



TU BERLIN

ADVANCED INFORMATION MANAGEMENT

HOMEWORK ASSIGNMENT 1

Programming in Hadoop and Clustering Exercises

Author:
Ward SCHODTS

Supervisor:
Juan SOTO

June 8, 2016

Programming in Hadoop

1. WordCount - "Hello World" of MapReduce

```
1 package de.tuberlin.dima.aim3.assignment1;
2
3 import de.tuberlin.dima.aim3.HadoopJob;
4 import org.apache.hadoop.fs.Path;
5 import org.apache.hadoop.io.IntWritable;
6 import org.apache.hadoop.io.Text;
7 import org.apache.hadoop.mapreduce.Job;
8 import org.apache.hadoop.mapreduce.Mapper;
9 import org.apache.hadoop.mapreduce.Reducer;
10 import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
11 import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
12
13 import java.io.IOException;
14 import java.util.ArrayList;
15 import java.util.Arrays;
16 import java.util.List;
17 import java.util.Map;
18 import java.util.function.Function;
19 import java.util.regex.Pattern;
20 import java.util.stream.StreamSupport;
21
22 import static java.util.stream.Collectors.counting;
23 import static java.util.stream.Collectors.groupingBy;
24
25 public class FilteringWordCount extends HadoopJob {
26
27     @Override
28     public int run(String[] args) throws Exception {
29         Map<String, String> parsedArgs = parseArgs(args);
30
31         Path inputPath = new Path(parsedArgs.get("--input"));
32         Path outputPath = new Path(parsedArgs.get("--output"));
33
34         Job wordCount = prepareJob(inputPath, outputPath, TextInputFormat.class,
35 FilteringWordCountMapper.class,
36 Text.class, IntWritable.class, WordCountReducer.class, Text.class,
37 IntWritable.class, TextOutputFormat.class);
38
39         wordCount.waitForCompletion(true);
40
41         return 0;
42     }
43
44     static class FilteringWordCountMapper extends Mapper<Object, Text, Text,
45 IntWritable> {
46
47         private ArrayList<String> filterList = new ArrayList<>();
48
49         /**
50          * Method to add words that should be filtered out.
51          *
52          * @param fl : the list with filtered words
53          */
54         public void addWordsToFilter(List<String> fl) {
55             this.filterList.addAll(fl);
56         }
57
58         public List getFilterList() {
59             return this.filterList;
60         }
61     }
62 }
```

```

60     @Override
61     protected void map(Object key, Text line, Context ctx) throws IOException,
InterruptedException {
62         String[] filterList = {"to", "and", "in", "the"};
63         addWordsToFilter(Arrays.asList(filterList));
64         Pattern.compile(" ").splitAsStream(line.toString().replace(",", ""))
        .map(String::toLowerCase).filter(l -> !getFilterList().contains(l))
        .collect(groupingBy(Function.identity(), counting())).forEach((word, count) ->
        writeToCtx(word, count, ctx));
65     }
66
67     private void writeToCtx(String word, Long val, Context ctx) {
68         try {
69             ctx.write(new Text(word), new IntWritable(val.intValue()));
70         } catch (InterruptedException | IOException ignored) {
71
72         }
73     }
74
75 }
76
77 static class WordCountReducer extends Reducer<Text, IntWritable, Text, IntWritable>
{
78
79     @Override
80     protected void reduce(Text key, Iterable<IntWritable> values, Context ctx)
81         throws IOException, InterruptedException {
82
83         ctx.write(key, new IntWritable(StreamSupport.stream(values.spliterator(),
84 false).mapToInt(IntWritable::get).sum()));
85     }
86
87 }
88 }

```

Listing 1: FilteringWordCount.java

2. A custom Writable

```

1 package de.tuberlin.dima.aim3.assignment1;
2
3 import org.apache.hadoop.hdfs.util.ByteArray;
4 import org.apache.hadoop.io.Writable;
5
6 import java.io.DataInput;
7 import java.io.DataOutput;
8 import java.io.IOException;
9 import java.util.Arrays;
10
11 public class PrimeNumbersWritable implements Writable {
12
13     private int[] numbers;
14
15     public PrimeNumbersWritable() {
16         numbers = new int[0];
17     }
18
19     public PrimeNumbersWritable(int... numbers) {
20         this.numbers = numbers;
21     }
22
23     @Override
24     public void write(DataOutput out) throws IOException {
25         out.writeInt(numbers.length);
26         Arrays.stream(numbers).forEach((v) -> writeToOut(out, v));
27     }

```

```

28
29     private void writeToOut(DataOutput out, int i) {
30         try {
31             out.writeInt(i);
32         } catch (IOException ignored) {
33         }
34     }
35
36
37     @Override
38     public void readFields(DataInput in) throws IOException {
39         int length = in.readInt();
40
41         int[] temp = new int[length];
42         for (int i = 0; i < length; i++) {
43             temp[i] = in.readInt();
44         }
45         this.numbers = temp;
46     }
47
48     @Override
49     public boolean equals(Object obj) {
50         if (obj instanceof PrimeNumbersWritable) {
51             PrimeNumbersWritable other = (PrimeNumbersWritable) obj;
52             return Arrays.equals(numbers, other.numbers);
53         }
54         return false;
55     }
56
57     @Override
58     public int hashCode() {
59         return Arrays.hashCode(numbers);
60     }
61 }

```

Listing 2: PrimeNumbersWritable.java

3. Average temperature per month

```

1 package de.tuberlin.dima.aim3.assignment1;
2
3
4 import de.tuberlin.dima.aim3.HadoopJob;
5 import org.apache.hadoop.conf.Configuration;
6 import org.apache.hadoop.fs.Path;
7 import org.apache.hadoop.io.DoubleWritable;
8 import org.apache.hadoop.io.IntWritable;
9 import org.apache.hadoop.io.Text;
10 import org.apache.hadoop.mapreduce.Mapper;
11 import org.apache.hadoop.mapreduce.Reducer;
12
13 import java.io.IOException;
14 import java.util.Map;
15
16 import org.apache.hadoop.mapreduce.Job;
17 import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
18 import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
19
20 public class AverageTemperaturePerMonth extends HadoopJob {
21
22     @Override
23     public int run(String[] args) throws Exception {
24         Map<String, String> parsedArgs = parseArgs(args);
25
26         Path inputPath = new Path(parsedArgs.get("--input"));
27         Path outputPath = new Path(parsedArgs.get("--output"));
28

```

```

29     double minimumQuality = Double.parseDouble(parsedArgs.get("--minimumQuality"));
30
31     Job averageTemperature = prepareJob(inputPath, outputPath, TextInputFormat.
class ,
32         AverageTemperatureMapper.class, Text.class, IntWritable.class,
33         AverageTemperatureReducer.class, Text.class, DoubleWritable.class,
34         TextOutputFormat.class);
35
36     averageTemperature.getConfiguration().set("minimumQuality", String.valueOf(
minimumQuality));
37
38     averageTemperature.waitForCompletion(true);
39
40     return 0;
41 }
42
43
44 static class AverageTemperatureMapper extends Mapper<Object, Text, Text,
IntWritable> {
45
46
47     @Override
48     protected void map(Object key, Text line, Context ctx) throws IOException,
InterruptedException {
49         Configuration conf = ctx.getConfiguration();
50         double minimumQuality = Double.parseDouble(conf.get("minimumQuality"));
51
52         String l = line.toString();
53         String[] ls = l.split("\t");
54         if (minimumQuality <= Double.parseDouble(ls[ls.length - 1])) {
55             String K = ls[0] + "\t" + ls[1];
56             ctx.write(new Text(K), new IntWritable(Integer.parseInt(ls[2])));
57         }
58     }
59 }
60
61 }
62
63 static class AverageTemperatureReducer extends Reducer<Text, IntWritable, Text,
DoubleWritable> {
64     @Override
65     protected void reduce(Text key, Iterable<IntWritable> values, Context ctx)
throws IOException, InterruptedException {
66         int sum = 0;
67         int length = 0;
68         for (IntWritable value : values) {
69             sum += value.get();
70             length++;
71         }
72         double average = (double) sum / length;
73         ctx.write(new Text(key), new DoubleWritable(average));
74     }
75 }
76 }
77 }

```

Listing 3: AverageTemperaturePerMonth.java

Clustering

1. Metrics I