Big Data Analytics (2CS702)

Practical 6	
Rollno: 19BCE055	Name: TEJAS DOBARIYA
Division: A	Batch: E1

AIM : Analyse impact of different number of mapper and reducer on same definition as practical 4. • Prepare a conclusive report on analysis.

CODE :

```
package wordcount;

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.Reducer.Context;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import wordcount.wordcount.IntSumReducer;
import wordcount.wordcount.TokenizerMapper;

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());
```

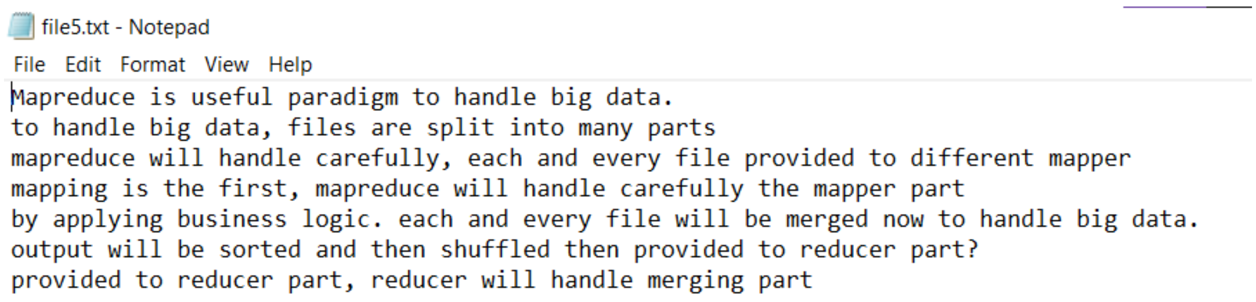
```
      context.write(word, one);
    }
  }
}

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.setNumReduceTasks(5);
 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);
}
}
```

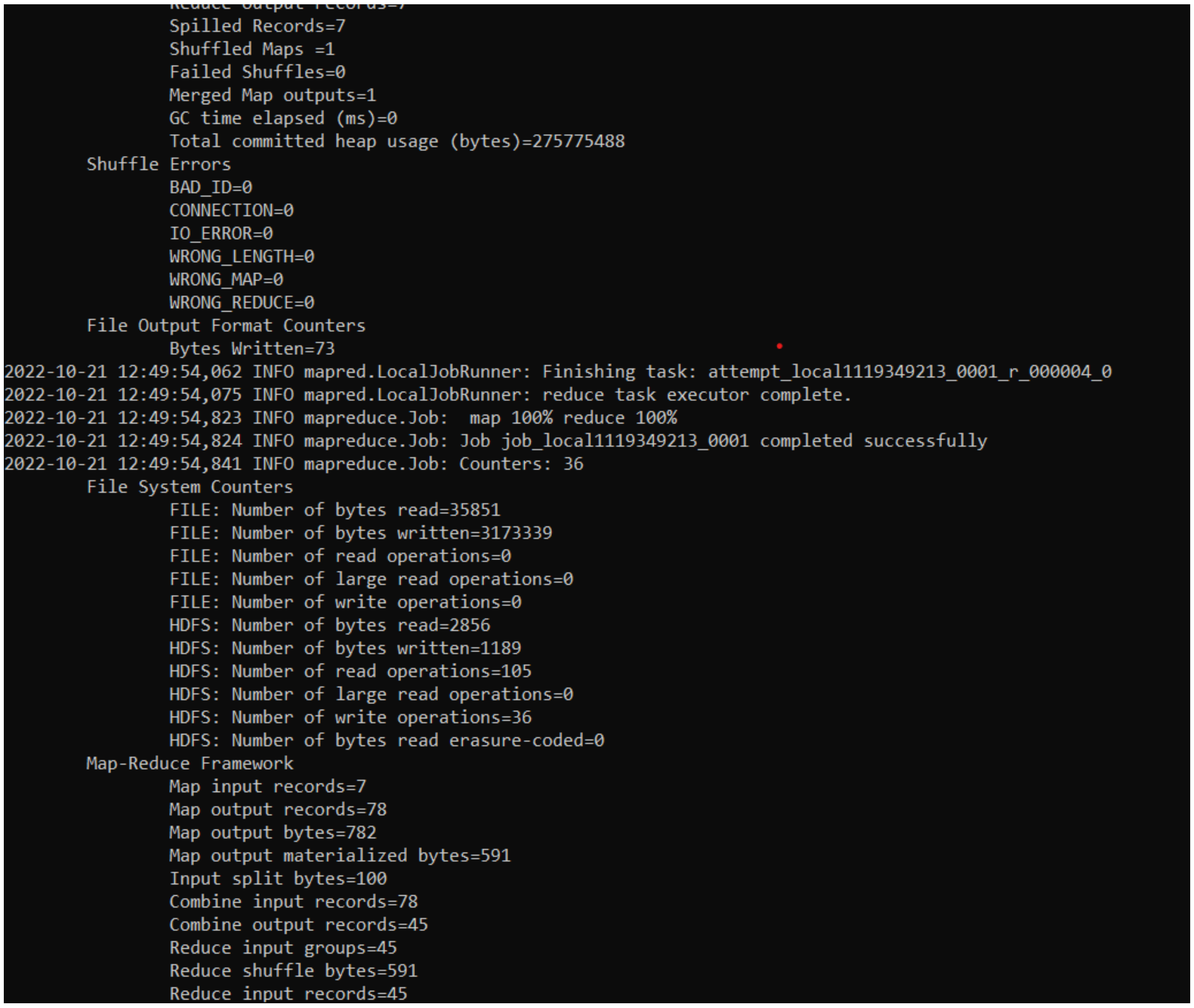
FILE6.TXT



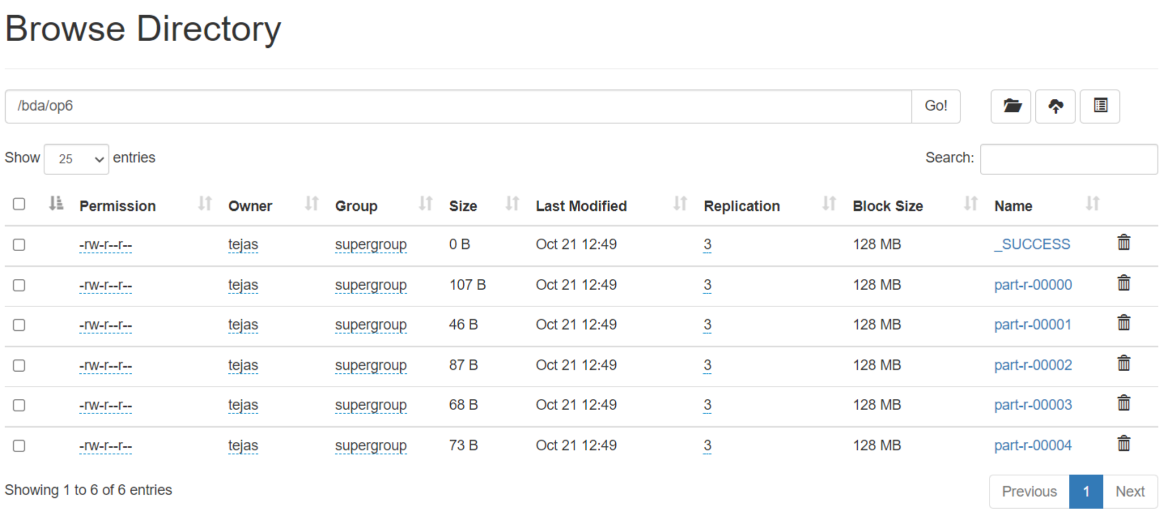
```
Mapreduce is useful paradigm to handle big data.
to handle big data, files are split into many parts
mapreduce will handle carefully, each and every file provided to different mapper
mapping is the first, mapreduce will handle carefully the mapper part
by applying business logic. each and every file will be merged now to handle big data.
output will be sorted and then shuffled then provided to reducer part?
provided to reducer part, reducer will handle merging part
```

SCREENSHOT

FOR job.setNumReduceTasks(5);



```
                Spilled Records=7
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=0
                Total committed heap usage (bytes)=275775488
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Output Format Counters
                Bytes Written=73
2022-10-21 12:49:54,062 INFO mapred.LocalJobRunner: Finishing task: attempt_local1119349213_0001_r_000004_0
2022-10-21 12:49:54,075 INFO mapred.LocalJobRunner: reduce task executor complete.
2022-10-21 12:49:54,823 INFO mapreduce.Job:  map 100% reduce 100%
2022-10-21 12:49:54,824 INFO mapreduce.Job: Job job_local1119349213_0001 completed successfully
2022-10-21 12:49:54,841 INFO mapreduce.Job: Counters: 36
        File System Counters
                FILE: Number of bytes read=35851
                FILE: Number of bytes written=3173339
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=2856
                HDFS: Number of bytes written=1189
                HDFS: Number of read operations=105
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=36
                HDFS: Number of bytes read erasure-coded=0
        Map-Reduce Framework
                Map input records=7
                Map output records=78
                Map output bytes=782
                Map output materialized bytes=591
                Input split bytes=100
                Combine input records=78
                Combine output records=45
                Reduce input groups=45
                Reduce shuffle bytes=591
                Reduce input records=45
```



Browse Directory

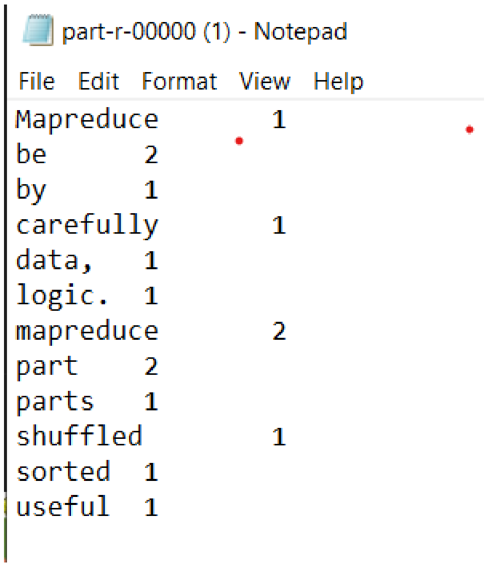
/bda/op6

Show 25 entries

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	tejas	supergroup	0 B	Oct 21 12:49	3	128 MB	_SUCCESS
-rw-r--r--	tejas	supergroup	107 B	Oct 21 12:49	3	128 MB	part-r-00000
-rw-r--r--	tejas	supergroup	46 B	Oct 21 12:49	3	128 MB	part-r-00001
-rw-r--r--	tejas	supergroup	87 B	Oct 21 12:49	3	128 MB	part-r-00002
-rw-r--r--	tejas	supergroup	68 B	Oct 21 12:49	3	128 MB	part-r-00003
-rw-r--r--	tejas	supergroup	73 B	Oct 21 12:49	3	128 MB	part-r-00004

Showing 1 to 6 of 6 entries

PART-R-00000 OUPUT DATA



```
Mapreduce	1
be	2
by	1
carefully	1
data,	1
logic.	1
mapreduce	2
part	2
parts	1
shuffled	1
sorted	1
useful	1
```

Changes to be made in code:

1. As mentioned to reduce the number of reducers to 0 we must edit the following sentence in the code:

```
job.setNumReduceTasks(0);
```

2. And to increase the number of reducers we need to edit the above sentence and instead of 0, we need to put in (n) the number we want to substitute. In my instance, I changed it to job.setNumReduceTasks(5);.

Changes observed:

1. So, it is usually preferred to keep the number of reducers such that it is a multiple of the block size, does not take a long time to execute, doesn't require a lot of file creations.

2. The number of reducers can't exceed the number of partitions, so we also need to take that into consideration when deciding the number of reducers. As for the instance, we make the number of reducers 10 but we have a single partition system then the output will be divided into 10 shards but those 10 outputs will be consolidated by a single reducer.

3. The benefits of having more reducers are that it increases load balancing, lowers the cost of failure.

4. But on the other hand, having too many reducers can also increase the framework overhead, slower start-up time, and increase the number of input for the next tasks

Conclusion:

Running the above map-reduce program we analyzed that when the number of reducers were set to 0 the output was not grouped and but when the number of reducers were set to 5 we got the output divided into 5 groups and it was all consolidated