8]:

```
windowedSpeeds = df_locations_parsed \
    .withWatermark("timestamp", "10 seconds") \
    .groupBy("json_value.ride_id", window("timestamp", "10 seconds",
    .mean('json_value.speed') \
    .withColumnRenamed("ride_id","key")\
    .withColumnRenamed("avg(json_value.speed AS `speed`)","value") \
    .select(col('key'),col('value'))

```

```
root
 |-- key: string (nullable = true)
 |-- value: double (nullable = true)
```

In [9]:

```
windowedSpeeds = df_locations_parsed \
    .withWatermark("timestamp", "10 seconds") \
    .groupBy("json_value.ride_id", window("timestamp", "10 seconds",
    .mean('json_value.speed') \
    .withColumnRenamed("ride_id","key")\
    .withColumnRenamed("avg(json_value.speed AS `speed`)","value") \
    .select(col('key'),col('value'))

```

```
root
 |-- key: string (nullable = true)
 |-- value: double (nullable = true)
```

In [10]:

```
windowedSpeeds = df_locations_parsed \
    .withWatermark("timestamp", "10 seconds") \
    .groupBy(window("timestamp", "10 seconds", "5 seconds"),"json_val
    .count() \
    .select(col('ride_id').alias("key"),col('count').alias("value"))


```

```
root
 |-- key: string (nullable = true)
 |-- value: long (nullable = false)
```

In [11]:

```
"""
windowedSpeeds = df_locations_parsed \
    .withWatermark('timestamp', "30 seconds") \
    .groupBy(
          window("timestamp", "30 seconds", "15 seconds"),
          "json_value.ride_id") \
    .mean("json_value.speed")

windowedSpeeds.printSchema()
windowedSpeeds.select(windowedSpeeds.columns[2])

#.select(col("json_value.ride_id").alias("key"), col("avg(speed)").al

```

Out[11]:

```
'\nwindowedSpeeds = df_locations_parsed     .withWatermark(\'timestam
p\', "30 seconds")     .groupBy(\n          window("timestamp", "30 se
conds", "15 seconds"),\n          "json_value.ride_id")     .mean("jso
n_value.speed") \n    \nwindowedSpeeds.printSchema()\nwindowedSpeeds.s
elect(windowedSpeeds.columns[2])\n\n#.select(col("json_value.ride_i
d").alias("key"), col("avg(speed)").alias("value"))\n\n'
```

In [12]:

```
"""
def foreach_batch_function(df, epoch_id):

    # Transform and write batchDF
    #print(df.printSchema())
    try:
        clear_output(wait=True)
        print(df.select(df.columns).show())
    except:
        pass
```

```



ds_locations = df_locations_parsed.writeStream.foreachBatch(foreach_b



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

In [13]:

```
windowedSpeeds = df_locations_parsed \
    .withWatermark("timestamp", "15 seconds") \
    .groupBy(
        window("timestamp", "30 seconds", "15 seconds"),
        "json_value.ride_id",
        'json_value.speed',
        'timestamp'
    )\
    .mean('json_value.speed')\
    .select(col('ride_id').alias('key'), col('speed').alias('value')


```

```
root
 |-- key: string (nullable = true)
 |-- value: double (nullable = true)
```

In [ ]:

In the previous Jupyter cells, you should have created the `windowedSpeeds` streaming query. Next, you will need to write that to the `LastnameFirstname-windowed` topic. If you created the `windowsSpeeds` streaming query correctly, the following should publish the results to the `LastnameFirstname-windowed` topic.

In [21]:

```
ds_locations_windowed = windowedSpeeds \
  .selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)") \
  .writeStream \
  .format("kafka") \
  .option("kafka.bootstrap.servers", "kafka.kafka.svc.cluster.local:9
  .option("topic", config['windowed_topic']) \
  .option("checkpointLocation", str(locations_windowed_checkpoint_dir
.start()

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

```
----------------------------------------------------------------------
-----
StreamingQueryException                   Traceback (most recent call
last)
<ipython-input-21-0e36252d7665> in <module>
      9
     10 try:
---> 11     ds_locations_windowed.awaitTermination()
     12 except KeyboardInterrupt:
     13     print("STOPPING STREAMING DATA")

/usr/local/spark/python/pyspark/sql/streaming.py in awaitTermination(s
elf, timeout)
    101             return self._jsq.awaitTermination(int(timeout * 10
00))
    102         else:
--> 103             return self._jsq.awaitTermination()
    104
    105     @property

/usr/local/spark/python/lib/py4j-0.10.9-src.zip/py4j/java_gateway.py i
n __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

/usr/local/spark/python/pyspark/sql/utils.py in deco(*a, **kw)
    135                 # Hide where the exception came from that show
s a non-Pythonic
    136                 # JVM exception message.
--> 137                 raise_from(converted)
    138             else:
    139                 raise

/usr/local/spark/python/pyspark/sql/utils.py in raise_from(e)

StreamingQueryException: Writing job aborted.
=== Streaming Query ===
Identifier: [id = 60f34d1c-fc52-4f84-aeec-cdbf62465592, runId = 65cb0f
a9-3da1-45a4-ba24-a59c88991847]
Current Committed Offsets: {KafkaV2[Subscribe[StoneburnerKurt-location
s]]: {"StoneburnerKurt-locations":{"0":11104}}}
Current Available Offsets: {KafkaV2[Subscribe[StoneburnerKurt-location
s]]: {"StoneburnerKurt-locations":{"0":11105}}}

Current State: ACTIVE
Thread State: RUNNABLE

Logical Plan:
WriteToMicroBatchDataSource org.apache.spark.sql.kafka010.KafkaStreami
ngWrite@6b89d9cc
+- Project [cast(key#144 as string) AS key#636, cast(value#145 as stri
ng) AS value#637]
```

```
   +- Project [ride_id#136 AS key#144, speed#137 AS value#145]
      +- Aggregate [window#138-T15000ms, json_value#43.ride_id, json_v
alue#43.speed, timestamp#12-T15000ms], [window#138-T15000ms AS window#
128-T15000ms, json_value#43.ride_id AS ride_id#136, json_value#43.spee
d AS speed#137, timestamp#12-T15000ms, avg(json_value#43.speed) AS avg
(json_value.speed AS `speed`)#133]
         +- Filter ((timestamp#12-T15000ms >= window#138-T15000ms.star
t) AND (timestamp#12-T15000ms < window#138-T15000ms.end))
            +- Expand [ArrayBuffer(named_struct(start, precisetimestam
pconversion((((CASE WHEN (cast(CEIL((cast((precisetimestampconversion
(timestamp#12-T15000ms, TimestampType, LongType) - 0) as double) / cas
t(15000000 as double))) as double) = (cast((precisetimestampconversion
(timestamp#12-T15000ms, TimestampType, LongType) - 0) as double) / cas
t(15000000 as double))) THEN (CEIL((cast((precisetimestampconversion(t
imestamp#12-T15000ms, TimestampType, LongType) - 0) as double) / cast
(15000000 as double))) + cast(1 as bigint)) ELSE CEIL((cast((preciseti
mestampconversion(timestamp#12-T15000ms, TimestampType, LongType) - 0)
as double) / cast(15000000 as double))) END + cast(0 as bigint)) - cas
t(2 as bigint)) * 15000000) + 0), LongType, TimestampType), end, preci
setimestampconversion((((((CASE WHEN (cast(CEIL((cast((precisetimestam
pconversion(timestamp#12-T15000ms, TimestampType, LongType) - 0) as do
uble) / cast(15000000 as double))) as double) = (cast((precisetimestam
pconversion(timestamp#12-T15000ms, TimestampType, LongType) - 0) as do
uble) / cast(15000000 as double))) THEN (CEIL((cast((precisetimestampc
onversion(timestamp#12-T15000ms, TimestampType, LongType) - 0) as doub
le) / cast(15000000 as double))) + cast(1 as bigint)) ELSE CEIL((cast
((precisetimestampconversion(timestamp#12-T15000ms, TimestampType, Lon
gType) - 0) as double) / cast(15000000 as double))) END + cast(0 as bi
gint)) - cast(2 as bigint)) * 15000000) + 0) + 30000000), LongType, Ti
mestampType)), timestamp#12-T15000ms, json_value#43), ArrayBuffer(name
d_struct(start, precisetimestampconversion((((CASE WHEN (cast(CEIL((c
ast((precisetimestampconversion(timestamp#12-T15000ms, TimestampType,
LongType) - 0) as double) / cast(15000000 as double))) as double) = (c
ast((precisetimestampconversion(timestamp#12-T15000ms, TimestampType,
LongType) - 0) as double) / cast(15000000 as double))) THEN (CEIL((cas
t((precisetimestampconversion(timestamp#12-T15000ms, TimestampType, Lo
ngType) - 0) as double) / cast(15000000 as double))) + cast(1 as bigin
t)) ELSE CEIL((cast((precisetimestampconversion(timestamp#12-T15000ms,
TimestampType, LongType) - 0) as double) / cast(15000000 as double)))
END + cast(1 as bigint)) - cast(2 as bigint)) * 15000000) + 0), LongTy
pe, TimestampType), end, precisetimestampconversion((((((CASE WHEN (ca
st(CEIL((cast((precisetimestampconversion(timestamp#12-T15000ms, Times
tampType, LongType) - 0) as double) / cast(15000000 as double))) as do
uble) = (cast((precisetimestampconversion(timestamp#12-T15000ms, Times
tampType, LongType) - 0) as double) / cast(15000000 as double))) THEN
(CEIL((cast((precisetimestampconversion(timestamp#12-T15000ms, Timesta
mpType, LongType) - 0) as double) / cast(15000000 as double))) + cast
(1 as bigint)) ELSE CEIL((cast((precisetimestampconversion(timestamp#1
2-T15000ms, TimestampType, LongType) - 0) as double) / cast(15000000 a
s double))) END + cast(1 as bigint)) - cast(2 as bigint)) * 15000000)
+ 0) + 30000000), LongType, TimestampType)), timestamp#12-T15000ms, js
on_value#43)], [window#138-T15000ms, timestamp#12-T15000ms, json_value
#43]
               +- EventTimeWatermark timestamp#12: timestamp, 15 secon
ds
                  +- Project [timestamp#12, <lambda>(value#8) AS json_
```

```
value#43]
                     +- StreamingDataSourceV2Relation [key#7, value#8,
topic#9, partition#10, offset#11L, timestamp#12, timestampType#13], or
g.apache.spark.sql.kafka010.KafkaSourceProvider$KafkaScan@31a12753, Ka
```