DEVELOPING MAP REDUCE APPLICATION

Input and Output types of a MapReduce job:

(input) <k1, v1> -> **map** -> <k2, v2> -> **combine** -> <k2, v2> -> **reduce** -> <k3, v3> (output)

**Example: WordCount v1.0**

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 {

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 {

// The **configuration** holds information about the job tracker, the input, output format and //the various other parameters of the **map reduce** job.

Configuration conf = new Configuration();

//configure the job, submit it, control its execution, and query the state

Job job = Job.getInstance(conf, "word count");

// build jar file

job.setJarByClass(WordCount.class);

**// Mapper maps input <key, value> pairs to a set of intermediate <key, value> pairs.**

job.setMapperClass(TokenizerMapper.class);

// **local aggregation of the intermediate outputs, which helps to cut down the amount of data //transferred from the Mapper to the Reducer.This is optional**

job.setCombinerClass(IntSumReducer.class);

**// reducer class**

job.setReducerClass(IntSumReducer.class);

// word in word count

job.setOutputKeyClass(Text.class);

// count of word in word count

job.setOutputValueClass(IntWritable.class);

//path to input file

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

//output file for word count is created by the program

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

//exit when the job is completed

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

}

}

Map Method

The Mapper implementation, via the map method, processes one line at a time, as provided by the specified TextInputFormat.

It then splits the line into tokens separated by whitespaces, via the StringTokenizer, and emits a key-value pair of < <word>, 1>.

##### Shuffle

Input to the Reducer is the sorted output of the mappers.

The shuffle and sort phases occur simultaneously; while map-outputs are being fetched they are merged

Reducer Method

The Reducer implementation, via the reduce method just sums up the values, which are the occurrence counts for each key (i.e. words in this example).

The output of the Reducer is *not sorted*.

**HADOOP CLUSTER CONFIGURATION**

1. FORMAT THE NAMENODE (NAMESPACE IN HDFS)

$bin/hdfs namenode –format

1. START THE DATA NODE AND YARN

sbin/start -dfs.sh

sbin/start-yarn.sh

1. PROCESSES IN EXECUTION

jps

3845 NameNode

3973 DataNode

4283 Jps

4175 SecondaryNameNode

1. STOP ALL PROCESS FOR EXIT

stop-all.sh

Input to Mapper

**<Line1,Hello world bye world >**

**<Line2, Hello hadoop goodbye hadoop>**

**Hello world bye world**

**Hello hadoop goodbye hadoop**

< **Hello, 1>**

**< Hadoop, 1>**

**< Goodbye, 1>**

**< Hadoop, 1>**

**< Hello, 1>**

**< World, 1>**

**< Bye, 1>**

**< World, 1>**

Mapper class

Word file with two lines

Output of mapper Map1

Map2

**job.setCombinerClass(IntSumReducer.class);**

**reduce()**

< **Hello, 1,1>**

**< Hadoop, 1,1>**

**< Goodbye, 1>**

**<World, 1>**

**< Bye, 1>**

**< Goodbye, 1>**

**< Hadoop, 2>**

**< Hello, 2>**

**< World, 2>**

**Reference:**

**https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html#Inputs\_and\_Outputs**