Experiment-6

Aim:

Run a basic Word Count Map Reduce program to understand Map Reduce Paradigm.

Solution

Source Code of WordCount.java

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 {

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);

}

}

**Run the wordcount example**

**Step-1**

Create a local directory for the application where the .JAVA files and input files can be stored. Save the source code of the word count program as WordCount.java.

**Step-2**

The next step is to compile the code and create JAR file.

Assuming the following environment variables are set:

JAVA\_HOME, HADOOP\_HOME and PATH

HADOOP\_CLASSPATH={JAVA\_HOME}/lib/tools.jar

Compile WordCount.java and create a jar:

hadoop com.sun.tools.javac.Main WordCount.java

jar cf wc.jar WordCount\*.class

**Step-3**

Assuming that:

Create a directory named as input in HDFS as given below

hdfs dfs –mkdir /wordcount/input

/wordcount/input - input directory in HDFS

Create two text files as sample input and copy them from local directory to the input directory in HDFS as given below:

hdfs dfs –put file01 /wordcount/input

hdfs dfs –put file01 /wordcount/input

see the list of files in the input directory in HDFS

hadoop fs -ls /wordcount/input/

/wordcount/input/file01

/wordcount/input/file02

Showing contents of input files

$ bin/hadoop fs -cat /user/joe/wordcount/input/file01

Hello World Bye World

$ bin/hadoop fs -cat /user/joe/wordcount/input/file02

Hello Hadoop Goodbye Hadoop

**Step-4**

Run the application:

Please note that your **wc.jar** file should not exist at the same location where other WordCount\*.class file are present. So, place the wc.jar file to some other directory and then execute the following command.

hadoop jar wc.jar WordCount /wordcount/input /wordcount/output

Note that the output directory will be automatically created by Hadoop.

**Output:**

hadoop fs -cat /wordcount/output/part-r-00000

Bye 1

Goodbye 1

Hadoop 2

Hello 2

World 2