Step 1¶

Initialize Spark Session

In [ ]:
```
import os
os.environ['PYSPARK_SUBMIT_ARGS'] = '--packages org.apache.spark:spark-streaming-
kafka-0-10_2.12:3.0.0,org.apache.spark:spark-sql-kafka-0-10_2.12:3.0.0 pyspark-shell'

from pyspark.sql import SparkSession
from pyspark.sql.functions import explode
from pyspark.sql.functions import split
from pyspark.sql import functions as F
from pyspark.sql.types import *

spark = SparkSession \
    .builder \
    .appName("Week 11 - Granularity Reduction") \
    .getOrCreate()
```

Step 2¶

Connection to Kafka Producer/Broker and subscribe to the topic and load data from Kafka topic with `readStream`

In [ ]:
```
topic = "week11_orig_data"
df = spark \
    .readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "127.0.0.1:9092") \
    .option("subscribe", topic) \
    .load()
```

In [ ]:
```
df.printSchema()
```

Step 3¶

Converting the value from the kafka data stream to string

In [ ]:
```
df = df.selectExpr("CAST(value AS STRING)")
```

In [ ]:
```
df.printSchema()
```

Step 4¶

Define a schema according to our data (as sent from the producer), Use `from_json` to parse the string to the json format based on the defined schema. Each message contains the value of the timestamp as "ts" field and a random integer value as "value" field, you can define a schema as follows

In [ ]:
```
schema = StructType([
    StructField('ts', TimestampType(), True),
    StructField('value', IntegerType(), True)
])
```

In [ ]:
```
df=df.select(F.from_json(F.col("value").cast("string"),
schema).alias('parsed_value'))
```

In [ ]:
```
df.printSchema()
```

The columns need to be renamed appropriately.

In [ ]:
```
df_formatted = df.select(
                    F.col("parsed_value.ts").alias("ts"),
                    F.col("parsed_value.value").alias("value")
                )
```
In [ ]:
```
df_formatted.printSchema()
```
Step 5¶

Reduce the value of the data by grouping the timestamp "ts" on a window of 5 seconds

In [ ]:
```
#Using the window function, we can perform the following aggregation
grouped_avg = df_formatted.groupBy(F.window("ts","5 second"))\
                    .agg(F.avg("value").alias("avg_value"))
```
In [ ]:
```
grouped_avg.printSchema()
```
Parsing and renaming the columns appropriately

In [ ]:
```
grouped_avg = grouped_avg.select(
                    F.col("window.end").alias("end_time"),
                    F.col("avg_value")
                )
```
In [ ]:
```
grouped_avg.printSchema()
```

Step 6¶

Create the **output sink** for the stream. For this case, we will output the data in memory. One will output the original random values in a table called "query_all" and the reduced values in another table called "reduced_values".

In [ ]:
```
#Change the output sink to "memory" and write output to the memory sink
query_all = df_formatted \
    .writeStream \
    .outputMode("append") \
    .format("memory") \
    .queryName("all_values") \
    .trigger(processingTime='5 seconds') \
    .start()
```

In [ ]:
```
#Change the output sink to "memory" and write output to the memory sink
query_reduced = grouped_avg \
    .writeStream \
    .outputMode("complete") \
    .format("memory") \
    .queryName("reduced_values") \
    .trigger(processingTime='5 seconds') \
    .start()
```
In [ ]:
```
spark.sql("select * from all_values order by ts asc").show()
spark.sql("select * from reduced_values order by end_time asc").show()
```
Visualizing streaming data¶

We have implemented the aggregation to get the average values of the random data in a window of 5 seconds. Let's write this this to the memory sink and query it using spark sql for visualizing it in real time.

Here, first we need to initialize an empty plot.

In [ ]:

```
def init_plots():
    try:
        width = 9.5
        height = 6
        fig = plt.figure(figsize=(width,height)) # create new figure
        fig.subplots_adjust(hspace=0.8)
        ax = fig.add_subplot(111) # adding the subplot axes to the given grid
position
        ax.set_xlabel('Time')
        ax.set_ylabel('Value')
        ax.title.set_text('Time Vs Value')
        fig.suptitle('Real-time uniform stream data visualization') # giving figure a
title
        fig.show() # displaying the figure
        fig.canvas.draw() # drawing on the canvas
        return fig, ax
    except Exception as ex:
        print(str(ex))
```

In [ ]:

```
import time
import matplotlib.pyplot as plt
%matplotlib notebook

fig, ax = init_plots()

while True:
    df_all = spark.sql("select * from all_values order by ts desc limit
90").toPandas()
    # Get starting timestamp to plot both graphs
    start_time = df_all['ts'][len(df_all)-1]
    df_reduced = spark.sql("select * from reduced_values where
end_time>='"+str(start_time)+"' order by end_time desc").toPandas()

    x_all = df_all['ts'].to_list()
    y_all = df_all['value'].to_list()
    x_reduced = df_reduced['end_time'].to_list()
    y_reduced = df_reduced['avg_value'].to_list()
    ax.clear()
    ax.plot(x_all, y_all, '-b', label='Original')
    ax.plot(x_reduced, y_reduced, '--r', label='Reduced')
    ax.set_xlabel('Time')
    ax.set_ylabel('Value')
    leg = ax.legend()
    fig.canvas.draw()

    time.sleep(5)
```

In [ ]:

In [ ]: