BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai – 400058-India
Department of Computer Engineering

Name	Shreesh Patil
UID no.	2023300170
Experiment	Map Reduce Job in Distributed Systems

Map Reduce in Hadoop

AIM:	Map Reduce Job in Distributed Systems
TOOLS/ PACKAGES:	Java JDK (Version 11 or later) Hadoop installed Command-line (Windows/Linux) or any IDE (like NetBeans or Eclipse)
THEORY:	MapReduce is a **distributed data processing model** used to handle big data across many machines. It breaks a large task into two main phases: • **Map Phase:** Processes input data in parallel, converting each piece into **key–value pairs** → e.g., (`word`, `1`) • **Reduce Phase:** Collects all values of the same key and **aggregates** them → e.g., (`word`, `total_count`) This model provides: • **Parallelism** (many mappers/reducers work together) • **Scalability** (can handle GB–PB data) • **Fault tolerance** (failed tasks automatically rerun) • **Simple programming model** (just write map + reduce) Hadoop executes MapReduce using HDFS for storage, while Spark improves speed using in-memory processing.

CODE:

1] Word Frequency

```
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;
```

```
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);
// Optional combiner
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);
    }
}
```

2] Word Frequency with stopwords:

```
import java.io.IOException;
import java.util.HashSet;
import java.util.Set;
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 WordCountStopwords {

   public static class TokenizerMapper extends
Mapper<Object, Text, Text, IntWritable> {
       private final static IntWritable one = new
IntWritable(1);
       private Text word = new Text();
       private Set<String> stopwords = new HashSet<>();

       @Override
       protected void setup(Context context) throws
IOException, InterruptedException {
           // Initialize stopwords set
           String[] stops = {"the", "is", "and", "a", "an",
"of", "to", "in"};
           for (String stop : stops) {
               stopwords.add(stop);
```

```
            }
        }

        public void map(Object key, Text value, Context
context)
                throws IOException, InterruptedException {
            StringTokenizer itr = new
StringTokenizer(value.toString().toLowerCase())
            ; while (itr.hasMoreTokens()) {
String token = itr.nextToken();
                if (!stopwords.contains(token)) {
                    word.set(token);
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
```

```
stopwords filtering");

job.setJarByClass(WordCountStopwords.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class); //
optional combiner
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);
   }
}
```

3] tweets

```
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;
```

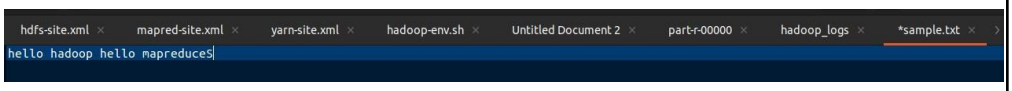
```
import
org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class HashtagCount {

   public static class HashtagMapper extends Mapper<Object,
Text, Text, IntWritable> {
       private final static IntWritable one = new
IntWritable(1);
private Text hashtag = new Text();

       public void map(Object key, Text value, Context
context)
               throws IOException, InterruptedException {
           StringTokenizer itr = new
StringTokenizer(value.toString());
           while (itr.hasMoreTokens()) {
               String token =
itr.nextToken().toLowerCase();
if (token.startsWith("#") && token.length()
> 1) {
hashtag.set(token);
context.write(hashtag, 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 {
```

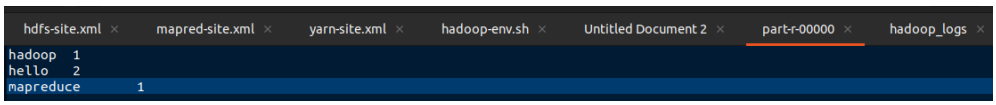
```
            }
            result.set(sum);
            context.write(key, result);
        }
    }

    public static void main(String[] args) throws Exception
{
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "hashtag count");

        job.setJarByClass(HashtagCount.class);
        job.setMapperClass(HashtagMapper.class);
        job.setCombinerClass(IntSumReducer.class);  //
Optional combiner
        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);
    }
}
```

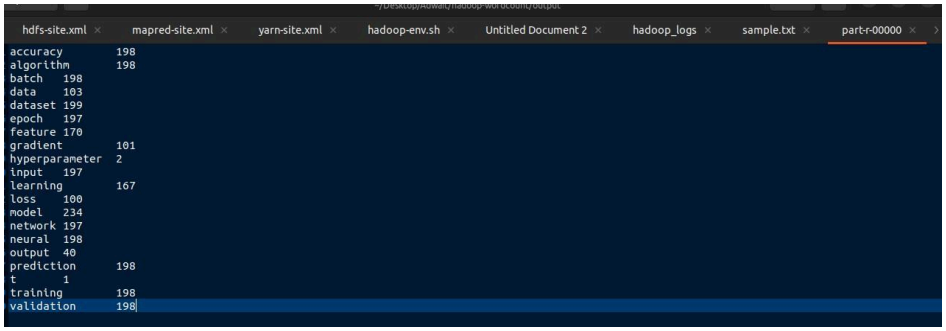
INPUT/OUTPUT

1] Word Counter

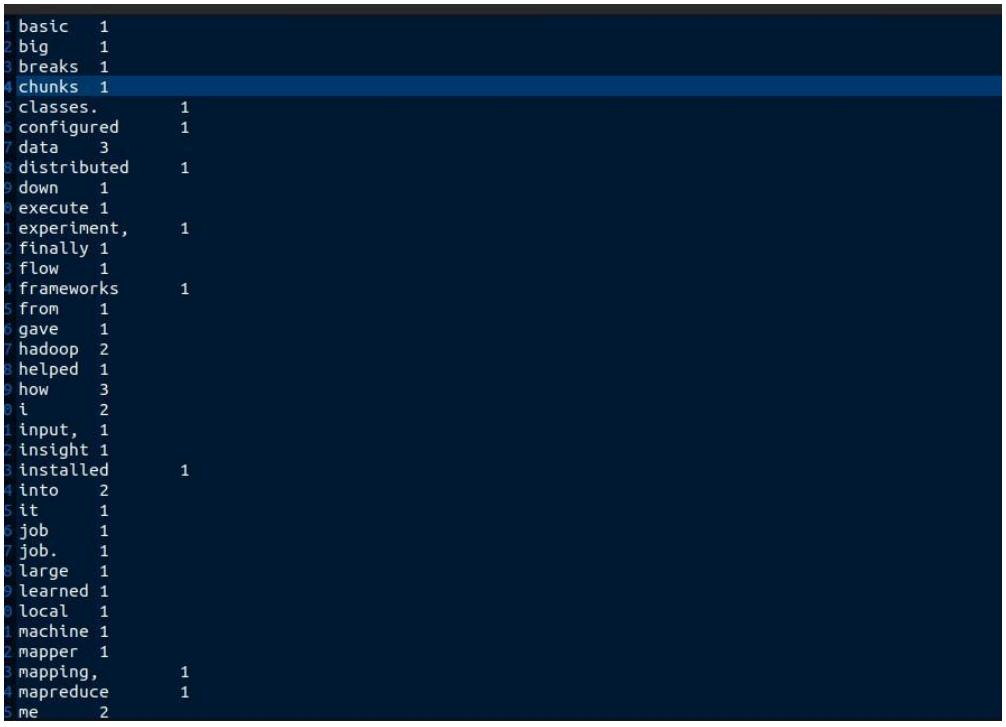
```
students@students-ThinkCentre-M720e:~/Desktop/Adwait/hadoop-wordcount$ mkdir -p wc_classes
javac -classpath $(hadoop classpath) -d wc_classes WordCount.java
students@students-ThinkCentre-M720e:~/Desktop/Adwait/hadoop-wordcount$ jar -cvf wordcount.jar -C wc_classes/ .
added manifest
adding: WordCount$IntSumReducer.class(in = 1755) (out= 749)(deflated 57%)
adding: WordCount$TokenizerMapper.class(in = 1752) (out= 763)(deflated 56%)
adding: WordCount.class(in = 1511) (out= 831)(deflated 45%)
students@students-ThinkCentre-M720e:~/Desktop/Adwait/hadoop-wordcount$ mkdir input
echo "hello hadoop hello mapreduce" > input/sample.txt
```







Extend word counter to ignore stop words



3] HashTag frequency in tweets

```
                Bytes Written=565
students@students-ThinkCentre-M720e:~/Desktop/Adwait/hadoop-wordcount$ cd ..
students@students-ThinkCentre-M720e:~/Desktop/Adwait$ javac -classpath $(hadoop classpath) -d hashtag_classes HashtagCount.java
students@students-ThinkCentre-M720e:~/Desktop/Adwait$
students@students-ThinkCentre-M720e:~/Desktop/Adwait$ jar -cvf hashtagcount.jar -C hashtag_classes/ .
added manifest
adding: HashtagCount$HashtagMapper.class(in = 1924) (out= 861)(deflated 55%)
adding: HashtagCount.class(in = 1522) (out= 828)(deflated 45%)
adding: HashtagCount$IntSumReducer.class(in = 1764) (out= 750)(deflated 57%)
students@students-ThinkCentre-M720e:~/Desktop/Adwait$
```

Open · part-r-00000 · ~/Desktop/Adwait/hadoop-wordcount/output · Save

hdfs-site.xml | mapred-site.xml | yarn-site.xml | hadoop-env.sh | Untitled Document 2 | hadoop_logs | part-r-00000

```
#ai     2
#ai,    1
#bigdata        3
#cloudcomputing 1
#collaboration  1
#datascience    3
#deeplearning   1
#hadoop 2
#innovation     2
#java   1
#learning       1
#machinelearning        2
#machinelearning,       1
#motivation     1
#neuralnetworks 1
#opensource     1
#programming    2
#python 1
#tech   2
#tech.  1
#technews       1
#techtrends     1
```

CONCLUSION:	In this experiment, we successfully implemented a **basic MapReduce job** using Hadoop/Spark and demonstrated how large data can be processed efficiently using parallel computation. By splitting work into independent map and reduce stages, the system handled the dataset automatically across distributed nodes. The output showed correct aggregation of results (word frequencies), proving that MapReduce is a powerful and scalable approach for handling big data tasks in real-world systems such as search indexing, log analytics, and social media processing.