Laboratorijska vežba 3 – Apache Spark

Cilj vežbe: Apache Spark structured streaming

Potrebni alati i instalacija

- Apache Spark klijent
- Apache Hadoop

Ovi alati podrazumevaju da je prethodno instalirana Java i podešena promenljiva okruženja JAVA_HOME.

Instalacija Apache Spark-a

- Preuzeti Apache Spark sa linka (treba odabrati verziju 3.5.1 i Pre-built for Apache Hadoop 3.3 and later tip paketa)
- Raspakovati arhivu na pogodnu lokaciju (npr. **C:\Spark**)
- Definisati promenljivu okruženja SPARK_HOME tako da ukazuje na direktorijum u kome se nalazi Spark, npr:
  C:\Spark

Apache Hadoop instalacija

Za potrebe ove laboratorijske vežbe dovoljno je instalirati **winutils.exe** za odgovarajuću verziju Hadoop-a.

- Preuzeti **winutils.exe**
- Kreirati **Hadoop** folder (npr. **C:\Hadoop**), a u njemu **bin** folder.
- Smestiti **winutils.exe** u bin folder
- Kreirati promenljivu okruženja HADOOP_HOME tako da ukazuje na lokaciju gde je ekstraktovan Hadoop (npr. **C:\Hadoop**):

U Path promenljivu okruženja dodati putanje do

- Hadoop\bin foldera,
- Spark\bin foldera ,
- Java\bin foldera.

Ostala podešavanja

- Instalirati PySpark: pip install pyspark==3.5.1

  PySpark verzija treba da odgovara instaliranoj verziji Spark-a.

Apache Spark

Apache Spark je sistem za obradu podataka na klasteru. Aplikacija koja koristi ovaj sistem se sastoji od glavnog programa, menadžera klastera i čvorova-radnika. Menadžer klastera vrši alokaciju resursa na čvorovima u klasteru i pokreće izvršenja na čvorovima. Kada se sistem pokrene, Spark šalje kod aplikacije na čvorove, a zatim i konkretne zadatke i podatke koje treba obraditi. Uz sistem se mogu koristiti i biblioteke koje se pribavljaju zajedno sa njim: **SparkSQL**, **Structured Streaming**, **MLlib** i **GraphX**.

Osnovna apstrakcija podataka koju Spark koristi jeste *resilient distributed dataset* (RDD). RDD je kolekcija elemenata koji se mogu paralelno obrađivati. Kreiraju se paralelizacijom postojećih elemenata (npr. paralelizacijom listi) ili iz nekog drugog izvora (npr. HDFS, Cassandra, Hbase, lokalni fajl sistem).

Kolekcije je moguće obraditi transformacijama (npr. mapiranje, filtirianje, unija, presek i dr.) koje kreiraju nove podatke (novi RDD) ili akcijama (npr. reduce(), collect()) koje izvršavaju neko izračunavanje nad podacima. Transformacijama se definiše šta će se dogoditi sa podacima kada obrade počne, dok pozivi akcija započinju obradu podataka.

DataFrame je distribuirana kolekcija podataka u tabelarnom formatu sa imenovanim kolonama. Deo je Spark SQL modula i ima highlevel API za distribuiranu obradu podataka.

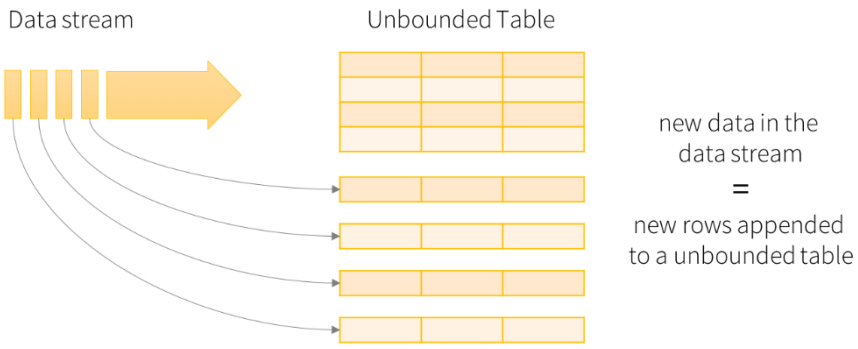
Tok Apache Spark programa podrazumeva:

- Kreiranje sesije – definisanje naziva aplikacije, master čvora i drugih konfiguracionih parametara,
- Kreiranje SparkContext promenljive koja predstavlja ulaznu tačku za korišćenje različitih Spark funkcionalnosti – konekcija prema Spark klasteru, kreiranje RDD-ova itd,
- Kreiranje transformacija i akcija nad podacima.

Structured Streaming

Structured Streaming je skalabilni sistem otporan na greške za obradu tokova podataka.

Osnovni koncept structured streaming-a jeste da se podaci koji se dobijaju iz nekog toka podataka tretiraju kao tabela u koju se konstantno dodaju novi podaci. Ovo omogućava da se tok podataka obradi na sličan način kao i serije podataka, iako se radi o neprekidnom prihvatanju podataka. Takođe, ovo omogućava upotrebu Dataset/DataFrame API za izvršavanje agregacije tokova podataka, prozora događaja u vremenu, spajanje između tokova i serija, i mnoge druge operacije.



Data stream as an unbounded table

Korisni linkovi:

Structured streaming
https://spark.apache.org/docs/latest/sql-programming-guide.html
https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/index.html#
https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame

Zadatak 1

Kreirati Apache Spark program koji čita informacije za 4 plastenika. Za svaki plastenik dobija se informacija o vremenu čitanja, imenu plastenika, temperaturi u plasteniku, vlažnosti vazduha i vlažnosti zemljišta. Ukoliko vrednost sa senzora nije mogla da se pročita umesto vrenosti prosleđuje se null. U konzoli prikazati nazive plastenika za koje nije pročitana neka od vrednosti.

```
from pyspark.sql import SparkSession

spark  = SparkSession.builder.master("local[2]") \
       .appName("SensorDataStreaming").getOrCreate()

socketDF = spark.readStream \
            .format("socket") \
            .option("host", "localhost") \
            .option("port", 9999) \
            .load()

# query = socketDF.writeStream \
    # .outputMode("append") \
    # .format("console") \
    # .option("truncate", False) \
    # .start()

parsed_df = socketDF.selectExpr("split(value, ', ') AS data")

#query = parsed_df.writeStream \
       #.outputMode("append") \
       #.format("console") \
       #.option("truncate", False) \
       #.start()

parsedDF = parsed_df.selectExpr(
    "data[0] as vreme",
    "data[1] as naziv",
    "CASE WHEN data[2] = 'null' THEN NULL ELSE CAST(data[2] AS DOUBLE) END as
temperatura",
    "CASE WHEN data[3] = 'null' THEN NULL ELSE CAST(data[3] AS DOUBLE) END as
vlaznostV",
    "CASE WHEN data[4] = 'null' THEN NULL ELSE CAST(data[4] AS DOUBLE) END as
vlaznostZ"
)

parsedDFF = parsedDF.where("temperatura is NULL or vlaznostV is NULL or vlaznostZ
is null").select("naziv")

query = parsedDF.writeStream \
       #.outputMode("append") \
       #.format("console") \
       #.option("truncate", False) \
       #.start()

query = parsedDFF.writeStream \
       .outputMode("append") \
       .format("console") \
       .option("truncate", False) \
       .start()

query.awaitTermination()
```

Zadatak 2

Kreirati Apache Spark program koji čita informacije za 4 plastenika. Za svaki plastenik dobija se informacija o vremenu čitanja, imenu plastenika, temperaturi u plasteniku, vlažnosti vazduha i

vlažnosti zemljišta. Ukoliko vrednost sa senzora nije mogla da se pročita umesto vrenosti prosleđuje se null. Prikazati sve plastenike u kojima je temperature bila veća od 40 stepani a vlažnost vazduha veća od 70%.

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.master("local[2]")\
       .appName("SensorDataStreaming").getOrCreate()

socketDF = spark.readStream \
            .format("socket") \
            .option("host", "localhost") \
            .option("port", 9999) \
            .load()

parsed_df = socketDF.selectExpr("split(value, ', ') AS data")

parsedDF = parsed_df.selectExpr(
    "data[0] as vreme",
    "data[1] as naziv",
    "CASE WHEN data[2] = 'null' THEN NULL ELSE CAST(data[2] AS DOUBLE) END as
temperatura",
    "CASE WHEN data[3] = 'null' THEN NULL ELSE CAST(data[3] AS DOUBLE) END as
vlaznostV",
    "CASE WHEN data[4] = 'null' THEN NULL ELSE CAST(data[4] AS DOUBLE) END as
vlaznostZ"
)

parsedDFF = parsedDF.where("temperatura > 40 and vlaznostV > 70")

query = parsedDFF.writeStream \
       .outputMode("append") \
       .format("console") \
       .start()

query = parsedDFF.writeStream \
.format("csv") \
.option("path", "izlaz2") \
.option("checkpointLocation", "checkpoint2") \
.start()

query.awaitTermination()
```

Zadatak 3

Kreirati Apache Spark program koji čita informacije za 4 plastenika. Za svaki plastenik dobija se informacija o vremenu čitanja, imenu plastenika, temperaturi u plasteniku, vlažnosti vazduha i vlažnosti zemljišta. Ukoliko vrednost sa senzora nije mogla da se pročita umesto vrenosti prosleđuje se null. Informacije sa senzora šalju se na svakih 5 sekundi. Prikazati prosečna čitanja vrednosti senzora na 15 sekundi, bez preklapanja prozora.

```
from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from pyspark.sql.functions import window, avg, col

spark = SparkSession.builder.master("local[2]")\
.appName("SensorDataStreaming").getOrCreate()

socketDF = spark.readStream \
            .format("socket") \
            .option("host", "localhost") \
            .option("port", 9999) \
            .load()
```

```
parsed_df = socketDF.selectExpr("split(value, ', ') AS data")

parsedDF = parsed_df.selectExpr(
    "data[0] as vreme",
    "data[1] as naziv",
    "CASE WHEN data[2] = 'null' THEN NULL ELSE CAST(data[2] AS DOUBLE) END as
temperatura",
    "CASE WHEN data[3] = 'null' THEN NULL ELSE CAST(data[3] AS DOUBLE) END as
vlaznostV",
    "CASE WHEN data[4] = 'null' THEN NULL ELSE CAST(data[4] AS DOUBLE) END as
vlaznostZ"
)

streaming_df = parsedDF.withColumn("vreme", F.to_timestamp("vreme", "yyyy-MM-
dd'T'HH:mm:ss"))

windowed_df = streaming_df.withWatermark("vreme", "30 seconds") \
        .groupBy(window(col("vreme"), "15 seconds", "15 seconds"), col("naziv"))\
        .agg(avg("temperatura").alias("avg_temperatura"),
avg("vlaznostV").alias("avg_vlaznostV"), avg("vlaznostZ").alias("avg_vlaznostZ"))

query = windowed_df.writeStream \
.outputMode("complete") \
.format("console") \
.option("truncate", False) \
.start()

query.awaitTermination()
```