MALNAD COLLEGE OF ENGINEERING

(An Autonomous Institute Affiliated to VTU, Belagavi)

HASSAN-573202, KARNATAKA, INDIA

MASTER OF TECHNOLOGY (M.Tech)

In

ARTIFICIAL INTELLIGENCE & DATA SCIENCE

Laboratory Manual

Course: Big Data Analytics Laboratory

Course Code: 25MCS27

Department of Computer Science and Engineering
Malnad College of Engineering
2025-2027

INDEX

1) Implement the following file management tasks in Hadoop:

i. Adding files and directories

ii. Retrieving files

iii. Deleting files

Hint: A typical Hadoop workflow creates data files (such as log files) elsewhere and copies them into HDFS using one of the above command line utilities.

i. Adding Files and Directories in HDFS

Create a Directory in HDFS

hdfs dfs -mkdir /user/data

Create Nested Directories

hdfs dfs -mkdir -p /user/data/logs/2026

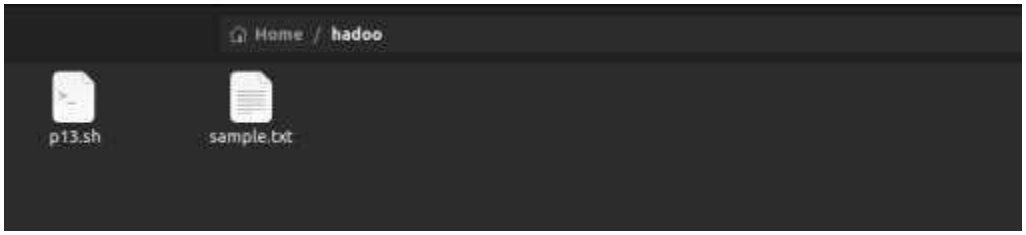
Copy File from Local System to HDFS

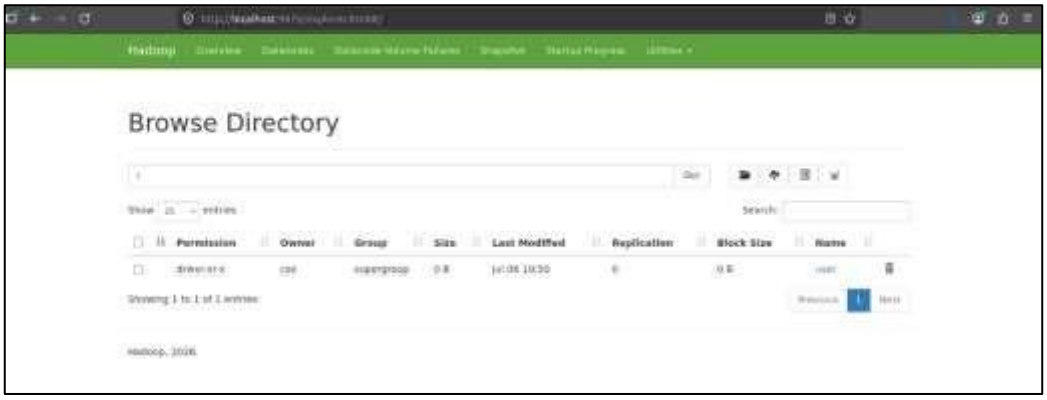
hdfs dfs -put sample.txt /user/data/

Alternative Command (copyFromLocal)

hdfs dfs -copyFromLocal sample.txt /user/data/

Execution Snapshots





ii. Retrieving Files from HDFS

View File Content

hdfs dfs -cat /user/data/sample.txt

Copy File from HDFS to Local System

hdfs dfs -get /user/data/sample.txt /home/user/

Alternative Command (copyToLocal)

hdfs dfs -copyToLocal /user/data/sample.txt /home/user/

hdfs dfs -copyToLocal /user/data/sample.txt ~/

List Files in Directory

hdfs dfs -ls /user/data/

Execution Snapshots

```
cse@cse-Vostro-3020-T:~$ hdfs dfs -cat /user/data/sample.txt
Helloi BigData
Hello Hadoop
cse@cse-Vostro-3020-T:~$
```

iii. Deleting Files and Directories in HDFS

Delete a File

hdfs dfs -rm /user/data/sample.txt

Delete Directory (Non-empty)

hdfs dfs -rm -r /user/data/

Force Delete (Skip Trash)

hdfs dfs -rm -r -skipTrash /user/data/

Execution Snapshots

```
cse@cse-Vostro-3020-T:~$ hdfs dfs -rm /user/data/sample.txt
Deleted /user/data/sample.txt
cse@cse-Vostro-3020-T:~$
```

```
cse@cse-Vostro-3020-T:~$ hdfs dfs -rm -r /user/data/
Deleted /user/data
cse@cse-Vostro-3020-T:~$
```

2) Run a basic word count Map Reduce program to understand Map Reduce Paradigm.

1. Prepare Input File
Create a sample text file:

echo "Hadoop MapReduce Hadoop Big Data" > input.txt

2. Start Hadoop Services
start-dfs.sh
start-yarn.sh

3. Create Input Directory in HDFS
hdfs dfs -mkdir /input

4. Upload File to HDFS
hdfs dfs -put input.txt /input

5. Run Word Count MapReduce Program

Hadoop provides a **built-in example JAR**:
hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-*.jar wordcount /*input*/input.txt /output

6. View Output

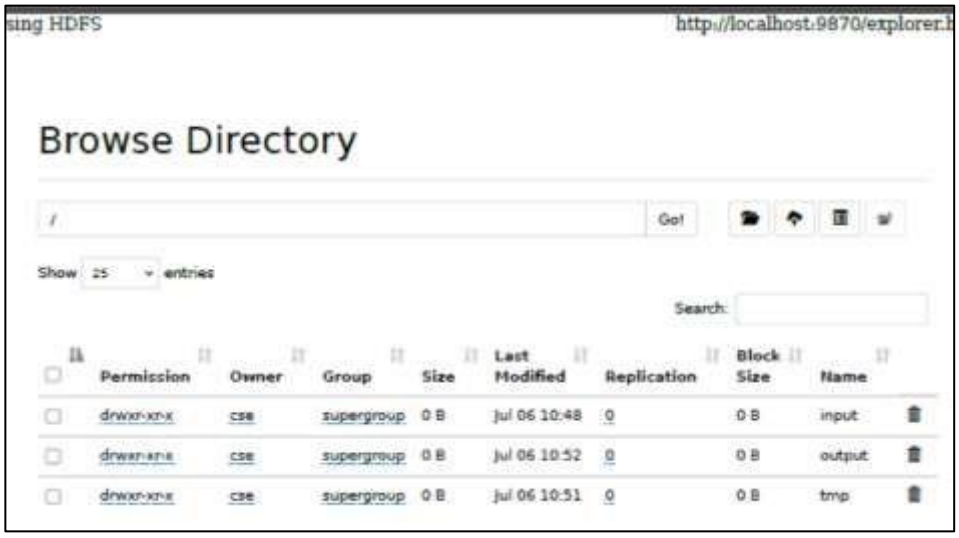
List output files:

hdfs dfs -ls /output

Display result:

hdfs dfs -cat /output/part-r-00000

Execution Snapshots




```
-rw-r--r--   1 cse sup
cse@cse:~$ hdfs dfs -c
Big     1
Data    1
Hadoop  2
MapReduce       1
cse@cse:~$
```

3) Write a Map Reduce program that mines weather data.
Hint: Weather sensors collecting data every hour at many locations across the globe gather alarge volume of log data, which is a good candidate for analysis with Map Reduce, since it issemi structured and record-oriented.

1) Input Data : input.txt

```
=======================
1950,0
1950,22
1950,15
1951,38
1951,42
1951,35
1952,30
1952,28

=======================
```

2) gedit mapper.py

```
=======================
#!/usr/bin/env python3
import sys

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

    if not line:
        continue

    year, temp = line.split(",")

    print(f"{year}\t{temp}")


=======================
```

3) gedit reducer.py

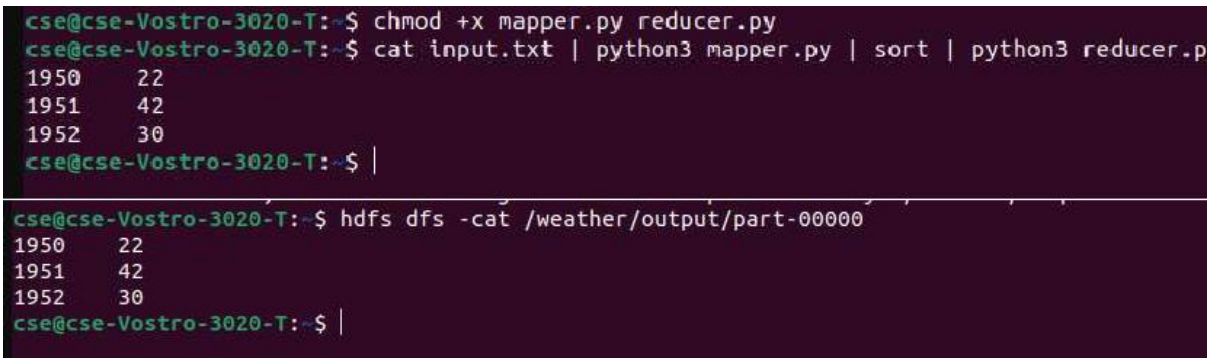
```
=======================
#!/usr/bin/env python3
import sys

current_year = None
max_temp = -9999

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

    year, temp = line.split("\t")
```

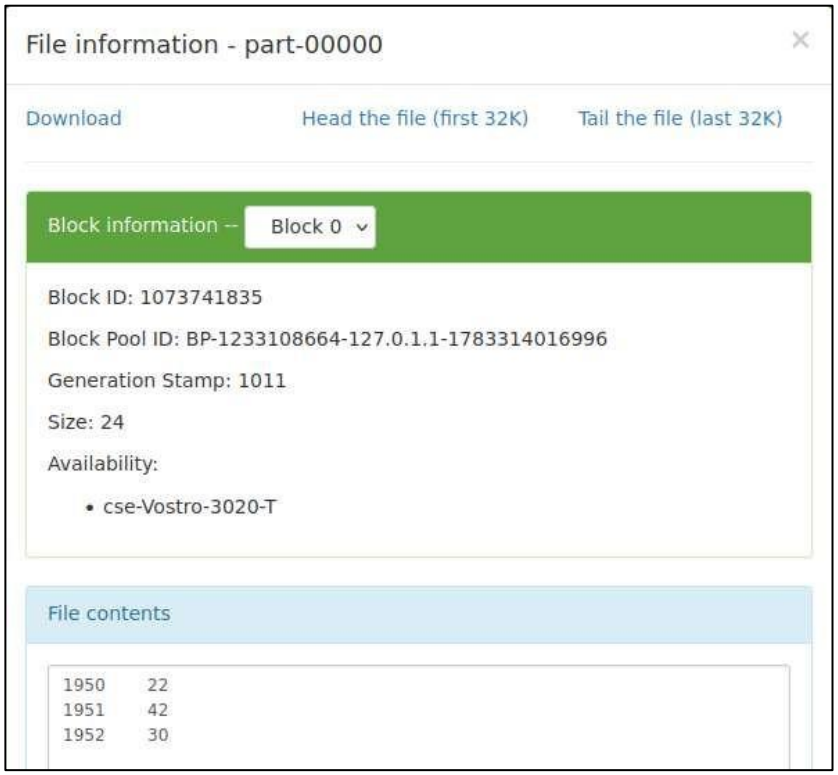
```
    temp = int(temp)

    if current_year == year:
        if temp > max_temp:
            max_temp = temp
    else:
        if current_year is not None:
            print(f"{current_year}\t{max_temp}")
        current_year = year
        max_temp = temp

if current_year is not None:
    print(f"{current_year}\t{max_temp}")
```

Execution:

```
chmod +x mapper.py reducer.py

cat input.txt | python3 mapper.py | sort | python3 reducer.py

hdfs dfs -mkdir /weather
hdfs dfs -put input.txt /weather/


hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming*.jar \
-input /weather/input.txt \
-output /weather/output \
-mapper mapper.py \
-reducer reducer.py \
-file mapper.py \
-file reducer.py

hdfs dfs -cat /weather/output/part-00000
```

Execution Snapshots





4) Implement matrix multiplication with Hadoop Map Reduce

Input File (mat.txt)

A,0,0,1
A,0,1,2
A,1,0,3
A,1,1,4
B,0,0,5
B,0,1,6
B,1,0,7
B,1,1,8

gedit maper.py

```
#!/usr/bin/env python3
import sys

n = 2

for line in sys.stdin:
    line = line.strip()
    parts = line.split(',')

    matrix = parts[0]
```

```
    row = int(parts[1])
    col = int(parts[2])
    val = int(parts[3])

    if matrix == 'A':
        for j in range(n):
            print(f"{row},{j}\tA,{col},{val}")
    else:
        for i in range(n):
            print(f"{i},{col}\tB,{row},{val}")
```

gedit reducer.py

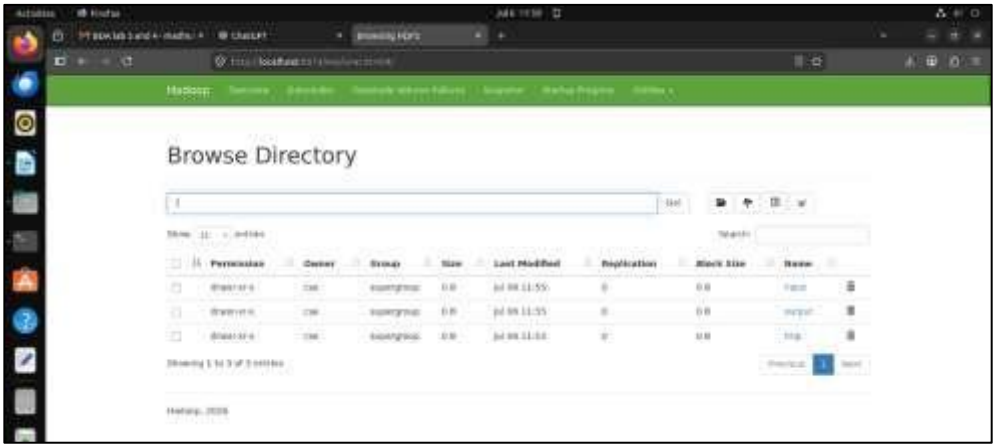
```
#!/usr/bin/env python3
import sys

current_key = None
values = []

def process(key, values):
    mapA = {}
    mapB = {}

    for val in values:
        parts = val.split(',')

        if parts[0] == 'A':
            mapA[int(parts[1])] = int(parts[2])
        else:
            mapB[int(parts[1])] = int(parts[2])

    result = 0

    for k in mapA:
        if k in mapB:
            result += mapA[k] * mapB[k]

    print(f"{key}\t{result}")

for line in sys.stdin:
    line = line.strip()
    key, value = line.split('\t')

    if current_key == key:
        values.append(value)
    else:
```

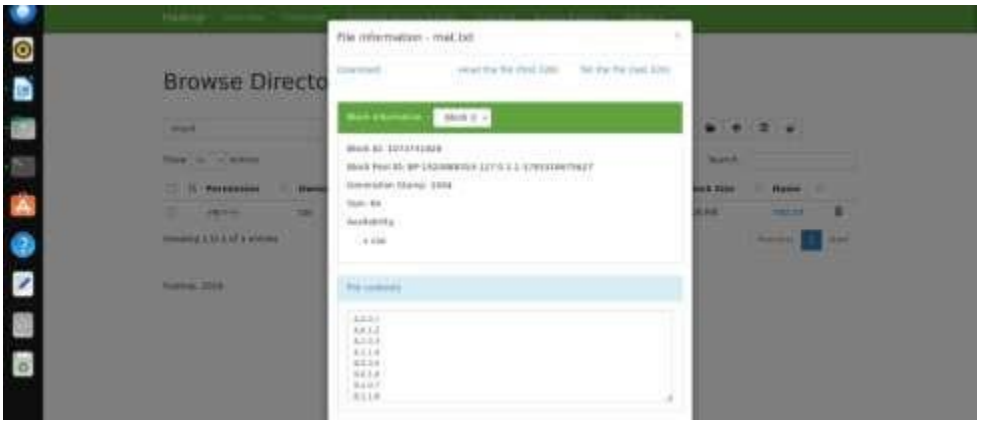
```
        if current_key:
            process(current_key, values)

        current_key = key
        values = [value]

if current_key:
    process(current_key, values)
```

Execution:

```
chmod +x maper.py reducer.py

hdfs dfs -mkdir /input

hdfs dfs -put mat.txt /input/

$hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming*.jar \
-input /input/mat.txt -output /output  -mapper maper.py -reducer reducer.py -file maper.py
-file reducer.py
```

Execution Snapshots





PIG INSTALLATION

Download pig 0.18.0
https://downloads.apache.org/pig/pig-0.18.0/

Extract
tar -xzf/home/cse/Downloads/pig-0.18.0.tar.gz

cd pig-0.18.0

nano ~/.bashrc

export PIG_HOME=$HOME/pig-0.18.0
export PATH=$PATH:$PIG_HOME/bin

source ~/.bashrc

cd ..

5) Run the Pig Latin Scripts to find Word Count.

gedit Sample.txt

Big data is powerful

Big data is easy

Pig is easy

gedit wordcount.pig

```
-- Load file
A = LOAD 'Sample.txt' USING TextLoader() AS (line:chararray);
-- Split lines into words
B = FOREACH A GENERATE FLATTEN(TOKENIZE(line)) AS word;
-- Group words
C = GROUP B BY word;
-- Count words
D = FOREACH C GENERATE group AS word, COUNT(B) AS count;
-- Store output
STORE D INTO 'output';
```

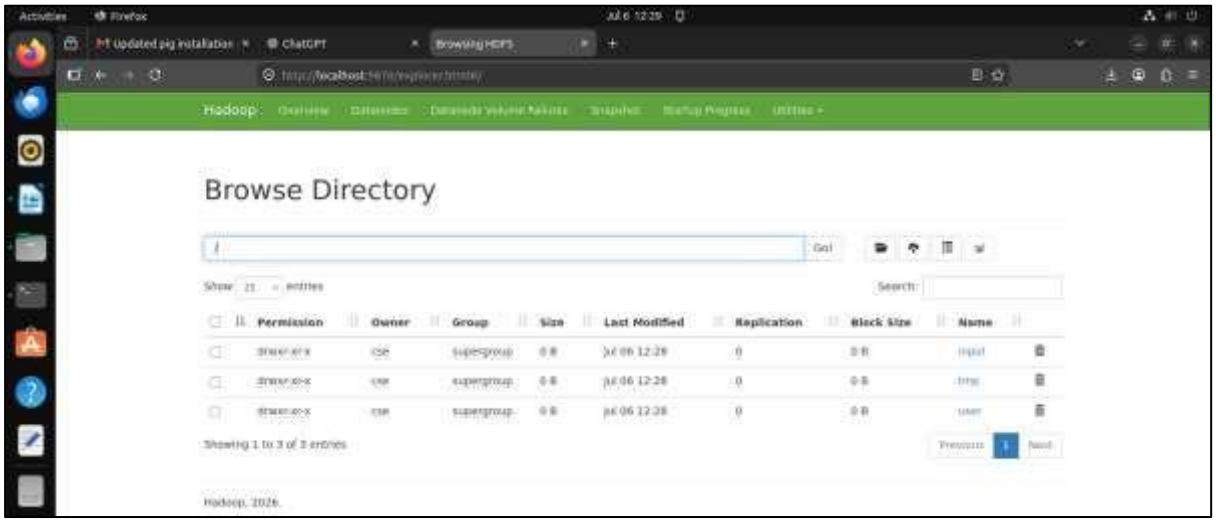
```
//to check locally
pig -x local wordcount.pig
```

```
cat output/part-r-00000
```

```
hdfs dfs -mkdir /input
hdfs dfs -put Sample.txt /input/
```

```
pig -x mapreduce wordcount.pig
```

Execution Snapshots




```
cse@cse:~$ cat output/part-r-00000
cat: output/part-r-00000: No such file or directory
cse@cse:~$ hdfs dfs -cat output/part-r-00000
is      3
Big     2
Pig     1
data    2
easy    2
powerful        1
cse@cse:~$
```

6) Run the Pig Latin Scripts to find a max temp for each and every year.

1. create a dataset weather.txt

1949 111 20
1949 112 25
1950 113 30
1950 114 28

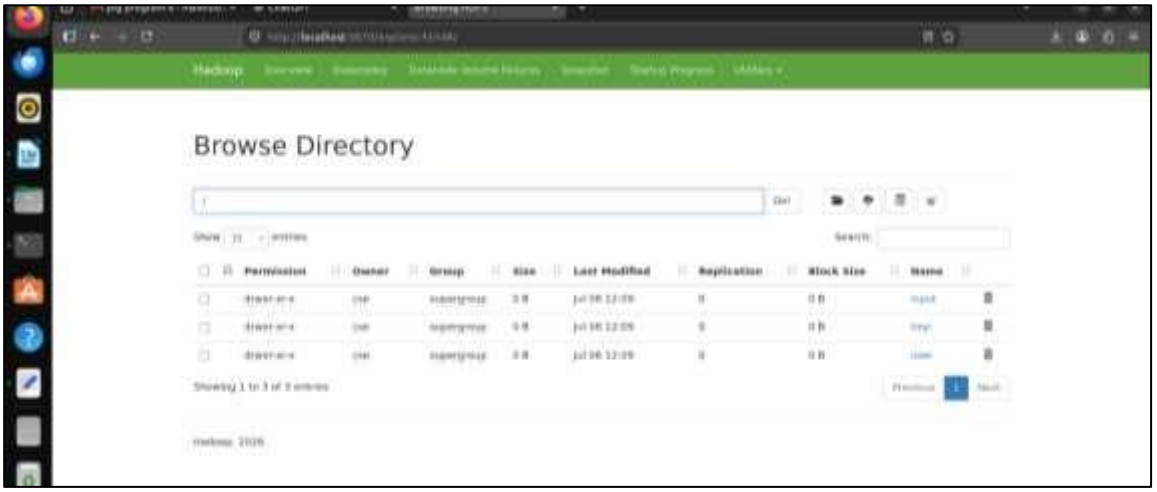
2. cretae pig file weather.pig

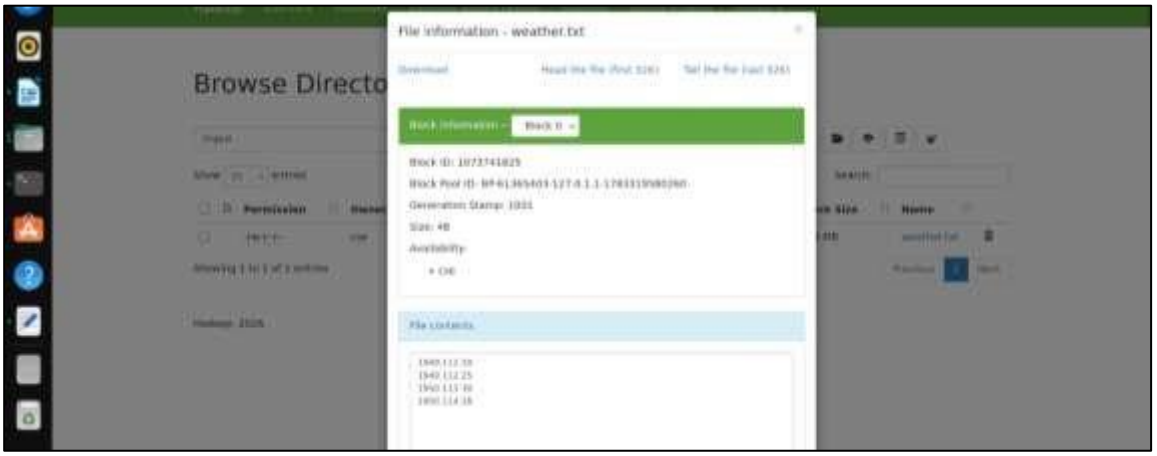
```
weather = LOAD '/input/weather.txt'
USING PigStorage(' ')
AS (year:int, station:int, temp:int);

grouped_data = GROUP weather BY year;

max_temp = FOREACH grouped_data GENERATE
   group AS year,
   MAX(weather.temp) AS max_temperature;

STORE max_temp INTO 'output';
```

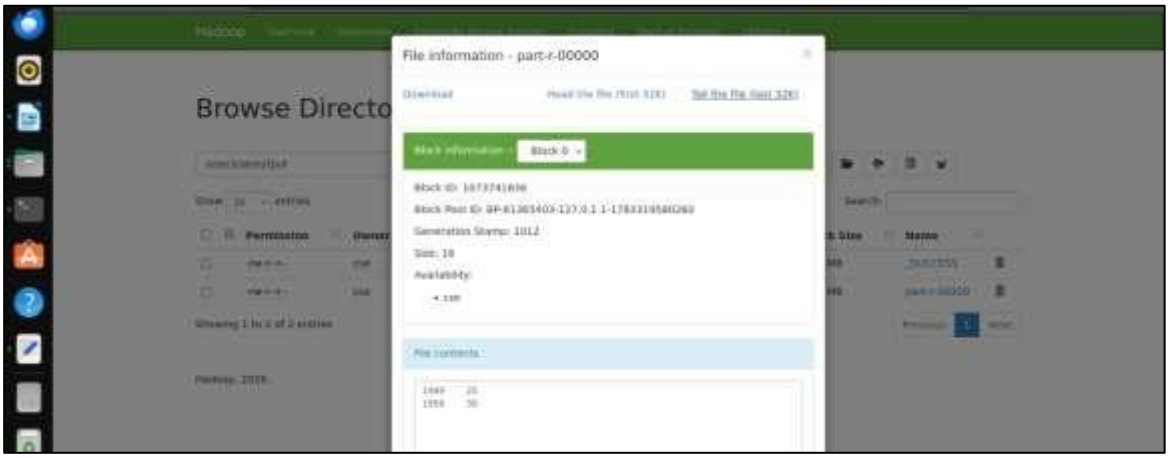
Execution:

```
hdfs dfs -mkdir  /input
hdfs dfs -put weather.txt /input

pig weather.pig

hdfs dfs -cat output/part-r-00000
```

Execution Snapshots








```
Completed, FinalApplicationStatus=SUCCEEDED. Redirecting to job history server
2026-07-06 12:11:19,162 [main] INFO  org.apache.hadoop.ipc.Client - Retrying connect to server: 0.0.0.0/0.0.0.0:10020. Already tried 0 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)
^Ccse@cse:~hdfs dfs -cat output/part-r-0000000
1949    25
1950    30
cse@cse:~$
```