Report Big Data

ANTONIO SESSA

January 2024

Email:a.sessa108@studenti.unisa.it
Matricola: 0622702305
Professore: Giuseppe D'Aniello gidaniello@unisa.it gidaniello@unisa.it

Indice

1 Hadoop

1.1 Descrizione del problema (traccia)

Realizzare l'indice degli arbitri, di una certa nazionalità, che consenta di sapere ogni arbitro quali partite ha arbitrato, considerando che un arbitro può ricoprire il ruolo di Referee, Assistant 1 e Assistant 2 in partite diverse. La nazionalità deve essere indicata dall'utente (ad esempio ITA).

1.2 Soluzione proposta Hadoop

Per risolvere il problema ho deciso di far utilizzo del pattern di summarization indice invertito e del pattern di filtering filtering. Le parole chiavi per cui costruiremo l'indice invertito sono i campi corrispondenti a Referre, Assistant 1 e Assistant 2, la lista di identificatori che assoceremo sono il campo MatchId (che interpreto come identificatore univoco per la partite). La condizione per cui filtriamo le parole chiavi è la presenza della stringa passata da utente (ad esempio ITA). Il dataset presenta alcuni valori con caratteri anomali, ma nessuno all'interno dei campi che prendiamo in considerazione e quindi non c'è necessità di modificarlo. Per utilizzare la stringa passata da utente all'interno del mapper, la configuriamo nel driver, così da poterci accedere nel mapper.

```
1 Configuration conf = new Configuration();
2 conf.set("nationality", args[2]);
```

Non è stato utizzato un combiner, dato che per il pattern indice invertito e filtro non partano alcun beneficio.

1.3 Codice Sorgente Mapper

```
class MapperReferee extends Mapper
        <LongWritable, // Input key type
    Text, // Input value type
    Text, // Output key type
    Text> {// Output value type
        private String nationality;
        @Override
        public void setup(Context context){
            // Get the searched nationality from the context
            nationality = context.getConfiguration().get("
                nationality");
        }
        @Override
    protected void map(LongWritable key, // Input key type
            Text value, // Input value type
            Context context) throws IOException,
                InterruptedException {
        // Split each sentence in words. Use whitespace"," as
            delimiter
        // The split method returns an array of strings
        String[] words = value.toString().split(",");
        // Check field 13,14,15 (respectively Referee,
            assistant 1 and assistant 2)
                // if it contains the searched nationality,
                    context write it + match Id
                // match Id is field 17
                if(words[13].contains(nationality)) context.
                    write(new Text(words[13]), new Text(words
                    [17]));
                if(words[14].contains(nationality)) context.
                    write(new Text(words[14]), new Text(words
                    [17]));
                if(words[15].contains(nationality)) context.
                    write(new Text(words[15]), new Text(words
                    [17]));

    }
}
```

Una possibile aggiunta che si può introdurre nel mapper e anche rimuovere dalla stringa che contiene il nome dell'arbitro/assistante l'indicatore della nazionalità.

1.4 Codice Sorgente Reducer

```
30 class ReducerReferee extends Reducer
31         <Text, // Input key type
32     Text, // Input value typeF
33     Text, // Output key type
34     Text> { // Output value type
35
36     @Override
37     protected void reduce(Text key, // Input key type (Referee
           name)
38             Iterable<Text> values, // Input value type (Match
                   Ids)
39             Context context) throws IOException,
                   InterruptedException {
40
41         String invIndex = new String();
42
43         // Iterate over the set of Match Ids and concatenate
               them
44         for (Text value : values) {
45             invIndex = invIndex.concat(value + ",");
46         }
47
48         // avoid writing the last comma by taking the substring
49         context.write(key, new Text(invIndex.substring(0,
               invIndex.length() - 1 )));
50     }
51 }
```

Il codice per il ruducer è praticamente identico a quello visto a lezione, l'unica aggiunta è la rimozione della virgola finale.

1.5 Codice Sorgente Driver

```
53 public class FinalProject {
54
55     /**
56      * @param args the command line arguments
57      */
58     public static void main(String[] args) throws IOException,
           InterruptedException, ClassNotFoundException {
59         Path inputPath;
60         Path outputDir;
61
62         if(args.length < 3) {
63             System.out.println("Usage : inputPath (hdfs) ,
                   outputPath (hdfs), nationality e.g ITA");
64             return ;
65         }
66
67         // Parse the parameters
68         inputPath = new Path(args[0]);
69         outputDir = new Path(args[1]);
70         Configuration conf = new Configuration();
71         conf.set("nationality", args[2]);
72
73         // Define a new job
74         Job job = Job.getInstance(conf);
75
76         // Assign a name to the job
77         job.setJobName("InvertedIndex");
78
79         // Set path of the input file/folder (if it is a folder
               , the job reads all the files in the specified
               folder) for this job
80         FileInputFormat.addInputPath(job, inputPath);
81
82         // Set path of the output folder for this job
83         FileOutputFormat.setOutputPath(job, outputDir);
84
85         // Specify the class of the Driver for this job
86         job.setJarByClass(FinalProject.class);
87
88         // Set job input format
89         job.setInputFormatClass(TextInputFormat.class);
90
91         // Set job output format
92         job.setOutputFormatClass(TextOutputFormat.class);
93
94         // Set map class
95         job.setMapperClass(MapperReferee.class);
96
```

```
97          // Set map input key and value classes
98          // Set map output key and value classes
99          job.setMapOutputKeyClass(Text.class);
100         job.setMapOutputValueClass(Text.class);
101
102         // Set reduce class
103         job.setReducerClass(ReducerReferee.class);
104
105         // Set reduce output key and value classes
106         job.setOutputKeyClass(Text.class);
107         job.setOutputValueClass(Text.class);
108
109         // Set number of reducers
110         //job.setNumReduceTasks(numberOfReducers);
111
112         // Execute the job and wait for completion
113         System.exit(job.waitForCompletion(true) ? 0 : 1);
114     }
115
116 }
```

2 Spark

2.1 Descrizione del problema (statistica a scelta)

Realizzare la classifica delle top X squadre che hanno vinto più partite con Y o più goal di differenza, ad esempio se Y = 1, ottengo la classifica delle squadre che hanno vinto più partite. Considerare la possibilità che una partita finisca ai rigori. X e Y devono essere indicati dall'utente e sono entrambi interi positivi. La classifica deve essere stampata a video.

2.2 Soluzione proposta Spark

Per risolvere il problema ho deciso di fare utilizzo dell'equivalente per Spark dei pattern di filtering, di Record count (per group) e top K. Per gestire le ties aggiungo anche un pattern di indice invertito, così da poter inserire più squadre nella stessa posizioni della classifica. Il dataset presenta alcuni valori anomali nei campi che indicano il nome di due squadre. In particolare, prima di Republic of Ireland e United Arab Emirates è presente il seguente typo rn">. Questo typo è consistente, ed ogni volta che una delle due squadre è presente nella riga è sempre preceduta dal typo, che quindi non è un problema per le statistiche (non dobbiamo gestire il caso in cui una volta la Repubblica di Irlanda è chiamata Republic of Ireland e un altra volta rn">Republic of Ireland). Si potrebbe gestire questo typo rimuovendolo prima di stampare a video la classifica.

2.3 Codice Sorgente Spark

La classe Match è realizzata per rendere il codice più leggibile, e per filtrare le partite che non terminano con Y goal di scarto tramite il metodo checkGoalDifference().

```
1  public static class Match implements Serializable {
2      String homeTeam;
3      String awayTeam;
4      Integer goalDifference;
5      Integer treshold;
6
7      public Match(String homeTeam, String awayTeam, Integer
           goalHomeTeam, Integer goalAwayTeam, Integer treshold) {
8          this.homeTeam = homeTeam;
9          this.awayTeam = awayTeam;
10         this.goalDifference = goalHomeTeam - goalAwayTeam;
11         this.treshold = treshold;
12     }
13     /**
14     * @return The team that won the match with 3 or more goal
           lead, null otherwise
15     */
16     public Tuple2<String, Integer> checkGoalDifference() {
17         if (goalDifference >= treshold) {
18             return new Tuple2(homeTeam, 1);
19         }
20         if (goalDifference <= -treshold) {
21             return new Tuple2(awayTeam, 1);
22         }
23         return null;
24     }
25 }
```

Creo l'oggetto Spark Context per leggere il file, e scarto l'header.

```
26 // Create a configuration object and set the name of the
       application
27 SparkConf conf = new SparkConf().setAppName("Spark World Cup
       Stats");
28
29 // Create a Spark Context object
30 JavaSparkContext sc = new JavaSparkContext(conf);
31
32 // Build an RDD of Strings from the input textual file
33 // Each element of the RDD is a line of the input file
34 JavaRDD<String> lines = sc.textFile(inputFile);
35 // filter out the header
36 JavaRDD<String> linesWithoutHeader = lines.filter(s -> !s.
       contains("Away Team Name"));
```

La lambda expression passata come parametro della funzione map, crea un RDD di matches, trasformando l'RDD di stringhe. All'interno della lambda expression viene anche considerata la possibilità che una partita finisca ai rigori.

```
// Create an RDD of matches
JavaRDD<Match> matches = linesWithoutHeader.map(s -> {
    String[] splitted = s.split(",");
    // check for penalties
    if(splitted[9].contains("penalties"))
    {
        // Assuming that the penalites are the only integers in
            the string, this leaves us with a string containing
        // the penalties shootout result separeted by "-" (e.g
            4-1)
        String [] penalties = splitted[9].replaceAll("[^0-9-]",
            "").split("-");
        // adding the result of the penalties to the scores
        return new Match(splitted[5], splitted[8], Integer.
            valueOf(splitted[6]) + Integer.valueOf(penalties[0])
            , Integer.valueOf(splitted[7])+ Integer.valueOf(
            penalties[1]),threshold);

    }
    // the value we are intrest in are homeTeam (5), awayTeam
        (6) and the result of the match (7 and 8)
    return new Match(splitted[5], splitted[8], Integer.valueOf(
        splitted[6]), Integer.valueOf(splitted[7]),threshold);
});
```

Creo ora il PairRDD contenente le tuple<nomeSquadra,1>. Devo usare il la funzione flatMapToPair perché la funzione può restituire tuple vuote, quando la partita finisce senza Y goal di differenza. Questo equivale al mapper del pattern record count (Per group)

```
// Create the RDD <teamName,1>
// flat map beacause not every match ends with a three goal
    difference (or larger)
JavaPairRDD<String, Integer> partialWinners = matches.
    flatMapToPair(f -> {
    List<Tuple2<String, Integer>> pairs = new ArrayList();
        if(f.checkGoalDifference() != null){
            pairs.add(f.checkGoalDifference());
        }
        return pairs.iterator();
});
```

Ora non resta che eseguire un reduceByKey che equivale al reduce del pattern record count (Per group) per ottenere il PairRDD contenente le tuple tuple<nomeSquadra,numeroDiVittoreConYGoalDiScarto>

```
// Create the RDD <teamName, numberOfWinsWithThreeGoalLead>
JavaPairRDD<String, Integer> reducedWinners = partialWinners.
    reduceByKey((a, b) -> {
    return a + b;
});
```

A questo punto si può eseguire la seguente azione[1], che equivale ad un pattern di top K, gestire le ties con il comparatore e stampare a video il risultato.

```
List<Tuple2<String, Integer>> Results = reducedWinners.top(K,
    new Tuple2Comparator());
```

In alternativa si è scelto di poter ottenere posizioni in classifica a pari merito, per fare ciò si possono invertire i nomi delle squadre e il numero di partite, eseguire una istruzione di reduce per concatenare i nomi delle squadre (simil reducer del pattern indice invertito), ordinarli per chiave in ordine discendente e prelevare i primi X. elementi.

```
// Create the RDD <numberOfWinsWithThreeGoalLead, teamName>
JavaPairRDD<Integer,String> invertedReducedWinners =
    reducedWinners.mapToPair(f -> {
    return new Tuple2(f._2(),f._1());
});


// Create the RDD <numberOfWinsWithThreeGoalLead, list of
    teamNames>
JavaPairRDD<Integer, String> invertedIndex =
    invertedReducedWinners.reduceByKey((a,b)->{
    return a + " " + b;
});
// Sort the RDD by key invertedIndex
JavaPairRDD<Integer, String> sortedInvertedIndex =
    invertedIndex.sortByKey(false);
// Take the top K of the sorted invertedIndex
List<Tuple2<Integer, String>> TopK = sortedInvertedIndex.take(K
    );
```

[1]Ho provato ad usare una lambda expression per il comparatore, ma causa un eccezzione 'task non serializzabile'. Sembra che l'espressione lambda che funge da comparatore non sia serializzabile e causi questo problema, che può essere risolto creando una classe che implementa Comparator<Tuple2<String, Integer>>, Serializable

Finimao stampando a video i risultati e chiudendo la spark Context

```
88 System.out.println("Le Top " + K + " Squadre dei mondiali di
       calcio tenutasi tra il 1930 e il 1998 che hanno vinto il
       maggior numero di partite con "+ threshold +" o maggior goal
        di scarto (rigori inclusi)");
89 for (Tuple2<Integer, String> t : TopK) {
90     System.out.println(t);
91 }
92 sc.close();
```

3 Risultati

3.1 Risultato Hadoop con input utente ITA

AGNOLIN Luigi (ITA) 551,610,120,564,398,627
ANGONESE Aurelio (ITA) 2066,1987,2004,1984
BALDAS Fabio (ITA) 3063
BARLASSINA Rinaldo (ITA) 1129,1130,1175,1119
BONIVENTO Ferruccio (ITA) 1133
CAIRONI Camillo (ITA) 1104,1105
CARMINATI Ettore (ITA) 1111
CARRARO Albino (ITA) 1105,1102
CASARIN Paolo (ITA) 741,764,959,900,922
COLLINA Pierluigi (ITA) 8762,8733
D ELIA Pietro (ITA) 111,129
DATTILO Generoso (ITA) 1199,1230,1202,1207,1119
GALEATI Giovanni (ITA) 1199,1231,1230,1202
GONELLA Sergio (ITA) 2201,2198,2433,2246
JONNI Cesare (ITA) 1564,1559,1480,1562
LANESE Tullio (ITA) 290,102,114,75
LO BELLO Concetto (ITA) 1659,1689,1634
LO BELLO Rosario (ITA) 152
LONGHI Carlo (ITA) 29
MAGNI Pierluigi (ITA) 175
MATTEA Francesco (ITA) 1108,1106
MAZZEI Gennaro (ITA) 8733,8759,8752
MELANDRI Ermenegildo (ITA) 1108
ORLANDINI Vincenzo (ITA) 1313,1233,1264,1304,1434,1423,1387,1278
PAIRETTO Pierluigi (ITA) 3088,25,198
RAMICONE Domenico (ITA) 3057,3063,3088
SASSI Otello (ITA) 1119
SBARDELLA Antonio (ITA) 1840,1893,1780,1843
SCARPI Giuseppe (ITA) 1176,1152,1141,1156
SCORZONI Raffaele (ITA) 1141
TURBIANI Giuseppe (ITA) 1102

3.2 Risultati Spark con Input X = 15 e Y = 1

(55, Brazil)
(39, Germany FR)
(38, Italy)
(32, Argentina)
(23, France)
(20, England)
(16, Yugoslavia Spain)
(15, Soviet Union Hungary Uruguay Sweden)
(14, Netherlands)
(13, Poland)
(12, Austria)
(11, Czechoslovakia)
(10, Belgium)
(9, Germany)
(8, Mexico Romania)

3.3 Risultati Spark con Input X = 15 e Y = 4

(7, Hungary Brazil)
(6, Germany FR)
(4, Argentina Uruguay)
(3, Netherlands Yugoslavia France)
(2, Soviet Union Spain Sweden Czechoslovakia Poland)
(1, Denmark Turkey Mexico Italy Bulgaria Russia Austria)

4 Come eseguire il codice

4.1 Eseguire Hadoop

```
# Inserire il dataset nel cartella data (dove e' montato il volume)
# Inserire il jar nella cartella data (dove e' montato il volume)
# Eseguire i seguenti comandi all'interno della bash del nodo master

# avviare HADOOP e YARN
$HADOOP_HOME/sbin/start-dfs.sh
$HADOOP_HOME/sbin/start-yarn.sh

# Inserire il dataset nel dfs
hdfs dfs -put worldcup1.csv /dataset

# Lancio Hadoop jar
# sostituire $0 con le iniziali di una nazione e.g ITA
```

```
14 hadoop jar FinalProject.jar /dataset /refereeIndex $0
15
16 # Prendo dal dfs il risultato di hadoop
17 hdfs dfs -get /refereeIndex
18
19 # Pulizia del dfs (cancello il dataset e il risultato di Hadoop
       )
20 hdfs dfs -rm -r /refereeIndex
21 hdfs dfs -rm /dataset
```

4.2 Eseguire Spark

1. Inserire il jar SparkWorldCupStats.jar, la cartella input, i due eseguibili shell launch_cluster.sh e launch_single.sh nella stessa cartella.

2. Se si vogliono cambiare i parametri di ingresso, modificando gli eseguibili shell, l'usage del programma è il seguente: Cartella che contiene il csv da processare, X, Y.

3. Eseguire uno dei due eseguibili shell.