WordCount

WordCount Example

WordCount example reads text files and counts how often words occur. The input is text files and the output is text files, each line of which contains a word and the count of how often it occured, separated by a tab.

Each mapper takes a line as input and breaks it into words. It then emits a key/value pair of the word and 1. Each reducer sums the counts for each word and emits a single key/value with the word and sum.

As an optimization, the reducer is also used as a combiner on the map outputs. This reduces the amount of data sent across the network by combining each word into a single record.

To run the example, the command syntax is

bin/hadoop jar hadoop-*-examples.jar wordcount [-m <#maps>] [-r <#reducers>] <in-dir> <out-dir>

All of the files in the input directory (called in-dir in the command line above) are read and the counts of words in the input are written to the output directory (called out-dir above). It is assumed that both inputs and outputs are stored in HDFS (see ImportantConcepts). If your input is not already in HDFS, but is rather in a local file system somewhere, you need to copy the data into HDFS using a command like this:

bin/hadoop dfs -mkdir <hdfs-dir>

bin/hadoop dfs -copyFromLocal <local-dir> <hdfs-dir>

As of version 0.17.2.1, you only need to run a command like this:

bin/hadoop dfs -copyFromLocal <local-dir> <hdfs-dir>

Word count supports generic options : see DevelopmentCommandLineOptions

Below is the standard wordcount example implemented in Java:

```java
package org.myorg;

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordCount {

 public static class Map 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 {
        String line = value.toString();
         StringTokenizer tokenizer = new StringTokenizer(line);
         while (tokenizer.hasMoreTokens()) {
             word.set(tokenizer.nextToken());
             context.write(word, one);
         }
    }
 }

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

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
      throws IOException, InterruptedException {
        int sum = 0;
         for (IntWritable val : values) {
             sum += val.get();
         }
         context.write(key, new IntWritable(sum));
    }
 }

 public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();

        Job job = new Job(conf, "wordcount");

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

    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

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

    job.waitForCompletion(true);
 }

}
```