Exercise 2: Hadoop MapReduce

TECHNISCHE UNIVERSITÄT DARMSTADT

Concepts and Technologies for Distributed Systems and Big Data Processing – SS 2017

Solution 2 Implementation

You can download the code for the solution for this task from the course website.

Solution 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

```java
public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable> {
  private final static IntWritable one = new IntWritable(1);
  private Text category = new Text();

  @Override
  protected void map(Object key, Text value, Context context) throws IOException, InterruptedException {
    StringTokenizer tokenizer = new StringTokenizer(value.toString(), ",;\\. \t\n\r\f");
    while (tokenizer.hasMoreTokens()) {
      String word = tokenizer.nextToken();

      int length = word.length();
      String c = ((length == 1) ? "tiny" :
              (length >= 2 && length <= 4) ? "small" :
              (length >= 5 && length <= 9) ? "medium": "big");
      category.set(c);
      context.write(category, one);

    }
  }
}

public static class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
  private IntWritable result = new IntWritable();

  @Override
  protected void reduce(Text key, Iterable<IntWritable> values, Context context)
      throws IOException, InterruptedException {

    int sum = 0;
    for (IntWritable val: values) {
        sum += val.get();
    }
    result.set(sum);
    context.write(key, result);

  }
}
```

Solution 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

```java
public static class TokenizerMapper extends Mapper<LongWritable, Text, Text, Text> {
  private Text word = new Text();
  private Text file = new Text();

  @Override
  protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
    FileSplit fileSplit = (FileSplit)context.getInputSplit();
    String fileName = fileSplit.getPath().getName();
    file.set(fileName);

    StringTokenizer tokenizer = new StringTokenizer(value.toString());
    while (tokenizer.hasMoreTokens()) {
      word.set(tokenizer.nextToken());
      context.write(word, file);
    }
  }
}

public static class InvertedReducer extends Reducer<Text, Text, Text, Text> {
  private Text result = new Text();

  @Override
  protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
    StringBuilder sb = new StringBuilder();
    for (Text val: values) {
      sb.append(val);

      if(values.iterator().hasNext()) {
        sb.append(",");
      }
    }
    result.set(sb.toString());
    context.write(key, result);
  }
}
```

The code computes the inverted index for the given documents, i.e., a list of references to documents for each word. It produces the following output:

```
Bye file01
Goodbye file02
Hadoop file02,file02
Hello file02,file01
World file01,file01
```