

Lab Lecture #2

Exercise (Old HADOOP APIs)

- We will see a basic Hadoop implementation of the word count application. Create the following WordCount.java source file in the \$HADOOP_HOME dir:

```
import java.io.*;
import java.util.*;

import org.apache.hadoop.fs.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;

public class WordCount {
    public static class Map extends MapReduceBase implements
        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,
            OutputCollector<Text, IntWritable> output, Reporter reporter)
            throws IOException {
            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                output.collect(word, one);
            }
        }
    }

    public static class Reduce extends MapReduceBase implements
        Reducer<Text, IntWritable, Text, IntWritable> {
        public void reduce(Text key, Iterator<IntWritable> values,
            OutputCollector<Text, IntWritable> output, Reporter reporter)
            throws IOException {
            int sum = 0;
            while (values.hasNext())
                sum += values.next().get();
            output.collect(key, new IntWritable(sum));
        }
    }

    public static void main(String[] args) throws Exception {
        JobConf conf = new JobConf(WordCount.class);
        conf.setJobName("wordcount");

        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);

        conf.setMapperClass(Map.class);
        conf.setCombinerClass(Reduce.class);
        conf.setReducerClass(Reduce.class);

        conf.setInputFormat(TextInputFormat.class);
        conf.setOutputFormat(TextOutputFormat.class);

        FileInputFormat.setInputPaths(conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));

        JobClient.runJob(conf);
    }
}
```

- Compile WordCount.java and create a jar file:


```
hadoop@localhost$ cd ${HADOOP_HOME}
hadoop@localhost$ mkdir classes
hadoop@localhost$ javac -cp hadoop-core-1.0.4.jar \
    -d classes WordCount.java
hadoop@localhost$ jar -cvf wordcount.jar -C classes/ .
```
- Create the following sample files in your \$HOME dir:


```
hadoop@localhost$ echo Hello World > file01
hadoop@localhost$ echo Hello Java > file02
hadoop@localhost$ echo Java and MapReduce > file03
```
- Create the HDFS input dir:


```
hadoop@localhost$ bin/hadoop fs -mkdir /user/hadoop/wordcount/input
```
- Copy the sample files in HDFS:


```
hadoop@localhost$ bin/hadoop fs -put file0? /user/hadoop/wordcount/input/
```
- Check the sample files have been copied:


```
hadoop@localhost$ bin/hadoop fs -ls /user/hadoop/wordcount/input/
hadoop@localhost$ bin/hadoop fs -cat /user/hadoop/wordcount/input/file01
hadoop@localhost$ bin/hadoop fs -cat /user/hadoop/wordcount/input/file02
hadoop@localhost$ bin/hadoop fs -cat /user/hadoop/wordcount/input/file03
```
- Run the application:


```
hadoop@localhost$ bin/hadoop jar wordcount.jar WordCount \
    /user/hadoop/wordcount/input /user/hadoop/wordcount/output
```
- Check the output:


```
hadoop@localhost$ bin/hadoop fs -cat \
    /user/hadoop/wordcount/output/part-00000
```
- Clean up


```
hadoop@localhost$ rm -r WordCount.java wordcount.jar classes/ file0?
hadoop@localhost$ bin/hadoop fs -rmr /user/hadoop/wordcount
```

Exercise (New HADOOP APIs)

- The following WordCount.java source file implements the previous exercises with the new Hadoop 0.20.2 APIs.

```
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 NewMapper 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 {
            StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens()) {
                word.set(itr.nextToken());
                context.write(word, one);
            }
        }
    }

    public static class NewReducer 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 = new Job(conf, "wordcount");
        job.setJarByClass(WordCountNew.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        job.setMapperClass(NewMapper.class);
        job.setCombinerClass(NewReducer.class);
        job.setReducerClass(NewReducer.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

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

}