Prvi projekat iz Sistema za obradu velike količine podataka

Student: Danilo Veljović 1120

Katedra za računarstvo i informatiku
Univerzitet u Nišu, Elektronski fakultet

February 18, 2021

Sadržaj

Opis dataseta

Dataset koji je korišćen u izradi aplikacije je Citibike NYC dataset, dat na linku https://www.citibikenyc.com/system-data. Dataset ima prostorno vremenske podatke koji se odnose na jedno iznajmljivanje bicikla.
Podaci koje sadrži su informacije o početnoj stanici (ime, id, vremenske i prostorne koordinate), koliko je trajala vožnja, koji bicikl je iznajmljen, pol, godina rođenja, i subscription service tj koji je tip korisnika koji je vozio itd.

Korišćeni alati

Kod je pisan u programskom jeziku Java (v1.8), korišćen je i Apache Spark framework (v1.6) i Hadoop DFS (v2.6.4).

Karakteristični delovi implementacije

Na slici je dat testcase. Napisan je jednostavan filter kojim će se izdvojiti samo vožnje čija je krajnja stanica Washington park. Zatim će se za jedan atribut (trip Duration) izračunati prosečna, minimalna i maksimalna vrednost.

```
//test case: End station Washington Park
System.out.println("Washington park dest: ");

JavaRDD<Record> recordsWhoseEndStationIsWashingtonPark = rdd_records.filter(t -> t.getEndStationName().equals(args[0]));

recordsWhoseEndStationIsWashingtonPark.collect().forEach(System.out::println);

JavaRDD<Integer> tripDurationsToWashingtonPark = recordsWhoseEndStationIsWashingtonPark.map(Record::getTripDuration);

Integer min = tripDurationsToWashingtonPark.min(Comparator.naturalOrder());

Integer max = tripDurationsToWashingtonPark.max(Comparator.naturalOrder());

long avg = tripDurationsToWashingtonPark.reduce(Integer::sum) / tripDurationsToWashingtonPark.count();

String s = "Shortest trip to Washington park, longest trip to Washington park and average length of trips to Washington park: "
        + min + ", " + max + ", " + avg;
```

Figure: Svi polasci čija je krajnja stanica Washington park

Karakteristični delovi implementacije

Na slici je dat task 1. Napisan je map reduce job koji će samo da beleži polaske sa svake stanice. I zatim se za svaku stanicu broj polazaka upisuje u fajl. Zatim se nalazi stanica sa najviše i najmanje polazaka, kao prosečan broj polazaka sa svake stanice.

```
//task 1: number of departures from a station
JavaPairRDD<String, Integer> numberOfDeparturesFromEachStation = rdd_records
        .mapToPair(i -> new Tuple2<>(i.getStartStationName(), 1))
        .reduceByKey(Integer::sum);

numberOfDeparturesFromEachStation.collect().forEach(t -> {
    try {
        fw.write( str: "Station: " + t._1() + " number of departures " + t._2() + " \n");
        fw.flush();
    } catch (IOException e) {
        e.printStackTrace();
    }
});

Integer maxStationAndNumberOfDepartures = numberOfDeparturesFromEachStation.map(t -> t._2).max(Comparator.naturalOrder());

Integer minStationAndNumberOfDepartures = numberOfDeparturesFromEachStation.map(t -> t._2).min(Comparator.naturalOrder());

Integer reduce = numberOfDeparturesFromEachStation.map(i -> i._2).reduce(Integer::sum);
double avgNumOfDeparturesFromEachStations = reduce / (double) numberOfDeparturesFromEachStation.count();
```

Karakteristični delovi implementacije

Na slici je dat task 2. Napisan je map reduce job koji će samo da beleži pristizanja na svaku stanicu. I zatim se za svaku stanicu broj pristizanja upisuje u fajl. Zatim se nalazi stanica sa najviše i najmanje dolazaka, kao prosečan broj dolazaka sa svake stanice.

```
//task 2: number of arrivals to a station
JavaPairRDD<String, Integer> numberOfArrivalsToEachStation = rdd_records
        .mapToPair(i -> new Tuple2<>(i.getEndStationName(), 1))
        .reduceByKey(Integer::sum);

fw.write( str: "\n \n");

numberOfArrivalsToEachStation.collect().forEach(t -> {
    try {
        fw.write( str: "Station: " + t._1() + " number of arrivals " + t._2() + " \n");
        fw.flush();
    } catch (IOException e) {
        e.printStackTrace();
    }
});

Integer maxStationAndNumberOfArrivals = numberOfArrivalsToEachStation.map(t -> t._2).max(Comparator.naturalOrder());

Integer minStationAndNumberOfArrivals = numberOfArrivalsToEachStation.map(t -> t._2).min(Comparator.naturalOrder());

Integer reduced = numberOfDeparturesFromEachStation.map(i -> i._2).reduce(Integer::sum);
double avgNumOfArrivalsFromEachStations = reduced / (double) numberOfArrivalsToEachStation.count();
```

Figure: Svi dolasci na stanicu

Karakteristični delovi implementacije

```
//task 3: number departures from each station at a certain time for every day of the week - may have null fields, should be ch...
//TODO: Same can be done for arrivals

JavaPairRDD<String, Integer> numberOfDeparturesFromEachStationAtACertainTimeAndForEveryDayOfTheWeek = rdd_records
        .filter(t -> !t.getStartTime().equals(LocalDateTime.MAX)) JavaRDD<Record>
        .mapToPair(t -> new Tuple2<>(t.getStartStationName(), t.getStartTime())) JavaPairRDD<String, LocalDateTime>
        .filter(t -> t._2().getHour() == 12)
        .mapToPair(t -> new Tuple2<>(t._1() + " " + t._2().getDayOfWeek().name(), 1)) JavaPairRDD<String, Integer>
        .reduceByKey(Integer::sum);

fw.write( str: "\n \n");

numberOfDeparturesFromEachStationAtACertainTimeAndForEveryDayOfTheWeek.collect().forEach(t -> {
    try {
        fw.write( str: "Station: " + t._1() + " number of arrivals " + t._2() + " at 12 o'clock \n");
        fw.flush();
    } catch (IOException e) {
        e.printStackTrace();
    }
});

Integer maxNumberOfDeparturesForEachStationAtCertainTimeAndEveryDay =
        numberOfDeparturesFromEachStationAtACertainTimeAndForEveryDayOfTheWeek.map(t -> t._2).max(Comparator.naturalOrder());

Integer minNumberOfDeparturesForEachStationAtCertainTimeAndEveryDay =
        numberOfDeparturesFromEachStationAtACertainTimeAndForEveryDayOfTheWeek.map(t -> t._2).min(Comparator.naturalOrder());

Integer reduc = numberOfDeparturesFromEachStation.map(i -> i._2).reduce(Integer::sum);
double avgNumberOfDeparturesForEachStationAtCertainTimeAndEveryDay = reduc
        / (double) numberOfDeparturesFromEachStationAtACertainTimeAndForEveryDayOfTheWeek.count();
```

Karakteristični delovi implementacije

Na slici na prethodnom slajdu je dat task 3. U ovom tasku se najpre izvlače svi polasci sa stanica i vreme polaska, izdvajaju se sve stanice čije je vreme polaska u 12, i nalazi se za svaku stanicu koliko je polazaka bilo u 12 časova.

Karakteristični delovi implementacije

```
//task 4: most popular bike for each station and length of each trip (get consumer info) at daytime ( > 17:00)
JavaRDD<Record> recordsWhoseStartTimeIsGreaterThen17 = rdd_records
        .filter(t -> !t.getStartTime().equals(LocalDateTime.MAX))
        .filter(t -> t.getStartTime().getHour() < 17);

JavaPairRDD<String, Integer> mostPopularDayTimeBikesForEachStation = recordsWhoseStartTimeIsGreaterThen17
        .mapToPair(t -> new Tuple2<>(t.getStartStationName() + " " + t.getBikeId(), 1))
        .reduceByKey(Integer::sum)
        .reduceByKey((a, b) -> a > b ? a : b);

fw.write( str: "\n \n");

mostPopularDayTimeBikesForEachStation.collect().forEach(t -> {
    try {
        fw.write( str: "Station: " + t._1() + " bike " + t._2() + " \n");
        fw.flush();
    } catch (IOException e) {
        e.printStackTrace();
    }
});

Integer mostPopular = mostPopularDayTimeBikesForEachStation.map(t -> t._2).max(Comparator.naturalOrder());

Integer leastPopular = mostPopularDayTimeBikesForEachStation.map(t -> t._2).min(Comparator.naturalOrder());

Integer sum = mostPopularDayTimeBikesForEachStation.map(i -> i._2).reduce(Integer::sum);
double avgBike = sum / (double) mostPopularDayTimeBikesForEachStation.count();
```

Karakteristični delovi implementacije

Na slici na prethodnom slajdu je dat task 4. U ovom delu koda se pronalazi najpopularniji bicikl za svaku stanicu, gde je vreme polaska pre 17:00. Zatim se nalazi minimalna, maksimalna i srednja vrednost za par stanica + bicikl.

Karakteristični delovi implementacije

Na slici je dat task 5. Ovaj task pronalazi broj polazaka sa svake stanice u podne, gde je stanica polaska istočno od Lafayette Ave St James Pl.

```
//task 5: number departures from each station at noon where the start station is east of the Lafayette Ave & St James Pl,
//id = 293, lat = 40.73020660529954, long = -73.99102628231049)
JavaPairRDD<String, Integer> eastOfLafayetteDepartures = rdd_records
        .filter(t -> !t.getStartTime().equals(LocalDateTime.MAX))
        .filter(t -> t.getStartStationLongitude().compareTo(BigDecimal.valueOf(-73.99102628231049)) > 0)
        .filter(t -> t.getStartTime().getHour() == 12)
        .mapToPair(t -> new Tuple2<>(t.getStartStationName(), 1))
        .reduceByKey(Integer::sum);

fw.write( str: "\n \n");

eastOfLafayetteDepartures.collect().forEach(t -> {
    try {
        fw.write( str: "EAST OF LAFAYETTE : Station: " + t._1() + " number of departures " + t._2() + " at 12 o'clock\n");
        fw.flush();
    } catch (IOException e) {
        e.printStackTrace();
    }
});

Integer mostDepartures = eastOfLafayetteDepartures.map(t -> t._2).max(Comparator.naturalOrder());

Integer leastDepartures = eastOfLafayetteDepartures.map(t -> t._2).min(Comparator.naturalOrder());

Integer sumDep = eastOfLafayetteDepartures.map(i -> i._2).reduce(Integer::sum);
double avgDepartures = sumDep / (double) eastOfLafayetteDepartures.count();
```

Karakteristični delovi implementacije

Na slici je dat task 6. Ovaj task pronalazi broj polazaka sa svake stanice u podne, gde je stanica polaska južno od Lafayette Ave St James Pl.

```
//task 6: number departures from each station at noon where the start station is south of Lafayette Ave & St James Pl

JavaPairRDD<String, Integer> southOfLafayetteDepartures = rdd_records
        .filter(t -> !t.getStartTime().equals(LocalDateTime.MAX))
        .filter(t -> t.getStartStationLatitude().compareTo(BigDecimal.valueOf(40.73020660529954)) < 0)
        .filter(t -> t.getStartTime().getHour() == 12)
        .mapToPair(t -> new Tuple2<>(t.getStartStationName(), 1))
        .reduceByKey(Integer::sum);

fw.write( str: "\n \n");

southOfLafayetteDepartures.collect().forEach(t -> {
    try {
        fw.write( str: "SOUTH OF LAFAYETTE : Station: " + t._1() + " number of departures " + t._2() + " \n");
        fw.flush();
    } catch (IOException e) {
        e.printStackTrace();
    }
});

Integer mostSouthDepartures = southOfLafayetteDepartures.map(t -> t._2).max(Comparator.naturalOrder());

Integer leastSouthDepartures = southOfLafayetteDepartures.map(t -> t._2).min(Comparator.naturalOrder());

double avgNumSouthDep = southOfLafayetteDepartures.map(i -> i._2).reduce(Integer::sum) / (double) southOfLafayetteDepartures.count();
```

Karakteristični delovi implementacije

Na slici je dat task 7. Ovaj task pronalazi pol ljudi koji iznajmljuju bicikle leti (jun, jul i avgust) u Washington parku.

```
//task 7: get gender of people who rent bikes in the summer at Washington Park (id = 262) (summer = June, July, August)

List<Integer> summerMonths = Arrays.asList(6, 7, 8);

JavaPairRDD<Integer, Integer> gendersWhoRentBikesInSummer = rdd_records
        .filter(t -> summerMonths.contains(t.getStartTime().getMonthValue()))
        .filter(t -> t.getStartStationId().equals(Long.parseLong(args[3])))
        .mapToPair(t -> new Tuple2<>(t.getGender(), 1))
        .reduceByKey(Integer::sum);

gendersWhoRentBikesInSummer.foreach(t -> System.out.println());

fw.write( str: "\n \n");

gendersWhoRentBikesInSummer.collect().forEach(t -> {
    try {
        fw.write( str: "GENDERS : GENDER : " + t._1() + " number " + t._2() + " \n");
        fw.flush();
    } catch (IOException e) {
        e.printStackTrace();
    }
});

Integer mostGenders = southOfLafayetteDepartures.map(t -> t._2).max(Comparator.naturalOrder());

Integer leastGenders = southOfLafayetteDepartures.map(t -> t._2).min(Comparator.naturalOrder());

double avgGenders = southOfLafayetteDepartures.map(i -> i._2).reduce(Integer::sum) / (double) southOfLafayetteDepartures.count();
```

The End