**Parallel Data Processing in MapReduce - CS 6240 – 01**

**HW-2**

**Govinda Bachani**

# **Source Code**

1. NO COMBINER

public class WordCount {

public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable>{

//Hashset used as filter the words that need to mapped.

public static Set<Integer> filter= new HashSet<Integer>(Arrays.asList(77,78,79,80,81,109,110,111,112,113));

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());

// Logic to separate out words starting with M,N,O,P,Q or m,n,o,p,q

String str = word.toString();

int a = str.charAt(0);

if(filter.contains(a)){

context.write(word, one);

}

}

}

}

// Custom Partitioner Class. This class extends the Hadoop Partitioner class thus it overrides the getPartition Method.

public static class CustomPartitioner extends Partitioner<Text, IntWritable>{

@Override

public int getPartition(Text key, IntWritable value, int numPartitions) {

// TODO Auto-generated method stub

if(numPartitions==0){

return 0;

}

String word=key.toString();

if (word.charAt(0)== 'M' || word.charAt(0)== 'm'){

return 0;

}

else if (word.charAt(0)== 'N'||word.charAt(0)== 'n'){

return 1 % numPartitions;

}

else if (word.charAt(0)== 'O'||word.charAt(0)== 'o'){

return 2 % numPartitions;

}

else if (word.charAt(0)== 'P'||word.charAt(0)== 'p'){

return 3 % numPartitions;

}

else{

return 4 % numPartitions;

}

}

}

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();

String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();

if (otherArgs.length != 2) {

System.err.println("Usage: wordcount <in> <out>");

System.exit(2);

}

Job job = new Job(conf, "word count");

job.setJarByClass(WordCount.class);

job.setMapperClass(TokenizerMapper.class);

// Setting the practitioner class into the job and the number of reduce tasks as well

job.setNumReduceTasks(5);

job.setPartitionerClass(CustomPartitioner.class);;

//Disabling the default combiner Job

//job.setCombinerClass(IntSumReducer.class);

job.setReducerClass(IntSumReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

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

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

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

}}

1. SiCombiner

public class WordCount {

public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable>{

//Hashset used as filter the words that need to mapped.

public static Set<Integer> filter= new HashSet<Integer>(Arrays.asList(77,78,79,80,81,109,110,111,112,113));

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());

// Logic to separate out words starting with M,N,O,P,Q or m,n,o,p,q

String str = word.toString();

int a = str.charAt(0);

if(filter.contains(a)){

context.write(word, one);

}

}

}

}

// Custom Partitioner Class. This class extends the Hadoop Partitioner class thus it overrides the getPartition Method.

public static class CustomPartitioner extends Partitioner<Text, IntWritable>{

@Override

public int getPartition(Text key, IntWritable value, int numPartitions) {

// TODO Auto-generated method stub

if(numPartitions==0){

return 0;

}

String word=key.toString();

if (word.charAt(0)== 'M' || word.charAt(0)== 'm'){

return 0;

}

else if (word.charAt(0)== 'N'||word.charAt(0)== 'n'){

return 1 % numPartitions;

}

else if (word.charAt(0)== 'O'||word.charAt(0)== 'o'){

return 2 % numPartitions;

}

else if (word.charAt(0)== 'P'||word.charAt(0)== 'p'){

return 3 % numPartitions;

}

else{

return 4 % numPartitions;

}

} }

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();

String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();

if (otherArgs.length != 2) {

System.err.println("Usage: wordcount <in> <out>");

System.exit(2);

}

Job job = new Job(conf, "word count");

job.setJarByClass(WordCount.class);

job.setMapperClass(TokenizerMapper.class);

// Setting the practitioner class into the job and the number of reduce tasks as well

job.setNumReduceTasks(5);

job.setPartitionerClass(CustomPartitioner.class);;

job.setCombinerClass(IntSumReducer.class);

job.setReducerClass(IntSumReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

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

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

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

}}

1. PerMap Tally

public class WordCount {

//Hashset used as filter the words that need to mapped.

public static Set<Integer> filter= new HashSet<Integer>(Arrays.asList(77,78,79,80,81,109,110,111,112,113));

public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable>{

private Text word = new Text();

public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

Map<String, Integer> map = new HashMap<String, Integer>();

StringTokenizer itr = new StringTokenizer(value.toString());

while (itr.hasMoreTokens()) {

word.set(itr.nextToken());

// Logic to separate out words starting with M,N,O,P,Q or m,n,o,p,q

String str = word.toString();

int a = str.charAt(0);

if(filter.contains(a)){

if(map.containsKey(str)){

int count = map.get(str);

count++;

map.put(str , count);

}

else{

map.put(str , 1);

}} }

for(Entry<String, Integer> e : map.entrySet()){

context.write( new Text(e.getKey()), new IntWritable(e.getValue()));

}}}

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);

}

}

// Custom Partitioner Class. This class extends the Hadoop Partitioner class thus it overrides the getPartition Method.

public static class CustomPartitioner extends Partitioner<Text, IntWritable>{

@Override

public int getPartition(Text key, IntWritable value, int numPartitions) {

// TODO Auto-generated method stub

if(numPartitions==0){

return 0;

}

String word=key.toString();

if (word.charAt(0)== 'M' || word.charAt(0)== 'm'){

return 0;

}

else if (word.charAt(0)== 'N'||word.charAt(0)== 'n'){

return 1 % numPartitions;

}

else if (word.charAt(0)== 'O'||word.charAt(0)== 'o'){

return 2 % numPartitions;

}

else if (word.charAt(0)== 'P'||word.charAt(0)== 'p'){

return 3 % numPartitions;

}

else{

return 4 % numPartitions;

}

} }

public static void main(String[] args) throws Exception {

Configuration conf = new Configuration();

String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();

if (otherArgs.length != 2) {

System.err.println("Usage: wordcount <in> <out>");

System.exit(2);

}

Job job = new Job(conf, "word count");

job.setJarByClass(WordCount.class);

job.setNumReduceTasks(5);

job.setPartitionerClass(CustomPartitioner.class);;

job.setMapperClass(TokenizerMapper.class);

//Disabling the default combiner Job

//job.setCombinerClass(IntSumReducer.class);

job.setReducerClass(IntSumReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

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

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

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

}}

1. PerTask Tally

public class WordCount {

//Hashset used as filter the words that need to mapped.

public static Set<Integer> filter= new HashSet<Integer>(Arrays.asList(77,78,79,80,81,109,110,111,112,113));

public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable>{

private Text word = new Text();

private Map<String, Integer> map;

//Setup Task

public void setup(Context context){

map=new HashMap<String, Integer>();

}

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());

// Logic to separate out words starting with M,N,O,P,Q or m,n,o,p,q

String str = word.toString();

int a = str.charAt(0);

if(filter.contains(a)){

if(map.containsKey(str)){

int count = map.get(str);

count++;

map.put(str, count);

}

else{

map.put(str, 1);

}}}}

//Cleanup Method

public void cleanup(Context context) throws IOException, InterruptedException{

for(Entry<String, Integer> e : map.entrySet()){

context.write( new Text(e.getKey()), new IntWritable(e.getValue()));

}}}

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);

}

}

// Custom Partitioner Class. This class extends the Hadoop Partitioner class thus it overrides the getPartition Method.

public static class CustomPartitioner extends Partitioner<Text, IntWritable>{

@Override

public int getPartition(Text key, IntWritable value, int numPartitions) {

// TODO Auto-generated method stub

if(numPartitions==0){

return 0;

}

String word=key.toString();

if (word.charAt(0)== 'M' || word.charAt(0)== 'm'){

return 0;

}

else if (word.charAt(0)== 'N'||word.charAt(0)== 'n'){

return 1 % numPartitions;

}

else if (word.charAt(0)== 'O'||word.charAt(0)== 'o'){

return 2 % numPartitions;

}

else if (word.charAt(0)== 'P'||word.charAt(0)== 'p'){

return 3 % numPartitions;

}

else{

return 4 % numPartitions;

}}}

public static void main(String[] args) throws Exception {

Configuration conf = new Configuration();

String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();

if (otherArgs.length != 2) {

System.err.println("Usage: wordcount <in> <out>");

System.exit(2);

}

Job job = new Job(conf, "word count");

job.setJarByClass(WordCount.class);

job.setNumReduceTasks(5);

job.setPartitionerClass(CustomPartitioner.class);;

job.setMapperClass(TokenizerMapper.class);

//Disabling the default combiner Job

//job.setCombinerClass(IntSumReducer.class);

job.setReducerClass(IntSumReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

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

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

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

}}

So in all the above variants of the WordCount Program the map function has input parameters “**key**” and “**value**”. The key is of type **Object** and the value is of type **Text.** So when I debug the program in the debugger mode my observations are:-

* The input parameter “**key**” is of type **Object** and when I look into its value it is 0. Furthermore the Object key is not even used during the map task.
* The input parameter “**value”** is of type Hadoop Text. Hadoop map reads the text in hand Line by Line. For every map task value is the sentence which is being provided to the map as input which then is broken into tokens and each token is processed separately. When I read the Hadoop Documentation it says the Text class is used to stores text using standard UTF8 encoding. It provides methods to serialize, deserialize, and compare texts at byte level.

# **Performance Comparison**

**Running Time of all programs**

|  |  |
| --- | --- |
| Task | Run Time(mm:ss) |
|  |  |
| NoCombiner Config 1 Run 1 | 05:28 |
| NoCombiner Config 1 Run 2 | 05:32 |
| NoCombiner Config 2 Run 1 | 04:10 |
| NoCombiner Config 2 Run 2 | 04:16 |
| SiCombiner Config 1 Run 1 | 04:46 |
| SiCombiner Config 1 Run 2 | 04:52 |
| SiCombiner Config 2 Run 1 | 03:36 |
| SiCombiner Config 2 Run 2 | 03:45 |
| PerMapTally Config 1 Run 1 | 05:49 |
| PerMapTally Config 1 Run 2 | 05:58 |
| PerMapTally Config 2 Run 1 | 04:39 |
| PerMapTally Config 2 Run 2 | 04:45 |
| PerTaskTally Config 1 Run 1 | 03:56 |
| PerTaskTally Config 1 Run 2 | 03:58 |
| PerTaskTally Config 2 Run 1 | 03:07 |
| PerTaskTally Config 2 Run 2 | 03:05 |

The answer to your questions are below:

1. The primary goal of a combiner is to optimize the no. of key-value pairs that would be moved across the network from mapper to the reducer. When I observe the timings of the Tasks I definitely feel that in the SiCombiner task the combiners were called. The Reduce Input Record value also seem to be very low.
2. So there is a definitely an observable difference between the run time of programs NoCombiner and SiCombiner. The SiCombiner Program works definitely faster compared to the program with the absent combiner. This also justifies the function of a combiner i.e. optimize the data that flows between the mapper and the reducer. There is also 0 value on the Combine Input Records and Combine Output Record on s the syslog of the NoCombiner variant. The Reduce Input records value is definitely too high for the NoCombiner variant.
3. The local aggregation inside the map function in the PerMapTally does not seem to be efficient when we compare the run time of that of the NoCombiner. In fact my run time observations show that the PerMapTally task is less efficient to that of NoCombiner. It might be due to the fact that it combines the count only within a single line of text not the whole map task. Also the Combine Input Records and Combine Output Record values are zero in both the case. The Reduce Input records value is definitely similar (more or less) in both the variants.
4. There is significant run time difference between PerMapTally and PerTaskTally. It is due to the fact that PerMapTally combines only the count within one single line of the whole text. Whereas the PerTaskTally increases the combining opportunities across the entire Map Input Split, not just a single line. The Reduce Input Record value also seem to be very low for PerTaskTally.
5. PerTaskTally has a much less run time as compared to that of a SiCombiner, because we have the control of the combining activity with us in case of the PerTaskTally variant. Whereas in case of SiCombiner it’s the case of Hadoop Combiner, this can’t be controlled by the user. The MapReduce System decides that when and on which Map task the combiner would be executed.
6. When I observe the run times of both the configurations on all the variants of the WordCount Program, there seems to be a significant difference between the run times on both the configurations with 6 machines and 11 machines. The configuration with 11 machines take lesser time than that of the 6 machines. It is because more the machines (i.e. worker machines) there is more computing power available in hand. Therefore there is better performance and MapReduce Programs Scale well on larger systems. Total committed heap usage and Physical memory (bytes) snapshot seems to be less for the larger config system.