

## Practical - 2

**Aim :** To implement a basic Map Reduce program.

A) Word Count

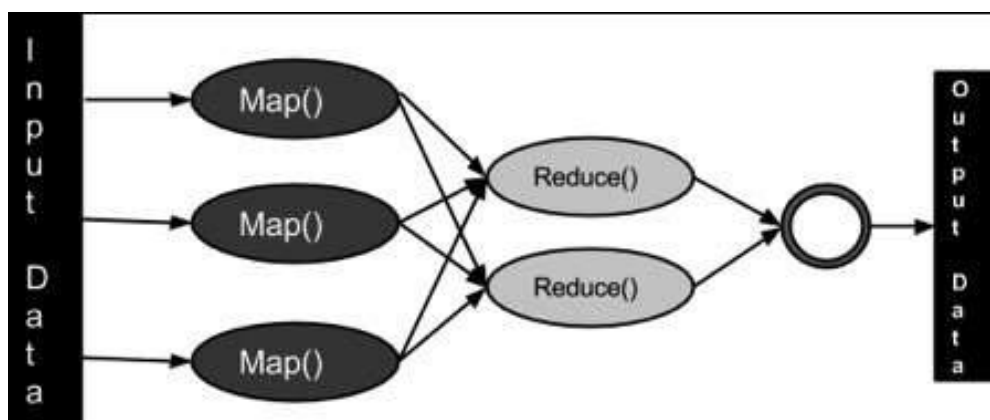
**Theory :**

### MapReduce :

- MapReduce is a software framework and programming model used for processing huge amounts of data.
- MapReduce programs work in two phases, namely, Map and Reduce. Map tasks deal with splitting and mapping of data while Reduce tasks shuffle and reduce the data.
- With MapReduce, rather than sending data to where the application or logic resides, the logic is executed on the server where the data already resides, to expedite processing.
- Data access and storage is disk-based, the input is usually stored as files containing structured, semi-structured, or unstructured data, and the output is also stored in files.

### MapReduce Algorithm :

- Generally MapReduce paradigm is based on sending the computer to where the data resides!
- MapReduce program executes in three stages, namely map stage, shuffle stage, and reduce stage.
  - ✧ **Map stage** – The map or mapper's job is to process the input data. Generally the input data is in the form of file or directory and is stored in the Hadoop file system (HDFS). The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.
  - ✧ **Reduce stage** – This stage is the combination of the Shuffle stage and the Reduce stage. The Reducer's job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.
- During a MapReduce job, Hadoop sends the Map and Reduce tasks to the appropriate servers in the cluster.
- The framework manages all the details of data-passing such as issuing tasks, verifying task completion, and copying data around the cluster between the nodes.
- Most of the computing takes place on nodes with data on local disks that reduces the network traffic.
- After completion of the given tasks, the cluster collects and reduces the data to form an appropriate result, and sends it back to the Hadoop server.



### **What is mapper, reducer and driver class? :**

#### **1. Mapper Class :**

- The first stage in Data Processing using MapReduce is the Mapper Class. Here, RecordReader processes each Input record and generates the respective key-value pair.
- Hadoop's Mapper store saves this intermediate data into the local disk.
- Input Split: It is the logical representation of data. It represents a block of work that contains a single map task in the MapReduce Program.
- RecordReader: It interacts with the Input split and converts the obtained data in the form of Key-Value Pairs.

#### **2. Reducer Class :**

- The Intermediate output generated from the mapper is fed to the reducer which processes it and generates the final output which is then saved in the HDFS.

#### **3. Driver Class :**

- The major component in a MapReduce job is a Driver Class.
- It is responsible for setting up a MapReduce Job to run-in Hadoop.
- We specify the names of Mapper and Reducer Classes long with data types and their respective job names.

**A) Word Count****Code :****WordCountDriver**

```

package hdfs;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

public class WordCountDriver {

    public static void main(String[] args) throws Exception {
        Job j1 = Job.getInstance(new Configuration());
        j1.setJarByClass(WordCountDriver.class);
        j1.setJobName("Average Word Count");

        FileInputFormat.addInputPath(j1, new Path(args[0]));
        FileOutputFormat.setOutputPath(j1, new Path(args[1]));

        j1.setMapperClass(WordCountMapper.class);
        j1.setReducerClass(WordCountReducer.class);
        j1.setOutputKeyClass(Text.class);
        j1.setOutputValueClass(IntWritable.class);

        System.exit(j1.waitForCompletion(true) ? 0 : 1);
    }
}

```

**WordCountMapper**

```

package hdfs;

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 WordCountMapper 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, Mapper<LongWritable, Text, Text,
IntWritable>.Context context)
        throws IOException, InterruptedException {

        String line = value.toString();
        StringTokenizer itr = new StringTokenizer(line);
        while (itr.hasMoreTokens()) {
            word.set(itr.nextToken());
            context.write(word, one);
        }
    }
}

```

**WordCountReducer**

```

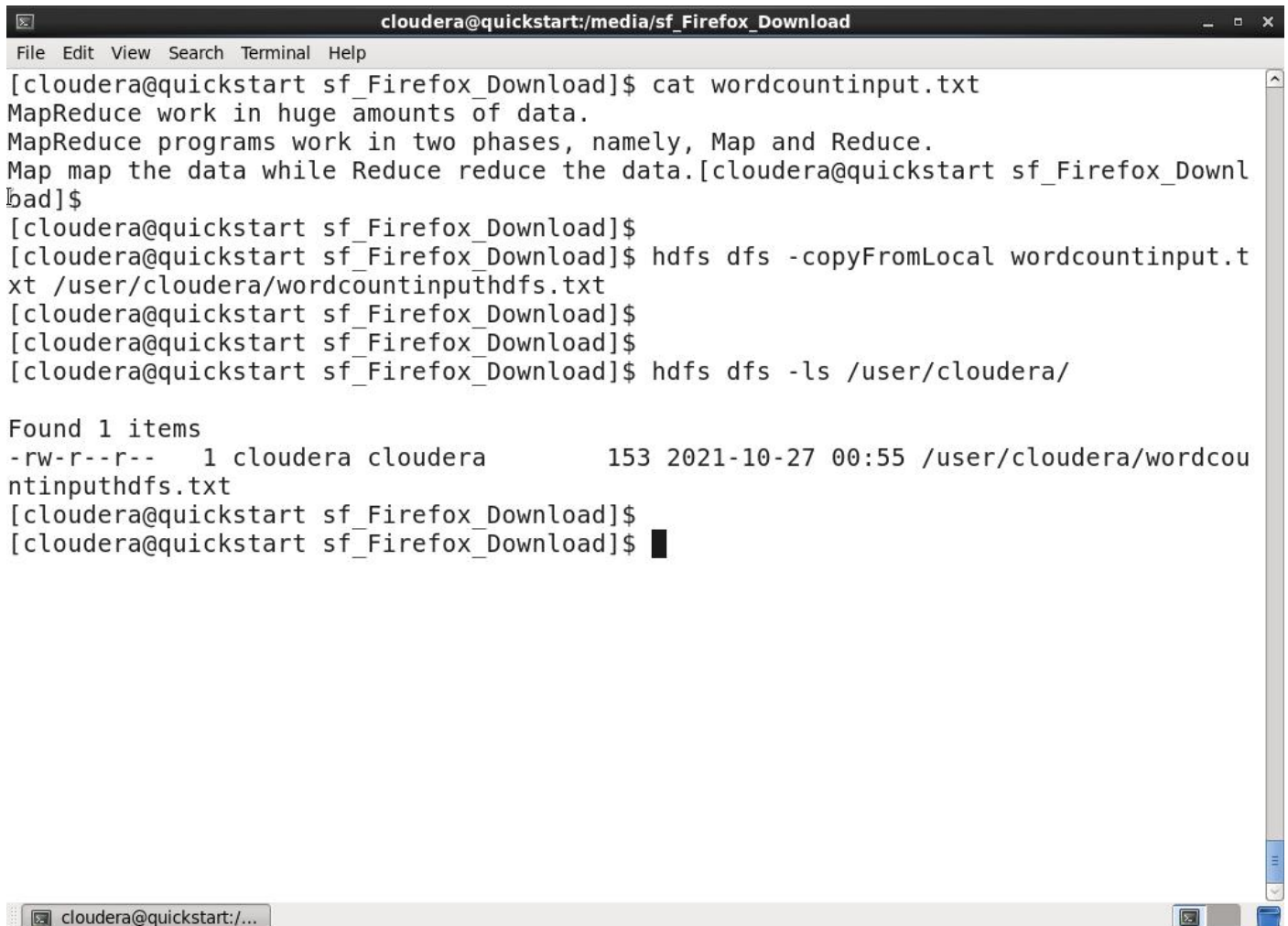
package hdfs;

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

public class WordCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    @Override
    protected void reduce(Text key, Iterable<IntWritable> values,
        Reducer<Text, IntWritable, Text, IntWritable>.Context context) throws IOException,
        InterruptedException {

        int sum = 0;
        for (IntWritable value : values) {
            sum += value.get();
        }
        context.write(key, new IntWritable(sum));
    }
}

```

**HDFS Commands :**


The screenshot shows a terminal window titled "cloudera@quickstart:/media/sf\_Firefox\_Download". The terminal displays the following commands and output:

```

[cloudera@quickstart sf_Firefox_Download]$ cat wordcountinput.txt
MapReduce work in huge amounts of data.
MapReduce programs work in two phases, namely, Map and Reduce.
Map map the data while Reduce reduce the data.[cloudera@quickstart sf_Firefox_Download]$
[cloudera@quickstart sf_Firefox_Download]$
[cloudera@quickstart sf_Firefox_Download]$ hdfs dfs -copyFromLocal wordcountinput.txt /user/cloudera/wordcountinputhdfs.txt
[cloudera@quickstart sf_Firefox_Download]$
[cloudera@quickstart sf_Firefox_Download]$
[cloudera@quickstart sf_Firefox_Download]$ hdfs dfs -ls /user/cloudera/

Found 1 items
-rw-r--r--  1 cloudera cloudera      153 2021-10-27 00:55 /user/cloudera/wordcountinputhdfs.txt
[cloudera@quickstart sf_Firefox_Download]$
[cloudera@quickstart sf_Firefox_Download]$

```

**Output :**

```

cloudera@quickstart:/media/sf_Firefox_Download
File Edit View Search Terminal Help
[cloudera@quickstart sf_Firefox_Download]$ hadoop jar word_count_jdk_v1.7.jar WordC
ountDriver /user/cloudera/wordcountinputhdfs.txt /user/cloudera/wordcountoutputhdfs
/
21/10/27 00:57:23 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:80
32
21/10/27 00:57:25 WARN mapreduce.JobResourceUploader: Hadoop command-line option pa
rsing not performed. Implement the Tool interface and execute your application with
ToolRunner to remedy this.
21/10/27 00:57:25 INFO input.FileInputFormat: Total input paths to process : 1
21/10/27 00:57:26 INFO mapreduce.JobSubmitter: number of splits:1
21/10/27 00:57:26 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_16353
13735377_0002
21/10/27 00:57:26 INFO impl.YarnClientImpl: Submitted application application_16353
13735377_0002
21/10/27 00:57:27 INFO mapreduce.Job: The url to track the job: http://quickstart.c
loudera:8088/proxy/application_1635313735377_0002/
21/10/27 00:57:27 INFO mapreduce.Job: Running job: job_1635313735377_0002
21/10/27 00:57:45 INFO mapreduce.Job: Job job_1635313735377_0002 running in uber mo
de : false
21/10/27 00:57:45 INFO mapreduce.Job: map 0% reduce 0%
21/10/27 00:59:09 INFO mapreduce.Job: map 100% reduce 0%

Message from syslogd@quickstart at Oct 27 00:59:08 ...
kernel:BUG: soft lockup - CPU#1 stuck for 72s! [java:15113]

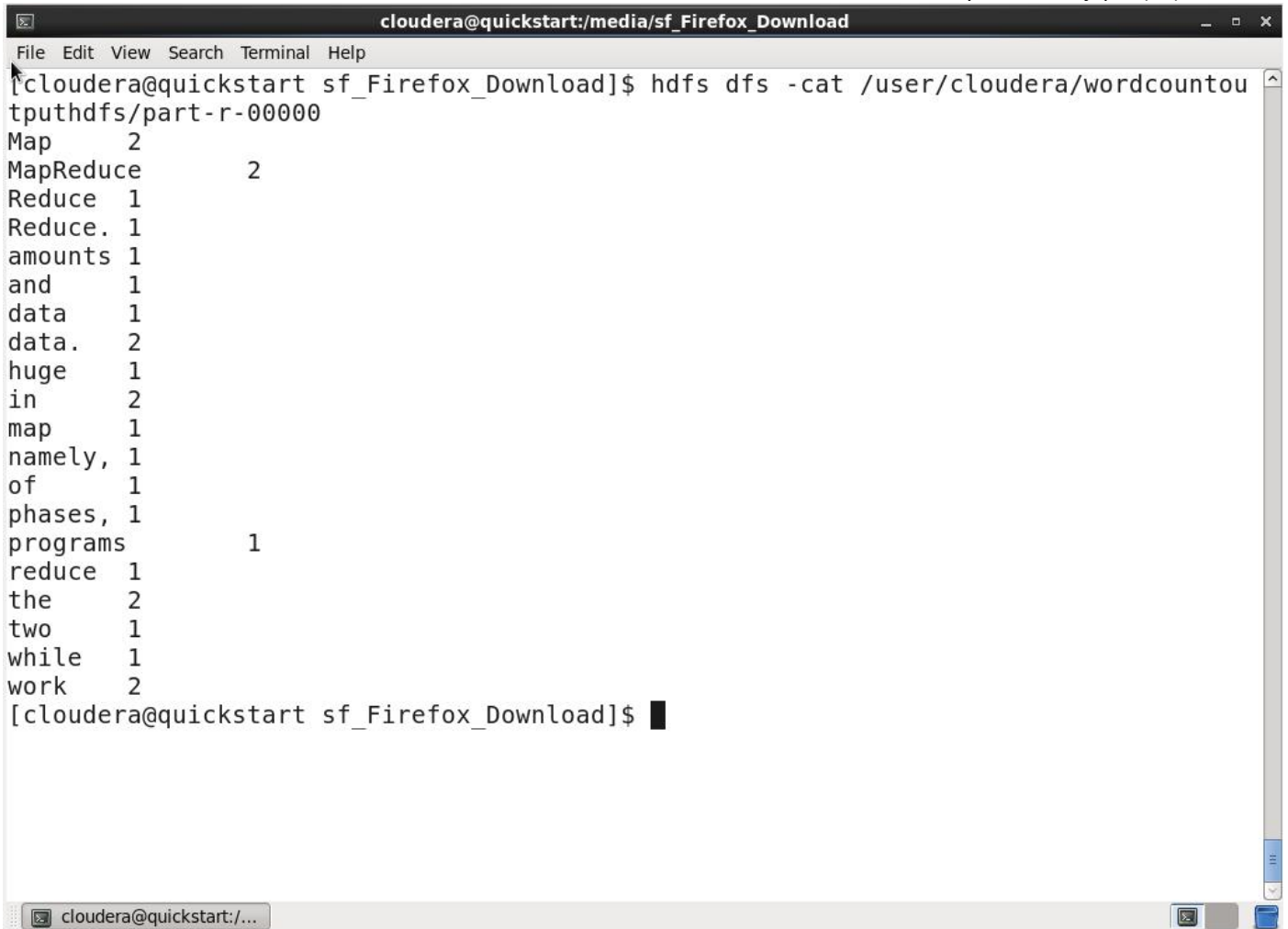
21/10/27 00:59:20 INFO mapreduce.Job: map 100% reduce 100%
21/10/27 00:59:20 INFO mapreduce.Job: Job job_1635313735377_0002 completed successf
ully
cloudera@quickstart:/...

```

```

cloudera@quickstart:/media/sf_Firefox_Download
File Edit View Search Terminal Help
Combine input records=0
Combine output records=0
Reduce input groups=20
Reduce shuffle bytes=312
Reduce input records=26
Reduce output records=20
Spilled Records=52
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=121
CPU time spent (ms)=1990
Physical memory (bytes) snapshot=477794304
Virtual memory (bytes) snapshot=3116580864
Total committed heap usage (bytes)=390594560
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
Bytes Read=153
File Output Format Counters
Bytes Written=158
[cloudera@quickstart sf_Firefox_Download]$
[cloudera@quickstart sf_Firefox_Download]$
cloudera@quickstart:/...

```



```
cloudera@quickstart:/media/sf_Firefox_Download
File Edit View Search Terminal Help
[cloudera@quickstart sf_Firefox_Download]$ hdfs dfs -cat /user/cloudera/wordcountou
tputhdfs/part-r-00000
Map      2
MapReduce      2
Reduce    1
Reduce.    1
amounts    1
and        1
data       1
data.      2
huge       1
in         2
map        1
namely,    1
of         1
phases,    1
programs   1
reduce     1
the        2
two        1
while      1
work       2
[cloudera@quickstart sf_Firefox_Download]$
```

**Conclusion :** Successfully Performed MapReduce Programs in Cloudera VM using Java and Hadoop.