Exercise 2:
Hadoop MapReduce

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Concepts and Technologies for Distributed Systems and Big Data Processing – SS 2016

Task 1 Paper Reading

Read the original MapReduce paper by Dean and Ghemawat [1]. You can find the paper at
`http://research.google.com/archive/mapreduce.html`

Answer the following questions:

a) How do the input keys/values, the intermediate keys/values and the output keys/values relate?

b) How does MapReduce deal with node failures?

c) What is the meaning and the implication of locality? How is it used?

d) Which problem is addressed by introducing a *combiner function* to the MapReduce model?

Task 2 Implementation

First have a look at the MapReduce examples from the lecture. You can download the code from the course website. The ZIP file contains a Maven project, which can be imported into your IDE. An Eclipse project is provided, too. If importing the Eclipse project should not work, please have a look into the informatory section at the end of the exercise to see how to import a Maven project into Eclipse.

You can execute the examples, like WordCount, from the *Run → Run Configurations* menu using the provided run configurations. After the execution has successfully completed, you will find a file `_SUCCESS` and a file `part-r-xxxxx` in the output directory. The latter contains your output.

Important note: Make sure you delete the output directory before running your application again. Otherwise you will get an error from Hadoop indicating that the directory already exists.

Taking into account the WordCount example from the lecture, implement FriendCount, which should count the number of friends for each given person. The list of friends is provided in the file `input/friendcount/friends`. Figure 1 shows the input given by the file and the expected output.

```
1 Jim,Sue
2 Sue,Jim
3 Lin,Joe
4 Joe,Lin
5 Jim,Kai
6 Kai,Jim
7 Jim,Lin
8 Lin,Jim
```

(a) Input

```
1 Jim 3
2 Joe 1
3 Kai 1
4 Lin 2
5 Sue 1
```

(b) Expected output.

Figure 1: FriendCount

Task 3 Completion

Complete the following code for WordLength, which should count how many words belong to each of the following four length categories:

tiny: 1 letter — *small*: 2–4 letters — *medium*: 5–9 letters — *big*: more than 10 letters

```
1  public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable> {
2    private final static IntWritable one = new IntWritable(1);
3    private Text category = new Text();
4
5    @Override
6    protected void map(Object key, Text value, Context context) throws IOException, InterruptedException {
7      StringTokenizer tokenizer = new StringTokenizer(value.toString(), ",;\\. \t\n\r\f");
8      while (tokenizer.hasMoreTokens()) {
9        String word = tokenizer.nextToken();
10
11
12
13
14
15
16
17
18
19
20
21     }
22   }
23 }
24
25 public static class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
26   private IntWritable result = new IntWritable();
27
28   @Override
29   protected void reduce(Text key, Iterable<IntWritable> values, Context context)
30       throws IOException, InterruptedException {
31
32
33
34
35
36
37
38
39
40
41
42   }
43 }
```

Task 4 Comprehension

Understand and explain what the following code does. What is the output of the program for the following input?

file1.txt: Hello World Bye World

file2.txt: Hello Hadoop Goodbye Hadoop

```
1  public static class TokenizerMapper extends Mapper<LongWritable, Text, Text, Text> {
2    private Text word = new Text();
3    private Text file = new Text();
4
5    @Override
6    protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
7      FileSplit fileSplit = (FileSplit)context.getInputSplit();
8      String fileName = fileSplit.getPath().getName();
9      file.set(fileName);
10
11     StringTokenizer tokenizer = new StringTokenizer(value.toString());
12     while (tokenizer.hasMoreTokens()) {
13       word.set(tokenizer.nextToken());
14       context.write(word, file);
15     }
16   }
17 }
18
19 public static class InvertedReducer extends Reducer<Text, Text, Text, Text> {
20   private Text result = new Text();
21
22   @Override
23   protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
24     StringBuilder sb = new StringBuilder();
25     for (Text val: values) {
26       sb.append(val);
27
28       if(values.iterator().hasNext()) {
29         sb.append(",");
30       }
31     }
32     result.set(sb.toString());
33     context.write(key, result);
34   }
35 }
```

Importing the Maven project into Eclipse

In Eclipse, open the *Import* dialog via *File* → *Import*.
Choose *Existing Maven Projects*.
Choose the root directory of the Maven project containing the `pom.xml` file.
Select the project and click *Finish*. This should also download the required dependencies.

Open the file you want to execute, e.g., `FriendCount.java`, in the Eclipse editor.
Open the *Run Configurations* dialog via *Run* → *Run Configurations*.
Right-click on *Java Application* and click *New* to create a new configuration for the currently open file.
The *Main class* should already be set up correctly, e.g., `de.tud.stg.FriendCount`.
Switch to the *Arguments* tab and give the input and output directories as *Program arguments*, e.g., `input/friendcount output/friendcount`.

References

[1] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.