Write Map Reduce Lab

In the Map Reduce Lab you learnt how to compile and run a java map-reduce job. In this section we will go in more detail and will write a simple map reduce word count program in Java step by step. This tutorial requires basic knowledge of java programming.

# Writing Simple Map Reduce

For a Hadoop Map Reduce job the basic minimum inputs that you need to provide is the input and output location. Lets write, run and understand a simple Map Reduce job with minimum input.

Input location – DFS location from where input files are read.

Output location – Empty DFS location to which the output will be written.

## Write Program

* Create program directory

Create a directory workspace in home directory and sub folder MySimpleMR.

|  |
| --- |
| cd /home/hadoop  mkdir workspace  mkdir workspace/MySimpleMR |

We will use the workspace directory for all the map reduce programs of this lab.

* Write the program step by step

1. Create a java program MySimpleMR.java inside MySimpleMR folder with the default main class.

|  |
| --- |
| public class MySimpleMR {  public static void main(String[] args) throws Exception {  *//code goes here*  }  } |

1. Create a job

|  |
| --- |
| public static void main(String[] args) throws Exception {  *Configuration conf = new Configuration();*  *Job job = new Job(conf,”mysimplemr”);*  } |

Hadoop job is configured using their own configuration APIs. A Configuration represents a

collection of properties and values.

1. Specify input and output locations

|  |
| --- |
| public static void main(String[] args) throws Exception {  Configuration conf = new Configuration();  Job job = new Job(conf,”mysimplemr”);  *FileInputFormat.addInputPath(job,new Path("/home/hadoop/workspace/data/dfsinput"));*  *FileInputFormat.addInputPath(job,new Path("/home/hadoop/workspace/data/dfsoutput"));*  } |

Sets the path of input and output directory for map reduce job.

1. Submit the job

|  |
| --- |
| public static void main(String[] args) throws Exception {  Configuration conf = new Configuration();  Job job = new Job(conf,”mysimplemr”);  FileInputFormat.addInputPath(job,  new Path("/home/hadoop/workspace/data/dfsinput"));  FileOutputFormat.setOutputPath(job,  new Path("/home/hadoop/workspace/data/dfsoutput"));  *job.waitForCompletion(true);*  } |

waitForCompletion submits the job and wait for it to finish.

1. Overall program

Importing the required packages in the code the overall code will look like

|  |
| --- |
| import java.io.IOException;      import java.util.\*;        import org.apache.hadoop.fs.Path;      import org.apache.hadoop.mapreduce.\*;      import org.apache.hadoop.conf.\*;      import org.apache.hadoop.io.\*;      import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;      import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;        public class MySimpleMR {        public static void main(String[] args) throws Exception {          Configuration conf = new Configuration();          Job job = new Job(conf,"simplemr");  FileInputFormat.addInputPath(job,  new Path("/home/hadoop/workspace/data/dfsinput"));  FileOutputFormat.setOutputPath(job,  new Path("/home/hadoop/workspace/data/dfsoutput"));          job.waitForCompletion(true);        }      } |

The program is now read. Let’s compile it and create a jar.

## Compile and Create Jar

* Compile the java code

|  |
| --- |
| $ cd /home/hadoop/workspace/MySimpleMR  $ javac -classpath $HADOOP\_HOME/hadoop-core-1.0.3.jar:$HADOOP\_HOME/hadoop-client-1.0.3.jar MySimpleMR.java |

* Create jar using the compiled classes

|  |
| --- |
| $ cd /home/hadoop/workspace/MySimpleMR  $ jar cvf MySimpleMR.jar MySimpleMR\*.class |

* Verify the jar file

|  |
| --- |
| $ cd /home/hadoop/workspace/MySimpleMR  $ jar tvf /home/hadoop/workspace/MySimpleMR/MySimpleMR.jar  0 Tue Jan 22 23:43:44 PST 2013 META-INF/  68 Tue Jan 22 23:43:44 PST 2013 META-INF/MANIFEST.MF  970 Tue Jan 22 23:42:56 PST 2013 MySimpleMR.class |

Next we will create the input hdfs files and run the job.

## Creating input data

* Create input files

Create data input folder inside workspace directory

|  |
| --- |
| $ cd /home/hadoop/workspace  $ mkdir data  $ mkdir data/input  $ vi /home/hadoop/workspace/data/input/input1.txt  Content:  Welcome to Hadoop Lab  Learning MapReduce  Save and close the file |

Another input file input2.txt

|  |
| --- |
| $ vi /home/hadoop/workspace/data/input/input2.txt  Content:  Learning Hadoop  Save and close the file |

* Copy the input files to input dfs location

The input dfs location used in our program is /home/hadoop/workspace/data/dfsinput

* Create the hdfs directory
* Copy the input text files to this input dfs location.
* Delete the dfsinput location if it already exists.
* Verify the input dfs files

|  |
| --- |
| $ hadoop fs -rmr /home/hadoop/workspace/data/dfsinput  $ hadoop fs -mkdir /home/hadoop/workspace/data/dfsinput  $ hadoop fs -copyFromLocal /home/hadoop/workspace/data/input/\* /home/hadoop/workspace/data/dfsinput  $ hadoop fs -ls /home/hadoop/workspace/data/dfsinput |

## Submit job

* Run the job

Lets remove the output directory if it already exists and run the job.

|  |
| --- |
| $ hadoop fs -rmr /home/hadoop/workspace/data/dfsoutput  Deleted hdfs://localhost:54310/home/hadoop/workshop/data/dfsoutput  $ cd /home/hadoop/workspace/MySimpleMR  $ hadoop jar MySimpleMR.jar MySimpleMR |

* Check the output

|  |
| --- |
| $ hadoop fs -cat /home/hadoop/workspace/data/dfsoutput/part-r-00000  Output:  0 Welcome to Hadoop  0 Learning Hadoop  22 Learning MapReduce |

## Understanding the flow

Lets go over the flow sequence for the MySimpleMR job step by step.

|  |  |  |  |
| --- | --- | --- | --- |
| **Flow sequence** | **Explanation** | **Example** | **Data** |
| 1. HDFS Input | Input hdfs data files. | input1.txt  input2.txt | Welcome to Hadoop Lab  Learning MapReduce  Learning Hadoop |
| 1. Input Formatter | Splits the files into tasks.  Provides methods to read the file content.  Generates input pairs K1,V1 for mapper. | Default TextInputFormat.class produces  Key (K1) – LongWritable (offset of the beginning of the line in the file)  Value (V1)– Text (line of the text) | Generated K1,V1 Pairs  0,Welcome to Hadoop Lab  22,Learning MapReduce  0,Learning Hadoop |
| 1. Mapper | Processes the input key value pair K1,V1  Emits intermediate key value pair K2,V2  The default mapper just copies input pair to output. | Default Types: K1-LongWritable, V1-Text  Input K1,V1 -> emits K2,V2(equals K1,V1)  So in this example K1=K2 and V1=V2 | Generated K2,V2 Pairs same as K1,V1  0,Welcome to Hadoop Lab  22,Learning MapReduce  0,Learning Hadoop |
| 1. Partition, Shuffle and Sort | The map task output is partitioned based on number of reducer.  Within each partition the pairs are sorted.  Shuffle determines and transfers the partitions to respective reducers. | As this is single node. There is only one reduce task and hence one partition.  The pairs gets sorted on key and the final pair K3,V3 is generated. | Sorted Pair K3,V3 based on key  0,Welcome to Hadoop Lab  0,Learning Hadoop  21,Learning MapReduce |
| 1. Reducer | Receives the final key value pairs K3,V3  The reduce function is called once for each key and their values.  The reducer processes the values for the key and generates output. | Default Types: K3-LongWritable, V3-Text  Sorted output becomes input to reducer.  Default reducer just copies the input to the output.  Input K3,V3 pairs becomes output of reduce task. | Output of reduce task  0,Welcome to Hadoop Lab  0,Learning Hadoop  21,Learning MapReduce |
| 1. Output Formatter | Writes the output to hdfs in defined format.  Each reducer writes a separate file part-r-nnnn.  nnnn – partition id | Default output formatter: TextOutputFormat - Converts the key value pair to string  Writes string one per line in the output hdfs location  Output file: part-r-00000 | Output of reduce task  part-r-00000  As there is only one partition. The first partition number is 00000. |
| 1. HDFS Output | The output is writen in specified output hdfs directory specified as parameter to the job. | The output location specified in the job is /home/hadoop/workspace  /data/dfsinput  The hdfs output location will have output file part-r-00000 | dfs cat command on the hdfs output file will show the output  0,Welcome to Hadoop Lab  0,Learning Hadoop  21,Learning MapReduce |

## Key Points

* The basic map reduce job requires only input and output hdfs location.
* The default input formatter is TextInputFormat which reads the input file and generates key (LongWritable) and value (Text) pairs.
* The default mapper copies the input key value pairs to output.
* The default reducer receives sorted key value pairs and copies it to output.
* The default output formatter is TextOutputFormat which converts key value pairs into string and writes it per line to the output dfs location.

# Writing Word Count Map Reduce

In the word count example for the same input files used in previous example the expected output will be count of all the words.

Input Files:

/home/hadoop/workspace/data/input/input1.txt

|  |
| --- |
| Welcome to Hadoop Lab  Learning MapReduce |

/home/hadoop/workspace/data/input/input2.txt

|  |
| --- |
| Learning Hadoop |

Expected output: Count of words from the input files

|  |  |
| --- | --- |
| Word | Count |
| Welcome  to  Hadoop  Lab  Learning  MapReduce | 1  1  2  1  2  1 |

Approach: The approach would be to

* Read the files line by line
* For each line get the count of each word
* Identify the common words
* Sum up the count of same word to get final count

Lets put that approach in map reduce job perspective. In addition to the job setup that we did in MySimpleMR example we will need to configure few more things and write the mapper and reducer tasks. The default mapper and reducer as we saw in previous example will not work here.

|  |  |  |  |
| --- | --- | --- | --- |
| **Activity** | **Activity Performer** | **What to use** | **How to configure** |
| Read the files line by line | InputFormatter.  The input formatter as we saw in previous example can read the files line by line. | There are various types of InputFormatter which one to use?  The default TextInputFormat.class produces  Key – LongWritable (offset of the beginning of the line in the file)  Value – Text (line of the text)  We can use the default one here.  Output pairs:  0,Welcome to Hadoop Lab  21,Learning MapReduce  0,Learning Hadoop | The InputFormatter is configured in the job.  *job.setInputFormatClass*  *(TextInputFormat.class);*  Configuring this would be optional as the default one is the same. |
| Split the lines into words. | This activity will be performed by mapper.  The input formatter feeds the mapper pairs K1,V1  Mapper function will get the input pairs and it will emit word and count pair. | Mapper’s map function will be called for each of the pair.  K1,V1 – 0,Welcome to Hadoop Lab  Mapper will emit  K2,V2 pairs -> <Welcome,1>,<to,1>,  <Hadoop,1>,<Lab,1>  This will be repeated for each K1,V1 | To implement mapper the map class has to extend the Mapper<K1,V1,K2,V2> class.  Map extends *Mapper<LongWritable,*  *Text,Text,IntWritable>*  The map function to use string tokenizer to split the line into words.  *StringTokenizer tokenizer = new StringTokenizer(line);*  *while (tokenizer.hasMoreTokens()) {*  *context.write(word, one);*  *}* |
| Identify the common words | This is performed by sort and shuffle. | The same key are coalesced.  K3,list(V3)  The default shuffle and sort mechanism does it. | As its handled by the internal shuffle and sort mechanism no configuration is needed. |
| Sum up the count of same word to get final count | This activity will be performed by reducer.  The reduce function will sum up the count from the list of values and emit the word and total count pairs. | For each of the coalesced pair K3,list(V3) the reduce function is called in loop.  Input: K3,list(V3) -> <Hadoop,[1,1]>  Reducer will sum the values to emit the output as  <Hadoop,2> | To implement reducer the reduce class has to extend the Reducer<K3,V3,OutKey,  OutValue> class.  Reduce extends *Reducer<Text,*  *Iterable<IntWritable>Text,*  *,IntWritable>*  The reduce function will sum up the values and write the output  *for (IntWritable val : values) {*  *sum += val.get();*  *}*  *context.write(key, new IntWritable(sum));* |

## Write Program

* Create directory

Create a directory MyWordCount example under workspace in home directory

|  |
| --- |
| cd /home/hadoop/workspace  mkdir MyWordCount |

* Write the job part of the code in the main class

1. Create a java program MyWordCound.java inside MyWordCount folder with the default main class. Set the job input and output path as done in previous example.

|  |
| --- |
| public class MyWordCount {  public static void main(String[] args) throws Exception {  Job job = new Job(conf, "wordcountsearch");  FileInputFormat.addInputPath(job,  new Path(“/home/hadoop/workspace/data/dfsinput”));  FileOutputFormat.setOutputPath(job,  new Path(“/home/hadoop/workspace/data/dfsoutput”));  job.waitForCompletion(true);  }  } |

1. Specify mapper and reducer classes

Lets call our mapper and reducer class as Map.class and Reduce.class. We will go through the details of these class implementation later.

|  |
| --- |
| . . .  FileOutputFormat.setOutputPath(job,  new Path(“/home/hadoop/workspace/data/dfsoutput”));  *job.setJarByClass(WordCountSearch.class);*  *job.setMapperClass(Map.class);*  *job.setReducerClass(Reduce.class);*  job.waitForCompletion(true);  . . . |

As we are not using the default mapper and reducer and going to have our own customization we need to configure the job to use our Map and Reduce class.

Also we need to specify the jar in which the Map and Reduce class exists using setJarByClass.

1. Set the output key and value class

|  |
| --- |
| . . .  job.setReducerClass(Reduce.class);  *job.setOutputKeyClass(Text.class);*  *job.setOutputValueClass(IntWritable.class);*  job.waitForCompletion(true);  . . . |

1. The complete main function will look like

|  |
| --- |
| public class MyWordCount {  public static void main(String[] args) throws Exception {  Configuration conf = new Configuration();  Job job = new Job(conf,"mywordcount");  job.setOutputKeyClass(Text.class);  job.setOutputValueClass(IntWritable.class);  job.setJarByClass(MyWordCount.class);  job.setMapperClass(Map.class);  job.setReducerClass(Reduce.class);  FileInputFormat.addInputPath(job,new  Path("/home/hadoop/workspace/data/dfsinput"));  FileOutputFormat.setOutputPath(job,new  Path("/home/hadoop/workspace/data/dfsoutput"));  job.waitForCompletion(true);  }  } |

* Write Mapper part as an inner class to our main MyWordCount class

As discussed in previous section the mapper class has following input output types

Input Key (K1) - LongWritable

Input Value (V1) – Text

Output Key (K2) – Text

Output Value (V2) – IntWritable

Input: <0,”Welcome to Hadoop”>

Output: <Welcome,1>,<To,1>,<Hadoop,1>

The mapper class should extend the Mapper class and override the map function as needed.

The map function will split the input line into words and emit the key value pairs. Using the map for loop as explained in previous section the Mapper implementation would be as

|  |
| --- |
| 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);  }  }  } |

* Write Reducer part which will be again implemented as an inner class

As discussed in previous section the reducer class has following input output types

Input Key (K3) - Text

Input Value (List V3) – IntWritable

Output Key – Text

Output Value – IntWritable

The reducer class implementation should extend the Reducer class and override the map

function as needed.

The reduce function will sum up all the values for a key and emits the output.

The reduce function is called for each key and their set of values

|  |
| --- |
| 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));  }  } |

* Complete Program

Importing the required packages the whole program will look like

|  |
| --- |
| 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.util.\*;  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 MyWordCount {  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,"mywordcount");  job.setOutputKeyClass(Text.class);  job.setOutputValueClass(IntWritable.class);  job.setJarByClass(MyWordCount.class);  job.setMapperClass(Map.class);  job.setReducerClass(Reduce.class);  FileInputFormat.addInputPath(job,new  Path("/home/hadoop/workspace/data/dfsinput"));  FileOutputFormat.setOutputPath(job,new  Path("/home/hadoop/workspace/data/dfsoutput"));  job.waitForCompletion(true);  }  } |

Similar program can be found at

/home/hadoop/Gajadata-Training/labs/mapreduce/WordCount/code/WordCount.java

## Compile and Create jar

* Compile the java code

|  |
| --- |
| $ cd /home/hadoop/workspace/MyWordCount  $ javac -classpath $HADOOP\_HOME/hadoop-core-1.0.3.jar:$HADOOP\_HOME/hadoop-client-1.0.3.jar MyWordCount.java |

* Create jar using the compiled classes

|  |
| --- |
| $ cd /home/hadoop/workspace/MyWordCount  $ jar cvf MyWordCount.jar MyWordCount\*.class |

* Verify the jar file

|  |
| --- |
| $ cd /home/hadoop/workspace/MyWordCount  $ jar cvf MyWordCount.jar \*.classadded manifest  adding: MyWordCount.class(in = 1376) (out= 736)(deflated 46%)  adding: MyWordCount$Map.class(in = 1861) (out= 781)(deflated 58%)  adding: MyWordCount$Reduce.class(in = 1633) (out= 686)(deflated 57%) |

## Creating input data

Skip this step if the input data is already done in previous program.

* Create input files

/home/hadoop/workspace/data/input/input1.txt with content

|  |
| --- |
| Welcome to Hadoop Lab  Learning MapReduce |

/home/hadoop/workspace/data/input/input2.txt with content

|  |
| --- |
| Learning Hadoop |

* Copy the input files to input dfs location

The input dfs location for our program is /home/hadoop/workspace/data/dfsinput

Lets create the directory and copy the input text files to this input dfs location

|  |
| --- |
| $ hadoop fs -mkdir /home/hadoop/workspace/data/dfsinput  $ hadoop fs -copyFromLocal /home/hadoop/workspace/data/input/\* /home/hadoop/workspace/data/dfsinput |

## Run job

* Run the job

Lets remove the output directory if it already exists and run the job.

|  |
| --- |
| $ hadoop fs -rmr /home/hadoop/workspace/data/dfsoutput  Deleted hdfs://localhost:54310/home/hadoop/workshop/data/dfsoutput  $ hadoop jar /home/hadoop/workspace/MyWordCount /MyWordCount.jar MyWordCount |

* Check the output

|  |
| --- |
| $ hadoop fs -cat /home/hadoop/workspace/data/dfsoutput/part-r-00000  Output:  Hadoop 2  Lab 1  Learning 2  MapReduce 1  Welcome 1  to 1 |