Raspodijeljena obrada velike količine podataka

3. Domaća zadaća

Filip Gulan - JMBAG: 0036479428

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

```
public class Task1 {

    // 1. 8358
    // 2. 1, 87, 0.44775397
    // Da, obje sadrže riječ doktor i slične medicinske termine višew puta
    // Da, po prethodno vidjenom primjeru trebala bi

    private static final String INPUT_FILE =
"/Users/filipgulan/Downloads/jester_dataset_2/jester_items.dat";
    private static final String OUTPUT_FILE = "item_similarity.csv";

    private static StandardAnalyzer analyzer = new StandardAnalyzer();
    private static Directory index = new RAMDirectory();

    public static void main(String[] args) throws IOException,
ParseException {
        List<String> lines =  Files.readAllLines(Paths.get(INPUT_FILE));
        Map<Integer, String> jokePairs = parseInput(lines);
        createLuceneDocuments(jokePairs);
        float[][] similarityMatrix = queryDocs(jokePairs);
```

```
        similarityMatrix = normalizeSimilarityMatrix(similarityMatrix);
        storeSimilarityMatrixAsCSV(similarityMatrix, OUTPUT_FILE);
    }

    private static void storeSimilarityMatrixAsCSV(float[][]
similarityMatrix, String path) throws IOException {
        try (BufferedWriter writer = new BufferedWriter(new
FileWriter(path))) {
            for (int i = 0; i < similarityMatrix.length; ++i) {
                for (int j = i + 1; j < similarityMatrix[i].length; ++j) {
                    if (similarityMatrix[i][j] > 0) {
                        String row = (i + 1) + "," + (j + 1) + "," +
similarityMatrix[i][j];
                        writer.write(row);
                        if (i == similarityMatrix.length - 1 && j ==
similarityMatrix[i].length -1) continue;
                        writer.newLine();
                    }
                }
            }
        }
    }

    private static float[][] normalizeSimilarityMatrix(float[][]
similarityMatrix) {
        for (Integer i = 0; i < similarityMatrix.length; i++) {
            Float max = similarityMatrix[i][i];
            for (Integer j = 0; j < similarityMatrix[i].length; j++) {
                similarityMatrix[i][j] /= max;
            }
        }

        for (int i = 0; i < similarityMatrix.length; i ++) {
            for (int j = 0; j <= i; j++) {
                similarityMatrix[j][i] = similarityMatrix[i][j] =
                        (similarityMatrix[i][j] + similarityMatrix[j][i]) /
2.0f;

            }
        }
        return similarityMatrix;
    }

    private static float[][] queryDocs(Map<Integer, String> jokePairs)
throws ParseException, IOException {
        Integer documentsCount = jokePairs.size();
        float[][] similarity = new float[documentsCount][documentsCount];
        IndexReader reader = DirectoryReader.open(index);
        IndexSearcher searcher = new IndexSearcher(reader);
```

```
        Integer currentDocumentIndex = 0;
        for (Map.Entry<Integer, String> entry : jokePairs.entrySet()) {
            Query query = new QueryParser("text",
analyzer).parse(QueryParser.escape(entry.getValue()));
            for (ScoreDoc hit : searcher.search(query,
documentsCount).scoreDocs) {
                Document document = reader.document(hit.doc);
                Integer documentID = Integer.parseInt(document.get("ID"));
                Integer documentIndex = documentID - 1;
                similarity[currentDocumentIndex][documentIndex] = hit.score;
            }
            currentDocumentIndex++;
        }
        reader.close();
        return similarity;
    }

    private static void createLuceneDocuments(Map<Integer, String>
jokePairs) throws IOException {
        FieldType idFieldType = new FieldType();
        idFieldType.setStored(true);
        idFieldType.setTokenized(false);
        idFieldType.setIndexOptions(IndexOptions.NONE);

        FieldType jokeFieldType = new FieldType();
        jokeFieldType.setStored(true);
        jokeFieldType.setTokenized(true);

jokeFieldType.setIndexOptions(IndexOptions.DOCS_AND_FREQS_AND_POSITIONS);

        IndexWriterConfig config = new IndexWriterConfig(analyzer);
        IndexWriter w = new IndexWriter(index, config);
        for (Map.Entry<Integer, String> entry : jokePairs.entrySet()) {
            Document document = new Document();
            document.add(new Field("ID", entry.getKey().toString(),
idFieldType));
            document.add(new Field("text", entry.getValue(),
jokeFieldType));
            w.addDocument(document);
        }
        w.close();
    }

    private static Map<Integer, String> parseInput(List<String> lines) {
        Iterator<String> iterator = lines.iterator();
        Map<Integer, String> jokes = new HashMap<>();

        while (iterator.hasNext()) {
```

```
            Integer id = Integer.parseInt(iterator.next().replace(":",
"").trim());
            String joke = "";
            while (iterator.hasNext()) {
                String line = iterator.next();
                if (line.isEmpty()) {
                    break;
                }
                joke += line;
            }
            joke =
StringEscapeUtils.unescapeXml(joke.toLowerCase().replaceAll("\\<.*?\\>",
""));
            jokes.put(id, joke);
        }
        return jokes;
    }
}
```

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

```
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);
    }
}
```