TU Berlin

Advanced Information Management

Homework assignment 1

Programming in Hadoop and Clustering Excercises

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,
    FilteringWordCountMapper.class,
35                 Text.class, IntWritable.class, WordCountReducer.class, Text.class,
    IntWritable.class, TextOutputFormat.class);
36
37         wordCount.waitForCompletion(true);
38
39         return 0;
40     }
41
42     static class FilteringWordCountMapper extends Mapper<Object, Text, Text,
    IntWritable> {
43
44         private ArrayList<String> filterList = new ArrayList<>();
45
46         /**
47          * Method to add words that should be filtered out.
48          *
49          * @param fl : the list with filtered words
50          */
51         public void addWordsToFilter(List<String> fl) {
52             this.filterList.addAll(fl);
53         }
54
55         public List getFilterList() {
56             return this.filterList;
57         }
58
59
```

```java
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(),
     false).mapToInt(IntWritable::get).sum()));
84          }
85
86      }
87
88 }
```

Listing 1: FilteringWordCount.java

2. A custom Writable

```java
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
```

```java
        double minimumQuality = Double.parseDouble(parsedArgs.get("--minimumQuality"));

        Job averageTemperature = prepareJob(inputPath, outputPath, TextInputFormat.
    class,
                AverageTemperatureMapper.class, Text.class, IntWritable.class,
                AverageTemperatureReducer.class, Text.class, DoubleWritable.class,
                TextOutputFormat.class);

        averageTemperature.getConfiguration().set("minimumQuality", String.valueOf(
    minimumQuality));

        averageTemperature.waitForCompletion(true);

        return 0;
    }


    static class AverageTemperatureMapper extends Mapper<Object, Text, Text,
    IntWritable> {


        @Override
        protected void map(Object key, Text line, Context ctx) throws IOException,
    InterruptedException {
            Configuration conf = ctx.getConfiguration();
            double minimumQuality = Double.parseDouble(conf.get("minimumQuality"));

            String l = line.toString();
            String[] ls = l.split("\t");
            if (minimumQuality <= Double.parseDouble(ls[ls.length - 1])) {
                String K = ls[0] + "\t" + ls[1];
                ctx.write(new Text(K), new IntWritable(Integer.parseInt(ls[2])));
            }

        }

    }

    static class AverageTemperatureReducer extends Reducer<Text, IntWritable, Text,
    DoubleWritable> {
        @Override
        protected void reduce(Text key, Iterable<IntWritable> values, Context ctx)
                throws IOException, InterruptedException {
            int sum = 0;
            int length = 0;
            for (IntWritable value : values) {
                sum += value.get();
                length++;
            }
            double average = (double) sum / length;
            ctx.write(new Text(key), new DoubleWritable(average));
        }
    }
}
```

Listing 3: AverageTemperaturePerMonth.java

Clustering

1. Metrics I

We have 3 points A(4,8), B(9,5) and C(2,2).
The *centroid* is then:
$$(x, y) = \left(\frac{4 + 9 + 2}{3}, \frac{8 + 5 + 2}{3} \right) = (5, 5)$$

We now calculate the error towards the centroid for every point:

A(4,8):
$$\sqrt{(4 - 5)^2 + (8 - 5)^2} = \sqrt{10}$$

B(9,5):
$$\sqrt{(9 - 5)^2 + (5 - 5)^2} = \sqrt{16} = 4$$

C(2,2):
$$\sqrt{(2 - 5)^2 + (2 - 5)^2} = \sqrt{18} = 3 \cdot \sqrt{2}$$

The SSE is then:
$$\sqrt{10}^2 + 4^2 + 3 \cdot \sqrt{2}^2 = 44$$

2. Metrics II

If you partition 3 points into 2 clusters, there's one with one element and one with two. The SSE of 1 point is always 0. So the total SSE is only determined by the SSE of the possible combinations of two points.

$$Centroid((3, 0), (0, 7)) = (1.5, 3.5)$$
$$SSE((3, 0), (0, 7)) = \sqrt{2.25 + 12.25}^2 + \sqrt{2.25 + 12.25}^2 = 29$$
$$Centroid(((0, 7), (6, 5)) = (3, 6)$$
$$SSE(((0, 7), (6, 5)) = \sqrt{9 + 1}^2 + \sqrt{9 + 1}^2 = 20$$
$$Centroid((3, 0), (6, 5)) = (4.5, 2.5)$$
$$SSE((3, 0), (6, 5)) = \sqrt{2.25 + 6.25}^2 + \sqrt{2.25 + 6.25}^2 = 17$$

The SSE is the smallest in the last configuration therefore the the most optimal split in two groups is the following:

Cluster 1: [(0,7)]
Cluster 2: [(3,0),(6,5)]