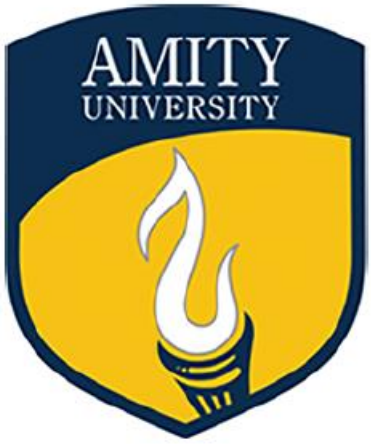
AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY

FUNDAMENTAL OF BIG DATA ANALYSIS LAB FILE

CSE 443

BACHELOR OF TECHNOLOGY

COMPUTER SCIENCE AND ENGINEERING



Submitted by:

Ashesh Prakash- 7CSE4X - A2305218219

Submitted to: Dr. Juhi Singh

AMITY UNIVERSITY UTTAR PRADESH

Table of Contents

PRACTICAL – 1

DATE: 19.07.2021
AIM

Write a brief introduction of Hadoop and show its step-by-step installation process.

THEORY

Hadoop:

The Apache™ Hadoop® project develops open-source software for reliable, scalable, distributed computing. The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly available service on top of a cluster of computers, each of which may be prone to failures.

Modules of Hadoop:

1. **HDFS:** The Hadoop Distributed File System (HDFS) is based on the Google File System (GFS) and provides a distributed file system that is designed to run on commodity hardware. It is highly fault-tolerant and is designed to be deployed on low-cost hardware.
2. **Yarn:** is a framework for job scheduling and cluster resource management.
3. **Map Reduce:** MapReduce is a parallel programming model for writing distributed applications devised at Google for efficient processing of large amounts of data (multi-terabyte data-sets), on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner.
4. **Hadoop Common:** These are Java libraries and utilities required by other Hadoop modules.

INSTALLATION PROCEDURE

Hadoop installation steps:

1. Download JAVA 8 from Oracle website and install.
2. Download Hadoop.tar file from - https://hadoop.apache.org/releases.html.
3. Extract Hadoop.tar file

4. Now open and edit 4 files core-site.xml, hdfs-site.xml, mapred-site.xml and yarn-site.xml that are present inside D:\hadoop\etc\hadoop. Add the following configuration settings:

core-site.xml

```
<configuration>
 <property>
 <name>fs.default.name</name>
 <value>hdfs://localhost:9000</value>
 </property>
</configuration>
```

hdfs-site.xml

```
<configuration>
 <property>
<name>dfs.replication</name>
 <value>1</value>
 </property>
 <property>
 <name>dfs.namenode.name.dir</name>
 <value>D:\hadoop\data</value>
 </property>
 <property>
 <name>dfs.datanode.data.dir</name>
 <value>D:\hadoop\data\namenode</value>
 </property>
</configuration>
```

mapred-site.xml

```
<configuration>
 <property>
```

<name>mapreduce.framework.name</name>
<value>yarn</value>
</property>
</configuration>

yarn-site.xml

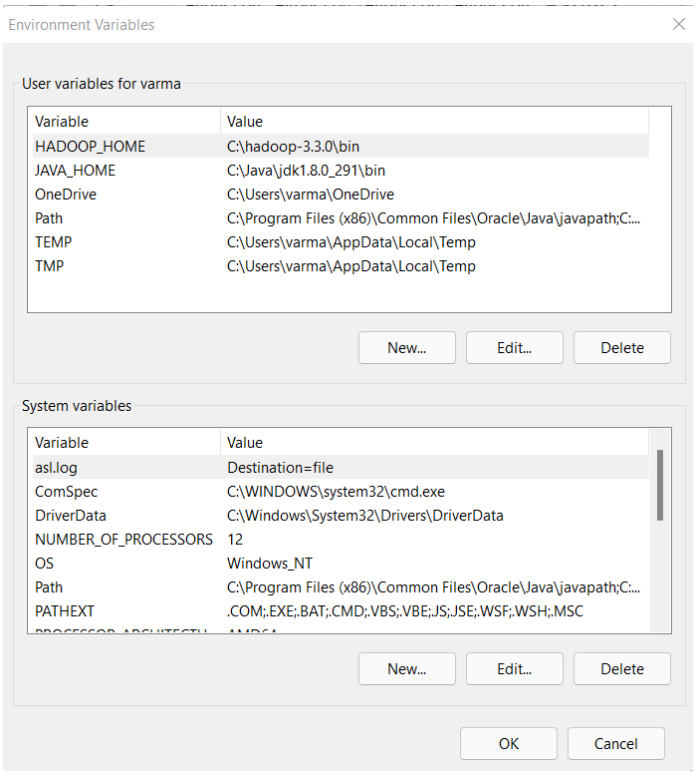
<configuration>
<property>
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle</value>
</property>
<property>
<name>yarn.nodemanager.auxservices.mapreduce.shuffle.class</name>
<value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
</configuration>

5. Open Hadoop-env.cmd using notepad and set the JAVA HOME path.

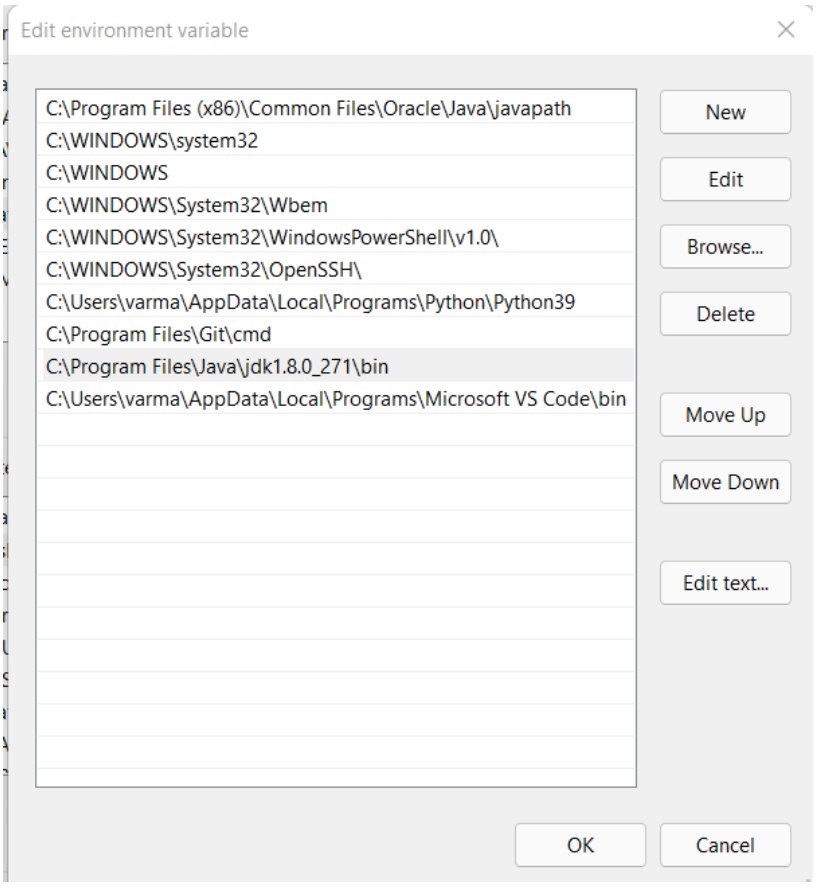
```
@rem The java implementation to use.  Required.
set JAVA_HOME=C:\Progra~1\Java\jdk1.8.0_261
```

6. Create a data folder inside the root Hadoop folder and open it. Inside the data folder create two folders named – datanode and namenode.

7. Add HADOOP_HOME and JAVA_HOME as system variables

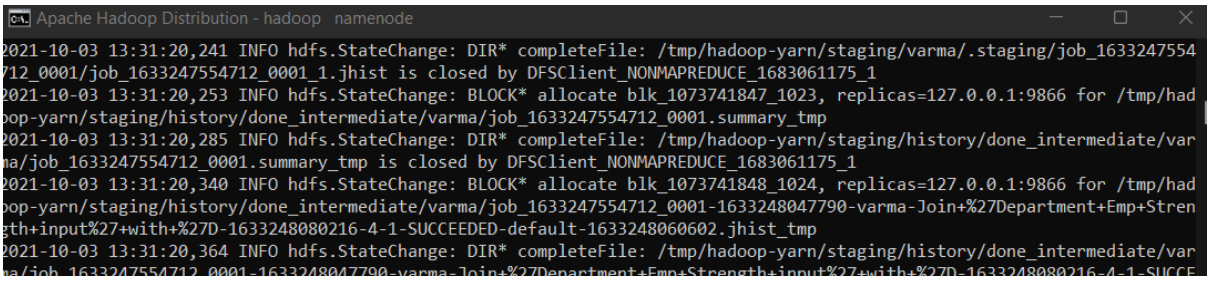


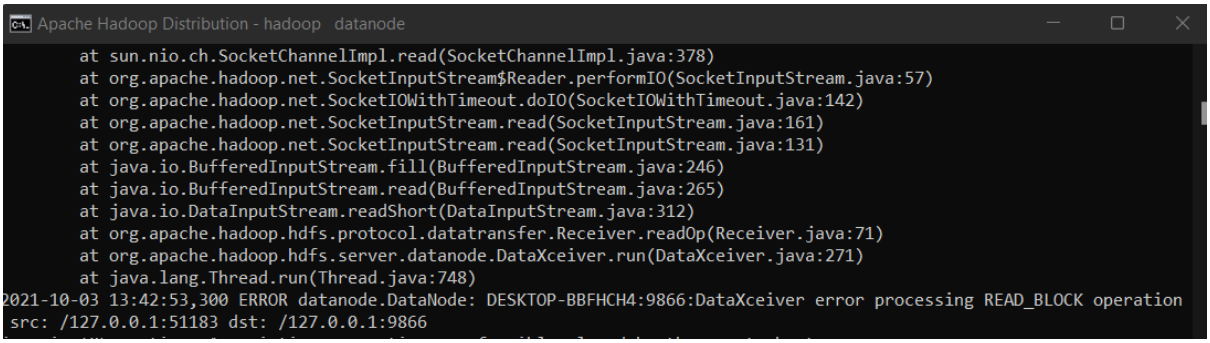
8. Add environment variables and add path of Hadoop\bin , Hadoop\sbin and Java\jdk.

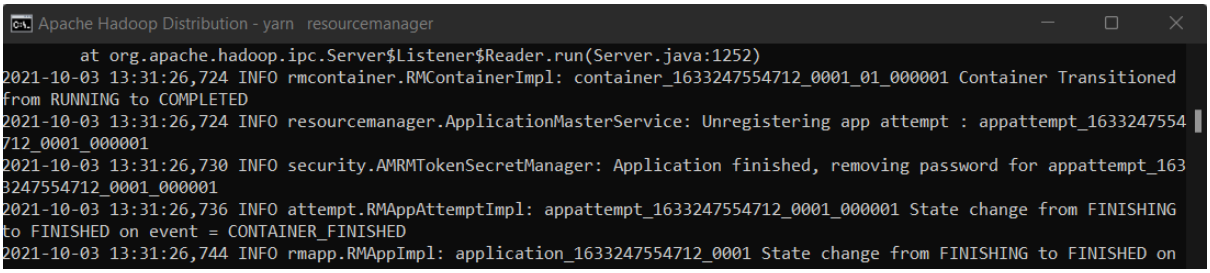


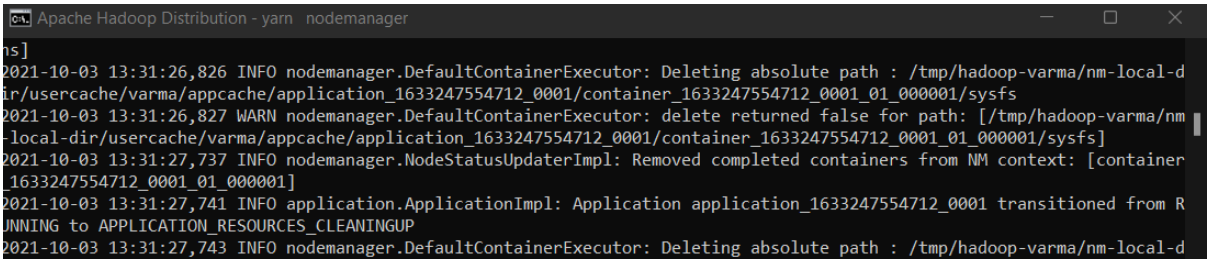
9. Download Hadoop winutils according to your Hadoop version from https://github.com/steveloughran/winutils and replace all files in Hadoop/bin folder.

10. Go to Hadoop/sbin and run start-all.cmd which will run 4 hadoop daemons.





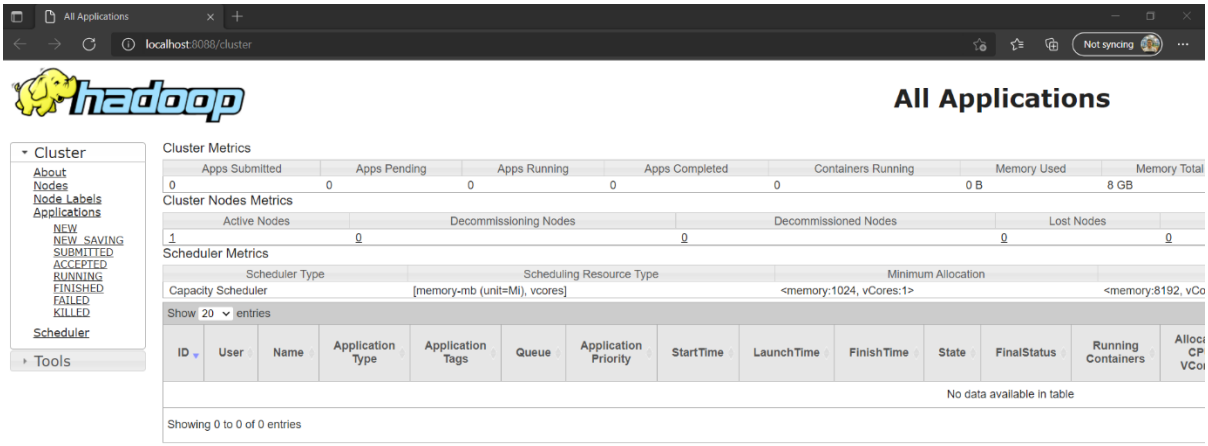


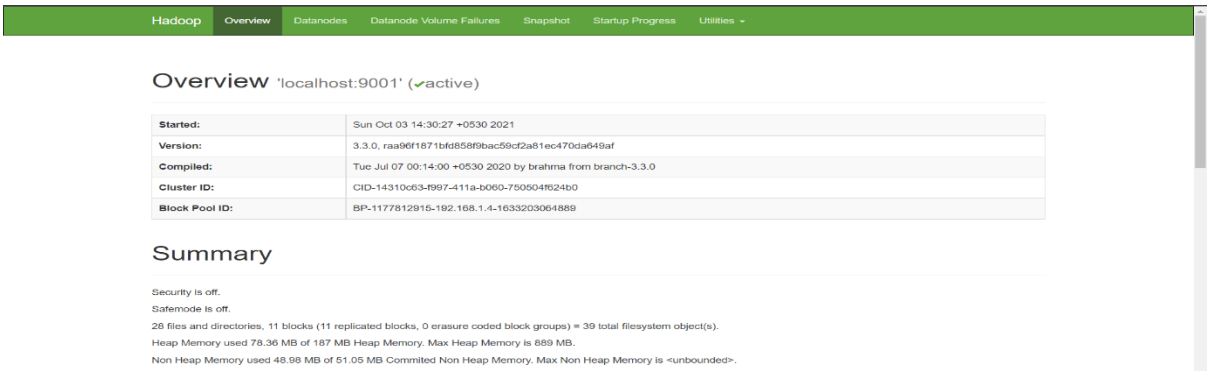


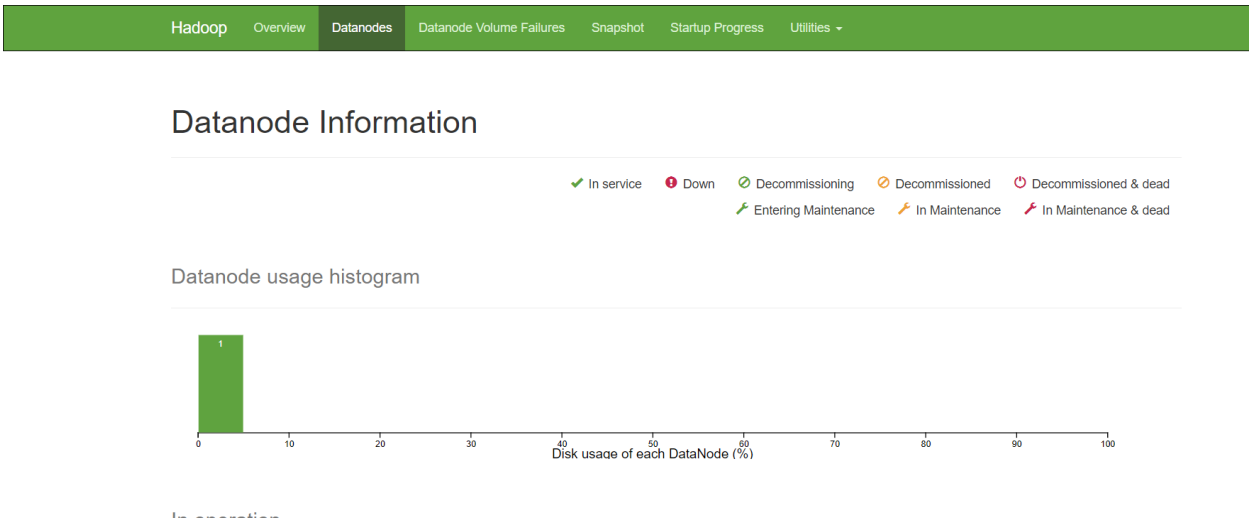
11. Open another command prompt and enter jps. If all daemons are running, the installation is successful.

```
C:\hadoop-3.3.0\sbin>jps
11248 Jps
14192
22240 DataNode
24016 RemoteMavenServer36
18312 NameNode
19020 ResourceManager
19436 NodeManager

C:\hadoop-3.3.0\sbin>
```

12. Open web browser and go to http://localhost:8088/ and http://localhost:9870/







RESULT

Hadoop is successfully installed and running.

PRACTICAL - 2

DATE: 02.08.2021

AIM - Explain some basic commands of Hadoop.

SOFTWARE USED
Powershell

COMMANDS

1. **mkdir**

Usage: hadoop fs -mkdir [-p] <paths>
Takes path uri's as argument and creates directories. Options:

- The -p option behavior is much like Unix mkdir -p, creating parent directories along the path.

Example:

- hadoop fs -mkdir /user/hadoop/dir1 /user/hadoop/dir2
- hadoop fs -mkdir hdfs://nn1.example.com/user/hadoop/dir hdfs://nn2.example.com/user/hadoop/dir

Exit Code:
Returns 0 on success and -1 on error.

```
PS F:\hadoop-3.2.1\hadoop-3.2.1\sbin> hdfs dfs -mkdir /newData
PS F:\hadoop-3.2.1\hadoop-3.2.1\sbin> hdfs dfs -ls /
Found 1 items
drwxr-xr-x   - varma supergroup          0 2021-09-27 23:55 /newData
PS F:\hadoop-3.2.1\hadoop-3.2.1\sbin>
```

2. **ls**

Usage: hadoop fs -ls [-C] [-d] [-h] [-q] [-R] [-t] [-S] [-r] [-u] [-e] <args> Options:

- -C: Display the paths of files and directories only.
- -d: Directories are listed as plain files.
- -h: Format file sizes in a human-readable fashion (e.g. 64.0m instead of 67108864).
- -q: Print? instead of non-printable characters.
- -R: Recursively list subdirectories encountered.
- -t: Sort output by modification time (most recent first).
- -S: Sort output by file size.
- -r: Reverse the sort order.
- -u: Use access time rather than modification time for display and sorting.
- -e: Display the erasure coding policy of files and directories only.

For a file ls returns stat on the file with the following format:

permissions number_of_replicas userid groupid filesize modification_date modification_time filename
For a directory it returns a list of its direct children as in Unix. A directory is listed as:
permissions userid groupid modification_date modification_time dirname

Files within a directory are ordered by filename by default. Example:

- hadoop fs -ls /user/hadoop/file1
- hadoop fs -ls -e /ecdir Exit Code:

Returns 0 on success and -1 on error.

```
PS F:\hadoop-3.2.1\hadoop-3.2.1\sbin> hdfs dfs -ls /
Found 1 items
drwxr-xr-x   - varma supergroup          0 2021-09-27 23:55 /newData
PS F:\hadoop-3.2.1\hadoop-3.2.1\sbin>
```

3. **lsr**

Usage: hadoop fs -lsr <args> Recursive version of ls.
Note: This command is deprecated. Instead use hadoop fs -ls –R

```
PS F:\hadoop-3.2.1\hadoop-3.2.1\sbin> hdfs dfs -lsr /
lsr: DEPRECATED: Please use 'ls -R' instead.
drwxr-xr-x   - varma supergroup          0 2021-09-27 23:55 /newData
PS F:\hadoop-3.2.1\hadoop-3.2.1\sbin>
```

4. **rmdir**

Usage: hadoop fs -rmdir [--ignore-fail-on-non-empty] URI [URI ...] Delete a directory.
Options:

- --ignore-fail-on-non-empty: When using wildcards, do not fail if a directory still contains files.

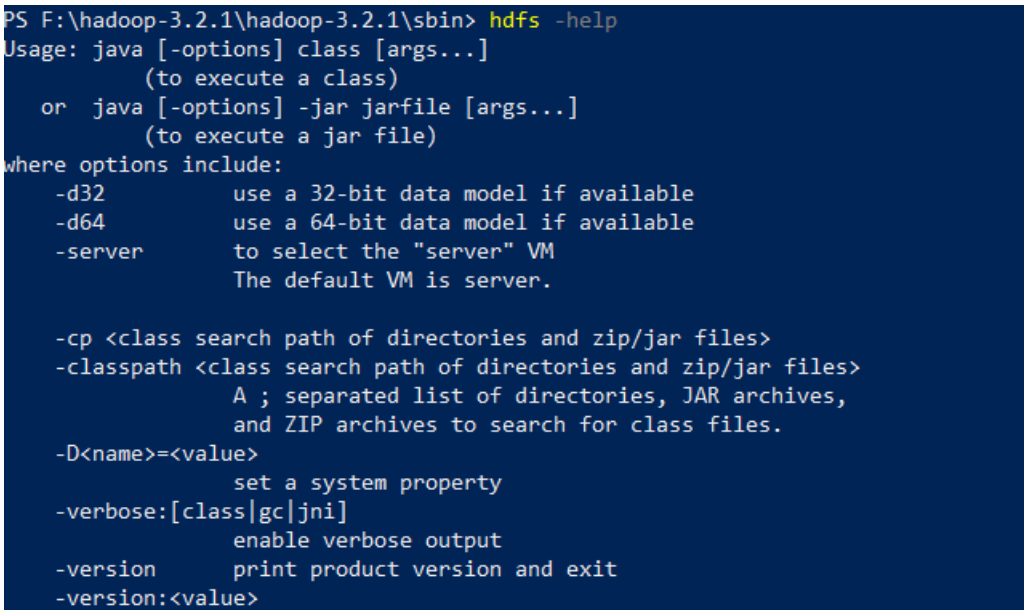
Example:

- hadoop fs -rmdir /user/hadoop/emptydir

```
PS F:\hadoop-3.2.1\hadoop-3.2.1\sbin> hdfs dfs -ls /
Found 2 items
drwxr-xr-x   - varma supergroup          0 2021-09-27 23:55 /newData
drwxr-xr-x   - varma supergroup          0 2021-09-28 00:17 /newData1
PS F:\hadoop-3.2.1\hadoop-3.2.1\sbin> hdfs dfs -rmdir /newData1
PS F:\hadoop-3.2.1\hadoop-3.2.1\sbin> hdfs dfs -ls /
Found 1 items
drwxr-xr-x   - varma supergroup          0 2021-09-27 23:55 /newData
PS F:\hadoop-3.2.1\hadoop-3.2.1\sbin>
```

5. **help**

Usage: hadoop fs -help Return usage output.



6. **du**

Usage: hadoop fs -du [-s] [-h] [-v] [-x] URI [URI ...]

Displays sizes of files and directories contained in the given directory or the length of a file in case it's just a file.

Options:

- The -s option will result in an aggregate summary of file lengths being displayed, rather than the individual files. Without the -s option, calculation is done by going 1-level deep from the given path.
- The -h option will format file sizes in a “human-readable” fashion (e.g 64.0m instead of 67108864)
- The -v option will display the names of columns as a header line.
- The -x option will exclude snapshots from the result calculation. Without the -x option (default), the result is always calculated from all INodes, including all snapshots under the given path.

The du returns three columns with the following format:

size disk_space_consumed_with_all_replicas full_path_name

Example:

- hadoop fs -du /user/hadoop/dir1 /user/hadoop/file1 hdfs://nn.example.com/user/hadoop/dir1 Exit Code: Returns 0 on success and -1 on error.

```
PS F:\hadoop-3.2.1\hadoop-3.2.1\sbin> hdfs dfs -du /newData
PS F:\hadoop-3.2.1\hadoop-3.2.1\sbin>
```

7. **count**

Usage: hadoop fs -count [-q] [-h] [-v] [-x] [-t [<storage type>]] [-u] [-e] [-s] <paths>

Count the number of directories, files and bytes under the paths that match the specified file pattern. Get the quota and the usage. The output columns with -count are: DIR_COUNT, FILE_COUNT,
CONTENT_SIZE, PATHNAME

The -u and -q options control what columns the output contains. -q means show quotas, -u limits the output to show quotas and usage only.
The output columns with -count -q are: QUOTA, REMAINING_QUOTA, SPACE_QUOTA, REMAINING_SPACE_QUOTA, DIR_COUNT, FILE_COUNT, CONTENT_SIZE, PATHNAME
The output columns with -count -u are: QUOTA, REMAINING_QUOTA, SPACE_QUOTA, REMAINING_SPACE_QUOTA, PATHNAME
The -t option shows the quota and usage for each storage type. The -t option is ignored if -u or -q option is not given. The list of possible parameters that can be used in -t option(case insensitive
except the parameter ""): "", “all”, “ram_disk”, “ssd”, “disk” or “archive”. The -h option shows sizes in human readable format.
The -v option displays a header line.

The -x option excludes snapshots from the result calculation. Without the -x option (default), the result is always calculated from all INodes, including all snapshots under the given path. The -x option is ignored if -u or -q option is given.
The -e option shows the erasure coding policy for each file.

The output columns with -count -e are: DIR_COUNT, FILE_COUNT, CONTENT_SIZE, ERASURECODING_POLICY, PATHNAME
The ERASURECODING_POLICY is the name of the policy for the file. If an erasure coding policy is setted on that file, it will return the name of the policy. If no erasure coding policy is setted, it will return "Replicated" which means it uses replication storage strategy.
The -s option shows the snapshot counts for each directory. Example:

- hadoop fs -count hdfs://nn1.example.com/file1 hdfs://nn2.example.com/file2
- hadoop fs -count -q hdfs://nn1.example.com/file1
- hadoop fs -count -q -h hdfs://nn1.example.com/file1
- hadoop fs -count -q -h -v hdfs://nn1.example.com/file1
- hadoop fs -count -u hdfs://nn1.example.com/file1
- hadoop fs -count -u -h hdfs://nn1.example.com/file1
- hadoop fs -count -u -h -v hdfs://nn1.example.com/file1
- hadoop fs -count -e hdfs://nn1.example.com/file1
- hadoop fs -count -s hdfs://nn1.example.com/file1 Exit Code:

Returns 0 on success and -1 on error.

8. **touchz**

Usage: hadoop fs -touchz URI [URI ...]

Create a file of zero length. An error is returned if the file exists with non-zero length. Example:

- hadoop fs -touchz pathname

Exit Code: Returns 0 on success and -1 on error.

```
PS F:\hadoop-3.2.1\hadoop-3.2.1\sbin> hdfs dfs -touchz /hello.txt
PS F:\hadoop-3.2.1\hadoop-3.2.1\sbin> hdfs dfs -ls /
Found 2 items
-rw-r--r--   1 varma supergroup          0 2021-09-28 21:22 /hello.txt
drwxr-xr-x   - varma supergroup          0 2021-09-27 23:55 /newData
PS F:\hadoop-3.2.1\hadoop-3.2.1\sbin>
```

9. **checksum**

Usage: hadoop fs -checksum [-v] URI Returns the checksum information of a file. Options

- The -v option displays blocks size for the file.

Example:

- hadoop fs -checksum hdfs://nn1.example.com/file1
- hadoop fs -checksum file:///etc/hosts

```
PS F:\hadoop-3.2.1\hadoop-3.2.1\sbin> hdfs dfs -checksum /
checksum: `/': Is a directory
PS F:\hadoop-3.2.1\hadoop-3.2.1\sbin> hdfs dfs -checksum /newData
checksum: `/newData': Is a directory
PS F:\hadoop-3.2.1\hadoop-3.2.1\sbin> hdfs dfs -checksum /hello.txt
/hello.txt      MD5-of-0MD5-of-0CRC32   00000000000000000000000070bc8f4b72a86921468bf8e8441dce51
PS F:\hadoop-3.2.1\hadoop-3.2.1\sbin>
```

10. cp

Usage: hadoop fs -cp [-f] [-p | -p[topax]] URI [URI ...] <dest>

Copy files from source to destination. This command allows multiple sources as well in which case the destination must be a directory.
'raw.*' namespace extended attributes are preserved if (1) the source and destination filesystems support them (HDFS only), and (2) all source and destination pathnames are in the /.reserved/raw hierarchy. Determination of whether raw.* namespace xattrs are preserved is independent of the -p (preserve) flag.
Options:

- The -f option will overwrite the destination if it already exists.
- The -p option will preserve file attributes [topx] (timestamps, ownership, permission, ACL, XAttr). If -p is specified with no *arg*, then preserves timestamps, ownership, permission. If -pa is specified, then preserves permission also because ACL is a super-set of permission. Determination of whether raw namespace extended attributes are preserved is independent of the -p flag.

Example:

- hadoop fs -cp /user/hadoop/file1 /user/hadoop/file2
- hadoop fs -cp /user/hadoop/file1 /user/hadoop/file2 /user/hadoop/dir Exit Code:

Returns 0 on success and -1 on error.

```
PS F:\hadoop-3.2.1\hadoop-3.2.1\sbin> hdfs dfs -ls /
Found 2 items
-rw-r--r--   1 varma supergroup          0 2021-09-28 21:22 /hello.txt
drwxr-xr-x   - varma supergroup          0 2021-09-27 23:55 /newData
PS F:\hadoop-3.2.1\hadoop-3.2.1\sbin> hdfs dfs -cp /hello.txt /newData
PS F:\hadoop-3.2.1\hadoop-3.2.1\sbin> hdfs dfs -ls /
Found 2 items
-rw-r--r--   1 varma supergroup          0 2021-09-28 21:22 /hello.txt
drwxr-xr-x   - varma supergroup          0 2021-09-28 21:58 /newData
PS F:\hadoop-3.2.1\hadoop-3.2.1\sbin> hdfs dfs -ls /newData
Found 1 items
-rw-r--r--   1 varma supergroup          0 2021-09-28 21:58 /newData/hello.txt
PS F:\hadoop-3.2.1\hadoop-3.2.1\sbin>
```

11. dus

Usage: hadoop fs -dus <args> Displays a summary of file lengths.
Note: This command is deprecated. Instead use hadoop fs -du -s.

```
PS F:\hadoop-3.2.1\hadoop-3.2.1\sbin> hdfs dfs -dus /
dus: DEPRECATED: Please use 'du -s' instead.
0  0  /
PS F:\hadoop-3.2.1\hadoop-3.2.1\sbin>
```

12. chmod

Usage: hadoop fs -chmod [-R] <MODE[,MODE]... | OCTAL_MODE> URI [URI ...]
Change the permissions of files. With -
R, make the change recursively through the directory structure. The user must be the owner of the file, or else a super-user.
Options

- The -R option will make the change recursively through the directory structure.

```
PS F:\hadoop-3.2.1\hadoop-3.2.1\sbin> hdfs dfs -ls /
Found 2 items
-rw-r--r--   1 varma supergroup          0 2021-09-28 21:22 /hello.txt
drwxr-xr-x   - varma supergroup          0 2021-09-28 21:58 /newData
PS F:\hadoop-3.2.1\hadoop-3.2.1\sbin> hdfs dfs -chmod -r /hello.txt
PS F:\hadoop-3.2.1\hadoop-3.2.1\sbin> hdfs dfs -ls /
Found 2 items
--w-------   1 varma supergroup          0 2021-09-28 21:22 /hello.txt
drwxr-xr-x   - varma supergroup          0 2021-09-28 21:58 /newData
PS F:\hadoop-3.2.1\hadoop-3.2.1\sbin>
```

13. stat
Usage: hadoop fs -stat [format] <path> ...
Print statistics about the file/directory at <path> in the specified format. Format accepts permissions in octal (%a) and symbolic (%A), file size in bytes (%b), type (%F), group name of owner (%g), name (%n), block size (%o), replication (%r), user name of owner(%u), access date(%x, %X), and modification date (%y, %Y). %x and %y show UTC date as "yyyy-MM-dd HH:mm:ss", and %X and
%Y show milliseconds since January 1, 1970 UTC. If the format is not specified, %y is used by default.

Example:

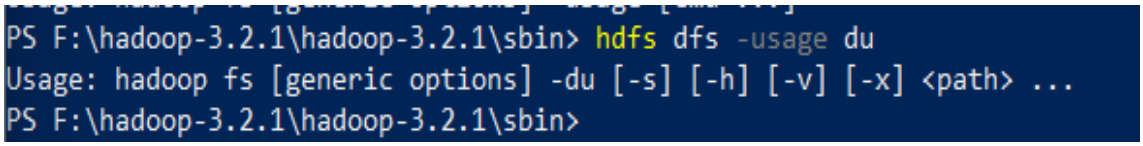
- hadoop fs -stat "type:%F perm:%a %u:%g size:%b mtime:%y time:%x name:%n" /file

Exit Code:
Returns 0 on success and -1 on error.

```
PS F:\hadoop-3.2.1\hadoop-3.2.1\sbin> hdfs dfs -stat /
2021-09-28 15:52:35
PS F:\hadoop-3.2.1\hadoop-3.2.1\sbin>
```

14. usage
Usage: hadoop fs -usage command

Return the help for an individual command.



RESULT - Hadoop commands running successfully.

PRACTICAL – 3

DATE: 09.08.2021

AIM

Write a java program to count the occurrence of each word in a file using:

a. Spit function
b. HashMap

SOFTWARE USED

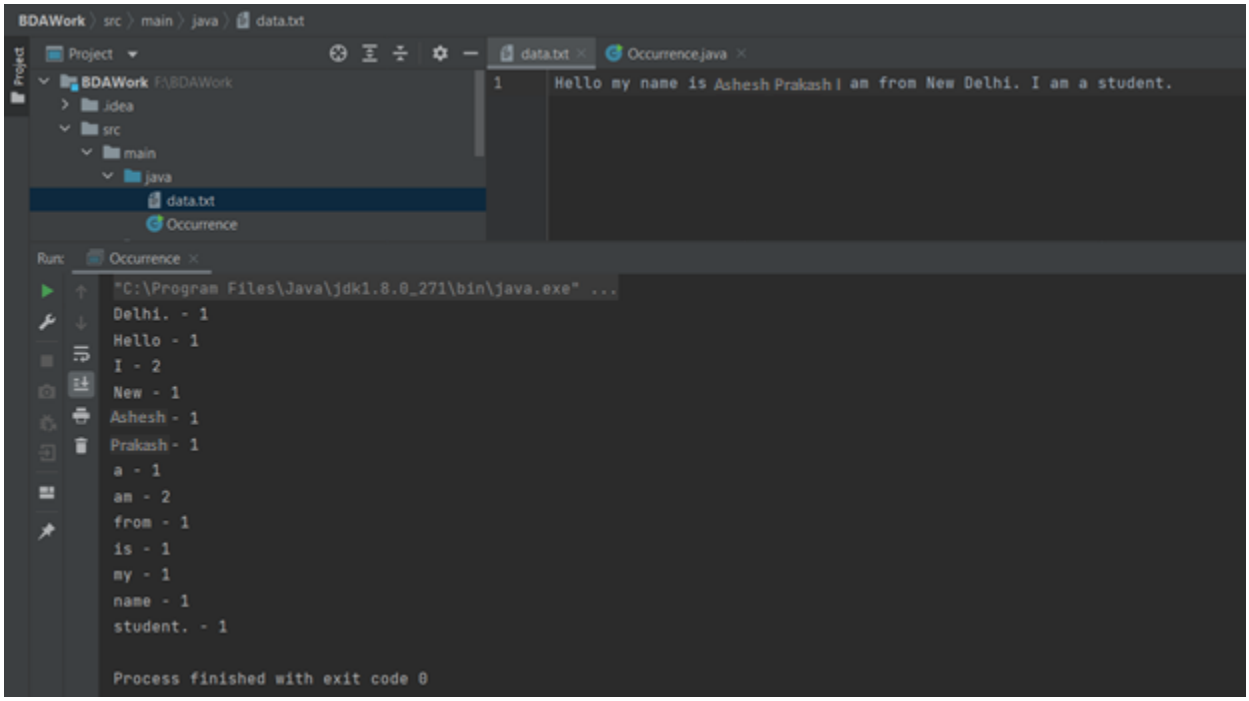
SOURCE CODE

```
import java.util.Map;

import java.util.TreeMap;



import java.io.BufferedReader;

import java.io.FileReader;



public class Occurrence {



   static void count_freq(String str)

   {

      Map<String,Integer> mp=new TreeMap<>();



      // Splitting to find the word

      String arr[]=str.split(" ");



      // Loop to iterate over the words
```

```
    for(int i=0;i<arr.length;i++)
    {
        // Condition to check if the array element is present the hash-map
        if(mp.containsKey(arr[i]))
        {
            mp.put(arr[i], mp.get(arr[i])+1);
        }
        else
        {
            mp.put(arr[i],1);
        }
    }

    // Loop to iterate over the elements of the map
    for(Map.Entry<String,Integer> entry:
            mp.entrySet())
    {
        System.out.println(entry.getKey()+
                " - "+entry.getValue());
    }
}

public static void main(String[] args) {
```

```
        try{
            String content="", line;
            //Opens a file in read mode
            FileReader file = new FileReader("F:\\BDAWork\\src\\main\\java\\data.txt");
            BufferedReader br = new BufferedReader(file);


            while((line = br.readLine()) != null) {
                content=content+line;
            }
            // Function Call
            count_freq(content);
        } catch(Exception e){
            System.out.print(e);
        }
    }
}
```

OUTPUT


BDAWork src main java data.txt
Project
data.txt
Occurrence.java
BDAWork F:\BDAWork
.idea
src
main
java
data.txt
Occurrence
1 Hello my name is Ashesh Prakash I am from New Delhi. I am a student.
Run: Occurrence
"C:\Program Files\Java\jdk1.8.0_271\bin\java.exe" ...
Delhi. - 1
Hello - 1
I - 2
New - 1
Ashesh - 1
Prakash - 1
a - 1
am - 2
from - 1
is - 1
my - 1
name - 1
student. - 1
Process finished with exit code 0

PRACTICAL – 4

DATE: 16.08.2021

AIM
Write a MapReduce script to count the occurrence of each word in a file.

SOFTWARE USED
Hadoop 3.3.0

THEORY

- Map-Reduce is a programming model that is mainly divided into two phases Map Phase and Reduce Phase. It is designed for processing the data in parallel which is divided on various machines (nodes).

- Hadoop Mapper is a function or task which is used to process all input records from a file and generate the output which works as input for Reducer. It produces the output by returning new key-value pairs.

- The input data has to be converted to key-value pairs as Mapper cannot process the raw input records or tuples (key-value pairs). The mapper also generates some small blocks of data while processing the input records as a key-value pair.

- But before sending this intermediate key-value pairs directly to the Reducer some process will be done which shuffle and sort the key-value pairs according to its key values, which means the value of the key is the main decisive factor for sorting.

- The output generated by the Reducer will be the final output which is then stored on HDFS (Hadoop Distributed File System). Reducer mainly performs some computation operation like addition, filtration, and aggregation.

- By default, the number of reducers utilized for process the output of the Mapper is 1 which is configurable and can be changed by the user according to the requirement.

SOURCE CODE

Mapper Class:

```
package bda4;// Importing libraries
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;

public class WCMapper extends MapReduceBase implements Mapper<LongWritable,
      Text, Text, IntWritable> {

   // Map function
   public void map(LongWritable key, Text value, OutputCollector<Text,
         IntWritable> output, Reporter rep) throws IOException
   {

      String line = value.toString();

      // Splitting the line on spaces
      for (String word : line.split(" "))
      {
         if (word.length() > 0)
         {
            output.collect(new Text(word), new IntWritable(1));
         }
      }
```

```
    }
}
```

Reducer Class:

```
package bda4;// Importing libraries
import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
public class WCReducer extends MapReduceBase implements Reducer<Text,
        IntWritable, Text, IntWritable> {

    // Reduce function
    public void reduce(Text key, Iterator<IntWritable> value,
                       OutputCollector<Text, IntWritable> output,
                       Reporter rep) throws IOException
    {

        int count = 0;
```

```
        // Counting the frequency of each words
        while (value.hasNext())
        {
            IntWritable i = value.next();
            count += i.get();
        }

        output.collect(key, new IntWritable(count));
    }
}
```

Driver Class:

```
package bda4;

import java.io.IOException;


import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
```

```
public class WCDriver extends Configured implements Tool {

   public int run(String args[]) throws IOException
   {
      if (args.length < 2)
      {
         System.out.println("Please give valid inputs");
         return -1;
      }

      JobConf conf = new JobConf(WCDriver.class);
      FileInputFormat.setInputPaths(conf, new Path(args[0]));
      FileOutputFormat.setOutputPath(conf, new Path(args[1]));
      conf.setMapperClass(WCMapper.class);
      conf.setReducerClass(WCReducer.class);
      conf.setMapOutputKeyClass(Text.class);
      conf.setMapOutputValueClass(IntWritable.class);
      conf.setOutputKeyClass(Text.class);
      conf.setOutputValueClass(IntWritable.class);
      JobClient.runJob(conf);
      return 0;
   }

   // Main Method
   public static void main(String[] args) throws Exception
   {
      int exitCode = ToolRunner.run(new WCDriver(), args);
      System.out.println(exitCode);
   }
}
```

OUTPUT

```
C:\hadoop-3.3.0\sbin>start-dfs.cmd

C:\hadoop-3.3.0\sbin>jps
19120 Jps
15272 NameNode
19436 DataNode

C:\hadoop-3.3.0\sbin>start-yarn.cmd
starting yarn daemons

C:\hadoop-3.3.0\sbin>jps
10304 NodeManager
2624 ResourceManager
15272 NameNode
13628 Jps
19436 DataNode

C:\hadoop-3.3.0\sbin> hdfs dfs -mkdir /test

C:\hadoop-3.3.0\sbin> hdfs dfs -ls /
Found 2 items
drwxr-xr-x   - varma supergroup          0 2021-10-03 01:16 /input_file
drwxr-xr-x   - varma supergroup          0 2021-10-03 11:55 /test
```

```
C:\hadoop-3.3.0\sbin>hadoop jar C:\Users\varma\Documents\MapReduceClient.jar  /test/BDAFile.txt /r_output
2021-10-03 11:59:31,020 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2021-10-03 11:59:31,222 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2021-10-03 11:59:31,919 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
2021-10-03 11:59:31,944 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/varma/.staging/job_1633242310008_0002
2021-10-03 11:59:32,342 INFO mapred.FileInputFormat: Total input files to process : 1
2021-10-03 11:59:32,455 INFO mapreduce.JobSubmitter: number of splits:2
2021-10-03 11:59:32,604 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1633242310008_0002
2021-10-03 11:59:32,604 INFO mapreduce.JobSubmitter: Executing with tokens: []
2021-10-03 11:59:32,812 INFO conf.Configuration: resource-types.xml not found
2021-10-03 11:59:32,813 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2021-10-03 11:59:33,250 INFO impl.YarnClientImpl: Submitted application application_1633242310008_0002
2021-10-03 11:59:33,296 INFO mapreduce.Job: The url to track the job: http://DESKTOP-BBFHCH4:8088/proxy/application_1633242310008_0002/
2021-10-03 11:59:33,297 INFO mapreduce.Job: Running job: job_1633242310008_0002
2021-10-03 11:59:43,486 INFO mapreduce.Job: Job job_1633242310008_0002 running in uber mode : false
2021-10-03 11:59:43,488 INFO mapreduce.Job:  map 0% reduce 0%
2021-10-03 11:59:51,759 INFO mapreduce.Job:  map 100% reduce 0%
2021-10-03 11:59:58,846 INFO mapreduce.Job:  map 100% reduce 100%
2021-10-03 11:59:58,857 INFO mapreduce.Job: Job job_1633242310008_0002 completed successfully
2021-10-03 11:59:58,971 INFO mapreduce.Job: Counters: 54
        File System Counters
                FILE: Number of bytes read=71
                FILE: Number of bytes written=796041
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=228
                HDFS: Number of bytes written=41
                HDFS: Number of read operations=11
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Launched map tasks=2
                Launched reduce tasks=1
                Data-local map tasks=2
                Total time spent by all maps in occupied slots (ms)=13057
                Total time spent by all reduces in occupied slots (ms)=4271
                Total time spent by all map tasks (ms)=13057
                Total time spent by all reduce tasks (ms)=4271
                Total vcore-milliseconds taken by all map tasks=13057
                Total vcore-milliseconds taken by all reduce tasks=4271
                Total megabyte-milliseconds taken by all map tasks=13370368
                Total megabyte-milliseconds taken by all reduce tasks=4373504
```

Hadoop Overview Datanodes Datanode Volume Failures Snapshot Startup Progress Utilities

Browse Directory

/r_output Go!

Show 25 entries Search:

	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
☐	-rw-r--r--	varma	supergroup	0 B	Oct 03 11:59	1	128 MB	_SUCCESS
☐	-rw-r--r--	varma	supergroup	41 B	Oct 03 11:59	1	128 MB	part-00000

Showing 1 to 2 of 2 entries Previous 1 Next

Hadoop, 2020.

```
C:\hadoop-3.3.0\sbin>hdfs dfs -cat /r_output/part-00000
Hello   1
My      1
is      1
name    1
Ashesh  1
Prakash 1

C:\hadoop-3.3.0\sbin>
```

PRACTICAL - 5

DATE: 06.09.2021

AIM - Write a MapReduce script to find the max and min temperature from record set stored in a text file.

SOFTWARE USED

Hadoop 3.3.0

THEORY

- **Map-Reduce** is a programming model that is mainly divided into two phases **Map Phase** and **Reduce Phase**. It is designed for processing the data in parallel which is divided on various machines (nodes).
- Hadoop Mapper is a function or task which is used to process all input records from a file and generate the output which works as input for Reducer. It produces the output by returning new key-value pairs.
- The input data has to be converted to key-value pairs as Mapper cannot process the raw input records or tuples (key-value pairs). The mapper also generates some small blocks of data while processing the input records as a key-value pair.
- But before sending this intermediate key-value pairs directly to the Reducer some process will be done which shuffle and sort the key-value pairs according to its key values, which means the value of the key is the main decisive factor for sorting.
- The output generated by the Reducer will be the final output which is then stored on HDFS (Hadoop Distributed File System). Reducer mainly performs some computation operation like addition, filtration, and aggregation.
- By default, the number of reducers utilized for process the output of the Mapper is 1 which is configurable and can be changed by the user according to the requirement.

CODE: -

```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.output.MultipleOutputs;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

/**
* @author devinline
*/
public class CalculateMaxAndMinTemeratureWithTime {
public static String calOutputName = "California";
public static String nyOutputName = "Newyork";
public static String njOutputName = "Newjersy";
public static String ausOutputName = "Austin";
public static String bosOutputName = "Boston";
public static String balOutputName = "Baltimore";

public static class WhetherForcastMapper extends
  Mapper<Object, Text, Text, Text> {

public void map(Object keyOffset, Text dayReport, Context con)
   throws IOException, InterruptedException {
```

```
StringTokenizer strTokens = new StringTokenizer(
  dayReport.toString(), "\t");
int counter = 0;
Float currnetTemp = null;
Float minTemp = Float.MAX_VALUE;
Float maxTemp = Float.MIN_VALUE;
String date = null;
String currentTime = null;
String minTempANDTime = null;
String maxTempANDTime = null;

while (strTokens.hasMoreElements()) {
 if (counter == 0) {
  date = strTokens.nextToken();
 } else {
  if (counter % 2 == 1) {
   currentTime = strTokens.nextToken();
  } else {
   currnetTemp = Float.parseFloat(strTokens.nextToken());
   if (minTemp > currnetTemp) {
    minTemp = currnetTemp;
    minTempANDTime = minTemp + "AND" + currentTime;
   }
   if (maxTemp < currnetTemp) {
    maxTemp = currnetTemp;
    maxTempANDTime = maxTemp + "AND" + currentTime;
   }
  }
 }
 counter++;
}
```

```
  // Write to context - MinTemp, MaxTemp and corresponding time
  Text temp = new Text();
  temp.set(maxTempANDTime);
  Text dateText = new Text();
  dateText.set(date);
  try {
   con.write(dateText, temp);
  } catch (Exception e) {
   e.printStackTrace();
  }

  temp.set(minTempANDTime);
  dateText.set(date);
  con.write(dateText, temp);

}
}

public static class WhetherForcastReducer extends
  Reducer<Text, Text, Text, Text> {
MultipleOutputs<Text, Text> mos;

public void setup(Context context) {
  mos = new MultipleOutputs<Text, Text>(context);
}

public void reduce(Text key, Iterable<Text> values, Context context)
   throws IOException, InterruptedException {
  int counter = 0;
  String reducerInputStr[] = null;
  String f1Time = "";
```

```
String f2Time = "";
String f1 = "", f2 = "";
Text result = new Text();
for (Text value : values) {

 if (counter == 0) {
  reducerInputStr = value.toString().split("AND");
  f1 = reducerInputStr[0];
  f1Time = reducerInputStr[1];
 }

 else {
  reducerInputStr = value.toString().split("AND");
  f2 = reducerInputStr[0];
  f2Time = reducerInputStr[1];
 }

 counter = counter + 1;
}
if (Float.parseFloat(f1) > Float.parseFloat(f2)) {

 result = new Text("Time: " + f2Time + " MinTemp: " + f2 + "\t"
   + "Time: " + f1Time + " MaxTemp: " + f1);
} else {

 result = new Text("Time: " + f1Time + " MinTemp: " + f1 + "\t"
   + "Time: " + f2Time + " MaxTemp: " + f2);
}
String fileName = "";
if (key.toString().substring(0, 2).equals("CA")) {
 fileName = CalculateMaxAndMinTemeratureTime.calOutputName;
```

```
  } else if (key.toString().substring(0, 2).equals("NY")) {
   fileName = CalculateMaxAndMinTemeratureTime.nyOutputName;
  } else if (key.toString().substring(0, 2).equals("NJ")) {
   fileName = CalculateMaxAndMinTemeratureTime.njOutputName;
  } else if (key.toString().substring(0, 3).equals("AUS")) {
   fileName = CalculateMaxAndMinTemeratureTime.ausOutputName;
  } else if (key.toString().substring(0, 3).equals("BOS")) {
   fileName = CalculateMaxAndMinTemeratureTime.bosOutputName;
  } else if (key.toString().substring(0, 3).equals("BAL")) {
   fileName = CalculateMaxAndMinTemeratureTime.balOutputName;
  }
  String strArr[] = key.toString().split("_");
  key.set(strArr[1]); //Key is date value
  mos.write(fileName, key, result);
}

@Override
public void cleanup(Context context) throws IOException,
   InterruptedException {
  mos.close();
}
}

public static void main(String[] args) throws IOException,
  ClassNotFoundException, InterruptedException {
Configuration conf = new Configuration();
Job job = Job.getInstance(conf, "Wheather Statistics of USA");
job.setJarByClass(CalculateMaxAndMinTemeratureWithTime.class);

job.setMapperClass(WhetherForcastMapper.class);
job.setReducerClass(WhetherForcastReducer.class);
```

```
job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(Text.class);

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

MultipleOutputs.addNamedOutput(job, calOutputName,
  TextOutputFormat.class, Text.class, Text.class);
MultipleOutputs.addNamedOutput(job, nyOutputName,
  TextOutputFormat.class, Text.class, Text.class);
MultipleOutputs.addNamedOutput(job, njOutputName,
  TextOutputFormat.class, Text.class, Text.class);
MultipleOutputs.addNamedOutput(job, bosOutputName,
  TextOutputFormat.class, Text.class, Text.class);
MultipleOutputs.addNamedOutput(job, ausOutputName,
  TextOutputFormat.class, Text.class, Text.class);
MultipleOutputs.addNamedOutput(job, balOutputName,
  TextOutputFormat.class, Text.class, Text.class);

// FileInputFormat.addInputPath(job, new Path(args[0]));
// FileOutputFormat.setOutputPath(job, new Path(args[1]));
Path pathInput = new Path(
  "hdfs://192.168.213.133:54310/weatherInputData/input_temp.txt");
Path pathOutputDir = new Path(
  "hdfs://192.168.213.133:54310/user/hduser1/testfs/output_mapred3");
FileInputFormat.addInputPath(job, pathInput);
FileOutputFormat.setOutputPath(job, pathOutputDir);

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

```
} catch (Exception e) {
  // TODO Auto-generated catch block
  e.printStackTrace();
}
}


}
```

OUTPUT: -

```
bash-4.1# hdfs dfs -mkdir /weatherInputData/
bash-4.1# vi input_temp.txt
bash-4.1# hdfs dfs -put input_temp.txt /weatherInputData/
bash-4.1# hdfs dfs -ls  /weatherInputData/
Found 1 items
-rw-r--r--   1 root supergroup      14474 2021-10-03 16:09 /weatherInputData/input_temp.txt
bash-4.1#
```

```
bash-4.1# hdfs dfs -cat /output_mapred3/Austin-r-00000
25-Jan-2014 Time: 12:34:542 MinTemp: -22.3 Time: 05:12:345 MaxTemp: 35.7
26-Jan-2014 Time: 22:00:093 MinTemp: -27.0 Time: 05:12:345 MaxTemp: 55.7
27-Jan-2014 Time: 02:34:542 MinTemp: -22.3 Time: 05:12:345 MaxTemp: 55.7
29-Jan-2014 Time: 14:00:093 MinTemp: -17.0 Time: 02:34:542 MaxTemp: 62.9
30-Jan-2014 Time: 22:00:093 MinTemp: -27.0 Time: 05:12:345 MaxTemp: 49.2
31-Jan-2014 Time: 14:00:093 MinTemp: -17.0 Time: 03:12:187 MaxTemp: 56.0
```

PRACTICAL - 6

DATE: 20.09.2021

AIM

Write a MapReduce script to implement Map side and Reduce side joins.

SOFTWARE USED

Hadoop 3.3.0

THEORY

MapReduce Join operation is used to combine two large datasets. However, this process involves writing lots of code to perform the actual join operation. Joining two datasets begins by comparing the size of each dataset. If one dataset is smaller as compared to the other dataset then smaller dataset is distributed to every data node in the cluster.

Once a join in MapReduce is distributed, either Mapper or Reducer uses the smaller dataset to perform a lookup for matching records from the large dataset and then combine those records to form output records.

Depending upon the place where the actual join is performed, joins in Hadoop are classified into-

1. Map-side join – When the join is performed by the mapper, it is called as map-side join. In this type, the join is performed before data is actually consumed by the map function. It is mandatory that the input to each map is in the form of a partition and is in sorted order. Also, there must be an equal number of partitions and it must be sorted by the join key.

2. Reduce-side join – When the join is performed by the reducer, it is called as reduce-side join. There is no necessity in this join to have a dataset in a structured form (or partitioned).

Here, map side processing emits join key and corresponding tuples of both the tables. As an effect of this processing, all the tuples with same join key fall into the same reducer which then joins the records with same join key.

1. DeptNameMapper.java

```
import java.io.IOException;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
public class DeptNameMapper extends MapReduceBase implements
      Mapper<LongWritable, Text, TextPair, Text>  {
   @Override
   public void map(LongWritable key, Text value,
               OutputCollector<TextPair, Text> output, Reporter reporter)
         throws IOException
   {
      String valueString = value.toString();
      String[] SingleNodeData = valueString.split("\t");
      output.collect(new TextPair(SingleNodeData[0], "0"), new
            Text(SingleNodeData[1]));
   } }
```

2. JoinReducer.java

```
import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.*;
public class JoinReducer extends MapReduceBase implements
      Reducer<TextPair, Text, Text, Text> {
   @Override
   public void reduce (TextPair key, Iterator<Text> values,
                  OutputCollector<Text, Text> output, Reporter reporter)
         throws IOException
   {
      Text nodeId = new Text(values.next());
      while (values.hasNext()) {
         Text node = values.next();
         Text outValue = new Text(nodeId.toString() + "\t\t" +
               node.toString());
         output.collect(key.getFirst(), outValue);
      }
   }
}
```

3. DeptEmpStrengthMapper.java

```
import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.io.IntWritable;
public class DeptEmpStrengthMapper extends MapReduceBase implements
      Mapper<LongWritable, Text, TextPair, Text> {
   @Override
   public void map(LongWritable key, Text value,
               OutputCollector<TextPair, Text> output, Reporter reporter)
         throws IOException
   {
      String valueString = value.toString();
      String[] SingleNodeData = valueString.split("\t");
      output.collect(new TextPair(SingleNodeData[0], "1"), new
            Text(SingleNodeData[1]));
   }
}
```

4. TextPair.java

```
import java.io.*;
import org.apache.hadoop.io.*;
public class TextPair implements WritableComparable<TextPair> {
   private Text first;
   private Text second;
   public TextPair() {
      set(new Text(), new Text());
   }
   public TextPair(String first, String second) {
      set(new Text(first), new Text(second));
   }
   public TextPair(Text first, Text second) {
      set(first, second);
   }
   public void set(Text first, Text second) {
      this.first = first;
      this.second = second;
   }
```

```
public Text getFirst() {
   return first;
}
public Text getSecond() {
   return second;
}
@Override
public void write(DataOutput out) throws IOException {
   first.write(out);
   second.write(out);
}
@Override
public void readFields(DataInput in) throws IOException {
   first.readFields(in);
   second.readFields(in);
}
@Override
public int hashCode() {
   return first.hashCode() * 163 + second.hashCode();
}
@Override
public boolean equals(Object o) {
   if (o instanceof TextPair) {
      TextPair tp = (TextPair) o;
      return first.equals(tp.first) && second.equals(tp.second);
   }
   return false;
}
@Override
public String toString() {
   return first + "\t" + second;
}
@Override
public int compareTo(TextPair tp) {
   int cmp = first.compareTo(tp.first);
   if (cmp != 0) {
      return cmp;
   }
   return second.compareTo(tp.second);
}
public static class Comparator extends WritableComparator {
   private static final Text.Comparator TEXT_COMPARATOR = new
         Text.Comparator();
   public Comparator() {
      super(TextPair.class);
   }
```

```
    @Override
    public int compare(byte[] b1, int s1, int l1,
                       byte[] b2, int s2, int l2) {
      try {
        int firstL1 = WritableUtils.decodeVIntSize(b1[s1]) + readVInt(b1,
            s1);
        int firstL2 = WritableUtils.decodeVIntSize(b2[s2]) + readVInt(b2,
            s2);
        int cmp = TEXT_COMPARATOR.compare(b1, s1, firstL1, b2, s2,
            firstL2);
        if (cmp != 0) {
          return cmp;
        }
        return TEXT_COMPARATOR.compare(b1, s1 + firstL1, l1 - firstL1,
            b2, s2 + firstL2, l2 - firstL2);
      } catch (IOException e) {
        throw new IllegalArgumentException(e);
      }
    }
  }
  static {
    WritableComparator.define(TextPair.class, new Comparator());
  }
  public static class FirstComparator extends WritableComparator {
    private static final Text.Comparator TEXT_COMPARATOR = new
        Text.Comparator();
    public FirstComparator() {
      super(TextPair.class);
    }
    @Override
    public int compare(byte[] b1, int s1, int l1,
                       byte[] b2, int s2, int l2) {
      try {
        int firstL1 = WritableUtils.decodeVIntSize(b1[s1]) + readVInt(b1,
            s1);
        int firstL2 = WritableUtils.decodeVIntSize(b2[s2]) + readVInt(b2,
            s2);
        return TEXT_COMPARATOR.compare(b1, s1, firstL1, b2, s2, firstL2);
      } catch (IOException e) {
        throw new IllegalArgumentException(e);
      }
    }
    @Override
    public int compare(WritableComparable a, WritableComparable b) {
      if (a instanceof TextPair && b instanceof TextPair) {
        return ((TextPair) a).first.compareTo(((TextPair) b).first);
```

```
        }
        return super.compare(a, b);
      }
   }
}
```

5. JoinDriver.java

```
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.mapred.lib.MultipleInputs;
import org.apache.hadoop.util.*;
public class JoinDriver extends Configured implements Tool {
   public static class KeyPartitioner implements Partitioner<TextPair,
         Text> {
      @Override
      public void configure(JobConf job) {}
      @Override
      public int getPartition(TextPair key, Text value, int
            numPartitions) {
         return (key.getFirst().hashCode() & Integer.MAX_VALUE) %
               numPartitions;
      }
   }
   @Override
   public int run(String[] args) throws Exception {
      if (args.length != 3) {
         System.out.println("Usage: <Department Emp Strength input><Department Name input>
<output>");
         return -1;
      }
      JobConf conf = new JobConf(getConf(), getClass());
      conf.setJobName("Join 'Department Emp Strength input' with 'Department Name input'");
      Path AInputPath = new Path(args[0]);
      Path BInputPath = new Path(args[1]);
      Path outputPath = new Path(args[2]);
      MultipleInputs.addInputPath(conf, AInputPath,
            TextInputFormat.class, DeptNameMapper.class);
      MultipleInputs.addInputPath(conf, BInputPath,
            TextInputFormat.class, DeptEmpStrengthMapper.class);
      FileOutputFormat.setOutputPath(conf, outputPath);
      conf.setPartitionerClass(KeyPartitioner.class);
      conf.setOutputValueGroupingComparator(TextPair.FirstComparator.class);
```

```
        conf.setMapOutputKeyClass(TextPair.class);
        conf.setReducerClass(JoinReducer.class);
        conf.setOutputKeyClass(Text.class);
        JobClient.runJob(conf);
        return 0;
    }
    public static void main(String[] args) throws Exception {
        int exitCode = ToolRunner.run(new JoinDriver(), args);
        System.exit(exitCode);
    }
}
```

OUTPUT

```
C:\hadoop-3.3.0\sbin>jps
14192
23888 RemoteMavenServer36
2880 Jps
18276 DataNode
7992 NameNode
12908 ResourceManager
17788 NodeManager

C:\hadoop-3.3.0\sbin>hdfs dfs -mkdir /joins

C:\hadoop-3.3.0\sbin>hdfs dfs -mkdir /joins/input

C:\hadoop-3.3.0\sbin>cd
C:\hadoop-3.3.0\sbin

C:\hadoop-3.3.0\sbin>cd
C:\hadoop-3.3.0\sbin

C:\hadoop-3.3.0\sbin>cd ...

C:\hadoop-3.3.0\sbin>cd ..

C:\hadoop-3.3.0>cd ..
```

```
C:\Users\ashesh\Desktop\MapReduceJoin\MapReduceJoin>hdfs dfs -ls /
Found 5 items
drwxr-xr-x   - ashesh supergroup          0 2021-10-03 01:16 /input_file
drwxr-xr-x   - ashesh supergroup          0 2021-10-03 13:23 /joins
drwxr-xr-x   - ashesh supergroup          0 2021-10-03 11:59 /r_output
drwxr-xr-x   - ashesh supergroup          0 2021-10-03 11:57 /test
drwx------   - ashesh supergroup          0 2021-10-03 11:58 /tmp

C:\Users\ashesh\Desktop\MapReduceJoin\MapReduceJoin>hdfs dfs -copyFromLocal DeptStrength.txt DeptName.txt /joins/input

C:\Users\ashesh\Desktop\MapReduceJoin\MapReduceJoin>hdfs dfs -ls /joins/input
Found 2 items
-rw-r--r--   1 ashesh supergroup         59 2021-10-03 13:28 /joins/input/DeptName.txt
-rw-r--r--   1 ashesh supergroup         50 2021-10-03 13:28 /joins/input/DeptStrength.txt
```

```
C:\Users\ashesh\Desktop\MapReduceJoin\MapReduceJoin>hdfs dfs -cat /joins/input/DeptName.txt
Dept_ID Dept_Name
A11     Finance
B12     HR
C13     Manufacturing

C:\Users\ashesh\Desktop\MapReduceJoin\MapReduceJoin>hdfs dfs -cat /joins/input/DeptStrength.txt
Dept_ID Total_Employee
A11     50
B12     100
C13     250
```

```
C:\Users\ashesh\Desktop\MapReduceJoin\MapReduceJoin>hdfs dfs -ls /joins/output
Found 2 items
-rw-r--r--   1 ashesh supergroup          0 2021-10-03 13:31 /joins/output/_SUCCESS
-rw-r--r--   1 ashesh supergroup         85 2021-10-03 13:31 /joins/output/part-00000

C:\Users\ashesh\Desktop\MapReduceJoin\MapReduceJoin>hdfs dfs -cat /joins/output/part-00000
A11     50              Finance
B12     100             HR
C13     250             Manufacturing
Dept_ID Total_Employee          Dept_Name
```

PRACTICAL - 7

DATE: 27.09.2021

AIM - Write a pig script to analyse the twitter data.

SOFTWARE USED

Hadoop 3.3.0 with Apache PIG

THEORY

Apache Pig is a high-level platform or tool which is used to process the large datasets. It provides a high-level of abstraction for processing over the MapReduce. It provides a high-level scripting language, known as Pig Latin which is used to develop the data analysis codes. First, to process the data which is stored in the HDFS, the programmers will write the scripts using the Pig Latin Language.

One limitation of MapReduce is that the development cycle is very long. Writing the reducer and mapper, compiling packaging the code, submitting the job and retrieving the output is a time-consuming task. Apache Pig reduces the time of development using the multi-query approach.

The steps to be followed are:

1. First of all, twitter imports the twitter tables (i.e. user table and tweet table) into the HDFS.
2. Then Apache Pig loads (LOAD) the tables into Apache Pig framework.
3. Tweets are analysed as per the desired problem statements using Pig Latin commands.
4. Finally, this result is stored back in the HDFS.

SOURCE CODE

```
twitter_data = LOAD '/Users/admin/Documents/tweets/' USING
com.twitter.elephantbird.pig.load.JsonLoader('-nestedLoad') AS myMap;*/

extract_details = FOREACH twitter_data GENERATE myMap#'user' as User, myMap#'id' AS
id ,myMap#'text' AS text;

tokens = FOREACH extract_details GENERATE id, text, FLATTEN (TOKENIZE(text)) AS
word;

dictionary = LOAD '/Users/admin/Documents/tweets/AFINN.txt' using
PigStorage('\t') AS (word:chararray, rating:int);

word_rating = JOIN tokens BY word left outer, dictionary BY word using'replicated';

rating = FOREACH word_rating GENERATE tokens::id as id, tokens::text astext,
dictionary::rating as rate;

word_group = GROUP rating BY (id, text);
avg_rate = FOREACH word_group GENERATE group, AVG(rating.rate) as
tweet_rating;

dump avg_rate;
```

RESULT

The PIG script has been written successfully.

PRACTICAL - 8

DATE: 04.10.2021

AIM - Write a hive script to analyse last 10 years of crime data.

SOFTWARE USED

Hadoop 3.3.0 with Apache HIVE

THEORY

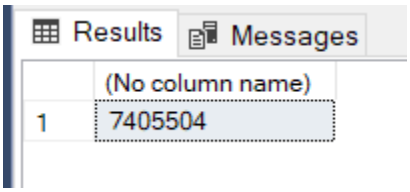
- Apache Hive is an open source data warehouse software for reading, writing and managing large data set files that are stored directly in either the Apache Hadoop Distributed File System (HDFS) or other data storage systems such as Apache HBase.
- Hive enables SQL developers to write Hive Query Language (HQL) statements that are similar to standard SQL statements for data query and analysis. HiveQL is similar to SQL for querying on schema info on the Metastore. It is one of the replacements of traditional approach for MapReduce program. Instead of writing MapReduce program in Java, we can write a query for MapReduce job and process it.
- Meta Store: Hive chooses respective database servers to store the schema or Metadata of tables, databases, columns in a table, their data types, and HDFS mapping.
- It is designed to make MapReduce programming easier because we don't have to know and write lengthy Java code. Instead, we can write queries more simply in HQL, and Hive can then create the map and reduce the functions.
- The steps to be followed are:
    1. First of all, load crime data into the HDFS.
    2. Create a table using HIVE
    3. Execute HIVE commands to analyse data.

SOURCE CODE

```
CREATE DATABASE crime_data


CREATE external TABLE chicago_crimes.all_crimes(
        id string,
        casenumber string,
        caldate string, block
        string,
        iucr string, primarytype
        string,description string,
        locationdescription string,arrest
        boolean,
        domestic boolean,beat
        string, district string,
        ward string,
        communityarea string,
        fbicode string, xcoordinate
        string, ycoordinate string,
        year string, updatedon
        string,
        latitude decimal(10, 0),
        longitude decimal(10, 0),
        location string
)
row format delimited fields terminated by ','stored as
textfile
location ‘/Users/admin/Documents’
```

OUTPUT

select count(*) from crime_data;



Results | Messages

	(No column name)
1	7405504

select * from crime_data where Location_Description = 'RESIDENCE';

Results | Messages

ID	Case_Number	Date	Block	IUCR	Primary_Type	Description	Location_Description	Arrest	Domestic	Beat	District	Ward	Community_Area	FBI_Code	X
10224738	HY411648	09/05/2015 01:30:00 PM	043XX S WOOD ST	0486	BATTERY	DOMESTIC BATTERY SIMPLE	RESIDENCE	0	1	924	009	12	61	08B	1
11646166	JC213529	09/01/2018 12:01:00 AM	082XX S INGLESIDE AVE	0810	THEFT	OVER $500	RESIDENCE	0	1	631	006	8	44	06	N
10224742	HY411435	09/05/2015 10:55:00 AM	082XX S LOOMIS BLVD	0610	BURGLARY	FORCIBLE ENTRY	RESIDENCE	0	0	614	006	21	71	05	1
10224757	HY411388	09/05/2015 09:55:00 AM	088XX S PAULINA ST	0610	BURGLARY	FORCIBLE ENTRY	RESIDENCE	1	0	2221	022	21	71	05	1
10224775	HY411666	09/05/2015 09:00:00 AM	066XX S KEATING AVE	2825	OTHER OFFENSE	HARASSMENT BY TELEPHONE	RESIDENCE	0	1	833	008	13	65	26	1
10224794	HY411721	09/05/2015 03:44:00 PM	074XX S WABASH AVE	143B	WEAPONS VIOLATION	UNLAWFUL POSS OTHER FIREARM	RESIDENCE	0	0	323	003	6	69	15	1
10224802	HY411753	09/05/2015 03:00:00 PM	051XX N AVERS AVE	2826	OTHER OFFENSE	HARASSMENT BY ELECTRONIC MEANS	RESIDENCE	0	1	1712	017	39	14	26	1
10224806	HY411806	09/03/2015 01:30:00 PM	010XX N HAMLIN AVE	1130	DECEPTIVE PRACTICE	FRAUD OR CONFIDENCE GAME	RESIDENCE	0	0	1112	011	27	23	11	1
10224810	HY411746	09/05/2015 03:15:00 PM	114XX S ST LAWRENCE AVE	0486	BATTERY	DOMESTIC BATTERY SIMPLE	RESIDENCE	1	1	531	005	9	50	08B	1
10224811	HY410834	09/04/2015 08:25:00 AM	001XX E 110TH ST	0610	BURGLARY	FORCIBLE ENTRY	RESIDENCE	0	0	513	005	9	49	05	1
10224812	HY411619	09/05/2015 12:30:00 AM	064XX S SAGINAW AVE	0820	THEFT	$500 AND UNDER	RESIDENCE	0	0	423	004	7	46	06	1
10224815	HY411431	09/05/2015 10:53:00 AM	0000X W 114TH PL	0420	BATTERY	AGGRAVATED:KNIFE/CUTTING INSTR	RESIDENCE	0	0	522	005	34	49	04B	1
10224830	HY411557	09/04/2015 11:00:00 PM	106XX S PRAIRIE AVE	2820	OTHER OFFENSE	TELEPHONE THREAT	RESIDENCE	0	0	512	005	9	49	26	1
10224832	HY411712	09/04/2015 08:00:00 PM	013XX W 50TH ST	0820	THEFT	$500 AND UNDER	RESIDENCE	0	1	933	009	19	61	06	1
11034701	JA366925	01/01/2001 11:00:00 AM	016XX E 86TH PL	1153	DECEPTIVE PRACTICE	FINANCIAL IDENTITY THEFT OVER $ 300	RESIDENCE	0	0	412	004	8	45	11	N
10224843	HY411734	09/05/2015 03:43:00 PM	101XX S PROSPECT AVE	0486	BATTERY	DOMESTIC BATTERY SIMPLE	RESIDENCE	0	1	2213	022	19	72	08B	1

select * from crime_data where WARD = '6';

Results | Messages

ID	Case_Number	Date	Block	IUCR	Primary_Type	Description	Location_Description	Arrest	Domestic	Beat	District	Ward	Community_Area
10224782	HY411661	09/05/2015 02:35:00 PM	066XX S PERRY AVE	1811	NARCOTICS	POSS: CANNABIS 30GMS OR LESS	ALLEY	1	0	722	007	6	69
10224784	HY411422	09/04/2015 07:30:00 PM	004XX W MARQUETTE RD	0810	THEFT	OVER $500	OTHER	0	0	722	007	6	66
10224794	HY411721	09/05/2015 03:44:00 PM	074XX S WABASH AVE	143B	WEAPONS VIOLATION	UNLAWFUL POSS OTHER FIREARM	RESIDENCE	0	0	323	003	6	69
10224801	HY411697	09/05/2015 03:05:00 PM	007XX E 90TH PL	0486	BATTERY	DOMESTIC BATTERY SIMPLE	APARTMENT	0	0	633	006	6	44
10224864	HY410860	09/04/2015 09:30:00 PM	001XX E 75TH ST	051A	ASSAULT	AGGRAVATED: HANDGUN	STREET	0	0	623	006	6	69
10224885	HY411859	09/05/2015 05:00:00 AM	076XX S COTTAGE GROVE AVE	1310	CRIMINAL DAMAGE	TO PROPERTY	COMMERCIAL / BUSINESS OFFICE	0	0	624	006	6	69
10224888	HY411820	09/05/2015 04:21:00 PM	003XX E 75TH ST	0460	BATTERY	SIMPLE	PARKING LOT/GARAGE(NON RESID.)	0	0	623	006	6	69
10224870	HY411737	09/05/2015 03:40:00 PM	080XX S LANGLEY AVE	0820	THEFT	$500 AND UNDER	APARTMENT	0	0	631	006	6	44
10225004	HY411885	09/04/2015 07:00:00 PM	006XX E 79TH ST	1150	DECEPTIVE PRACTICE	CREDIT CARD FRAUD	RESTAURANT	0	0	624	006	6	69
10225012	HY412057	09/05/2015 09:15:00 PM	069XX S NORMAL BLVD	0486	BATTERY	DOMESTIC BATTERY SIMPLE	SIDEWALK	0	1	732	007	6	66
10225015	HY412040	09/05/2015 09:04:00 PM	008XX E 75TH ST	0486	BATTERY	DOMESTIC BATTERY SIMPLE	SIDEWALK	1	1	624	006	6	69
10225044	HY412066	09/05/2015 09:25:00 PM	085XX S STATE ST	2026	NARCOTICS	POSS: SYNTHETIC DRUGS	STREET	1	0	632	006	6	44
10225061	HY412077	09/05/2015 09:22:00 PM	0000X W 79TH ST	1811	NARCOTICS	POSS: CANNABIS 30GMS OR LESS	CTA STATION	1	0	623	006	6	44
10225074	HY412099	09/05/2015 09:10:00 PM	001XX W 72ND ST	2024	NARCOTICS	POSS: HEROIN(WHITE)	STREET	1	0	731	007	6	69
11645557	JC212655	04/01/2018 12:01:00 AM	083XX S VERNON AVE	1153	DECEPTIVE PRACTICE	FINANCIAL IDENTITY THEFT OVER $ 300	RESIDENCE	0	0	631	006	6	44
10225149	HY412228	09/05/2015 11:30:00 PM	003XX E 75TH ST	0820	THEFT	$500 AND UNDER	STREET	0	0	323	003	6	69

RESULT

The crime data has been analyzed successfully.

PRACTICAL – 9

DATE: 11.10.2021

AIM – Write script how can you get the structured dataset from rdbms using the sqoop.

SOFTWARE USED

Hadoop 3.3.0

THEORY

Apache Sqoop is a tool in Hadoop ecosystem which is designed to transfer data between HDFS (Hadoop storage) and relational database servers like MySQL, Oracle RDB, SQLite, Teradata, Netezza, Postgres etc. Apache Sqoop imports data from relational databases to HDFS, and exports data from HDFS to relational databases. It efficiently transfers bulk data between Hadoop and external data stores such as enterprise data warehouses, relational databases, etc. Additionally, Sqoop is used to import data from external datastores into Hadoop ecosystem's tools like Hive & HBase.

When we submit our Job, it is mapped into Map Tasks which brings the chunk of data from HDFS. These chunks are exported to a structured data destination. Combining all these exported chunks of data, we receive the whole data at the destination, which in most cases is an RDBMS. Apache Sqoop just imports and exports the data; it does not perform any aggregations. Map job launch multiple mappers depending on the number defined by the user. For Sqoop import, each mapper task will be assigned with a part of data to be imported. Sqoop distributes the input data among the mappers equally to get high performance. Then each mapper creates a connection with the database using JDBC and fetches the part of data assigned by Sqoop and writes it into HDFS or Hive or HBase based on the arguments provided in the CLI.

SOURCE CODE

1.First of all create a table in RDBMS.

```
CREATE TABLE employee (
ID INT NOT NULL,
NAME VARCHAR (20) NOT NULL,
AGE INT NOT NULL,
ADDRESS CHAR (25),
PRIMARY KEY (ID)
);
show tables;
INSERT INTO employee (id, name, age, address) VALUES (101, 'Srinath', 27, 'London');
```

INSERT INTO employee (id, name, age, address) VALUES (102, 'Anne', 33, 'London');

INSERT INTO employee (id, name, age, address) VALUES (103, 'Banne', 27, 'London');

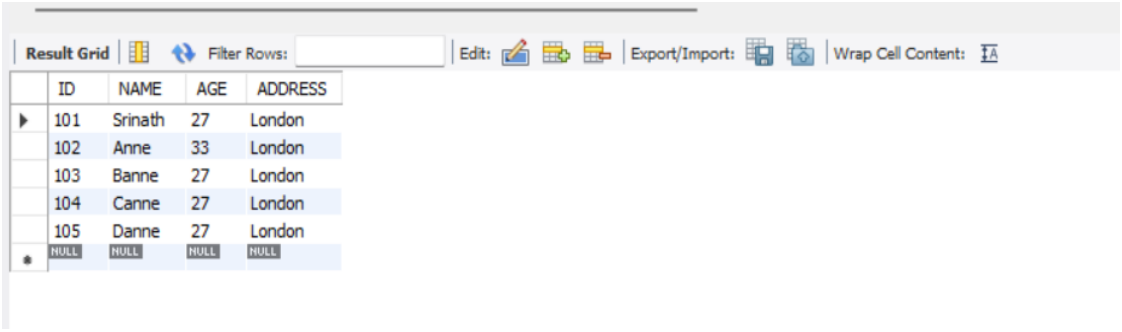
INSERT INTO employee (id, name, age, address) VALUES (104, 'Canne', 27, 'London');

INSERT INTO employee (id, name, age, address) VALUES (105, 'Danne', 27, 'London');

2. Import table using Sqoop.

sqoop import --connect jdbc: mysql://localhost/bdafile --username root --table employee

OUTPUT



ID	NAME	AGE	ADDRESS
101	Srinath	27	London
102	Anne	33	London
103	Banne	27	London
104	Canne	27	London
105	Danne	27	London
NULL	NULL	NULL	NULL

```
04/10/21 14:13:43 INFO mapreduce.Job: Counters: 31
        File System Counters
                FILE: Number of bytes read=0
                FILE: Number of bytes written=620704
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=464
                HDFS: Number of bytes written=13821993
                HDFS: Number of read operations=16
                HDFS: Number of large read operations=0
                HDFS: Number ofwrite operations=8
        Job Counters
                Killed map tasks=1
                Launched map tasks=5
                Other local map tasks=5
                Total time spent by all maps in occupied slots (ms)=217032
                Total time spent by all reduces in occupied slots (ms)=0
                Total time spent by all map tasks (ms) =217032
                Total vcore-milliseconds taken by all map tasks=217032
                Total megabyte-milliseconds taken by all map tasks=222240768
        Map-Reduce Framework
                Map input records=300024
```

RESULT

Successfully retrieved structured dataset from rdbms using the sqoop.

PRACTICAL – 10

DATE: 18.10.2021

AIM – Write the script to get the structured dataset from different sources using flume.

SOFTWARE USED

Hadoop 3.3.0

THEORY

Apache Flume is a distributed, reliable, and available system for efficiently collecting, aggregating, and moving large amounts of data from many different sources to a centralized data store, such as HDFS. Even though historically lots of use cases of Flume are involved with log data collection/aggregation, Flume can be used together with Kafka and turn itself into a real-time event processing pipeline. When we must collect and transfer unstructured data from various sources to HDFS or HBase then we use Apache Flume.

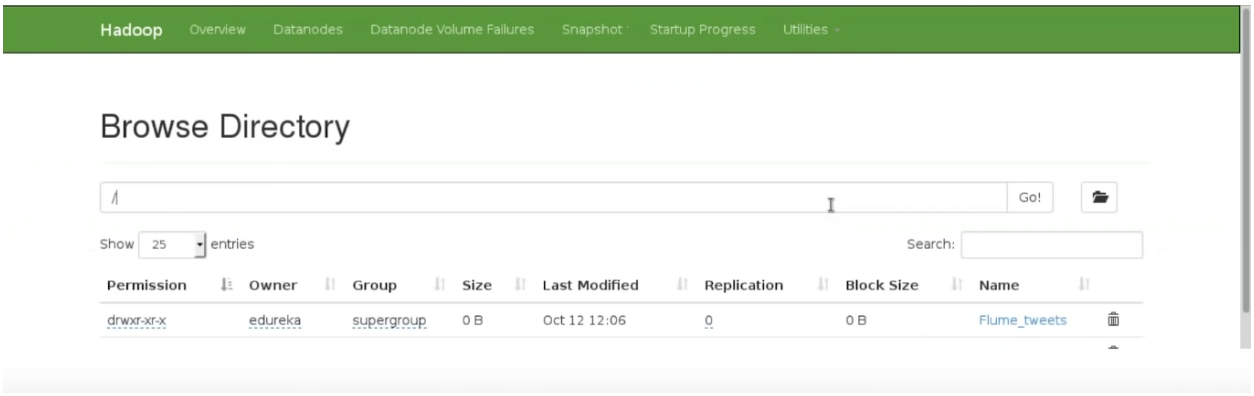
CODE: -

flume.conf file


```
hdoop@DeeJay-Zorin:~$ flume-ng agent --conf conf --conf-file /home/hdoop/flume.conf --name TwitterAgent -Dflume.root.logger=DEBUG.console
Info: Including Hadoop libraries found via (/usr/lib/hadoop-2.8.1/bin/hadoop) for HDFS access
Info: Including Hive libraries found via () for Hive access
+ exec /usr/lib/jvm/jdk1.8.0_144/bin/java -Xmx20m -Dflume.root.logger=DEBUG.console -cp 'conf:/usr/lib/apache-flume-1.7.0-bin/lib/*:/usr/lib/hadoop-2.
8.1/etc/hadoop:/usr/lib/hadoop-2.8.1/share/hadoop/common/lib/*:/usr/lib/hadoop-2.8.1/share/hadoop/common/*:/usr/lib/hadoop-2.8.1/share/hadoop/hdfs:/us
r/lib/hadoop-2.8.1/share/hadoop/hdfs/lib/*:/usr/lib/hadoop-2.8.1/share/hadoop/hdfs/*:/usr/lib/hadoop-2.8.1/share/hadoop/yarn/lib/*:/usr/lib/hadoop-2.8
.1/share/hadoop/yarn/*:/usr/lib/hadoop-2.8.1/share/hadoop/mapreduce/lib/*:/usr/lib/hadoop-2.8.1/share/hadoop/mapreduce/*:/usr/lib/hadoop-2.8.1/contrib
/capacity-scheduler/*.jar:/lib/*' -Djava.library.path=:/usr/lib/hadoop-2.8.1/lib/native org.apache.flume.node.Application --conf-file /home/edureka/fl
ume.conf --name TwitterAgent
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/apache-flume-1.7.0-bin/lib/slf4j-log4j12-1.6.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/hadoop-2.8.1/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
```

```
        at org.apache.hadoop.io.retry.RetryInvocationHandler$Call.invoke(RetryInvocationHandler.java:155)
        at org.apache.hadoop.io.retry.RetryInvocationHandler$Call.invokeOnce(RetryInvocationHandler.java:95)
        at org.apache.hadoop.io.retry.RetryInvocationHandler.invoke(RetryInvocationHandler.java:335)
        at com.sun.proxy.$Proxy14.create(Unknown Source)
        at org.apache.hadoop.hdfs.DFSOutputStream.newStreamForCreate(DFSOutputStream.java:246)
        at org.apache.hadoop.hdfs.DFSClient.create(DFSClient.java:1257)
        at org.apache.hadoop.hdfs.DFSClient.create(DFSClient.java:1199)
        at org.apache.hadoop.hdfs.DistributedFileSystem$8.doCall(DistributedFileSystem.java:472)
        at org.apache.hadoop.hdfs.DistributedFileSystem$8.doCall(DistributedFileSystem.java:469)
        at org.apache.hadoop.fs.FileSystemLinkResolver.resolve(FileSystemLinkResolver.java:81)
        at org.apache.hadoop.hdfs.DistributedFileSystem.create(DistributedFileSystem.java:469)
        at org.apache.hadoop.hdfs.DistributedFileSystem.create(DistributedFileSystem.java:410)
        at org.apache.hadoop.fs.FileSystem.create(FileSystem.java:928)
        at org.apache.hadoop.fs.FileSystem.create(FileSystem.java:909)
        at org.apache.hadoop.fs.FileSystem.create(FileSystem.java:806)
        at org.apache.hadoop.fs.FileSystem.create(FileSystem.java:795)
        at org.apache.flume.sink.hdfs.HDFSDataStream.doOpen(HDFSDataStream.java:81)
        at org.apache.flume.sink.hdfs.HDFSDataStream.open(HDFSDataStream.java:108)
        at org.apache.flume.sink.hdfs.BucketWriter$1.call(BucketWriter.java:242)
        at org.apache.flume.sink.hdfs.BucketWriter$1.call(BucketWriter.java:232)
        at org.apache.flume.sink.hdfs.BucketWriter$9$1.run(BucketWriter.java:668)
        at org.apache.flume.auth.SimpleAuthenticator.execute(SimpleAuthenticator.java:50)
        at org.apache.flume.sink.hdfs.BucketWriter$9.call(BucketWriter.java:665)
        at java.util.concurrent.FutureTask.run(FutureTask.java:266)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
```

OUTPUT: -



RESULT

Successfully retrieved the structured dataset from different sources using flume.

PRACTICAL – 11

DATE: 25.10.2021

AIM – Create a database, table, and insert data into the Hbase.

SOFTWARE USED

Hadoop 3.3.0

THEORY

HBase is a column-oriented non-relational database management system that runs on top of Hadoop Distributed File System (HDFS). HBase provides a fault-tolerant way of storing sparse data sets, which are common in many big data use cases. It is well suited for real-time data processing or random read/write access to large volumes of data. Unlike relational database systems, HBase does not support a structured query language like SQL; in fact, HBase isn't a relational data store at all. HBase applications are written in Java™ much like a typical Apache MapReduce application. HBase does support writing applications in Apache Avro, REST and Thrift. An HBase system is designed to scale linearly. It comprises a set of standard tables with rows and columns, much like a traditional database. Each table must have an element defined as a primary key, and all access attempts to HBase tables must use this primary key. Avro, as a component, supports a rich set of primitive data types including: numeric, binary data and strings; and a number of complex types including arrays, maps, enumerations and records. A sort order can also be defined for the data. HBase relies on ZooKeeper for high-performance coordination. ZooKeeper is built into HBase, but if you're running a production cluster, it's suggested that you have a dedicated ZooKeeper cluster that's integrated with your HBase cluster. HBase works well with Hive, a query engine for batch processing of big data, to enable fault-tolerant big data applications.

STEPS

1. Create a table schema.
2. Define Column Family
3. Insert data
4. Read data

OUTPUT

```
hbase(main):002:0> create 'emp', 'personal data', 'professional data'

hbase(main):005:0> put 'emp','1','personal data:name','raju'
hbase(main):006:0> put 'emp','1','personal data:city','hyderabad'
hbase(main):007:0> put 'emp','1','professional data:designation','manager'
hbase(main):007:0> put 'emp','1','professional data:salary','50000'

hbase(main):012:0> get 'emp', '1'

   COLUMN                        CELL
personal : city timestamp = 1417521848375, value = hyderabad
personal : name timestamp = 1417521785385, value = ramu
professional: designation timestamp = 1417521885277, value = manager
professional: salary timestamp = 1417521903862, value = 50000
```

RESULT

Successfully created database, table and insert data into the Hbase.

PRACTICAL – 12

DATE: 01.11.2021

AIM – Write an script to run an application using oozie.

SOFTWARE USED

Hadoop 3.3.0

THEORY

Apache Oozie is the tool in which all sort of programs can be pipelined in a desired order to work in Hadoop's distributed environment. Oozie also provides a mechanism to run the job at a given schedule.

This tutorial explains the scheduler system to run and manage Hadoop jobs called Apache Oozie. It is tightly integrated with Hadoop stack supporting various Hadoop jobs like Hive, Pig, Sqoop, as well as system specific jobs like Java and Shell.

This tutorial explores the fundamentals of Apache Oozie like workflow, coordinator, bundle and property file along with some examples. By the end of this tutorial, you will have enough understanding on scheduling and running Oozie jobs on Hadoop cluster in a distributed environment.

CODE

workflow.xml file

```
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License.
-->
<workflow-app xmlns="uri:oozie:workflow:0.2" name="map-reduce-wf">
    <start to="mr-node"/>
    <action name="mr-node">
        <map-reduce>
            <job-tracker>${jobTracker}</job-tracker>
            <name-node>${nameNode}</name-node>
            <prepare>
                <delete path="${nameNode}/user/${wf:user()}/${examplesRoot}/output-data/${outputDir
"/>
            </prepare>
            <configuration>
                <property>
                    <name>mapred.job.queue.name</name>
                    <value>${queueName}</value>
                </property>
                <property>
                    <name>mapred.mapper.class</name>
                    <value>org.apache.oozie.example.SampleMapper</value>
                </property>
                <property>
                    <name>mapred.reducer.class</name>
```

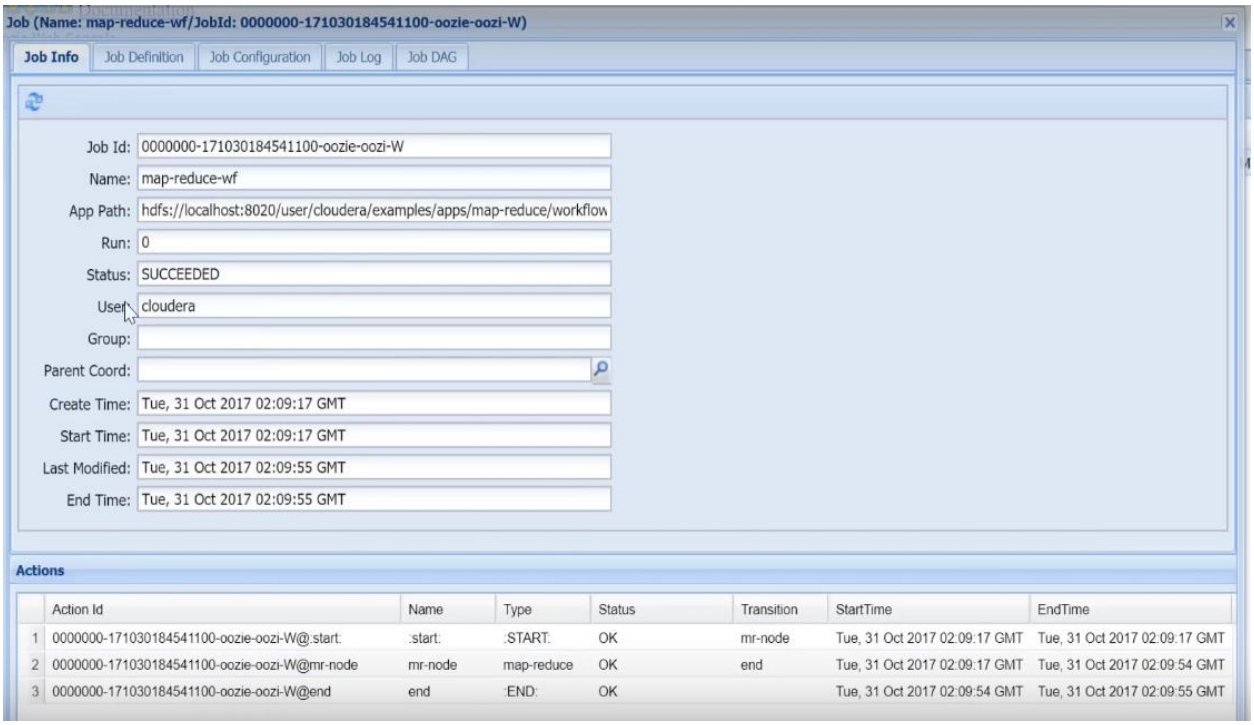
```
            <property>
                <name>mapred.reducer.class</name>
                <value>org.apache.oozie.example.SampleReducer</value>
            </property>
            <property>
                <name>mapred.map.tasks</name>
                <value>1</value>
            </property>
            <property>
                <name>mapred.input.dir</name>
                <value>/user/${wf:user()}/${examplesRoot}/input-data/text</value>
            </property>
            <property>
                <name>mapred.output.dir</name>
                <value>/user/${wf:user()}/${examplesRoot}/output-data/${outputDir}</value>
            </property>
        </configuration>
    </map-reduce>
    <ok to="end"/>
    <error to="fail"/>
</action>
<kill name="fail">
    <message>Map/Reduce failed, error message[${wf:errorMessage(wf:lastErrorNode())}]</message>
</kill>
<end name="end"/>
```

job.properties file

```
#
# Licensed to the Apache Software Foundation (ASF) under one
# or more contributor license agreements.  See the NOTICE file
# distributed with this work for additional information
# regarding copyright ownership.  The ASF licenses this file
# to you under the Apache License, Version 2.0 (the
# "License"); you may not use this file except in compliance
# with the License.  You may obtain a copy of the License at
#
#      http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#

nameNode=hdfs://localhost:8020
jobTracker=localhost:8032
queueName=default
examplesRoot=examples

oozie.wf.application.path=${nameNode}/user/${user.name}/${examplesRoot}/apps/map-reduce/workflow
l
outputDir=map-reduce
```

OUTPUT: -

```
[cloudera@quickstart oozie]$ oozie job -oozie http://localhost:11000/oozie -config examples/apps/ma
p-reduce/job.properties -run
job: 0000000-171030184541100-oozie-oozi-W
[cloudera@quickstart oozie]$
```



RESULT

Successfully to run an application using oozie.