PROGRAM 1

**AIM: Understanding and using basic HDFS commands**

**Hadoop HDFS** is a distributed file system which provides redundant storage space for files having huge sizes. It is used for storing files which are in the range of terabytes to petabytes. Command Line is one of the simplest interface to HDFS. Below are the basic HDFS File System Commands which are similar to Linux file system commands. Once the hadoop daemons are started running, HDFS file system is ready and file system operations like creating directories, moving files, deleting files, reading files and listing directories.

1. **mkdir:** To create a directory in hadoop file system. Syntax: hdfs fs -mkdir <folder-name>

Example: hdfs fs -mkdir /hello

1. **ls:** This command is used to list all the files and directories. Syntax: hdfs fs -ls <path>

Example: hdfs dfs -ls /

1. **put:** To copy files/folders from local file system to hdfs store. Syntax: hdfs fs -put <local file path> <dest present on hdfs> Example: hdfs fs -put ../Desktop/file.txt /hello
2. **get:** To copy files/folders from hdfs store to local file system. Syntax: hdfs fs -get <srcfile on hdfs> <local file dest> Example: hdfs fs -get /hello/file.txt ..Desktop/folder
3. **cat:** To print file contents. Syntax: hdfs fs -put <path>

Example: hdfs fs -cat /hello/file.txt

1. **cp:** This command is used to copy files within hdfs. Syntax: hdfs fs -cp <src on hdfs> <dest on hdfs> Example: hdfs -cp /hello /hello1
2. **mv:** This command is used to move files within hdfs. Syntax: hdfs fs -mv <src on hdfs> <dest on hdfs> Example: hdfs -mv /hello/myfile.txt /hello1
3. **rm**: This command removes the file or empty directory identified by path. Syntax: hdfs fs -rm <path>

Example: hdfs fs -rm /hello/file.txt

1. **getmerge:** This command retrieves all files that match the path src in HDFS, and copies them to a single, merged file in the local file system identified by localDest.

Syntax: hdfs fs -getmerge <src> <localDest> Example:

1. **setrep**: This command is used to change the replication factor of a file/directory in HDFS. By default it is 3 for anything which is stored in HDFS (as set in hdfs core- site.xml).

Syntax: hadoop fs -setrep <rep\_factor> <path> Example: hadoop fs -setrep 6 hello/sample

1. **touchz:** It creates an empty file. Syntax: hdfs fs -touchz <file-path>

Example: hdfs fs -touchz /hello/myfile.txt

1. **test:** Returns 1 if path exists; has zero length; or is a directory or 0 otherwise. Options Description

-d Check whether the path given by the user is a directory or not, return 0 if it is a directory.

-e Check whether the path given by the user exists or not, return 0 if the path exists.

-f Check whether the path given by the user is a file or not, return 0 if it is a file.

-s Check if the path is not empty, return 0 if a path is not empty.

-r return 0 if the path exists and read permission is granted

-w return 0 if the path exists and write permission is granted

-z Checks whether the file size is 0 byte or not, return 0 if the file is of 0 bytes. Syntax: hdfs fs -test -[ezd] <path>

Example: hdfs fs -test -d /sample

1. **appendToFile:** The HDFS fs shell command appendToFile appends the content of single or multiple local files specified in the localsrc to the provided destination file on the HDFS.

Syntax: hadoop fs -appendToFile <localsrc> <dest> Example: hadoop fs -appendToFile /hello/file /hello1/sample

1. **df:** The Hadoop fs shell command df shows the capacity, size, and free space available on the HDFS file system.The -h option formats the file size in the human-readable format.

Syntax: hadoop fs -df [-h] <path> Example: hadoop fs -df -h

1. **du**: This Hadoop fs shell command du prints a summary of the amount of disk usage

of all files/directories in the path. Syntax: hdfs fs -du <dirName> Example: hdfs fs -du /hello

1. **count:** The Hadoop fs shell command count counts the number of files, directories, and bytes under the paths that matches the specified file pattern.

Options:

-q – shows quotas(quota is the hard limit on the number of names and amount of space used for individual directories)

-u – it limits output to show quotas and usage only

-h – shows sizes in a human-readable format

-v – shows header line

Syntax: hadoop fs -count [options] <path> Example: hadoop fs -count -v /

1. **chgrp:** The Hadoop fs shell command chgrp changes the group of the file specified in the path. The user must be the owner of the file or superuser. Sets group recursively if

-R is specified.

Syntax: hdfs fs -chgrp [-R] group <path>... Example: hdfs fs -chgrp newgroup /hello

1. **chmod:** Changes the file permissions associated with one or more objects identified by path. Performs changes recursively with -R. Mode is a 3-digit octal mode, or

{augo}+/-{rwxX}. Assumes if no scope is specified and does not apply an umask. Syntax: hdfs fs -chmod [-R] mode,mode,.. <path>

Example: hdfs fs -chmod 744 /sample/file.txt

1. **chown**: Sets the owning user and/or group for files or directories identified by path....

Sets owner recursively if -R is specified.

Syntax: hdfs fs -chown [-R] [owner][:[group]] <path> Example: hadoop fs -chown newsample /hello

PROGRAM 2

**AIM: Word count application using Mapper-Reducer on single node cluster.**

In Hadoop, MapReduce is a computation that decomposes large manipulation jobs into individual tasks that can be executed in parallel cross a cluster of servers. The results of tasks can be joined together to compute final results.MapReduce works by breaking the processing into two phases: the map phase and the reduce phase. Each phase has key-value pairs as input and output, the types of which may be chosen by the programmer.

**MapReduce** is the core component of [**Hadoop**](http://data-flair.training/blogs/hadoop-tutorial-for-beginners/) that process huge amount of data in parallel by dividing the work into a set of independent tasks. In MapReduce data flow in step by step from Mapper to Reducer.

**Input Files:** The data for a MapReduce task is stored in input files, and input files typically lives in HDFS. The format of these files is arbitrary, while line-based log files and binary format can also be used.

**InputFormat** InputFormat defines how these input files are split and read. It selects the files or other objects that are used for input. InputFormat creates InputSplit. Learn MapReduce InputFormat in detail.

**InputSplits:** It is created by InputFormat, logically represent the data which will be processed by an individual Mapper (We will understand mapper below). One map task is created for each split; thus the number of map tasks will be equal to the number of InputSplits. The split is divided into records and each record will be processed by the mapper. Learn MapReduce InputSplit in detail.

**RecordReader:** It communicates with the InputSplit in Hadoop MapReduce and converts the data into key-value pairs suitable for reading by the mapper. By default, it uses TextInputFormat for converting data into a key-value pair. RecordReader communicates with the InputSplit until the file reading is not completed. It assigns byte offset (unique number) to each line present in the file. Further, these key-value pairs are sent to the mapper for further processing.

Mapper:

It processes each input record (from RecordReader) and generates new key-value pair, and this key-value pair generated by Mapper is completely different from the input pair. The output of Mapper is also known as intermediate output which is written to the local disk. The output of the Mapper is not stored on HDFS as this is temporary data and writing on HDFS will create unnecessary copies (also HDFS is a high latency system). Mappers output is passed to the combiner for further process

MAP REDUCE FLOW CHART

**Combiner**

The combiner is also known as ‘Mini-reducer’. Hadoop MapReduce Combiner performs local aggregation on the mappers’ output, which helps to minimize the data transfer between mapper and reducer (we will see reducer below). Once the combiner functionality is executed, the output is then passed to the partitioner for further work. Learn MapReduce Combiner in detail.

Partitioner

Hadoop MapReduce, Partitioner comes into the picture if we are working on more than one reducer (for one reducer partitioner is not used).Partitioner takes the output from combiners and performs partitioning. Partitioning of output takes place on the basis of the key and then sorted. By hash function, key (or a subset of the key) is used to derive the partition. According to the key value in MapReduce, each combiner output is partitioned, and a record having the same key value goes into the same partition, and then each partition is sent to a reducer. Partitioning allows even distribution of the map output over the reducer.

Shuffling and Sorting

Now, the output is Shuffled to the reduce node (which is a normal slave node but reduce phase will run here hence called as reducer node). The shuffling is the physical movement of the data which is done over the network. Once all the mappers are finished and their output is shuffled on the reducer nodes, then this intermediate output is merged and sorted, which is then provided as input to reduce phase.

Reducer

It takes the set of intermediate key-value pairs produced by the mappers as the input and then runs a reducer function on each of them to generate the output. The output of the reducer is the final output, which is stored in HDFS. Follow this link to learn about Reducer in detail.

SHUFFLE PHASE

**RecordWriter**

It writes these output key-value pair from the Reducer phase to the output files.

OutputFormat

The way these output key-value pairs are written in output files by RecordWriter is determined by the OutputFormat. OutputFormat instances provided by the Hadoop are used to write files in HDFS or on the local disk. Thus the final output of reducer is written on HDFS by OutputFormat instances.Hence, in this manner, a Hadoop MapReduce works over the cluster.

**Steps for execution in python:**

**Step 1:** Open terminal and create a text file and move it to the hadoop store using the following command.

Make a directory

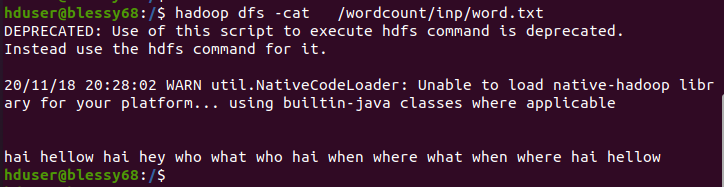
$ hadoop fs -mkdir /wordcount

$hadoop fs -mkdir /wordput/inp

Move the text file onto the hadoop store.

$ hadoop fs -put /home /Desktop/Lab/word.txt

**Step 2:** Type in the mapper code and reducer code.



**my\_mapper.py** #!/usr/bin/python import sys;

for line in sys.stdin: line = line.strip() words = line.split() for word in words:

print ('%s-%s' %(word,1)) **my\_reducer.py** #!/usr/bin/python

import sys;

from operator import itemgetter; prev\_word = None

prev\_count = 0 word = None

for line in sys.stdin: line = line.strip()

word,count = line.split('-',1) count=int(count)

if word == prev\_word:

prev\_count+=1;

else:

if prev\_word:

print('%s-%s' %(prev\_word,prev\_count)) prev\_word = word;

prev\_count = count;

print('%s-%s' %(prev\_word,prev\_count))

**Step 3:** Run the following command.

$ **hadoop jar** /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.9.0.jar **-file**

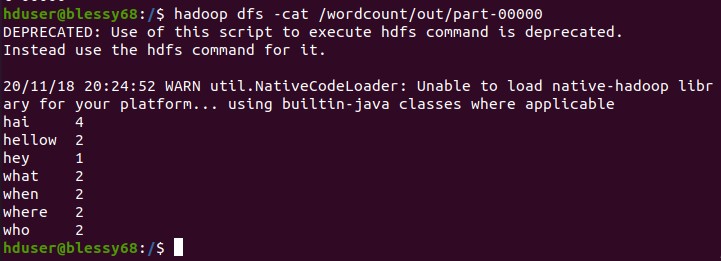
/home/Desktop/Lab/word.txt **-mapper** “python3 mapper.py” **-file** / home

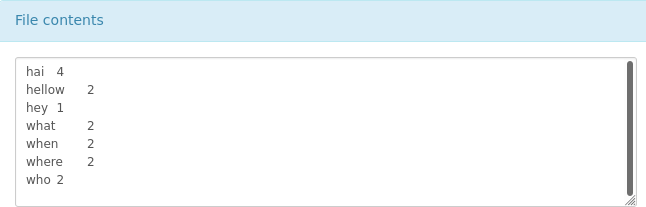
/Desktop/Lab/reducer.py **-reducer** “python3 reducer.py**” -input** /wordcount/inp **-output**

/wordcount/out

**Step 4:** Open result on the terminal Or you can see your result in Hadoop Web Interface http://localhost:50070/

Goto utilities> Browse file System> /wordcount/out

Output on the terminal



PROGRAM 3

**AIM: Analysis of weather dataset using Mapper-Reducer on single node cluster.**

MapReduce is a programming model for data processing. The model is simple, yet not too simple to express useful programs in. Hadoop can run MapReduce programs written in various languages

Data Format

The data is from the National Climatic Data Center (NCDC, [http://www](http://www/) .ncdc.noaa.gov/). The data is stored using a line-oriented ASCII format, in which each line is a record. The format supports a rich set of meteorological elements, many of which are optional or with variable data lengths. For simplicity, we shall focus on the basic elements, such as temperature, which are always present and are of fixed width.

Example:

0057

332130 # USAF weather station identifier 99999 # WBAN weather station identifier 19500101 # observation date

0300 # observation time

Since there are tens of thousands of weather stations, the whole dataset is made up of a large number of relatively small files. It’s generally easier and more efficient to process a smaller number of relatively large files, so the data was preprocessed so that each year’s readings were concatenated into a single file.The script loops through the compressed year files, first printing the year, and then processing each file using awk. The awk script extracts two fields from the data: the air temperature and the quality code.

Steps for execution in python:

**Step 1:** Open terminal and create a text file and move it to the hadoop store using the following command.

Make a directory

$ hadoop fs -mkdir /weather\_data

$hadoop fs -mkdir /weather\_data/input Move the text file onto the hadoop store.

$ hadoop fs -put /home /Desktop/Lab/weather/\* /weather/inp **Step 2:** Type in the mapper code and reducer code. **my\_mapper.py**

#!/usr/bin/python import sys l=list()

for line in sys.stdin:

line = line.strip() year=int(line[15:19]) temp=int(line[87:92]) l.append([year,temp])

for record in l:

print('%s %s' %(record[0],record[1]))

**my\_reducer.py** #!/usr/bin/python import sys dmax=dict() y\_list=list() dmin=dict()

for line in sys.stdin:

line = line.strip()

year,temp = map(int,line.split()) if year not in y\_list:

y\_list.append(year) dmax[year]=temp dmin[year]=temp

else:

if dmax[year]<temp:

dmax[year]=temp if dmin[year]>temp:

dmin[year]=temp

print(' ') print('year max\_temp min\_temp') print(' ')

for i in dmax.keys():

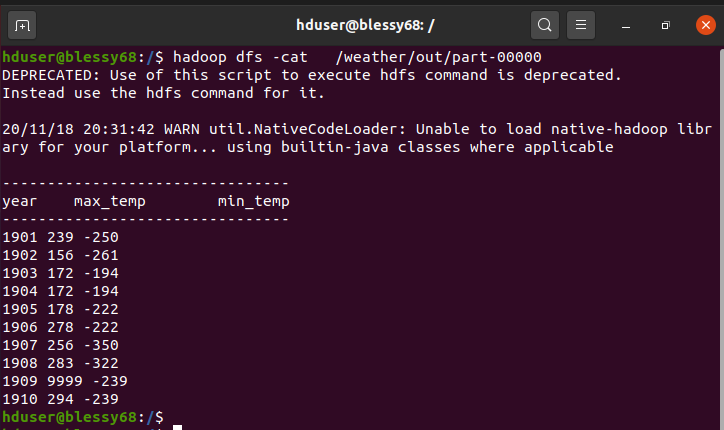
print('%s----------%s %s' %(i,dmax[i],dmin[i]))

**Step 3:** Run the following command.

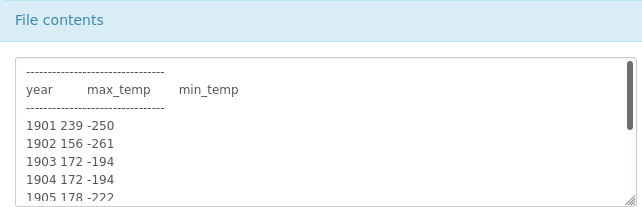
$ **hadoop jar** /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.9.0.jar **-file** mapperpath **-mapper** “python3 mapper.py” **-reducer path -reducer** “python3 reducer.py” **- input** /weather /inp/\* **-output** /weather/out

**Step 4**: Open result on the terminal Or you can see your result in Hadoop Web Interface http://localhost:50070/

Goto utilities> Browse file System> /weather\_data/OUT



Output on the terminal



Result from the Hadoop web interface

PROGRAM 4

**AIM: Web Log Analysis using Mapper-Reducer on single node cluster.**

Web logs are data that is generated by web servers for requests they receive. There are various web servers such as Apache, Nginx, Tomcat, and so on. Each web server logs data in a specific

format. The server access log records all requests processed by the server. Of course, storing the information in the access log is only the start of log management. The next step is to analyze this information to produce useful statistics. We can store log data in many different format but here we are going to use data from the Apache Web Server, which is in combined access logs. **Combined Log Format**

A typical configuration for the access log might look as follows.

LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{Useragent}i\"" combined CustomLog log/acces\_log combined

**Example:** 127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache\_pb.gif HTTP/1.0" 200 2326 "[http://www.example.com/start.html"](http://www.example.com/start.html) "Mozilla/4.08 [en] (Win98; I ;Nav)"

127.0.0.1 (%h)

This is the IP address of the client (remote host) which made the request to the server.

- (%l)

The "hyphen" in the output indicates that the requested piece of information is not available

frank (%u)

This is the userid of the person requesting the document as determined by HTTP authentication.

[10/Oct/2000:13:55:36 -0700] (%t)

The time that the server finished processing the request. The format is:

[day/month/year:hour:minute:second zone] day = 2\*digit

month = 3\*letter year = 4\*digit hour = 2\*digit minute = 2\*digit second = 2\*digit

zone = (`+' | `-') 4\*digit

"GET /apache\_pb.gif HTTP/1.0" (\"%r\")

The request line from the client is given in double quotes. The request line contains a great deal of useful information. First, the method used by the client is GET. Second, the client requested the resource /apache\_pb.gif, and third, the client used the protocol HTTP/1.0.

200 (%>s)

This is the status code that the server sends back to the client. This information is very valuable, because it reveals whether the request resulted in a successful response (codes beginning in 2),

a redirection (codes beginning in 3), an error caused by the client (codes beginning in 4), or an error in the server (codes beginning in 5).

2326 (%b)

The last entry indicates the size of the object returned to the client (in bytes), not including the response headers.

**"**[**http://www.example.com/start.html"**](http://www.example.com/start.html) **(\"%{Referer}i\")**

The "Referer" (sic) HTTP request header. This gives the site that the client reports having been referred from. (This should be the page that links to or includes /apache\_pb.gif). This is the page that is linked to this URL.

"Mozilla/4.08 [en] (Win98; I ;Nav)" (\"%{User-agent}i\")

This is the browser identification string.

Write a map reduce program that reads a weblog file to give URL address and their counts. (URL, Total Count)

**Steps for execution in python:**

**Step 1:** Open terminal and create a text file and move it to the hadoop store using the following command.

Make a directory

$ hadoop fs -mkdir /web\_log

$hadoop fs -mkdir /web\_log/inputdata Move the text file onto the hadoop store.

$ hadoop fs -put /home/Desktop/lab/web\_log/web\_log.txt /web\_log/inputdata

**Step 2:** Type in the mapper code and reducer code.

**my\_mapper.py** #!/usr/bin/python import sys; import re;

for line in sys.stdin: line = line.strip() words=line.split()

x=re.findall("http://",words[10]) if (x):

print('%s %s' %(words[10],1))

my\_reducer.py

#!/usr/bin/python import sys;

from operator import itemgetter; prev\_word = None

prev\_count = 0 word = None

for line in sys.stdin:

line = line.strip() word,count = line.split('\t',1) count=int(count)

if word == prev\_word:

prev\_count+=1;

else:

if prev\_word:

print('%s %s' %(prev\_word,prev\_count)) prev\_word = word;

prev\_count = count;

print('%s %s' %(prev\_word,prev\_count))

**Step 3:** Run the following command.

$ **hadoop jar** /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.9.0.jar **-file** /home

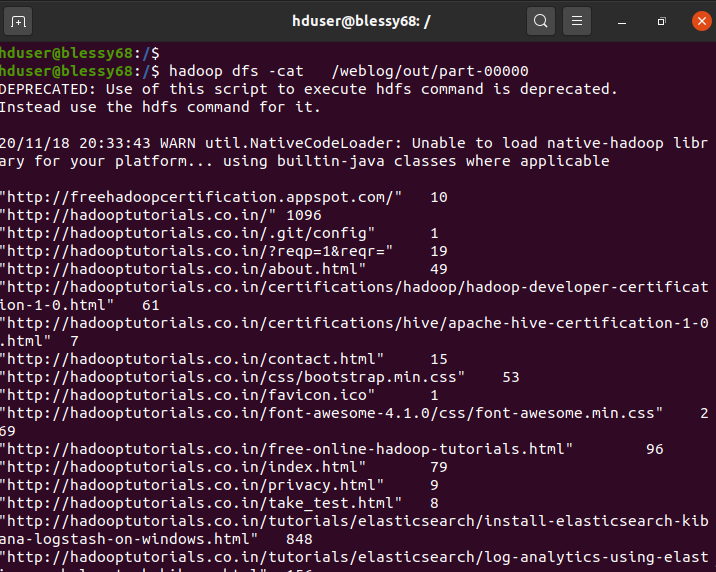
/Desktop/Lab/mapper.py **-mapper** “python3 mapper.py” **-file** / home/

/Desktop/Lab/reducer.py **-reducer** “python3 reducer.py” **-input** /weblog/inp **-output**

/weblog/out

**Step 4:** Open result on the terminal Or you can see your result in Hadoop Web Interface http://localhost:50070/

Goto utilities> Browse file System> /web\_log/out



Output

