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

```java
package de.tuberlin.dima.aim3.assignment1;

import de.tuberlin.dima.aim3.HadoopJob;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Map;
import java.util.function.Function;
import java.util.regex.Pattern;
import java.util.stream.StreamSupport;

import static java.util.stream.Collectors.counting;
import static java.util.stream.Collectors.groupingBy;

public class FilteringWordCount extends HadoopJob {

    @Override
    public int run(String[] args) throws Exception {
        Map<String, String> parsedArgs = parseArgs(args);

        Path inputPath = new Path(parsedArgs.get("--input"));
        Path outputPath = new Path(parsedArgs.get("--output"));

        Job wordCount = prepareJob(inputPath, outputPath, TextInputFormat.class,
    FilteringWordCountMapper.class,
                Text.class, IntWritable.class, WordCountReducer.class, Text.class,
    IntWritable.class, TextOutputFormat.class);

        wordCount.waitForCompletion(true);

        return 0;
    }

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

        private ArrayList<String> filterList = new ArrayList<>();

        /**
         * Method to add words that should be filtered out.
         *
         * @param fl : the list with filtered words
         */
        public void addWordsToFilter(List<String> fl) {
            this.filterList.addAll(fl);
        }

        public List getFilterList() {
            return this.filterList;
        }


```

```
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

```
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)
66                  throws IOException, InterruptedException {
67              int sum = 0;
68              int length = 0;
69              for (IntWritable value : values) {
70                  sum += value.get();
71                  length++;
72              }
73              double average = (double) sum / length;
74              ctx.write(new Text(key), new DoubleWritable(average));
75          }
76      }
77  }
```

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