Drugi projekat iz Sistema za obradu velike količine podataka

Danilo Veljović, 1120

Katedra za računarstvo i informatiku
Univerzitet u Nišu, Elektronski fakultet

February 18, 2021

Sadržaj

Opis baze

Cassandra je distribuirana baza podataka koja ispunjava AP uslove, CAP teoreme. Kao takva je pogodna za korišćenje u distribuiranim aplikacijama koje se zahtevaju high availability, ali eventualnu konzistentnost. U ovoj aplikaciji se koristila Cassandra u kojoj je bio kreiran jedan keyspace *citibike*, i jedna tabela *citibike*. Ta tabela ima kolone id, lokaciju, min, max, avg, i message koja predstavlja neku poruku. Na slici na sledećem slajdu je dat deo koda koji služi kao konektor aplikacije na bazu.

Cassandra Connector

```java
public void connect(String node, Integer port) {
    Cluster.Builder b = Cluster.builder().addContactPoint(node);
    if (port != null) {
        b.withPort(port);
    }
    cluster = b.build();

    session = cluster.connect();
}

public Session getSession() { return this.session; }

public void close() {
    session.close();
    cluster.close();
}

public void connect() {
    CassandraConnector client = new CassandraConnector();
    client.connect( node: "localhost",  port: 9042);
    this.session = client.getSession();
    System.out.println("CONNECTED!");
}
```

Kafka Producer

Kafka producer će čitati podatke na par sekundi i slati ih na topic. Deo koda koji to ilustruje je dat na slici.

```
while(line != null){

    ProducerRecord<String, String> rec = new ProducerRecord<>(topic,  line);

    kafkaProducer.send(rec);

    System.out.println("[PRODUCER] Sent message: " + line);
    //LOGGER.debug("[PRODUCER] Sent message: " + line);
    line = reader.readLine();

    Thread.sleep(millisToSleep);
}
```

Kafka Listener

Boilerplate deo koda Kafka Listener/Analyzer klase će biti izostavljen (konektovanje na bazu, konektovanje na topic itd). U narednim slajdovima biće samo dat deo koda koji ilustruje različite map-reduce jobove koji se izvršavaju.

Kafka Listener

Prvi job koji se izvršava jeste testcase job koji samo filtrira sve stanice kojima je odredišna stanica Washington park.Nalazi se zatim min, max i avg vrednost trajanja puta. A onda se za za tako filtriran skup nalazi user type koji je najčešće dolazio na tu stanicu (takođe i min i avg).

```
JavaRDD<Record> recordsWhoseEndStationIsWashingtonPark = rdd_records.filter(t -> t.getEndStationName().equals("Washington Park"));

if (!recordsWhoseEndStationIsWashingtonPark.isEmpty()) {
    JavaRDD<Integer> tripDurationsToWashingtonPark = recordsWhoseEndStationIsWashingtonPark.map(Record::getTripDuration);

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

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

    long avg = tripDurationsToWashingtonPark.reduce(SUM_REDUCER) / 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;

    connector.insertInto( location: "Washington Park", min, max, (double) avg, s);

    JavaPairRDD<String, Integer> userTypeRDD =
            recordsWhoseEndStationIsWashingtonPark.mapToPair(t -> new Tuple2<>(t.getUserType(), 1)).reduceByKey(Integer::sum);

    Integer min1 = userTypeRDD.map(Tuple2::_2).min(Comparator.naturalOrder());

    Integer max1 = userTypeRDD.map(Tuple2::_2).max(Comparator.naturalOrder());

    Long avg1 = userTypeRDD.map(Tuple2::_2).reduce(SUM_REDUCER) / userTypeRDD.count();

    connector.insertInto( location: "Washington Park", min1, max1, (double) avg1, message: "most popular bike");
}
```

Kafka Listener

Sledeći job koji se izvršava je job koji uzima stanice koje u tom batch koji je pročitan sa topica imaju više od pet polazaka. I za takve podatke se onda nalazi min, max i avg.

```
//task 1: all stations for which there are more than 5 departures
JavaPairRDD<String, Integer> numberOfDeparturesFromEachStation = rdd_records
        .mapToPair(i -> new Tuple2<>(i.getStartStationName(), 1))
        .reduceByKey(Integer::sum)
        .filter(t -> t._2 > 5);

if (!numberOfDeparturesFromEachStation.isEmpty()) {
    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();

    connector.insertInto( location: null, minStationAndNumberOfDepartures,
            maxStationAndNumberOfDepartures, avgNumOfDeparturesFromEachStations,  message: "Departures from any station");

}
```

Kafka Listener

Sledeći task koji se izvršava jeste da u tom batchu koji je primljen filtrira one čija je startna lokacija istočno od Lafayette Ave St James Pl. Nalazi se najpopularniji bicikl među takvim podacima (takođe i najmanje popularni bicikl) i srednja vrednost izjamljivanja bicikala.

```
//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)
JavaRDD<Record> eastOfLafayetteDepartures = rdd_records
        .filter(t -> !t.getStartTime().equals(LocalDateTime.MAX))
        .filter(t -> t.getStartStationLongitude().compareTo(BigDecimal.valueOf(-73.99102628231049)) > 0);

if (!eastOfLafayetteDepartures.isEmpty()) {
    JavaPairRDD<Long, Integer> mostPopularBikesEastOfLafayette = eastOfLafayetteDepartures.mapToPair(t ->
            new Tuple2<>(t.getBikeId(), 1)).reduceByKey(Integer::sum);

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

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

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

    connector.insertInto( location: "East of Lafayette", leastDepartures, mostDepartures, avgDepartures,  message: null);

}
```

Kafka Listener

U jobu na slici ispod se filtriraju svi recordi iz batcha gde je startna stanica južno od Lafayette Ave St James Pl. Takođe se i za ove redove uzima dužina putovanja i nalaze najduže, najkraće i prosečna vrednost putovanja.

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

JavaRDD<Record> southOfLafayetteDepartures = rdd_records
        .filter(t -> !t.getStartTime().equals(LocalDateTime.MAX))
        .filter(t -> t.getStartStationLatitude().compareTo(BigDecimal.valueOf(40.73020660529954)) < 0);

if (!southOfLafayetteDepartures.isEmpty()) {
    JavaRDD<Integer> tripDurations = southOfLafayetteDepartures.map(Record::getTripDuration);

    Integer mostSouthDepartures = tripDurations.max(Comparator.naturalOrder());

    Integer leastSouthDepartures = tripDurations.min(Comparator.naturalOrder());

    double avgNumSouthDep = tripDurations.reduce(Integer::sum) / (double) tripDurations.count();

    connector.insertInto( location: "South of Lafayette", leastSouthDepartures, mostSouthDepartures, avgNumSouthDep,  message: null);

}
```

Kafka Listener

U nastavku je dato top N kombinacija početna-krajnja stanica, top N startnih lokacija, top N krajnjih lokacija i top N bicikala. Na ovoj slici je dato top N kombinacija početnih i krajnjih stanica.

```
//b. top N start locations and top N end locations and top N bikes

JavaPairRDD<String, Integer> topNLocations = rdd_records
        .mapToPair(t -> new Tuple2<>(t.getStartStationName() + " - " + t.getEndStationName(), 1))
        .reduceByKey(Integer::sum)
        .sortByKey();
List<Tuple2<String, Integer>> listOfLocations = topNLocations.collect();
List<Tuple2<String, Integer>> sortable = new ArrayList<>(listOfLocations);
Comparator<Tuple2<String,  Integer>> comparator = (tupleA, tupleB) -> tupleB._2().compareTo(tupleA._2());

sortable.sort(comparator);

for (int i = 0; i < N && !sortable.isEmpty() && sortable.get(i) !=null; i++) {
    connector.insertInto(sortable.get(i)._1,  min: null,  max: null,  avg: null,  message: "Number of occurrences " + sortable.get(i)._2);
}
```

Kafka Listener

Na ovoj slici je dato top N kombinacija početnih stanica.

```
List<Tuple2<String, Integer>> mostCommonStartStation = rdd_records
        .mapToPair(t -> new Tuple2<>(t.getStartStationName(), 1)).reduceByKey(Integer::sum).collect();

sortable.clear();
sortable.addAll(mostCommonStartStation);

sortable.sort(comparator);

for (int i = 0; i < N && !sortable.isEmpty() && sortable.get(i) !=null; i++) {
    connector.insertInto(sortable.get(i)._1,  min: null,  max: null,  avg: null,  message: "Number of occurrences " + sortable.get(i)._2);
}
```

Kafka Listener

Na ovoj slici je dato top N kombinacija krajnih stanica.

```
List<Tuple2<String, Integer>> mostCommonEndStation = rdd_records
        .mapToPair(t -> new Tuple2<>(t.getEndStationName(), 1)).reduceByKey(Integer::sum).collect();

sortable.clear();
sortable.addAll(mostCommonEndStation);

sortable.sort(comparator);

for (int i = 0; i < N && !sortable.isEmpty() && sortable.get(i) !=null; i++) {
    connector.insertInto(sortable.get(i)._1, min: null, max: null, avg: null, message: "Number of occurrences " + sortable.get(i)._2);
}
```

Kafka Listener

Na slici je dat job koji nalazi bicikl koji je najviše puta bio korišćen u tom batchu koji je pročitan iz topica, bicikl koji je najmanje puta iznajmljen iz topica i prosečan broj iznajmljivanja bicikala.

```
List<Tuple2<Long, Integer>> topNBikes = rdd_records
        .mapToPair(t -> new Tuple2<>(t.getBikeId(), 1)).reduceByKey(Integer::sum).collect();

List<Tuple2<Long, Integer>> sorted = new ArrayList<>(topNBikes);

sorted.sort((a, b) ->  b._2().compareTo(a._2()));

for (int i = 0; i < N && !sorted.isEmpty() && sorted.get(i) !=null; i++) {
    connector
            .insertInto( location: sorted.get(i)._1+"", min: null, max: null, avg: null, message: "Number of occurrences " +
                    sortable.get(i)._2);
}
```

The End