Ex.No: 8	**Implementing a Combiner in Word Count**
Date: 15.9.25	

Aim:

To implement a Combiner in Hadoop MapReduce for Word Count, and analyze its effect on performance and intermediate data transfer.

Procedure:

1. **Set up environment**
   - Ensure Hadoop (3.x) is installed and HDFS/YARN is running in pseudo-distributed or cluster mode.
   - Java (JDK 1.8+) and Hadoop client libraries should be on your classpath.
2. **Create three Java source files**
   - WC_Mapper.java — Mapper class
   - WC_Reducer.java — Reducer and Combiner
   - WC_Driver.java — Job configuration and submission
3. **Build the JAR**
   - Use Maven, Ant, or javac + jar to package the classes into wordcount-combiner.jar.
4. **Prepare input**
   - On your local machine, create input text (e.g., input.txt).
   - Upload to HDFS:

```
hadoop fs -mkdir /wordcount/input
hadoop fs -put input.txt /wordcount/input
```

5. **Run the job**
   - Command:

```
hadoop jar wordcount-combiner.jar WC_Driver /wordcount/input /wordcount/output
```

6. **Observe output**
   - Command

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

7. **Analyze results**
   - Compare job runtime **with and without** the combiner; measure intermediate map outputs to see how the combiner reduces shuffle traffic.

Program:

WC_Mapper.java

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
public class WC_Mapper extends Mapper<LongWritable, Text, Text, IntWritable> {
   private final static IntWritable one = new IntWritable(1);
   private Text word = new Text();
   @Override
   protected void map(LongWritable 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);
      }
   }
}
```

WC_Reducer.java

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class WC_Reducer extends Reducer<Text, IntWritable, Text, IntWritable> {
   private IntWritable result = new IntWritable();



   @Override
```

```
    protected 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);
    }
}
```

WC_Driver.java

```
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.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
public class WC_Driver {
    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println("Usage: WC_Driver <input> <output>");
            System.exit(2);
        }
        Job job = Job.getInstance(new Configuration(), "Word Count with Combiner");
        job.setJarByClass(WC_Driver.class);
        job.setMapperClass(WC_Mapper.class);
        job.setCombinerClass(WC_Reducer.class);  // Combiner enabled
        job.setReducerClass(WC_Reducer.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);
    }
}
```

Output:

input.txt : hello world hello Hadoop Combiner

Final reducer output:

```
Combiner 1
Hadoop    1
hello     3
world     1
```

Result:

Thus, the program has been executed successfully and the output is also verified.