ANALIZA BIG MOBILITY DATASTREAM-A U APACHE SPARKU I FLINKU

Danilo Milošević 1732

PODACI

- Koristimo isti skup podataka kao u prethodnom zadatku
- 2 CSV fajla veličine ~ 1GB.
- Fajlovi sadrže podatke o izduvnim gasovima kao i geografskim podacima kretanja automobila, motora, bicikli i pešaka

PREGLED DELOVA

Producer

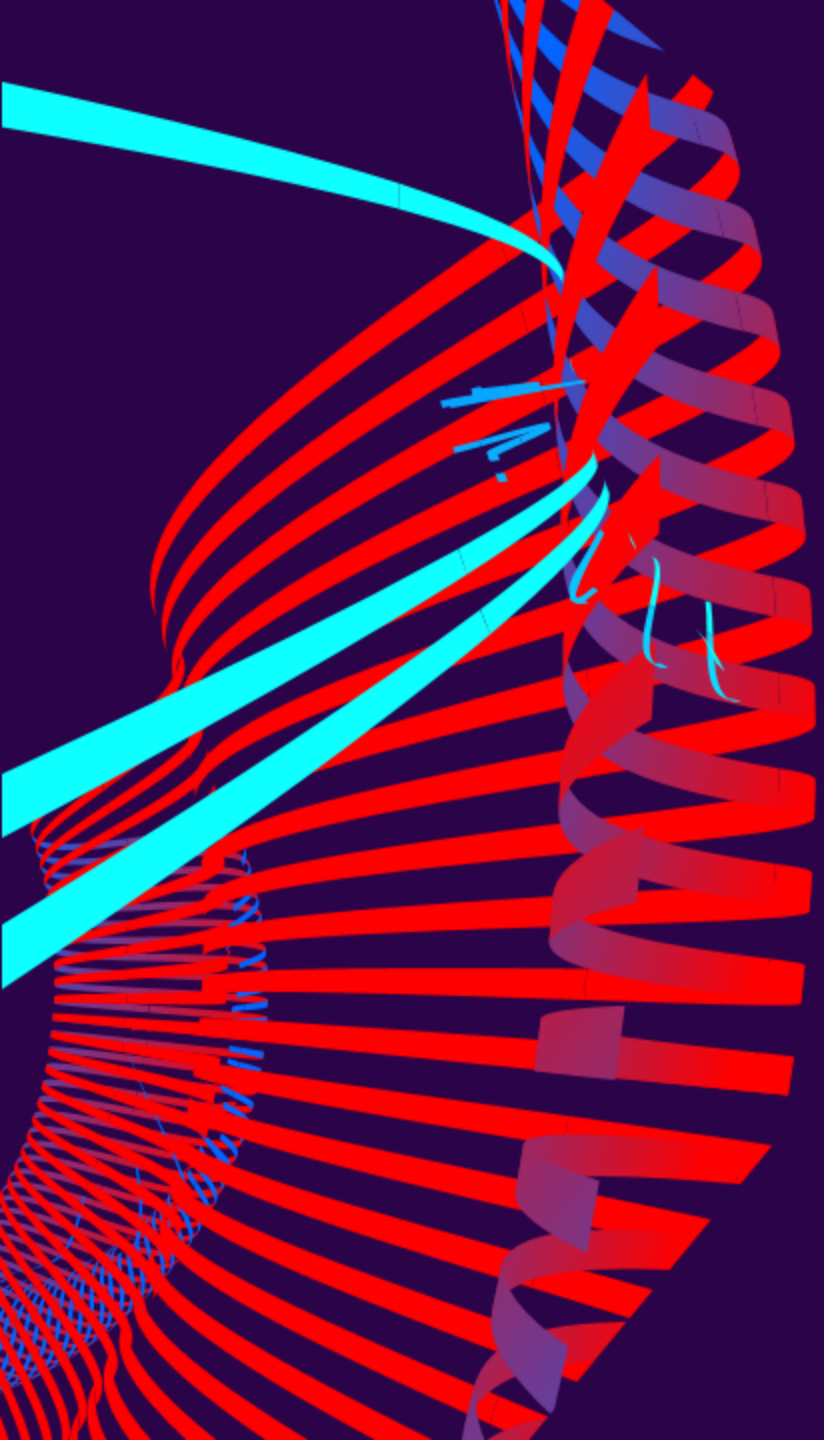
- Python skripta koja šalje podatke na Kafka stream
- Može se izabrati Kafka stream u zavisnosti da li su geografski ili emisioni podaci

Flink Consumer

- Python skripta koja analizira emisione podatke i geografske podatke sa Kafka topic-a korišćenjem Apache Flinka

Spark Consumer

- Python skripte koja analizira emisione podatke i geografske podatke sa Kafka topic-a korišćenjem Apache Sparka



PRODUCER

PRODUCER

Konfiguracija

- __init__ funkcijom kreiramo instancu klase Producer gde se poziva i određivanje konfiguracije
- U okviru funkcije get_app_config određujemo gru pu kao i topic na koji se subscribujemo.
- Pored toga se određuju i drugi argumenti kao što su
  - putanja CSV fajla,
  - da li štampamo podatke za spark ili flink,
  - da li štampamo poruke na konzolu
  - Error flag, indikuje da li je konfiguracija ispravna

```python
class Producer:
    def __init__(self, args, createProducer = True):
        self.config={}
        self.get_app_config(args=args)
        if not self.config['error'] and createProducer:
            self.producer = self.configure_producer()

    def configure_producer(self):
        producer_config = {
            'bootstrap_servers': '0.0.0.0:9094',
            'client_id': self.config['client_id'],
            'acks': 'all',
            'linger_ms': 10,
        }
        return KafkaProducer(**producer_config)
```

- U okviru get_app_config funkcije se takođe vrši provera unetih argumenata
- Ukoliko argumenti fale konfiguracija je označena da poseduje error
- Ukoliko su argumenti pogrešno upisani korisnik se obaveštava i postavlja se error flag.

```python
def get_app_config(self, args):
    start_path = '/Users/danilomilosevic/Documents/Danilo/VS/'
    self.config = {
        'client_id':'emission-producer',
        'group':'emission-group',
        'to_print':False,
        'is_spark':True,
        'sleep_time':0,#s
        'topic':'emsTopic',
        'file':start_path + 'emissions.csv',
        'error':False
    }

    if len(args) < 2:
        print("\tUsage: python msg_producer.py [ems|fcd](type) [spark|flink] [print|noprint] [sleep_time(ms)]")
        self.config['error'] = True
        return

    try:
        type = args[1]
        self.config['client_id'] = 'fcd-producer' if type=='fcd' else 'emission-producer'
        self.config['group'] = 'fcd-group' if type=='fcd' else 'emission-group'
        self.config['topic'] = 'fcdTopic' if type=='fcd' else 'emsTopic'
        self.config['file'] = start_path+'fcd.csv' if type=='fcd' else start_path+'emissions.csv'
        self.config['is_spark'] = args[2] == 'spark'
        self.config['to_print'] = args[3] == 'print'
        try:
            self.config['sleep_time'] = float(args[4])/1000.0
        except ValueError:
            self.config['error'] = True
            print('\tSleep time has to be in milliseconds!')
    except:
        pass
```

PRODUCER

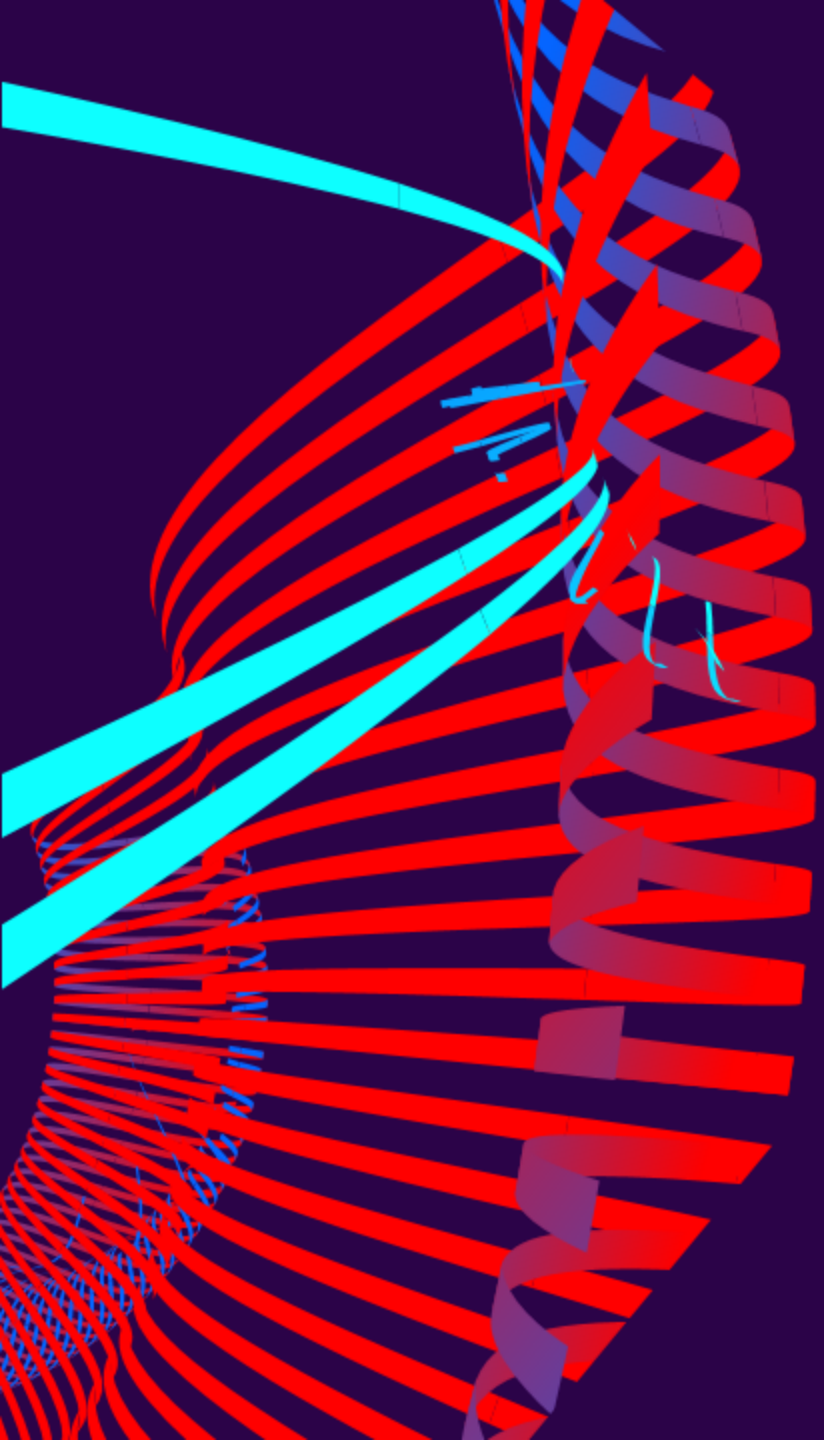
Record production

- U okviru funkcije produce_records se obavlja sledeće:
  - Proverava se greška u konfiguraciji i štampa odgovarajuća poruka
  - Otvara se fajl i preskače prva linija (CSV header)
  - Generišemo random početni datum kao timestamp
  - Čita se linija po liniju
  - Linija se formatira u specifičnom formatu
  - Linija se šalje na topic
  - Ukoliko je print flag true, štampamo na ekran šta se šalje

```
def produce_records(self):
    if(self.config['error']):
        print('\tError in config!')
        return
    with open(self.config['file']) as file:
        line = file.readline() #skip first line
        start_date = Producer.get_random_date()
        line = file.readline()
        while line:
            processed_line, ok = self.format_line(line, start_date=start_date)
            if ok:
                if self.config['to_print']:
                    print('Sending to topic: ', processed_line)
                self.producer.send(
                    topic=self.config['topic'],
                    value=processed_line.strip().encode('utf-8'))
                self.producer.flush()
                if self.config['to_print']:
                    print('\tSent!')
            time.sleep(self.config['sleep_time'])
            line = file.readline()
    self.producer.flush()
    self.producer.close()
```

- Podaci se formatiraju u funkciji format line
- U slučaju geo podataka se podaci ne formatiraju
- Kod emisionih podataka se dodaje timestamp – kod Sparka se doda offset na generisani početni datum dok kod Flink-a kreiramo timestamp u milisekundama

```
def format_line(self, line, start_date=None):
    if 'fcd' in self.config['topic'] and line.split(";")[1] == "":
        return (line,False)

    seconds = float(line.split(";")[0])
    return (Producer.get_date_timestamp(start_date, seconds, self.config['is_spark']) + line[line.index(";"):],True)
```



SPARK CONSUMER

SPARK CONSUMER

Konfiguracija

- __init__ funkcijom kreiramo instance klasa EmissionConsumerSpark i FCDConsumer spark gde se postavljaju veličine blokova kojima delimo poziciju vozila na indekse blokova
- U okviru funkcije get_app_config određujemo gru pu kao i topic na koji se subscribujemo.
- Pored toga se određuju i drugi argumenti kao što su
  - Broj zona po x i y osi
  - Veličina vremenskog prozora
  - Da li koristimo sliding window i koja je njegova veličina
  - Koliko procesa koristimo

```
class FCDConsumerSpark:

    minY = -1.68
    minX = -74.273621
    maxX = 25447.74
    maxY = 36412.67

    xZones = 4
    yZones = 4
    blockSizeX = None
    blockSizeY = None
    producer = None

    def __init__(self, topic, xZones, yZones):
        self.topic = topic
        FCDConsumerSpark.xZones = xZones
        FCDConsumerSpark.yZones = yZones
        FCDConsumerSpark.blockSizeX = (FCDConsumerSpark.maxX - FCDConsumerSpark.minX) / FCDConsumerSpark.xZones
        FCDConsumerSpark.blockSizeY = (FCDConsumerSpark.maxY - FCDConsumerSpark.minY) / FCDConsumerSpark.yZones
```

```
class EmissionConsumerSpark:

    minY = -1.68
    minX = -74.273621
    maxX = 25447.74
    maxY = 36412.67

    xZones = 4
    yZones = 4
    blockSizeX = None
    blockSizeY = None
    producer = None

    def __init__(self, topic, xZones, yZones):
        self.topic = topic
        EmissionConsumerSpark.xZones = xZones
        EmissionConsumerSpark.yZones = yZones
        EmissionConsumerSpark.blockSizeX = (EmissionConsumerSpark.maxX - EmissionConsumerSpark.minX) / EmissionConsumerSpark.xZones
        EmissionConsumerSpark.blockSizeY = (EmissionConsumerSpark.maxY - EmissionConsumerSpark.minY) / EmissionConsumerSpark.yZones
```

```
@staticmethod
def get_app_config(args):
    config = {
        'topic':'emsTopic',
        'group':'emission-group',
        'to_print':False,
        'out_topic':None,
        'secondsWindow':5,
        'slideWindowSeconds':1,#s
        'xZones':4,
        'yZones': 4,
        'error':False,
        'num_proc':'*'
    }
    i=1
    try:
        if (args[i]=="--print"):
            config['to_print'] = True
            i+=1

        config['secondsWindow'] = int(args[i])
        i+=1

        if(args[i]=="--slide"):
            i+=1
            config['slideWindowSeconds'] = int(args[i])
            i+=1

        if(args[i]=="--out"):
            config['out_topic'] = "ems_out_topic_spark"
            i+=1

        if(args[i]=='--n'):
            i+=1
            try:
                config['num_proc'] = str(int(args[i]))
            except:
                config['num_proc'] = '*'
            i+=1

        config['xZones'] = int(args[i])
        i+=1

        config['yZones'] = int(args[i])
        i+=1
    except:
        pass
    return config
```

```
@staticmethod
def get_app_config(args):
    config = {
        'topic':'fcdTopic',
        'group':'fcd-group',
        'to_print':False,
        'out_topic':None,
        'secondsWindow':5,
        'slideWindowSeconds':1,#s
        'xZones':4,
        'yZones': 4,
        'error':False,
    }
    i=1
    try:
        if (args[i]=="--print"):
            config['to_print'] = True
            i+=1

        config['secondsWindow'] = int(args[i])
        i+=1

        if(args[i]=="--slide"):
            i+=1
            config['slideWindowSeconds'] = int(args[i])
            i+=1

        if(args[i]=="--out"):
            config['out_topic'] = "fcd_out_topic_spark"
            i+=1

        if(args[i]=='--n'):
            i+=1
            try:
                config['num_proc'] = str(int(args[i]))
            except:
                config['num_proc'] = '*'
            i+=1

        config['xZones'] = int(args[i])
        i+=1

        config['yZones'] = int(args[i])
        i+=1
    except:
        pass
    return config
```

- Kao i u produceru, pribavljanje argumenata se vrši uz proveru da li su argumenti uneti ispravno
- Ukoliko nisu koristi se default-na vrednost

SPARK CONSUMER

Consume

- U okviru funkcije consume se obavlja sledeće:
  - Uspostavlja se stream sa odgovarajućeg topica na kafka:9092
  - Sve vrednosti se prvo čitaju kao stringovi
  - Zatim se string splituje po karakteru ; i pribavljaju odgovarajuća polja
  - Pribavljena polja se konvertuju u svoje tipove podataka (timestamp kao timestamp, ostali podaci su double=
  - Dodaje se zona kao atribut
  - Izvrši se grupacija po zoni kao i window-ing
  - Primenjuje se avg operacija za emisije dok se za geografske podatke koristi count za broj vozila u određenim oblastima i ulicama
  - Konačno dodajemo kolone za start i kraj prozora
  - Na kraju sve formatiramo kao string i šaljemo na output topic kao i na konzolu.

```
def consume(self, config):
    start = time.time()
    spark = SparkSession.builder.appName("EMS").master(f"local[{config['num_proc']}]").getOrCreate()

    kafkaDF = spark.readStream.format("kafka")\
        .option("kafka.bootstrap.servers", "kafka:9092")\
        .option("subscribe", config['topic'])\
        .option("startingOffsets", "earliest")\
        .load()

    emissionsStream = kafkaDF.selectExpr("CAST(value AS STRING)").alias("value")

    parsedData = emissionsStream.selectExpr("split(value, ';') as parsed")\
        .selectExpr(
            "parsed[0] AS timestamp",
            "parsed[1] AS CO",
            "parsed[2] AS CO2",
            "parsed[3] AS HC",
            "parsed[4] AS NOx",
            "parsed[5] AS PMx",
            "parsed[18] AS x",
            "parsed[19] AS y"
        )

    emissionsData = parsedData.select(
        col("timestamp").cast("timestamp"),
        col("CO").cast(DoubleType()),
        col("CO2").cast(DoubleType()),
        col("HC").cast(DoubleType()),
        col("NOx").cast(DoubleType()),
        col("PMx").cast(DoubleType()),
        col("x").cast(DoubleType()),
        col("y").cast(DoubleType())
    )

    determine_zone_udf = udf(self.determine_zone, IntegerType())
    emissionsData = emissionsData.withColumn("zone", determine_zone_udf(col("x"), col("y")))
```

```
reducedData = emissionsData.groupBy(
    col("zone"),
    self.get_window(config['secondsWindow'], slideWindowSeconds=config['slideWindowSeconds'])
).agg(
    avg("CO").alias("avg_CO"),
    avg("CO2").alias("avg_CO2"),
    avg("HC").alias("avg_HC"),
    avg("NOx").alias("avg_NOx"),
    avg("PMx").alias("avg_PMx")
)

windowData = reducedData.withColumn("window_start", from_unixtime(unix_timestamp(col("window.start"), "yyyy-MM-dd HH:mm:ss"), "yyyy-MM-dd HH:mm:ss")) \
                        .withColumn("window_end", from_unixtime(unix_timestamp(col("window.end"), "yyyy-MM-dd HH:mm:ss"), "yyyy-MM-dd HH:mm:ss"))\
                        .drop('window')

formated_data = windowData.withColumn("value", concat(
        col("zone"), lit(";"),
        col("window_start"), lit(";"),
        col("window_end"), lit(";"),
        col("avg_CO"), lit(";"),
        col("avg_CO2"), lit(";"),
        col("avg_HC"), lit(";"),
        col("avg_NOx"), lit(";"),
        col("avg_PMx")
    )).selectExpr("CAST(value AS STRING)")

producer = KafkaProducer(bootstrap_servers='kafka:9092')
try:
    query = formated_data \
            .writeStream \
            .outputMode("complete") \
            .foreachBatch(lambda batch_df, batch_id:
                [(producer.send(config['out_topic'], value=row.value.encode('utf-8'))) for row in batch_df.collect() if config['out_topic'] is not None]+
                [print(EmissionConsumerSpark.format_console_output(row)) for row in batch_df.collect()]
            ) \
            .start()

    query.awaitTermination()
except Exception as e:
    end = time.time()
    print("Time in s: ",(end-start))
producer.close()
```

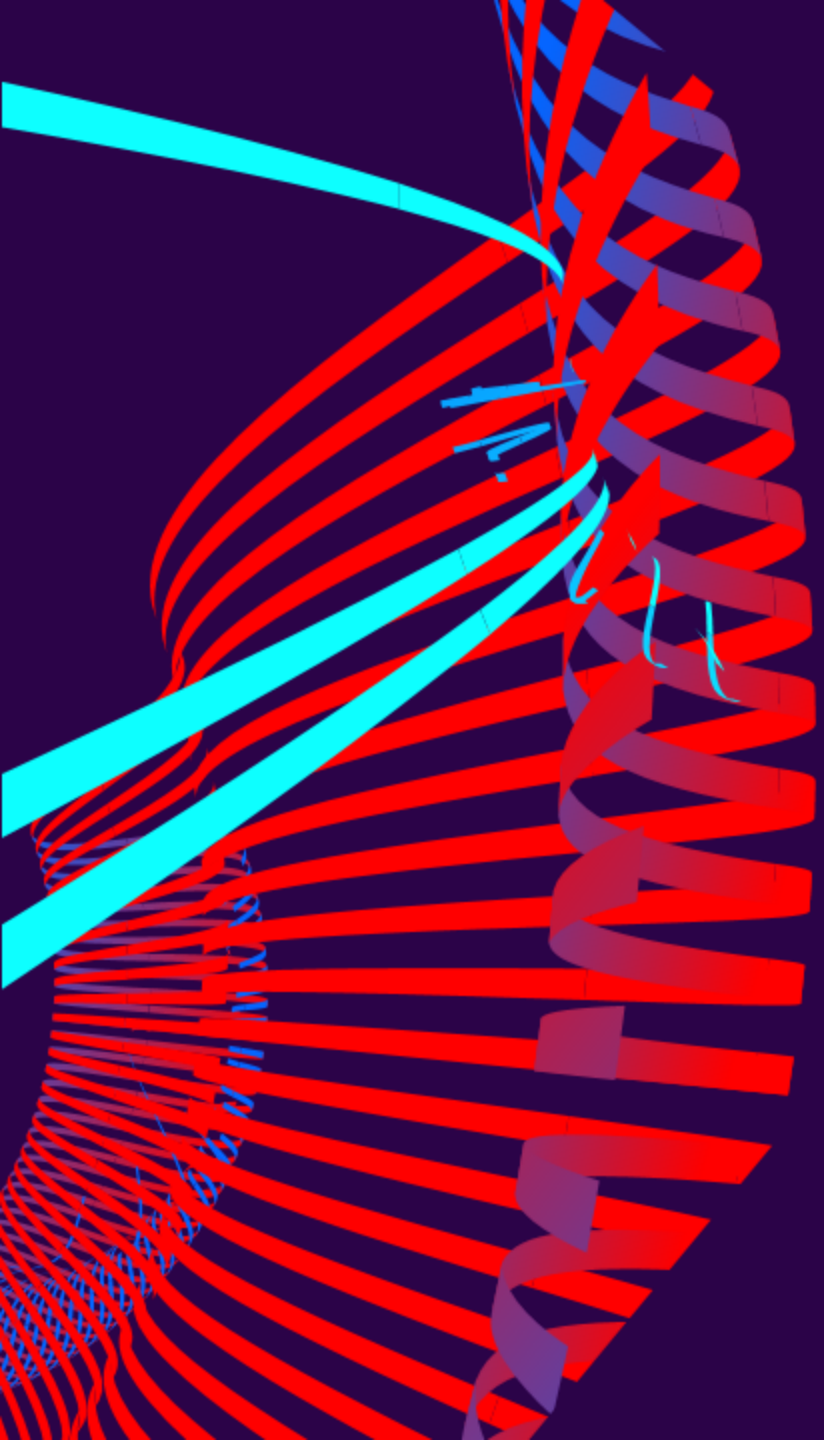
SPARK CONSUMER

Consume

- U slučaju fcd podataka, vrši se ista obrada, s tim da sada uzimamo timestamp, traku vozila i njegove koordinate
- Sada vršimo count operaciju i na kraju ispisujemo rezultate kao u prethodnom primeru

```
def consume(self, config):
    start = time.time()
    spark = SparkSession.builder.appName("FCD").master(f"local[{config['num_proc']}]").getOrCreate()

    kafkaDF = spark.readStream.format("kafka")\
        .option("kafka.bootstrap.servers", "kafka:9092")\
        .option("subscribe", config['topic'])\
        .option("startingOffsets", "earliest")\
        .load()

    emissionsStream = kafkaDF.selectExpr("CAST(value AS STRING)").alias("value")

    parsedData = emissionsStream.selectExpr("split(value, ';') as parsed")\
        .selectExpr(
            "parsed[0] AS timestamp",
            "substring_index(parsed[3], '#', 1) AS lane",
            "parsed[8] AS x",
            "parsed[9] AS y"
        )

    emissionsData = parsedData.select(
        col("timestamp").cast("timestamp"),
        col("lane"),
        col("x").cast(DoubleType()),
        col("y").cast(DoubleType())
    )

    determine_zone_udf = udf(self.determine_zone, IntegerType())
    emissionsData = emissionsData.withColumn("zone", determine_zone_udf(col("x"), col("y")))

    reducedData = emissionsData.groupBy(
        col("lane"),
        col("zone"),
        self.get_window(config['secondsWindow'], slideWindowSeconds=config['slideWindowSeconds'])
    ).count().orderBy(desc("count"))

    windowData = reducedData.withColumn("window_start", from_unixtime(unix_timestamp(col("window.start"), "yyyy-MM-dd HH:mm:ss"), "yyyy-MM-dd HH:mm:ss")) \
                            .withColumn("window_end", from_unixtime(unix_timestamp(col("window.end"), "yyyy-MM-dd HH:mm:ss"), "yyyy-MM-dd HH:mm:ss"))\
                            .drop('window')
```

```
formated_data = windowData.withColumn("value", concat(
        col("zone"), lit(";"),
        col("window_start"), lit(";"),
        col("window_end"), lit(";"),
        col("lane"), lit(";"),
        col("count"), lit(";"),
    )).selectExpr("CAST(value AS STRING)")

producer = KafkaProducer(bootstrap_servers='kafka:9092')
try:
    query = formated_data \
            .writeStream \
            .outputMode("complete") \
            .foreachBatch(lambda batch_df, batch_id:
                [(producer.send(config['out_topic'], value=row.value.encode('utf-8'))) for row in batch_df.collect() if config['out_topic'] is not None]+
                [print(FCDConsumerSpark.format_console_output(row)) for row in batch_df.collect()]
            )\
            .start()
    query.awaitTermination()
except:
    end = time.time()
    print("Time in s: ",(end-start))
producer.close()
```



FLINK CONSUMER

FLINK CONSUMER

Konfiguracija

- __init__ funkcijom kreiramo instance klasa EmissionConsumerSpark i FCDConsumer spark gde se postavljaju veličine blokova kojima delimo poziciju vozila na indekse blokova
- U okviru funkcije get_app_config određujemo gru pu kao i topic na koji se subscribujemo.
- Pored toga se određuju i drugi argumenti kao što su
  - Broj zona po x i y osi
  - Veličina vremenskog prozora
  - Da li koristimo sliding window i koja je njegova veličina
  - Koliko procesa koristimo
  - Tuple-ovi kojima ćemo vršiti izbor podatak u zavisnosti od topica
  - Lambda izraz za key-by izraz

```
class ConsumerFlink:
    minY = -1.68
    minX = -74.273621
    maxX = 25447.74
    maxY = 36412.67

    def __init__(self, topic, xZones, yZones):
        self.topic = topic
        self.xZones = xZones
        self.yZones = yZones
        self.blockSizeX = (self.maxX - self.minX) / xZones
        self.blockSizeY = (self.maxY - self.minY) / yZones

    def determine_zone(self, x, y):
        xInd = int((x - self.minX) / self.blockSizeX)
        yInd = int((y - self.minY) / self.blockSizeY)
        return yInd * self.xZones + xInd
```

```
@staticmethod
def get_app_config(args):
    ems_type_ind_tuples = ((0,str),(1,float),(2,float),(3,float),(4,float),(5,float),(18,float),(19, float))
    fcd_type_ind_tuples = ((0,str),(1,str,'split'),(8,float),(9,float))
    ems_type = Types.TUPLE([Types.STRING(), Types.DOUBLE(), Types.DOUBLE(), Types.DOUBLE(), Types.DOUBLE(), Types.DOUBLE(), Types.DOUBLE(), Types.DOUBLE()])
    fcd_type = Types.TUPLE([Types.STRING(), Types.STRING(), Types.DOUBLE(), Types.DOUBLE()])
    ems_zone = Types.TUPLE([Types.STRING(), Types.DOUBLE(), Types.DOUBLE(), Types.DOUBLE(), Types.DOUBLE(), Types.DOUBLE(), Types.DOUBLE(), Types.DOUBLE(),Types.INT()])
    fcd_zone = Types.TUPLE([Types.STRING(), Types.STRING(), Types.DOUBLE(), Types.DOUBLE(), Types.INT()])
    config = {
        'topic': 'emsTopic',
        'group': 'emission-group',
        'to_print': False,
        'out_topic': None,
        'secondsWindow': 5,
        'slideWindowSeconds': None,
        'xZones': 4,
        'yZones': 4,
        'error': False,
        'type_ind_tuples': ems_type_ind_tuples,
        'xy_inds':(6,7),
        'stream_types':ems_type,
        'zoned_types':ems_zone,
        'key_by': lambda x: x[8],
        'num_proc':8
    }
```

- Kao i u produceru, pribavljanje argumenata se vrši uz proveru da li su argumenti uneti ispravno
- Ukoliko nisu koristi se default-na vrednost

```
i = 1
try:
    if args[i] == "fcd":
        config['topic'] = 'fcdTopic'
        config['group'] = 'fcd-group'
        config['type_ind_tuples'] = fcd_type_ind_tuples
        config['xy_inds']=(2,3)
        config['stream_types'] = fcd_type
        config['zoned_types'] = fcd_zone
        config['key_by'] = lambda x: (x[1],x[4])
        i+=1

    if args[i] == "--print":
        config['to_print'] = True
        i += 1

    config['secondsWindow'] = int(args[i])
    i += 1

    if args[i] == "--slide":
        i += 1
        config['slideWindowSeconds'] = int(args[i])
        i += 1

    if args[i] == "--out":
        config['out_topic'] = 'ems_out_topic_flink' if config['topic']=='emsTopic' else 'fcd_out_topic_flink'
        i += 1

    if(args[i]=='--n'):
        i+=1
        try:
            config['num_proc'] = int(args[i])
        except:
            config['num_proc'] = 8
        i+=1

    config['xZones'] = int(args[i])
    i += 1

    config['yZones'] = int(args[i])
    i += 1
except:
    pass
return config
```

FLINK CONSUMER

Consume

- U okviru funkcije consume se obavlja sledeće:
  - Uspostavlja se stream sa odgovarajućeg topica na kafka:9092
  - Sve vrednosti se prvo čitaju kao stringovi
  - Zatim se string splituje po karakteru ; i pribavljaju odgovarajuća polja u funkciji parse data
  - Pribavljena polja se konvertuju u svoje tipove podataka
  - Dodaje se zona kao atribut
  - Izvrši se grupacija na osnovu lambda izraza u konfiguraciji kao i window-ing
  - Primenjuje se odgovarajuća funkcija za procesiranje podataka u zavisnosti da li je fcd ili ems
  - Tu se dodaje i početak i kraj prozora
  - Na kraju sve podatke šaljemo na output topic kao i na konzolu.

```
def consume(self, config):
    start = time.time()
    env = StreamExecutionEnvironment.get_execution_environment()
    env.set_parallelism(config['num_proc'])
    self.add_jars(env)
    kafka_stream = self.init_stream(config, env)

    xy_inds = config['xy_inds']
    type_ind_tuples = config['type_ind_tuples']

    def parse_data(line):
        fields = line.split(';')
        result = ()
        for t in type_ind_tuples:
            ind = t[0]
            ty = t[1]
            if ind >= len(fields):
                print("Parsing went wrong, trying to access element ",ind," length is ",len(fields))
                exit(1)
            value = fields[ind]
            if (len(t)>2):
                value = value.split('#')[0]
            result += (ty(value),)
        return result

    def assign_zone(value):
        x, y = xy_inds
        zone = self.determine_zone(value[x], value[y])
        return (*value, zone)

    process_f = ConsumerFlink.process_ems if config['topic']=='emsTopic' else ConsumerFlink.process_fcd

    try:
        parsed_stream = kafka_stream.map(parse_data, output_type=config['stream_types'])
    except Exception as e:
        print("Failed creating the parsed stream: ",e)
        exit(1)
```

```
try:
    enriched_stream = parsed_stream.map(assign_zone, output_type=config['zoned_types'])
except Exception as e:
    print("Failed adding the zone attribute: " , e)
    exit(1)

try:
    windowed_stream = enriched_stream.assign_timestamps_and_watermarks(
        WatermarkStrategy.
        for_bounded_out_of_orderness(Duration.of_seconds(10)).
        with_timestamp_assigner(Assigner())
    ).key_by(config['key_by']).window(
        self.get_window(config['secondsWindow'], config.get('slideWindowSeconds'))
    ).apply(ProcessWindowFunction(process_f), output_type=Types.STRING())

    windowed_stream.print()
except Exception as e:
    print("Failed adding windows: ", e)
    exit(1)

if config['out_topic']:
    producer = FlinkKafkaProducer(
        topic=config['out_topic'],
        serialization_schema=SimpleStringSchema(),
        producer_config={'bootstrap.servers': 'kafka:9092'}
    )
    windowed_stream.add_sink(producer)

try:
    env.execute()
except KeyboardInterrupt:
    end = time.time()
    print("Execution time in s: ", (end-start))
except Exception as e:
    print("Execute failed: ",e)
```

FLINK CONSUMER

Consume

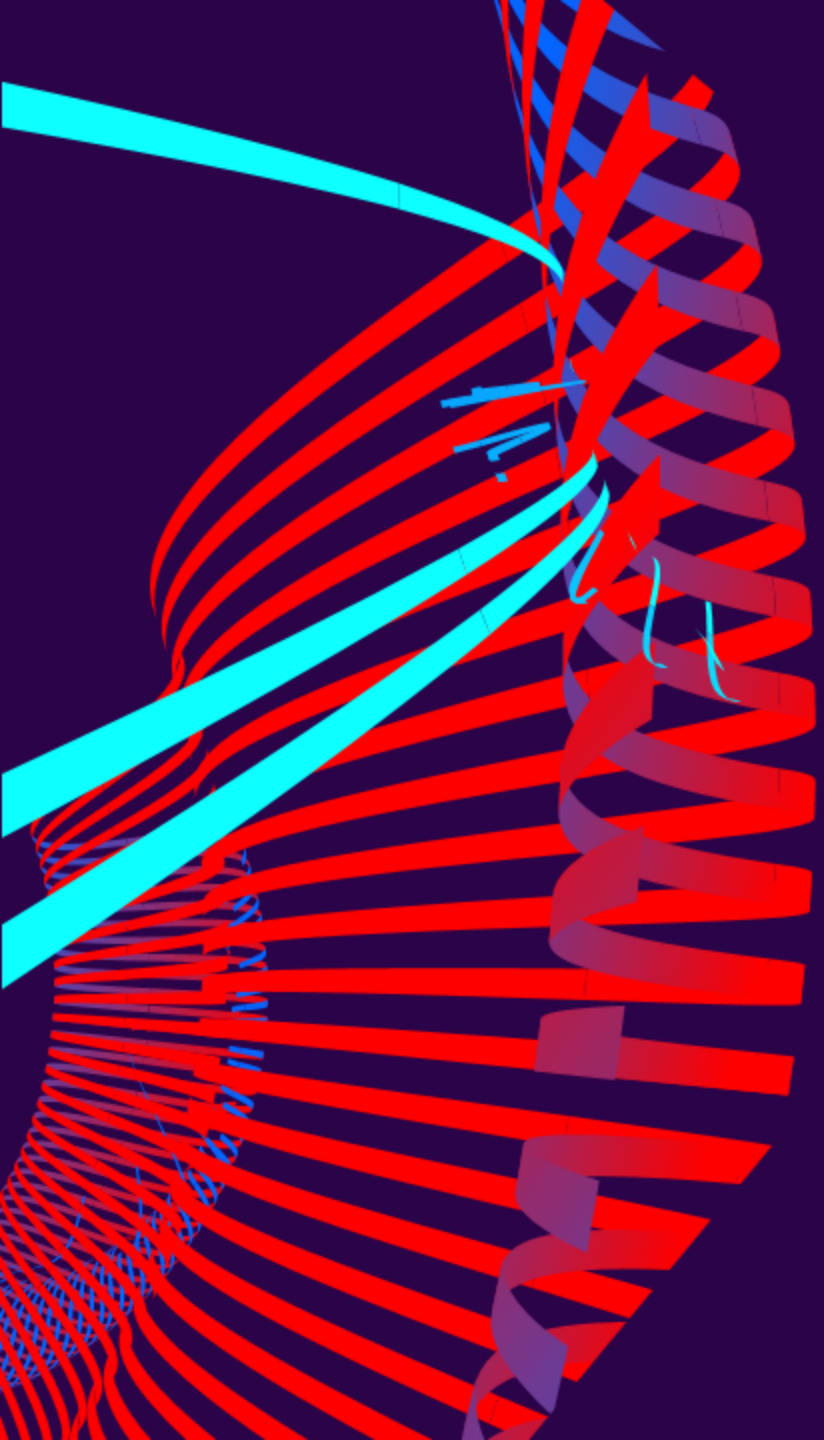
- Procesiranje podataka se vrši funkcijama process_ems i process_fcd, dok se određivanje zona i prozora vrši kroz determine_zone i get_ window

- Bitan dodatak je da je neophodno uključiti biblioteke kako bi mogli Flink i Kafka topic da komuniciraju

```
def determine_zone(self, x, y):
    xInd = int((x - self.minX) / self.blockSizeX)
    yInd = int((y - self.minY) / self.blockSizeY)
    return yInd * self.xZones + xInd

def get_window(self, windowSeconds, slideWindowSeconds = None):
    if slideWindowSeconds is None:
        return TumblingProcessingTimeWindows.of(Time.milliseconds(windowSeconds*1000))
    return SlidingProcessingTimeWindows.of(Time.milliseconds(windowSeconds*1000), Time.milliseconds(slideWindowSeconds*1000))
```

```
def process_ems(window, iterable):
    elements = list(iterable)
    zone = elements[0][8]
    window_start = window.start
    window_end = window.end
    avg_CO = round(sum([x[1] for x in elements]) / len(elements),2)
    avg_CO2 = round(sum([x[2] for x in elements]) / len(elements),2)
    avg_HC = round(sum([x[3] for x in elements]) / len(elements),2)
    avg_NOx = round(sum([x[4] for x in elements]) / len(elements),2)
    avg_PMx = round(sum([x[5] for x in elements]) / len(elements),2)
    res = str(f"|Zone: {zone};\tWindowStart: {window_start};\tWindowEnd: {window_end};\tCO: {avg_CO};\tCO2: {avg_CO2};\tHC: {avg_HC};\tNOx: {avg_NOx};\tPMx: {avg_PMx}|")
    return res

def process_fcd(window, iterable):
    elements = list(iterable)
    zone = elements[0][4]
    window_start = window.start
    window_end = window.end
    lane = elements[0][1]
    count = len(elements)
    res = str(f"|Zone: {zone};\tWindowStart:{window_start};\tWindowEnd: {window_end};\tLane: {lane};\tCount: {count}|")
    return res
```

```
def add_jars(self, env):
    file = "file://"
    jars = ["/flink/flink-python-1.19.0.jar",
            "/flink/flink-sql-connector-kafka-3.2.0-1.19.jar",
            "/flink/kafka-clients-3.2.0.jar",
            "/flink/flink-connector-kafka-3.2.0-1.19.jar"]

    for p in jars:
        if not os.path.isfile(Path(p)):
            print(p," doesn't exist!")
            exit(1)
        env.add_jars(file+p)
```



DEPLOY I PERFORMANSE

PERFORMANSE

- Testiranje vršeno na Docker containeru.
- Program se izvršava na 1,2,4 ili 8 niti

Spark

EMS(s)	1	2	3	Prosek	STD
1	17.92720604	17.83009005	17.38372159	17.71367256	0.2898423886
2	13.46768074	13.91785574	13.64364958	13.67639535	0.2268669143
4	11.70595717	11.68580675	12.03429985	11.80868793	0.1956452541
8	11.19891715	11.4132061	10.57503891	11.06238739	0.4354438449

FCD(s)	1	2	3	Prosek	STD
1	16.0277462	16.6584363	16.29742265	16.32786838	0.3164454249
2	13.14872003	13.05463839	13.06817317	13.09051053	0.05086311928
4	11.5325644	11.39065695	11.4484148	11.45721205	0.07136157947
8	11.03940177	10.6115458	11.02381516	10.89158758	0.2426484743

Flink

EMS(s)	1	2	3	Prosek	STD
1	15.99004364	13.89095855	15.60166454	15.16088891	1.116804336
2	15.65553212	15.45427918	12.43877101	14.51619411	1.801913074
4	14.47164798	14.8783226	12.79246616	14.04747891	1.105729992
8	12.58246493	14.97194958	12.5955627	13.38332574	1.37580419

FCD (lin/s)	1	2	3	Prosek	STD
1	129.25	117.3591	135.2421	127.2837333	9.102201695
2	339.1665	321.8732	345.3012	335.4469667	12.14882678
4	376.9345	380.1134	363.8391	373.629	8.626004742
8	217.7928	205.8212	221.7642	215.1260667	8.299301914

HVALA NA PAŽNJI