Assignment No. 6

Pipeline multiple Map Reduce jobs

Example: Pipelining Two Jobs

Job 1: Word Count (Word frequency count)

This first job counts the occurrences of each word in the input text files.

```
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.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<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[] words = value.toString().split("\\s+");
            for (String wordStr : words) {
                word.set(wordStr);
                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);
```

```
    }
}
```

Job 2: Filter Words with Frequency Greater Than 2

The second job processes the output of the first job to filter and only output words that have a frequency greater than 2

```
public class FilterWords {
    public static class FilterMapper extends Mapper<Object, Text, Text,
IntWritable> {
        private IntWritable count = new IntWritable();

        public void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
            String[] fields = value.toString().split("\t");
            String word = fields[0];
            int wordCount = Integer.parseInt(fields[1]);

            // Output only words with count greater than 2
            if (wordCount > 2) {
                count.set(wordCount);
                context.write(new Text(word), count);
            }
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "filter words");
        job.setJarByClass(FilterWords.class);
        job.setMapperClass(FilterMapper.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));  // Input
path from the first job's output
        FileOutputFormat.setOutputPath(job, new Path(args[1]));  // Output
path

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

- **Output format**: The output of each job should be compatible with the input format of the next job. For instance, if the output of Job 1 is a simple key-value pair (word and count), Job 2 should be able to process that format directly.

Step 1: Export Java Eclipse Project Jar File to Cloudera
Step 2. Make firstfile.txt file vi editor ->Write data
Step 3: Perform Below commands on terminal

Command Map Reduce Code

1) Transfer all local file to hadoop

```
Hdfs dfs –put firstfile.txt /user/cloudera

Hdfs dfs –put PipLine1.jar /user/cloudera
```

2) Run First job of Java Jar File for Map Reduce Operation

```
hadoop jar PipLine1.jar wordcount firstfile.txt outpip1
```

3) Run Second job of Java Jar File for Map Reduce Operation

```
hadoop jar PipLine1.jar FilterWords outpip1 outpip2
```

4) List outputfile

```
hdfs dfs –ls /user/cloudera/outpip2
```

5) Show outputfile

```
hdfs dfs –cat /user/cloudera/outpip2/part-r-00000
```