PRACTICAL 11

Roll No. - 31404 K4 DSBDAL

PROBLEM STATEMENT -
Write a code in JAVA for a simple Word Count application that counts the number of occurrences of each word in a given input set using the Hadoop Map-Reduce framework on local-standalone set-up.

1. This command creates a directory `/user/te` in HDFS. If `/user` doesn't already exist, it creates that too.

te@sel-a1-216-18:~/Desktop/31404/practicals/WordCountProject$ hadoop fs -mkdir -p /user/te

2. Creating a folder called `input` inside `/user/te` on HDFS. This is usually where you'll upload your data files (like `.txt`) to be processed by your Hadoop jobs (like WordCount).

te@sel-a1-216-18:~/Desktop/31404/practicals/WordCountProject$ hdfs dfs -mkdir -p /user/te/input

3. Copying all files from the local `input` folder into the HDFS folder `/user/te/input/`.
So after this, your input files are ready on HDFS to be used by a MapReduce job like WordCount.

te@sel-a1-216-18:~/Desktop/31404/practicals/WordCountProject$ hdfs dfs -put input/* /user/te/input/

4. List the contents of the `/user/te/input/` directory on HDFS.
this confirms your file `input.txt` is successfully uploaded and ready for processing in your WordCount job.

te@sel-a1-216-18:~/Desktop/31404/practicals/WordCountProject$ hdfs dfs -ls /user/te/input/
Found 1 items
-rw-r--r-- 3 te supergroup 253 2025-04-08 09:49 /user/te/input/input.txt

5. Create a new directory named `build` in your current working directory, which is:
`bash`
`~/Desktop/31404/practicals/WordCountProject`

The build directory is typically used to store compiled .class files or the final JAR file that you'll run with Hadoop.

te@sel-a1-216-18:~/Desktop/31404/practicals/WordCountProject$ mkdir build

6. The `WordCount.class` and potentially other related `.class` files will be created in the `build` folder.

te@sel-a1-216-18:~/Desktop/31404/practicals/WordCountProject$ javac -classpath $(hadoop classpath) -d build WordCount.java

7. A `wordcount.jar` file is created in your current directory, containing all compiled `.class` files from the `build` folder. This JAR can now be submitted to Hadoop for execution.

te@sel-a1-216-18:~/Desktop/31404/practicals/WordCountProject$ jar -cvf wordcount.jar -C build/ .

added manifest
adding: WordCount$TokenizerMapper.class(in = 1948) (out= 866)(deflated 55%)
adding: WordCount$IntSumReducer.class(in = 1751) (out= 747)(deflated 57%)
adding: WordCount.class(in = 1532) (out= 845)(deflated 44%)

8. Hadoop runs your `WordCount` job by reading data from `/user/te/input`, processing it with your map and reduce logic, and writing the final word counts to `/user/te/output`.

te@sel-a1-216-18:~/Desktop/31404/practicals/WordCountProject$ hadoop jar wordcount.jar WordCount /user/te/input /user/te/output

9. Confirms that your Word Count program worked properly and gives output.

te@sel-a1-216-18:~/Desktop/31404/practicals/WordCountProject$ hdfs dfs -cat /user/te/output/part-r-00000
a 1
allows 1
an 1
and 1
applications 1
big 1
clusters 1
component 1
computers 1
core 1
data 2
distributed 1
for 1
framework 2
hadoop4
in 1
is 3
large 1
mapreduce 1
of 3
on 1
opensource 1
powerful 1
processing 1
runs 1
sets 1
that 1
the 2

used 1
widely 1

10. The contents of `/user/te/output` in HDFS will now be available on your machine inside a folder called `output_local`.

te@sel-a1-216-18:~/Desktop/31404/practicals/WordCountProject$tte@ste@sel-tete@te@tette

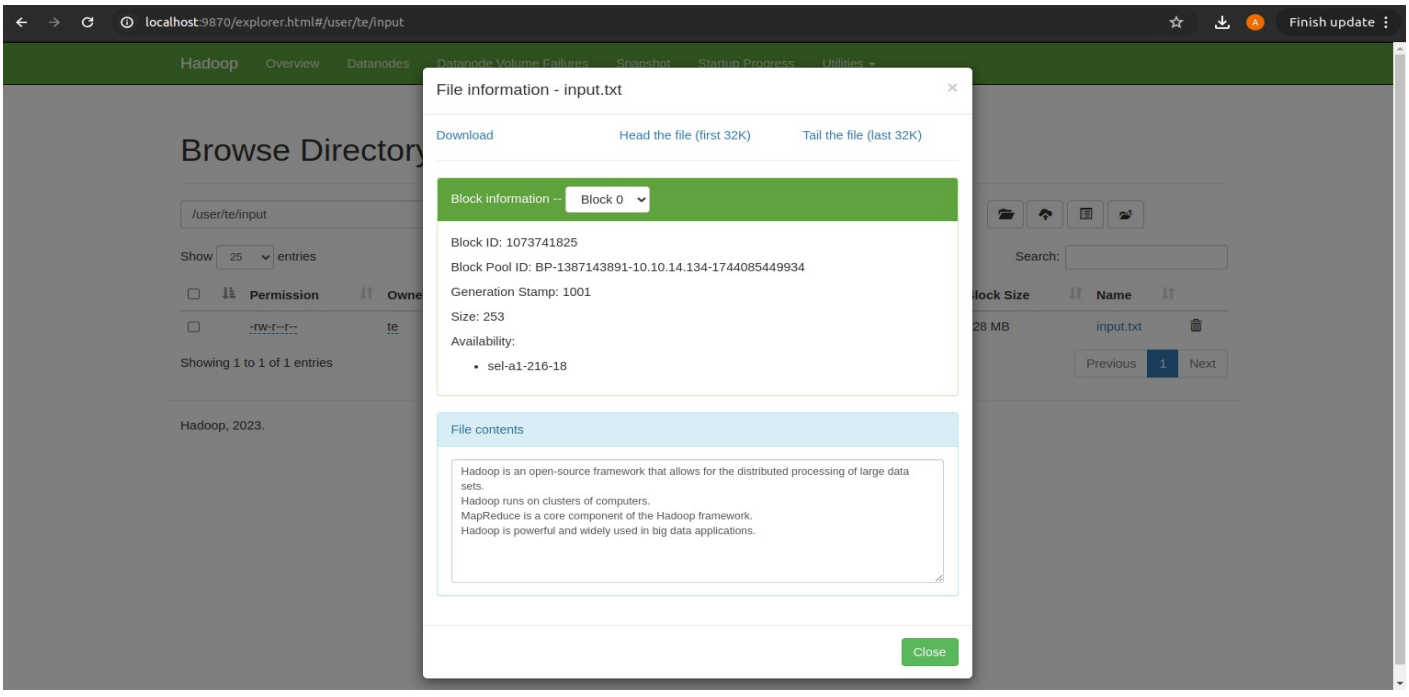
te@sel-a1-216-18:~/Desktop/31404/practicals/WordCountProject$ hadoop fs -get /user/te/output ./output_local

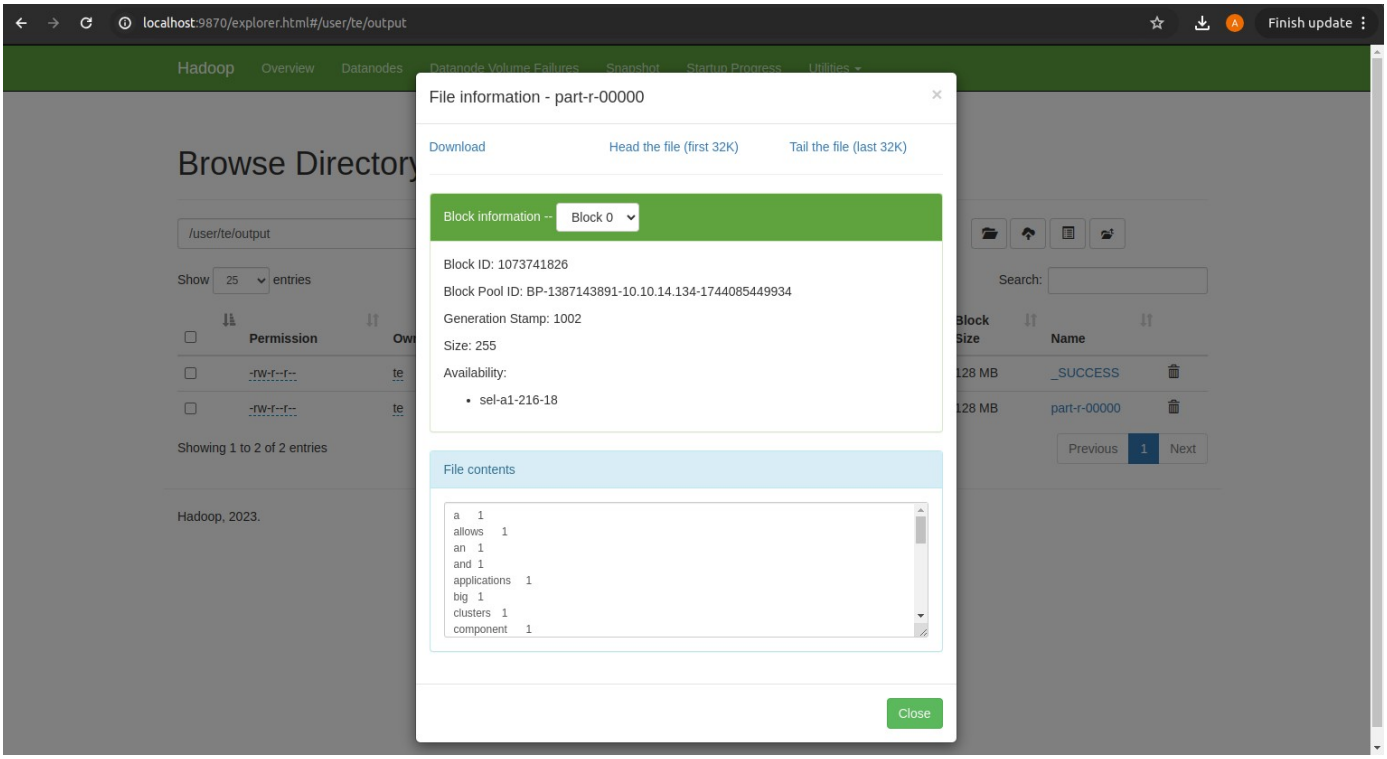
WordCount.java -

```
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 {

   // Mapper Class
   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().replaceAll("[^a-zA-Z]", "").toLowerCase()); // clean up
            if (!word.toString().isEmpty()) {
               context.write(word, one);
            }
         }
      }
   }

   // Reducer Class
   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);
    }
  }

  // Driver Code
  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 optimization
    job.setReducerClass(IntSumReducer.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    FileInputFormat.addInputPath(job, new Path("input"));
    FileOutputFormat.setOutputPath(job, new Path("output"));

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

Outputs -




localhost:9870/explorer.html#/user/te/output
Finish update
Hadoop
Overview
Datanodes
Datanode Volume Failures
Snapshot
Startup Progress
Utilities
Browse Directory
/user/te/output
Show 25 entries
Search:
Permission
Block Size
Name
-rw-r--r--
128 MB
_SUCCESS
-rw-r--r--
128 MB
part-r-00000
Showing 1 to 2 of 2 entries
Previous
1
Next
Hadoop, 2023.
File information - part-r-00000
Download
Head the file (first 32K)
Tail the file (last 32K)
Block information --
Block 0
Block ID: 1073741826
Block Pool ID: BP-1387143891-10.10.14.134-1744085449934
Generation Stamp: 1002
Size: 255
Availability:
sel-a1-216-18
File contents
a 1
allows 1
an 1
and 1
applications 1
big 1
clusters 1
component 1
Close