Raspodijeljena obrada velike količine podataka

4. Domaća zadaća

Filip Gulan - JMBAG: 0036479428

1. zadatak

1. 97 ulaznih datoteka.
2. Broj zapisa - 4 726 643
3. Veličina izlazne datoteke - 411,4MB.

Izvorni kod

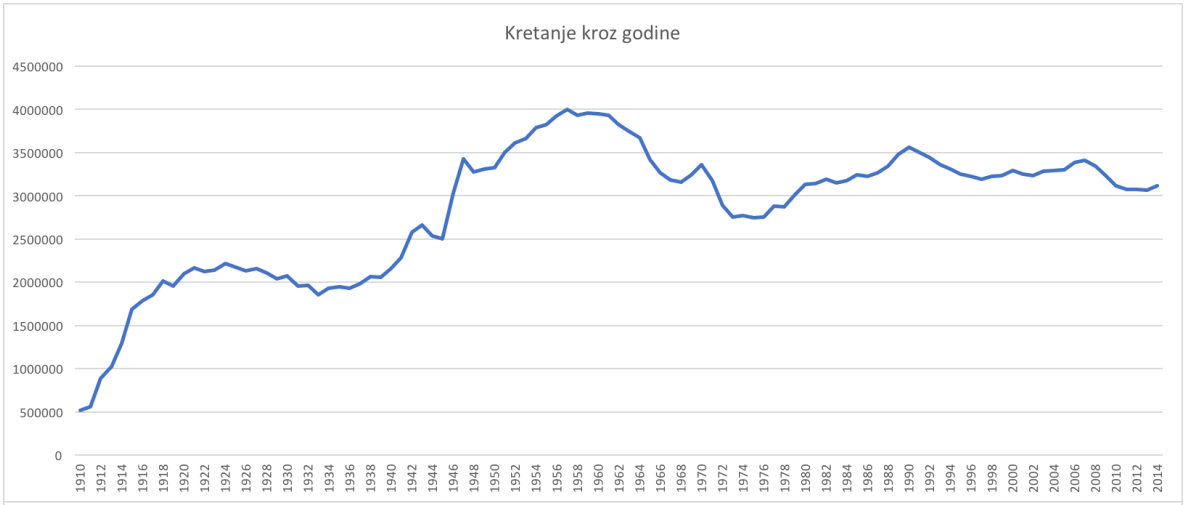
Task1.java

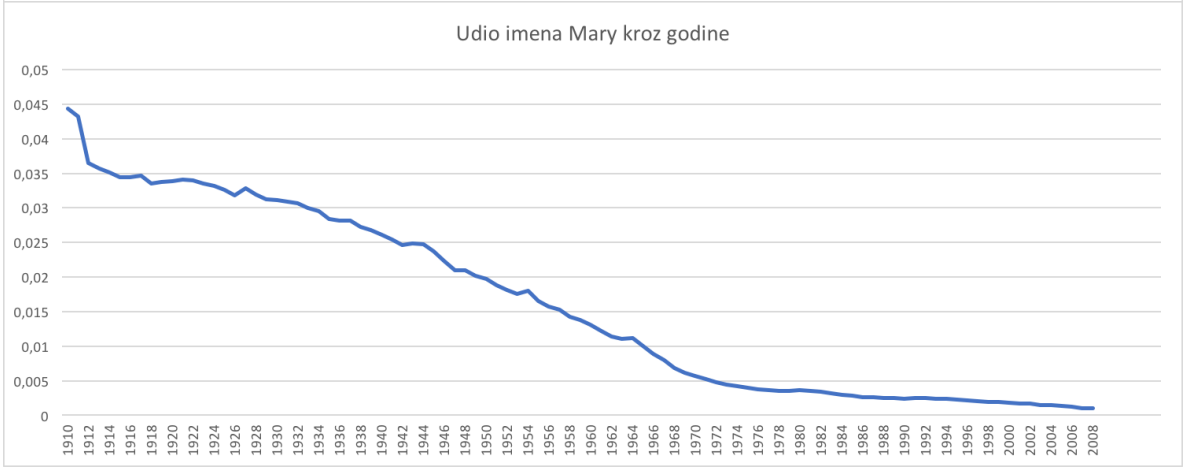
```
public class Task1 {

    private static final String INPUT_FOLDER =
"/Users/filipgulan/Downloads/sensorscope-monitor";
    private static final String OUTPUT_FILE = "senesorscope-monitor-
all.csv";

    public static void main(String[] args) throws IOException {
        File[] files = new File(INPUT_FOLDER).listFiles();
        Stream<String> stream = null;
        for (File file : files) {
            if (file.isFile() && file.getName().endsWith("txt")) {
                if (stream == null) {
                    stream = Files.lines(Paths.get(file.getAbsolutePath()));
                } else {
                    stream = Stream.concat(stream,
Files.lines(Paths.get(file.getAbsolutePath())));
                }
            }
        }

        Stream<String> resultStream = stream
                .map(line -> line.split("\\s+"))
                .filter(items -> SensorscopeReading.isParsable(items))
                .map(items -> new SensorscopeReading(items))
                .parallel()

.sorted(Comparator.comparingInt(SensorscopeReading::getTimestamp))
                .map(reading -> reading.toCSVLine());

        Files.write(Paths.get(OUTPUT_FILE),
                (Iterable<String>)resultStream::iterator);
    }
}
```

SensorscopeReading.java

```
public class SensorscopeReading {

    private String[] items;
    private Integer timestamp;

    public SensorscopeReading(String[] items) {
        this.items = items;
        this.timestamp = Integer.parseInt(items[7]);
    }

    public Integer getTimestamp() {
        return timestamp;
    }

    public String toCSVLine() {
        return Joiner.on(",").join(items);
    }

    public static boolean isParsable(String[] items) {
        try {
            Integer.parseInt(items[0]);
            Integer.parseInt(items[1]);
            Integer.parseInt(items[2]);
            Integer.parseInt(items[3]);
            Integer.parseInt(items[4]);
            Integer.parseInt(items[5]);
            Integer.parseInt(items[6]);
            Integer.parseInt(items[7]);
            Float.parseFloat(items[8]);
            Float.parseFloat(items[9]);
            Float.parseFloat(items[10]);
            Float.parseFloat(items[11]);
            Float.parseFloat(items[12]);
            Float.parseFloat(items[13]);
            Float.parseFloat(items[14]);
            Float.parseFloat(items[15]);
            Float.parseFloat(items[16]);
            Float.parseFloat(items[17]);
            Float.parseFloat(items[18]);

            return true;
        } catch (Exception e) {
            return false;
        }
    }
}
```

2. zadatak

1. Najnepopularnije žensko ime - Anaissa.
2. 10 najpopularnijih muških imena - James, John, Robert, Michael, William, David, Richard, Joseph, Charles, Thomas.
3. Najviše djece rođeno 1946. je u državi New York.
4. 

5. 

6. Ukupno rođeno djece - 298883326.
7. Broj različitih imena - 30274

Izvorni kod

Task2.java

```
public class Task2 {

    private static final String INPUT_FILE =
"/Users/filipgulan/Downloads/StateNames.csv";

    public static void main(String[] args) {
        SparkConf conf = new SparkConf().setAppName("Task2");
        try {
            conf.get("spark.master");
        } catch (NoSuchElementException ex) {
            conf.setMaster("local");
        }
        JavaSparkContext sc = new JavaSparkContext(conf);

        JavaRDD<String> lines = sc.textFile(INPUT_FILE);
```

```
        JavaRDD<USBabyNameRecord> result = lines
                .map(line -> line.split(","))
                .filter(items -> USBabyNameRecord.isParsable(items))
                .map(items -> new USBabyNameRecord(items))
                .cache();
        printUnpopularFemaleName(result);
        printMostPpularMaleNames(result);
        printBestCountry(result, 1946);
        printYearStatistic(result);
        printNameStatistic(result, "Mary");
        printSum(result);
        printDistinctNames(result);
    }

    private static void printDistinctNames(JavaRDD<USBabyNameRecord> result)
{
        Long sum = result.map(record ->
record.getName()).distinct().count();
        System.out.println("Distinct names: " + sum);
    }

    private static void printSum(JavaRDD<USBabyNameRecord> result) {
        Integer sum = result.map(record -> record.getCount()).reduce((x, y)
-> x + y);
        System.out.println("Sum: " + sum);
    }

    private static void printNameStatistic(JavaRDD<USBabyNameRecord> result,
String name) {
        Map<Integer, Integer> records = result
                .mapToPair(record -> new Tuple2<>(record.getYear(),
record.getCount()))
                .reduceByKey((x, y) -> x + y)
                .mapToPair(x -> x.swap()).sortByKey(false).mapToPair(x ->
x.swap()).collectAsMap();

        List<Tuple2<Integer, Double>> nameRecords = result
                .filter(record -> record.getName().equals(name))
                .mapToPair(record -> new Tuple2<>(record.getYear(),
record.getCount()))
                .reduceByKey((x, y) -> x + y)
                .map(tuple -> new Tuple2<>(tuple._1,
                        new Double(tuple._2) /
                                new Double(records.getOrDefault(tuple._1,
tuple._2)))).collect();

        nameRecords.forEach(record -> {
            System.out.println(record._1 + "," + record._2);
```

```
        });
    }

    private static void printYearStatistic(JavaRDD<USBabyNameRecord> result)
{
        List<Tuple2<Integer, Integer>> records = result
                .mapToPair(record -> new Tuple2<>(record.getYear(),
record.getCount()))
                .reduceByKey((x, y) -> x + y)
                .mapToPair(x -> x.swap()).sortByKey(false).mapToPair(x ->
x.swap()).collect();
        records.forEach(record -> {
            System.out.println(record._1 + "," + record._2);
        });
    }

    private static void printBestCountry(JavaRDD<USBabyNameRecord> result,
int year) {
        JavaPairRDD records = result
                .filter(record -> record.getYear() == year)
                .mapToPair(record -> new Tuple2<>(record.getState(),
record.getCount()))
                .reduceByKey((x, y) -> x + y)
                .mapToPair(x -> x.swap()).sortByKey(false).mapToPair(x ->
x.swap());
        System.out.println("State with most children: " +
records.first()._1);

    }

    private static void printMostPpularMaleNames(JavaRDD<USBabyNameRecord>
result) {
        List<Tuple2<String, Integer>> records = result
                .filter(record -> !record.isFemale())
                .mapToPair(record -> new Tuple2<>(record.getName(),
record.getCount()))
                .reduceByKey((x, y) -> x + y)
                .mapToPair(x -> x.swap()).sortByKey(false).mapToPair(x ->
x.swap()).take(10);
        records.forEach(record -> {
            System.out.println(record._1 + " - " + record._2);
        });
    }

    private static void printUnpopularFemaleName(JavaRDD<USBabyNameRecord>
result) {
        JavaPairRDD rdd = result
                .filter(record -> record.isFemale())
```

```
                .mapToPair(record -> new Tuple2<>(record.getName(),
record.getCount()))
                .reduceByKey((x, y) -> x + y)
                .mapToPair(x -> x.swap()).sortByKey(true).mapToPair(x ->
x.swap());
        System.out.println("Most unpopular female name: " + rdd.first()._1);
    }
}
```

USBabyNameRecord.java

```
public class USBabyNameRecord {

    private Integer id;
    private String name;
    private Integer year;
    private String gender;
    private String state;
    private Integer count;
    private boolean female;

    public USBabyNameRecord(String[] items) {
        this.id = Integer.parseInt(items[0]);
        this.name = items[1];
        this.year = Integer.parseInt(items[2]);
        this.gender = items[3];
        this.state = items[4];
        this.count = Integer.parseInt(items[5]);
        this.female = gender.equals("F");
    }

    public Integer getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public Integer getYear() {
        return year;
    }

    public String getGender() {
        return gender;
    }

    public Boolean isFemale() {
        return female;
```

```
    }

    public String getState() {
        return state;
    }

    public Integer getCount() {
        return count;
    }

    public static boolean isParsable(String[] items) {
        try {
            Integer.parseInt(items[0]);
            Integer.parseInt(items[2]);
            Integer.parseInt(items[5]);
            return true;
        } catch (Exception e) {
            return false;
        }
    }
}
```

3. zadatak

1. Novi direktorij nastaje svakih 10 sekundi.
2. Izračun se pokreće svakih 10 sekundi.
3. Može, jer je ulazna datoteka sortirana po vremenu, ne po solarPanelCurrent.
4. Prvih par vrijednosti su iste. Razlog tomu je veća veličina windowa, nego slide duration.

Task3.java

```java
public class Task3 {

    private static final String OUTPUT = "output";

    public static void main(String[] args) {
        SparkConf conf = new SparkConf().setAppName("Task3");
        try {
            conf.get("spark.master");
        } catch (NoSuchElementException ex) {
            conf.setMaster("local");
        }

        JavaStreamingContext jssc = new JavaStreamingContext(conf,
Durations.seconds(5));
        JavaDStream<String> records = jssc.socketTextStream("localhost",
                SensorStreamGenerator.PORT);

        JavaPairDStream<Integer, Double> result = records
                .map(line -> line.split("\\s+"))
                .filter(SensorscopeReading::isParsable)
                .map(SensorscopeReading::new)
                .mapToPair(record -> new Tuple2<>(record.getStationID(),
record.getSolarCurrent()))
                .reduceByKeyAndWindow(Math::max, Durations.seconds(60),
Durations.seconds(10));

        result.dstream().saveAsTextFiles(OUTPUT, "txt");
        jssc.start();
        jssc.awaitTermination();
    }
}
```