Raspodijeljena obrada velike količine podataka

2. Domaća zadaća

Filip Gulan - JMBAG: 0036479428

Velika većina koda se dijeli između zadataka stoga su razredi koji se korsite kroz ostale zadatke navedeni u izvornom kodu prvog zadatka. Također, treći zadatak koristi pomoćne `getJob` metode prvog i drugog zadatka.

1. zadatak

1. U ulaznoj datoteci se nalazi 9834 različitih vozila.
2. Najdulja ukupna vožnja jednog taksija je trajala 225600 sekundi. Minimalna vožnja tog taksija je 0 sekundi, a najdulja 3360 sekundi.
3. Kako je u ovom zadatku funkcionalnost `Combinera` bila identična onoj od `Reducera` bilo je potrebno sam u `main` metodi dodati liniju `job.setCombinerClass(TripTimeReducer.class);`.
4. Program bez `Combinera` se izvodio `30.63` sekunde, dok s istim se izvodio `23.62` sekunde. Da, to je u skladu s očekivanjima jer manje se podataka slalo na `Reducer`, no ta razlika nije toliko značajna jer koristimo Hadoop u pseudo-raspodijeljenom načinu radu. Razlika bi bila značajnija ukoliko bi se program izvršavao u pravom raspodijeljenom načinu radu.

Izvorni kod

Task1.java

```java
public class Task1 {

    private static final String INPUT_PATH = "/user/rovkp/trip_data.csv";
    private static final String OUTPUT_PATH = "/user/rovkp/trip_result";

    public static void main(String[] args) throws IOException,
            ClassNotFoundException, InterruptedException {
        long startTime = System.nanoTime();
        Job job = getJob(INPUT_PATH, OUTPUT_PATH, Task1.class);
        job.waitForCompletion(true);
        System.out.println("Total time: " + (System.nanoTime() - startTime)/10e8 + '
    }

    public static Job getJob(String inputPath, String outputPath, Class<?> cls)
            throws IOException {
        Job job = Job.getInstance();
        job.setJarByClass(cls);
        job.setJobName("TripTime");
        job.setNumReduceTasks(1);

        FileInputFormat.addInputPath(job, new Path(inputPath));
        FileOutputFormat.setOutputPath(job, new Path(outputPath));

        job.setMapperClass(TripTimeMapper.class);
        job.setCombinerClass(TripTimeReducer.class);
        job.setReducerClass(TripTimeReducer.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(TripTimeTuple.class);
        return job;
    }
}
```

TripTimeMapper.java

```
public class TripTimeMapper extends Mapper<LongWritable, Text, Text, TripTimeTuple>

    @Override
    protected void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        // First line is CSV header
        String row = value.toString();
        if (row.startsWith("medallion,")) return;
        DEBSRecordParser parser = new DEBSRecordParser();
        parser.parse(row);
        Integer duration = parser.getDuration();
        context.write(new Text(parser.getMedallion()),
                new TripTimeTuple(duration, duration, duration));
    }
}
```

TripTimeReducer.java

```
public class TripTimeReducer extends Reducer<Text, TripTimeTuple, Text, TripTimeTupl

    private TripTimeTuple result = new TripTimeTuple();

    @Override
    protected void reduce(Text key, Iterable<TripTimeTuple> values, Context context)
            throws IOException, InterruptedException {
        Integer totalDuration = 0, minDuration = -1, maxDuration = -1;
        Integer currentMin, currentMax;

        for (TripTimeTuple value : values) {
            totalDuration += value.getTotalDuration().get();

            if (minDuration == -1) {
                minDuration = value.getMinDuration().get();
                maxDuration = value.getMaxDuration().get();
            } else {
                currentMin = value.getMinDuration().get();
                currentMax = value.getMaxDuration().get();
                minDuration = currentMin < minDuration ? currentMin : minDuration;
                maxDuration = currentMax > maxDuration ? currentMax : maxDuration;
            }
        }
        result.setData(totalDuration, minDuration, maxDuration);
        context.write(key, result);
    }
}
```

TripTimeTuple.java

```
public class TripTimeTuple implements WritableComparable<TripTimeTuple> {

    private IntWritable minDuration;
    private IntWritable maxDuration;
    private IntWritable totalDuration;

    public TripTimeTuple() {
        this(new IntWritable(), new IntWritable(),
                new IntWritable());
    }

    public TripTimeTuple(IntWritable minDuration, IntWritable maxDuration,
                         IntWritable totalDuration) {
        this.minDuration = minDuration;
        this.maxDuration = maxDuration;
        this.totalDuration = totalDuration;
    }

    public TripTimeTuple(Integer minDuration, Integer maxDuration,
                         Integer totalDuration) {
        setData(totalDuration, minDuration, maxDuration);
    }

    @Override
    public int compareTo(TripTimeTuple tuple) {
        Integer result = totalDuration.compareTo(tuple.totalDuration);
        if (result != 0 ) {
            return result;
        }
        result = maxDuration.compareTo(tuple.maxDuration);
        if (result != 0 ) {
            return result;
        }

        return minDuration.compareTo(tuple.minDuration);
    }

    @Override
    public void write(DataOutput dataOutput) throws IOException {
        minDuration.write(dataOutput);
        maxDuration.write(dataOutput);
        totalDuration.write(dataOutput);
    }

    @Override
    public void readFields(DataInput dataInput) throws IOException {
```

```
        minDuration.readFields(dataInput);
        maxDuration.readFields(dataInput);
        totalDuration.readFields(dataInput);
    }

    public IntWritable getMinDuration() {
        return minDuration;
    }

    public IntWritable getMaxDuration() {
        return maxDuration;
    }

    public IntWritable getTotalDuration() {
        return totalDuration;
    }

    public void setData(Integer totalDuration, Integer minDuration,
                        Integer maxDuration) {
        this.totalDuration = new IntWritable(totalDuration);
        this.minDuration = new IntWritable(minDuration);
        this.maxDuration = new IntWritable(maxDuration);
    }

    public int hashCode() {
        return totalDuration.hashCode() * 163
                + maxDuration.hashCode() + minDuration.hashCode();
    }

    @Override
    public boolean equals(Object o) {
        if (o instanceof TripTimeTuple) {
            TripTimeTuple tp = (TripTimeTuple) o;
            return totalDuration.equals(tp.totalDuration)
                    && minDuration.equals(tp.minDuration)
                    && maxDuration.equals(tp.maxDuration);
        }
        return false;
    }

    @Override
    public String toString() {
        return totalDuration.toString() + "\t" + minDuration.toString()
                + "\t" + maxDuration.toString();
    }
}
```

DEBSRecordParser.java

```
public class DEBSRecordParser {

    private static final Double UPPER_LONGITUDE = -73.95;
    private static final Double LOWER_LONGITUDE = -74.0;
    private static final Double UPPER_LATITUDE = 40.8;
    private static final Double LOWER_LATITUDE = 40.75;

    private String medallion;
    private Integer duration;
    private Boolean inCenter;
    private Integer category;

    public void parse(String record) {
        String[] tokens = record.split(",");
        medallion = tokens[0];
        duration = Integer.parseInt(tokens[8]);
        Double pickupLongitude = Double.parseDouble(tokens[10]);
        Double pickupLatitude = Double.parseDouble(tokens[11]);
        Double dropoffLongitude = Double.parseDouble(tokens[12]);
        Double dropoffLatitude = Double.parseDouble(tokens[13]);
        inCenter = inCenter(pickupLongitude, pickupLatitude,
                dropoffLongitude, dropoffLatitude);
        Integer passengerCount = Integer.parseInt(tokens[7]);

        if (passengerCount == 1) {
            category = 1;
        } else if (passengerCount == 2 || passengerCount == 3) {
            category = 2;
        } else if (passengerCount >= 4) {
            category = 3;
        } else {
            category = -1;
        }
    }

    private Boolean inCenter(Double pickupLongitude, Double pickupLatitude,
                             Double dropoffLongitude, Double dropoffLatitude) {
        return inRange(pickupLongitude, LOWER_LONGITUDE, UPPER_LONGITUDE)
                && inRange(pickupLatitude, LOWER_LATITUDE, UPPER_LATITUDE)
                && inRange(dropoffLongitude, LOWER_LONGITUDE, UPPER_LONGITUDE)
                && inRange(dropoffLatitude, LOWER_LATITUDE, UPPER_LATITUDE);
    }

    private boolean inRange (Double value, Double lowerBound, Double upperBound){
        return value >= lowerBound && value <= upperBound;
    }
```

```
    public String getMedallion() {
        return medallion;
    }

    public Integer getDuration() {
        return duration;
    }

    public Boolean isInCenter() {
        return inCenter;
    }

    public Integer getCategory() {
        return category;
    }
}
```

2. zadatak

Napomena: Kod ovog zadatka moguće su razlike u broju vožnji ovisno o implementaciji. U početnoj csv datoteci postoje vožnje koje su bez putnika pa ovisno o if uvjetu gdje se određuje kateogrija moguće su manje razlike u brojevima. Ovdje je implementirano da vožnje koje su bez putnika ne spadaju u niti jednu od danih kategorija u zadatku.

1. Užem centru pripada 401,566 vožnji, dok širem gradskom području pripada 861,815 vožnji što ukupno daje 1,263,381 vožnji (u csv datoteci postoji 1,263,388 vožnji no neke od njih kako je već navedeno su bez putnika te nisu uvrštene u gornje brojke).
2. Osim same implementacije Partitionera, potrebno je prilikom konfiguracije posla dodati navedenog Partitionera na sljedeći način:
   `job.setPartitionerClass(LocationCategoryPartitioner.class);`
3. Vožnje po skupinama:
   - 1 putnik:
     - uži centar: 244814
     - izvan centra: 525517
   - 2-3 putnika:
     - uži centar: 70458
     - izvan centra: 151509
   - 4 ili više putnik:
     - uži centar: 86294

- izvan centra: 184789

Izvorni kod

Task2.java

```
public class Task2 {

    private static final String INPUT_PATH = "/user/rovkp/trip_data.csv";
    private static final String OUTPUT_PATH = "/user/rovkp/location_result";

    public static void main(String[] args) throws IOException,
            ClassNotFoundException, InterruptedException {
        Job job = getJob(INPUT_PATH, OUTPUT_PATH, Task2.class);
        job.waitForCompletion(true);
    }

    public static Job getJob(String inputPath, String outputPath, Class<?> cls)
            throws IOException {
        Job job = Job.getInstance();
        job.setJarByClass(cls);
        job.setJobName("Location");

        FileInputFormat.addInputPath(job, new Path(inputPath));
        FileOutputFormat.setOutputPath(job, new Path(outputPath));

        job.setMapperClass(LocationMapper.class);
        job.setPartitionerClass(LocationCategoryPartitioner.class);
        job.setReducerClass(LocationReducer.class);
        job.setNumReduceTasks(6);

        job.setOutputKeyClass(NullWritable.class);
        job.setOutputValueClass(Text.class);
        job.setMapOutputKeyClass(IntWritable.class);
        job.setMapOutputValueClass(Text.class);
        return job;
    }
}
```

LocationMapper.java

```
public class LocationMapper extends Mapper<LongWritable, Text, IntWritable, Text> {

    @Override
    protected void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        String row = value.toString();
        // First line is CSV header
        if (row.startsWith("medallion,")) return;
        DEBSRecordParser parser = new DEBSRecordParser();
        parser.parse(row);
        // Invalid passenger count
        if (parser.getCategory() == -1) return;
        context.write(new IntWritable(parser.getCategory()), new Text(row));
    }
}
```

LocationReducer.java

```
public class LocationReducer extends Reducer<IntWritable, Text, NullWritable, Text>
    @Override
    protected void reduce(IntWritable key, Iterable<Text> values, Context context)
            throws IOException, InterruptedException {
        for (Text value : values) {
            context.write(NullWritable.get(), value);
        }
    }
}
```

LocationCategoryPartitioner.java

```java
public class LocationCategoryPartitioner extends Partitioner<IntWritable, Text> {

    @Override
    public int getPartition(IntWritable category, Text value , int i) {
        DEBSRecordParser parser = new DEBSRecordParser();
        String row = value.toString();
        if (row.startsWith("medallion,")) return 0;
        parser.parse(row);
        Boolean isInCenter = parser.isInCenter();
        switch (category.get()) {
            case 1:
                if (isInCenter) return 0;
                else return 1;
            case 2:
                if (isInCenter) return 2;
                else return 3;
            default:
                if (isInCenter) return 4;
                else return 5;
        }
    }
}
```

3. zadatak

Napomena: Kod ovog zadatka moguće su razlike u broju vožnji ovisno o implementaciji. U početnoj csv datoteci postoje vožnje koje su bez putnika pa ovisno o if uvjetu gdje se određuje kateogrija moguće su manje razlike u brojevima. Ovdje je implementirano da vožnje koje su bez putnika ne spadaju u niti jednu od danih kategorija u zadatku.

1. Izvršeno je 7 MapReduce poslova. Jedan posao za particioniranje podataka te šest poslova za obradu pojedine particije.
2. Broj različitih taksija po skupinama:
    - 1 putnik:
        - uži centar: 6601
        - izvan centra: 8714
    - 2-3 putnika:
        - uži centar: 4138
        - izvan centra: 5487
    - 4 ili više putnik:

- uži centar: 3425
- izvan centra: 3983

Izvorni kod:

```
public class Task3 {

    private static final String INTERMEDIATE_PATH = "intermediate";
    private static final String INPUT_PATH = "/user/rovkp/trip_data.csv";
    private static final String OUTPUT_PATH = "/user/rovkp/jr/joined_result";

    public static void main(String[] args) throws IOException,
            ClassNotFoundException, InterruptedException {
        Job partitionJob = Task2.getJob(INPUT_PATH, INTERMEDIATE_PATH, Task3.class);

        Integer partitionResult = partitionJob.waitForCompletion(true) ? 0 : 1;
        if (partitionResult != 0) {
            FileSystem.get(partitionJob.getConfiguration())
                    .delete(new Path(INTERMEDIATE_PATH), true);
            return;
        }
        for (int i = 0; i < 6; i++) {
            Job timeJob = Task1.getJob(INTERMEDIATE_PATH + "/part-r-0000"+i,
                    OUTPUT_PATH+i, Task3.class);
            timeJob.waitForCompletion(true);
            FileSystem.get(timeJob.getConfiguration())
                    .delete(new Path(INTERMEDIATE_PATH + "/part-r-0000"+i), true);
        }
        FileSystem.get(partitionJob.getConfiguration())
                .delete(new Path(INTERMEDIATE_PATH), true);
    }
}
```