Analiza big-mobility podataka u Apache Spark-u

Danilo Milošević 1732

Kreiranje skupa podataka

- **Generisani podaci**:
  - Period: 3600 timestamp-a
  - Tipovi vozila: automobili, motori, autobusi, bicikli, pešaci
  - Podaci generisani za New York
- **Podaci se sastoje iz dva fajla:**
  - Emisije (izduvni gasovi)
  - Koordinate ulica (kojim ulicama su se kretali)
- **Veličina fajlova**:
  - Svaki fajl ima veličinu od 1 GB

Zadatak 1
Određivanje broja vozila po zonama

Uvod i postavke okruženja

Postavljanje okruženja:

- U prvom koraku podešavamo okruženja za PySpark korišćenjem **os.environ** za definisanje Python drivera.

```
import os
import sys

os.environ['PYSPARK_PYTHON'] = sys.executable
os.environ['PYSPARK_DRIVER_PYTHON'] = sys.executable
```

Pribavljanje argumenata

U funkciji get_args čitamo argumente komandne linije, koje ćemo kasnije koristiti u programu. Tu su definisana imena argumenata koje očekujemo u konzoli kao i njihovi tipovi. Takođe proverava da li su svi neophodni argumenti obezbeđeni.

Lista exp_arg sadrži tuple-ove sa imenima argumenta i njihovim tipom. Argumenti uključuju informacije kao što su koordinate, vreme, tip vozila, itd.

```
def get_args():
    exp_arg = [("master","string"),
               ("geo_x","float"),
               ("geo_y","float"),
               ("distance","float"),
               ("time_s","float"),
               ("time_e", "float"),
               ("type","string"),
               ("print","bool")]
```

Konverzija podataka:
Kada se argumenti dobiju iz konzole prevode se u svoje odgovarajuce tipove

```
def convert_to(data, type):
    if type=="int":
        return int(data)
    elif type=="float" or type=="float":
        return float(data)
    elif type=="bool":
        return data=="print"
    return data
```

Čitanje i validacija argumenata:
Argumenti se čitaju iz komandne linije (sys.argv). Proverava se da li je obezbeđen dovoljan broj argumenata. Argumenti se konvertuju i skladište u rečnik arg_dict.

```
args = sys.argv[1:]
print(args)
if len(args)< len(exp_arg)-1:
    print("Nedovoljno argumenata!")
    sys.exit(-1)

arg_dict={}
for i in range(len(args)):
    arg_dict[exp_arg[i][0]] = convert_to(args[i], exp_arg[i][1])
if not "type" in arg_dict.keys():
    arg_dict["type"] = None
if not "print" in arg_dict.keys():
    arg_dict["print"] = True

return arg_dict
```

Kreiranje Spark sesije i učitavanje podataka

Kreira se Spark sesija i učitavaju se podaci iz CSV fajla koji se nalazi u HDFS-u.

Kreiranje Spark sesije:

- Spark sesija se kreira sa zadatim master argumentom (local[*] za lokalno testiranje na svim threadovima.

Učitavanje podataka:

- Podaci se učitavaju iz CSV fajla koristeći spark.read.csv metodu, gde se specificira separator (sep=';') i putanja gde čuvamo fajl fcd.csv

```
spark = SparkSession.builder.master(arg_dict["master"]).appName("hdfs_city").getOrCreate()
start = time.time()
fcdData = spark.read.csv("hdfs://localhost:9000/data/fcd.csv", sep=';', inferSchema=True, header=True)
```

Filtriranje podataka

Podaci se filtriraju na osnovu vremenskog perioda, geografskih koordinata i opcionalno tipa vozila.

Filtriranje po vremenu i lokaciji:
Filtrira se tako da se uzimaju samo zapisi unutar zadatog vremenskog perioda (time_s do time_e) i unutar zadate distance od tačke (geo_x, geo_y).

```
rez = fcdData.filter((fcdData.timestep_time>=arg_dict["time_s"]) &
                     (fcdData.timestep_time<=arg_dict["time_e"]) &
                     (get_distance(fcdData.vehicle_x,fcdData.vehicle_y, arg_dict["geo_x"],arg_dict["geo_y"])<arg_dict["distance"]))
```

Filtriranje po tipu vozila:
Ako je zadan tip vozila, dodatno se filtriraju podaci po tom tipu.

```
if arg_dict["type"]!=None:
    rez = rez.filter(fcdData.vehicle_type.contains(arg_dict["type"]))
rez.show()
end = time.time()
```

Logovanje i prikaz rezultata

Funkcija print_output prikazuje rezultate filtriranja podataka u terminalu.
Prikazuje broj zapisa (rez.count()) i ispisuje prvih nekoliko redova rezultata (rez.show()).

```
def print_output(rez):
    print(rez.count())
    rez.show()
```

Logovanje parametara i vremena:
Parametri upita (npr. koordinate, vreme, tip vozila) i vreme izvršenja se zapisuju u datoteku logs.txt.

```
with open("logs.txt",'a') as f:
    time_s = arg_dict["time_s"]
    time_e = arg_dict["time_e"]
    tip = "Samo master" if arg_dict["master"] == "local" else "Cluster"
    params = "X: "+str(arg_dict["geo_x"])+\
             " Y: "+str(arg_dict["geo_y"])+\
             " D: "+str(arg_dict["distance"])
    if arg_dict["type"] !=None:
        params +=" Type: "+arg_dict["type"]
```

Zadatak 2
Statistika emisija

Postavljanje okruženja

Importovanje modula:

- Uvođenje potrebnih modula: SparkSession, functions as F, time, os, i sys.

Postavljanje okruženja za PySpark:

- Definisanje Python driver-a za rad sa Spark-om.

```
import time
import os
import sys

os.environ['PYSPARK_PYTHON'] = sys.executable
os.environ['PYSPARK_DRIVER_PYTHON'] = sys.executable
```

Funkcija get_args

- Kao i u prethodnom zadatku prvo pribavljamo argumente sa konzole
- Exp_arg sadrzi niz tuple-ova koji definišu očekivane argumente i njihove tipove
- Argumenti se čitaju, cast-uju i zatim čuvaju u dictionary-u.

```
def get_args():
    exp_arg = [ ("master","string"),
                ("time_s","float"),
                ("time_e", "float"),
                ("print","bool")]

    def convert_to(data, type):
        if type=="int":
            return int(data)
        elif type=="float" or type=="float":
            return float(data)
        elif type=="bool":
            return data=="print"
        return data

    args = sys.argv[1:]
    print(args)
    if len(args)< len(exp_arg)-1:
        print("Nedovoljno argumenata!")
        sys.exit(-1)

    arg_dict={}
    for i in range(len(args)):
        arg_dict[exp_arg[i][0]] = convert_to(args[i], exp_arg[i][1]
    if not "type" in arg_dict.keys():
        arg_dict["type"] = None
    if not "print" in arg_dict.keys():
        arg_dict["print"] = True

    return arg_dict
```

Određivanje min/max, proseka i std

Funkcija get_processing generiše listu agregatnih funkcija (minimum, maksimum, srednja vrednost i standardna devijacija) za zadato ime kolone. Time automatski generišemo funkcije za specifiranu kolonu u skupu podataka.

Kako kolone sadrže prefiks "vehicle_" to ne moramo ručno da unosimo, samo ime izduvnog gasa.

Funkcija vraća listu Spark SQL funkcija za minimum, maksimum, srednju vrednost i standardnu devijaciju za datu kolonu, sa odgovarajućim imenima (alias).

```
n = "vehicle_"+name

return [F.min(n).alias("min_"+name),
        F.max(n).alias("max_"+name),
        F.mean(n).alias("mean_"+name),
        F.stddev(n).alias("stddev_"+name)]
```

Obrada više kolona

Funkcija get_all_processing koristi funkciju get_processing za generisanje agregatnih funkcija za više kolona. Vraća listu svih agregatnih funkcija koje se primenjuju na zadate kolone.

Tako generisane agregatne funkcije se kasnije koriste za određivanje minimuma, maksimuma, proseka i standardne devijacije za sve kolone skupa podataka.

```
def get_all_processing(names):
    rez = []
    for name in names:
        for x in get_processing(name):
            rez.append(x)
    return rez
```

Kreiranje spark sesije i učitavanje podataka

Kreiranje Spark sesije:
Definisanje master čvora i ime aplikacije.

Učitavanje podataka:
Učitavanje CSV fajla iz HDFS-a sa zadatom šemom i separatorom.

```
spark = SparkSession.builder.master(arg_dict["master"]).appName("hdfs_city").getOrCreate()
start = time.time()
emsData = spark.read.csv("hdfs://localhost:9000/data/emissions.csv", sep=';', inferSchema=True, header=True)
```

Filtriranje i grupisanje podataka

Filtriranje po vremenskom periodu:
Filtriranje zapisa na osnovu vremenskog intervala.

Grupisanje i agregacija podataka:
Grupisanje po koloni vehicle_lane i primena agregatnih funkcija za emisije i druge parametre.

```
rez = emsData.filter((emsData.timestep_time>=arg_dict["time_s"]) &
                     (emsData.timestep_time<=arg_dict["time_e"]))\
                    .groupBy("vehicle_lane").agg(
                        *get_all_processing(["CO2","CO","HC","NOx","PMx","noise","fuel","electricity"])
                    )
```

Prikaz i logovanje rezultata

- Kao u prvom zadatku, na konzoli logujemo rezultate ukoliko je uključen print flag.
- Pored toga u logs.txt štampamo da li se izvršava na masteru ili clusteru kao i vreme izvršenja programa

```
with open("logs.txt",'a') as f:
    time_s = arg_dict["time_s"]
    time_e = arg_dict["time_e"]
    tip = "Samo master" if arg_dict["master"] == "local" else "Cluster"
    print("Zadatak 2, "+tip + " "+str(time_s)+"-"+str(time_e),file=f)
    print("\t Vreme: "+str(end-start)+" s", file=f)

if arg_dict["print"]:
    print_output(rez)
```

Vremena izvršenja

Zadatak1 -74 40 10

Jedan proces

	1	2	3	4	5	Prosek	STD
0-1200	111.8596802	108.3859243	105.8026011	106.785362	108.8900397	108.3447215	2.320990212
0-2400	107.5575154	106.0937114	107.2767134	108.0439541	111.1989486	108.0341686	1.909334916
0-3600	111.326587	110.4575393	106.9689426	106.0574784	107.4748726	108.457084	2.300721604

Cluster

	1	2	3	4	5	Prosek	STD
0-1200	96.3934381	93.10338807	93.56953359	94.76137471	93.38869166	94.24328523	1.35786699
0-2400	95.56738663	94.0015676	93.77035809	93.8135488	96.62371755	94.75531573	1.282145393
0-3600	94.73175788	95.32735777	97.00252557	98.07662582	97.69628787	96.56691098	1.470439578

Zadatak2

Jedan proces

	1	2	3	4	5	Prosek	STD
0-1200	106.5054684	97.89993477	102.5133958	99.94637895	98.17880392	101.0087964	3.580496857
0-2400	102.640183	110.4362311	103.4232664	103.946753	103.72753	104.8347927	3.170152836
0-3600	114.4299436	108.3794641	110.6586826	110.1452165	110.6674139	110.8561441	2.207148295

Cluster

	1	2	3	4	5	Prosek	STD
0-1200	65.48954058	64.24853086	66.39000392	65.44477677	66.2622633	65.56702309	0.8544041553
0-2400	66.46297669	67.7753706	65.60588574	68.03232408	66.89619756	66.95455093	0.9873307462
0-3600	69.54416299	69.02573299	68.0081625	69.24691653	69.04114151	68.9732233	0.5787038568

HVALA NA PAŽNJI