22MCA0139

Rajat Singh

Cloud Computing DA 1

Question 1: Counting the number of occurrences of each word in a collection of documents.

Solution:

To count the number of occurrences of each word in a collection of documents using the MapReduce algorithm in Java, you can follow these steps:

- Map Phase: The mapper function takes in the input data (in this case, each document) and extracts each word from it. For each word, the mapper emits a key-value pair, where the key is the word and the value is 1.
- Shuffle Phase: The MapReduce framework shuffles and sorts the key-value pairs generated by the mapper, so that all values with the same key are grouped together.
- Reduce Phase: The reducer function takes in the key-value pairs generated by the shuffle phase and aggregates them. For each key (word), the reducer sums up all the values (counts) associated with that key to get the total count for that word.

Code:

```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
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.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {
```

```
  public static class TokenizerMapper extends Mapper<LongWritable, Text, Text,
IntWritable> {

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

  public static class IntSumReducer extends Reducer<Text, IntWritable, Text,
IntWritable> {

    private IntWritable result = new IntWritable();

    public 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);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

In this code, the TokenizerMapper class is responsible for extracting each word from the input data and emitting a key-value pair for each word. The IntSumReducer class aggregates the key-value pairs generated by the mapper and produces the final output, which is the count of each word in the input data.

To run this code, we need to pass the input directory as the first command line argument and the output directory as the second command line argument. For example:

```
$ hadoop jar wordcount.jar WordCount /input /output
```

This will run the MapReduce job on the input directory and produce the output in the output directory

Question 2: Counting the number of occurrences of words having the same size, or the same number of letters, in a collection of documents.

Solution:

The approach remains nearly the same, but the code is tweaked a little bit to suit this problem.

Code:

```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
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.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordLengthCount {
```

```
  public static class TokenizerMapper extends Mapper<LongWritable, Text,
IntWritable, IntWritable> {

    private final static IntWritable one = new IntWritable(1);
    private IntWritable wordLength = new IntWritable();

    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        String word = itr.nextToken();
        wordLength.set(word.length());
        context.write(wordLength, one);
      }
    }
  }

  public static class IntSumReducer extends Reducer<IntWritable, IntWritable,
IntWritable, IntWritable> {

    private IntWritable result = new IntWritable();

    public void reduce(IntWritable 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);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word length count");
    job.setJarByClass(WordLengthCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(IntWritable.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

In this modified code, the TokenizerMapper class extracts each word from the input data and emits a key-value pair for each word. The key is the length of the word and the value is 1. The IntSumReducer class aggregates the key-value pairs generated by the mapper and produces the final output, which is the count of words having the same length in the input data.

Question 3: Counting the number of occurrences of Palindrome in a collection of documents.

Solution:

```
import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
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.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class PalindromeCount {
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "palindrome count");
        job.setJarByClass(PalindromeCount.class);
        job.setMapperClass(PalindromeMapper.class);
        job.setCombinerClass(PalindromeReducer.class);
        job.setReducerClass(PalindromeReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }

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

```
        public void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
            String[] tokens = value.toString().split("\\s+");
            for (String token : tokens) {
                if (isPalindrome(token)) {
                    word.set(token);
                    context.write(word, one);
                }
            }
        }

        private boolean isPalindrome(String str) {
            int i = 0, j = str.length() - 1;
            while (i < j) {
                if (str.charAt(i) != str.charAt(j)) {
                    return false;
                }
                i++;
                j--;
            }
            return true;
        }
    }

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

        public 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);
        }
    }
}
```