TASK 1

a) Understanding and using basic HDFS commands

Apache Hadoop hadoop fs or hdfs dfs are file system commands to interact with HDFS, these commands are very similar to Unix Commands. Note that some Syntax and output formats may differ between Unix and HDFS Commands.
Hadoop is a open-source distributed framework that is used to store and process a large set of datasets. To store data, Hadoop uses HDFS, and to process data, it uses MapReduce & Yarn. In this article, I will mainly focus on Hadoop HDFS commands to interact with the files.

Hadoop provides two types of commands to interact with File System; hadoop fs or hdfs dfs. Major difference being hadoop commands are supported with multiple file systems like S3, Azure and many more.

Start Hadoop Services
In order to run hdfs dfs or hadoop fs commands, first, you need to start the Hadoop services by running the start-dfs.sh script from the Hadoop installation.

ubuntu@namenode:~$ start-dfs.sh
Starting namenodes on [namenode.socal.rr.com]
Starting datanodes
Starting secondary namenodes [namenode]
ubuntu@namenode:~$

Basic HDFS DFS Commands

Below are basic hdfs dfs or hadoop fs Commands.

COMMAND	DESCRIPTION
`-ls`	List files with permissions and other details
`-mkdir`	Creates a directory named path in HDFS
`-rm`	To Remove File or a Directory
`-rmr`	Removes the file that identified by path / Folder and subfolders
`-rmdir`	Delete a directory
`-put`	Upload a file / Folder from the local disk to HDFS
`-cat`	Display the contents for a file
`-du`	Shows the size of the file on hdfs.
`-dus`	Directory/file of total size
`-get`	Store file / Folder from HDFS to local file

`-getmerge`	Merge Multiple Files in an HDFS
`-count`	Count number of directory, number of files and file size
`-setrep`	Changes the replication factor of a file
`-mv`	HDFS Command to move files from source to destination
`-moveFromLocal`	Move file / Folder from local disk to HDFS
`-moveToLocal`	Move a File to HDFS from Local
`-cp`	Copy files from source to destination
`-tail`	Displays last kilobyte of the file
`-touch`	create, change and modify timestamps of a file
`-touchz`	Create a new file on HDFS with size 0 bytes
`-appendToFile`	Appends the content to the file which is present on HDF
`-copyFromLocal`	Copy file from local file system
`-copyToLocal`	Copy files from HDFS to local file system
`-usage`	Return the Help for Individual Command
`-checksum`	Returns the checksum information of a file
`-chgrp`	Change group association of files/change the group of a file or a path
`-chmod`	Change the permissions of a file
`-chown`	change the owner and group of a file
`-df`	Displays free space
`-head`	Displays first kilobyte of the file
`-Create Snapshots`	Create a snapshot of a snapshottable directory
`-Delete Snapshots`	Delete a snapshot of from a snapshottable directory
`-Rename Snapshots`	Rename a snapshot
`-expunge`	create new checkpoint
`-Stat`	Print statistics about the file/directory
`-truncate`	Truncate all files that match the specified file pattern to the specified length
`-find`	Find File Size in HDFS

ls – List Files and Folder

HDFS ls command is used to display the list of Files and Directories in HDFS, This ls command shows the files with permissions, user, group, and other details. For more information follow ls- List Files and Folder

```
$hadoop fs -ls
```

```
or
$hdfs dfs -ls
```

mkdir – Make Directory

HDFS mkdir command is used to create a directory in HDFS. By default, this directory would be owned by the user who is creating it. By specifying "/" at the beginning it creates a folder at root directory.

```
$hadoop fs -mkdir /directory-name
or
$hdfs dfs -mkdir /directory-name
```

rm – Remove File or Directory

HDFS rm command deletes a file and a directory from HDFS recursively.

```
$hadoop fs -rm /file-name
or
$hdfs dfs -rm /file-name
```

rmr – Remove Directory Recursively

Rmr command is used to deletes a file from Directory recursively, it is a very useful command when you want to delete a non-empty directory.

```
$hadoop fs -rmr /directory-name
or
$hdfs dfs -rmr /directory-name
```

rmdir – Delete a Directory

Rmdir command is used to removing directories only if they are empty.

```
$hadoop fs -rmdir /directory-name
or
$hdfs dfs -rmdir /directory-name
```

put – Upload a File to HDFS from Local

Copy file/folder from local disk to HDFS. On put command specifies the local-file-path where you wanted to copy from and then hdfs-file-path where you wanted to copy to on hdfs.

```
$ hadoop fs -put /local-file-path /hdfs-file-path
or
$ hdfs dfs -put /local-file-path /hdfs-file-path
```

cat – Displays the Content of the File

The cat command reads the specified file from HDFS and displays the content of the file on console or stdout.

```
$ hadoop fs -cat /hdfs-file-path
or
$ hdfs dfs -cat /hdfs-file-path
```

du – File Occupied in Disk

Du command is used to How much file Occupied in the disk. The field is the base size of the file or directory before replication.

```
$ hadoop fs -du  /hdfs-file-path
or
$ hdfs dfs -du  /hdfs-file-path
```

dus – Directory/file of the total size

Dus command is used to will give the total size of directory/file.

```
$ hadoop fs -dus  /hdfs-directory
or
$ hdfs dfs -dus  /hdfs-directory
```

get – Copy the File from HDFS to Local

Get command is used to store filess from HDFS to the local file. HDFS file gets the local machine.

```
$ hadoop fs -get /local-file-path /hdfs-file-path
or
$ hdfs dfs -get /local-file-path /hdfs-file-path
```

getmerge – Merge Multiple Files in an HDFS

If you have multiple files in an HDFS, use -getmerge option command. All these multiple files merged into one single file and downloads to local file system.

```
$ hadoop fs -getmerge [-nl] /source /local-destination
or
$ hdfs dfs -getmerge [-nl] /source /local-destination
```

count – Number of Directory

The count command is used to count a number of directories, a number of files, and file size on HDFS.

```
$ hadoop fs -count /hdfs-file-path
or
$ hdfs dfs -count /hdfs-file-path
```

mv – Moves Files from Source to Destination

MV (move) command is used to move files from one location to another location in HDFS. Move command allows multiple sources as well in which case the destination needs to be a director.

```
$ hadoop fs -mv /local-file-path /hdfs-file-path
or
$ hdfs dfs -mv /local-file-path /hdfs-file-path
```

moveFromLocal – Move file / Folder from Local disk to HDFS

Similar to the put command, `moveFromLocal` moves the file or source from the local file path to the destination in the HDFS file path. After this command, you will not find the file on the local file system.

```
$ hadoop fs -moveFromLocal /local-file-path /hdfs-file-path
or
$ hdfs dfs -moveFromLocal /local-file-path /hdfs-file-path
```

moveToLocal – Move a File to HDFS from Local

Similar to the get command, moveToLocal moves the file or source from the HDFS file path to the destination in the local file path.

```
```

```
$ hadoop fs -moveToLocal /hdfs-file-path /local-file-path
or
$ hdfs dfs -moveToLocal /hdfs-file-path /local-file-path
```

Cp – Copy Files from Source to Destination

Copy File-one location to another location in HDFS. Copy files from source to destination, Copy command allows multiple sources as well in which case the destination must be a directory.

```
$ hadoop fs -cp /local-file-path /hdfs-file-path
or
$ hdfs dfs -cp /local-file-path /hdfs-file-path
```

setrep – Changes the Replication Factor of a File

This HDFS command is used to change the replication factor of a file. If the path is a directory then the command recursively changes the replication factor of all files under the directory tree rooted at the path.

```
$ hadoop fs -setrep /number /file-name
or
$ hdfs dfs -setrep /number /file-name
```

tail – Displays Last Kilobyte of the File

Tail command is used to Display last kilobyte of the file to stdout.

```
$ hadoop fs -tail /hdfs-file-path
or
$ hdfs dfs -tail /hdfs-file-path
```

touch – Create and Modify Timestamps of a File

It is used to create a file without any content. The file created using the touch command is empty. updates the access and modification times of the file specified by the URI to the current time, the file does not exist then a zero-length file is created at URI with the current time as the timestamp of that URI.

```
$ hadoop fs -touch /hdfs-file-path
or
```

```
$ hdfs dfs -touch /hdfs-file-path
```

touchz – Create a File of zero Length

Create a new file on HDFS with size 0 bytes. create a file of zero length, an error is returned if the file exists with non-zero length.

```
$ hadoop fs -touchz /hdfs-file-path
or
$ hdfs dfs -touchz /hdfs-file-path
```

appendToFile – Appends the Content to the File

Appends the content to the file which is present on HDFS. Append single source. or multiple sources from the local file system to the destination file system. this command appends the contents of all the given local files to the provided destination file on the HDFS filesystem.

```
$ hadoop fs -appendToFile /hdfs-file-path
or
$ hdfs dfs -appendToFile /hdfs-file-path
```

copyFromLocal – Copy File from Local file System

Copying file from a local file to HDFS file system. Similar to the fs -put command and copyFromLocal command both are Store files from local disk to HDFS. Except that the source is restricted to a local file reference.

```
$ hadoop fs -copyToLocal /hdfs-file-path /local-file-path
or
$ hdfs dfs -copyToLocal  /hdfs-file-path /local-file-path
```

copyToLocal – Copy Files from HDFS to Local file System

Copying files from HDFS file to local file system. Similar to the fs -get command and copyToLocal command both are Store files from hdfs to local files. Except that the destination is restricted to a local file reference.

```
$ hadoop fs -copyToLocal /hdfs-file-path /local-file-path
or
$ hdfs dfs -copyToLocal  /hdfs-file-path /local-file-path
```

usage – Return the Help for Individual Command

Usage command is used to Provide you help for indidual commands.

```
$ hadoop fs -usage mkdir
or
$ hdfs dfs -usage mkdir
```

checksum -Returns the Checksum Information of a File

The checksum command is used to Returns the Checksum Information of a File. Returns the checksum information of a file.

```
$ hadoop fs -checksum [-v] URI
or
$ hdfs dfs -checksum [-v] URI
```

chgrp – Change Group Association of Files

chgrg command is used to change the group of a file or a path. The user must be the owner of files, or else a super-user.

```
$ hadoop fs -chgrp [-R] groupname
or
$ hdfs dfs -chgrp [-R] groupname
```

chmod – Change the Permissions of a File

This command is used to change the permissions of a file. With -R Used to modify the files recursively and it is the only option that is being supported currently.

```
$ hadoop fs -chmod [-R] hdfs-file-path
or
$ hdfs dfs -chmod [-R] hdfs-file-path
```

chown – Change the Owner and Group of a File

Chown command is used to change the owner and group of a file. This command is similar to the shell's chown command with a few exceptions.

```
$ hadoop fs -chown [-R] [owner][:[group]] hdfs-file-path
or
```

```
$ hdfs dfs -chown [-R] [owner][:[group]] hdfs-file-path
```

df – Displays free Space

Df is the Displays free space. This command is used to show the capacity, free and used space available on the HDFS filesystem. Used to format the sizes of the files in a human-readable manner rather than the number of bytes.

```
$ hadoop fs -df /user/hadoop/dir1
or
$ hdfs dfs -df /user/hadoop/dir1
```

head – Displays first Kilobyte of the File

Head command is use to Displays first kilobyte of the file to stdout.

```
$ hadoop fs -head /hdfs-file-path
or
$ hdfs dfs -head /hdfs-file-path
```

createSnapshots – Create Snapshottable Directory

This operation requires owner privilege of the snapshot table directory. The path of the snapshot table directory, snapshot name is The snapshot name a default name is generated using a timestamp.

```
$ hadoop fs -createSnapshot /path /snapshotName
or
$ hdfs dfs -createSnapshot /path /snapshotName
```

deleteSnapshots – Delect Snapshottable Directory

This operation requires owner privilege of the snapshot table directory. The path of the snapshot table directory, snapshot name is The snapshot name.

```
$ hadoop fs -deleteSnapshot /path /snapshotName
or
$ hdfs dfs -deleteSnapshot /path /snapshotName
```

renameSnapshots – Rename a Snapshot

This operation requires owner privilege of the snapshottable directory.

```
$ hadoop fs -renameSnapshot /path /oldName /newName
```

```
or
$ hdfs dfs  -renameSnapshot /path /oldName /newName
```

expunge – Create New Checkpoint

This command is used to empty the trash available in an HDFS system. Permanently delete files in checkpoints older than the retention threshold from the trash directory.

```
$ hadoop fs –expunge -immediate -fs  /hdfs-file-path
or
$ hdfs dfs –expunge -immediate -fs  /hdfs-file-path
```

Stat – File/Directory Print Statistics

This command is used to print the statistics about the file/directory in the specified format. Print statistics about the file/directory at in the specified format.

```
$ hadoop fs -stat /format
or
$ hdfs dfs -stat /format
```

Truncate – Specified File Pattern and Length

Truncate all files that match the specified file pattern to the specified length.

```
$ hadoop fs -truncate [-w] /length /hdfs-file-path
or
$ hdfs dfs -truncate [-w] /length /hdfs-file-path
```

Find – Find File Size in HDFS

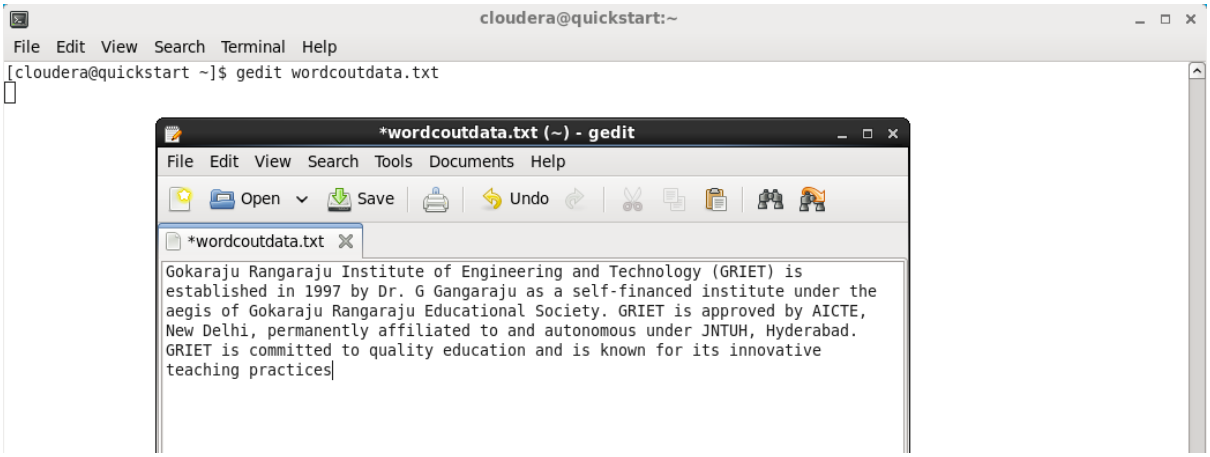
In Hadoop, `hdfs dfs -find` or `hadoop fs -find` commands are used to get the size of a single file or size for all files specified in an expression or in a directory. By default, it points to the current directory when the path is not specified.

```
$hadoop fs -find / -name test -print
or
$hdfs dfs -find / -name test -print
```

Task1b Word Count Program using python in Hadoop Streaming

Step 1: Create a file with the name wordcountdata.txt and add some data to it.

[cloudera@quickstart ~]$ gedit wordcoutdata.txt

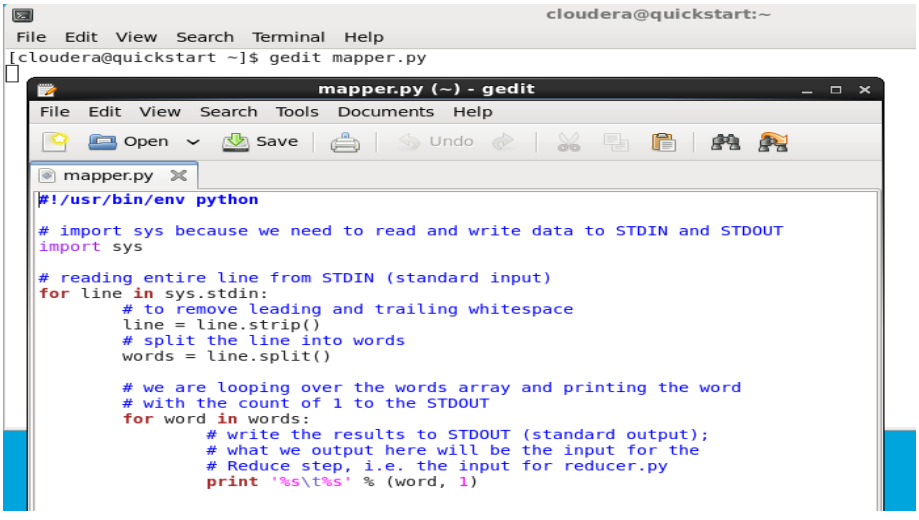


Click on save and press enter

Step 2: Create a mapper.py file that implements the mapper logic. It will read the data from STDIN and will split the lines into words, and will generate an output of each word with its individual count.

[cloudera@quickstart ~]$ gedit mapper.py

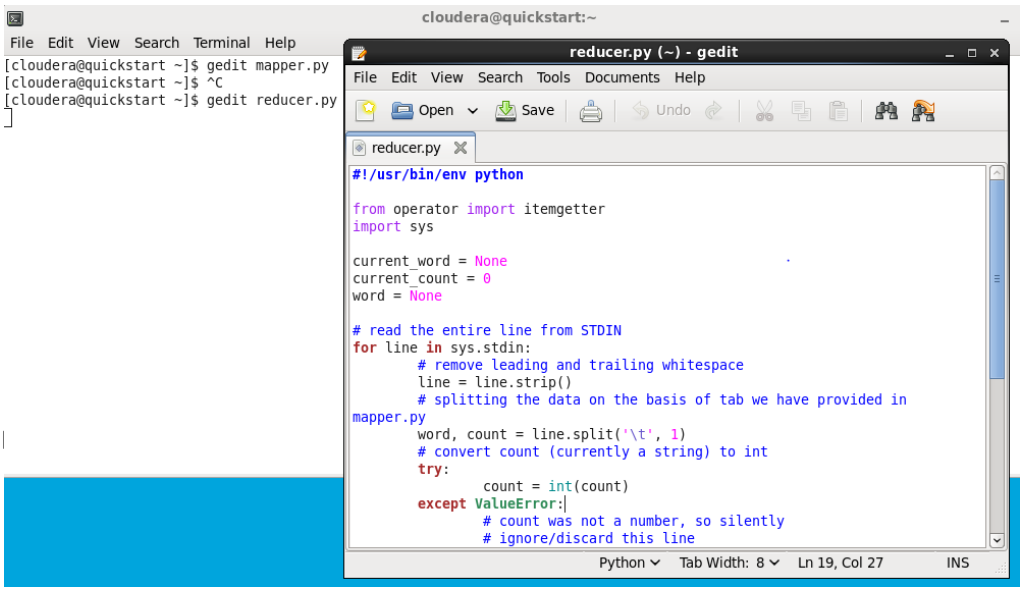
```
#!/usr/bin/env python
# import sys because we need to read and write data to STDIN and STDOUT
import sys
# reading entire line from STDIN (standard input)
for line in sys.stdin:
        # to remove leading and trailing whitespace
        line = line.strip()
        # split the line into words
        words = line.split()
        # we are looping over the words array and printing the word
        # with the count of 1 to the STDOUT
        for word in words:
                # write the results to STDOUT (standard output);
                # what we output here will be the input for the
                # Reduce step, i.e. the input for reducer.py
                print '%s\t%s' % (word, 1)
```



Step 3: Create a reducer.py file that implements the reducer logic. It will read the output of mapper.py from STDIN(standard input) and will aggregate the occurrence of each word and will write the final output to STDOUT.

[cloudera@quickstart ~]$ gedit reducer.py

```
#!/usr/bin/env python

from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

# read the entire line from STDIN
for line in sys.stdin:
        # remove leading and trailing whitespace
        line = line.strip()
        # splitting the data on the basis of tab we have provided in mapper.py
        word, count = line.split('\t', 1)
        # convert count (currently a string) to int
        try:
                count = int(count)
        except ValueError:
                # count was not a number, so silently
                # ignore/discard this line
                continue

        # this IF-switch only works because Hadoop sorts map output
        # by key (here: word) before it is passed to the reducer
        if current_word == word:
                current_count += count
        else:
                if current_word:
                        # write result to STDOUT
                        print '%s\t%s' % (current_word, current_count)
                current_count = count
                current_word = word

# do not forget to output the last word if needed!
if current_word == word:
        print '%s\t%s' % (current_word, current_count)
```

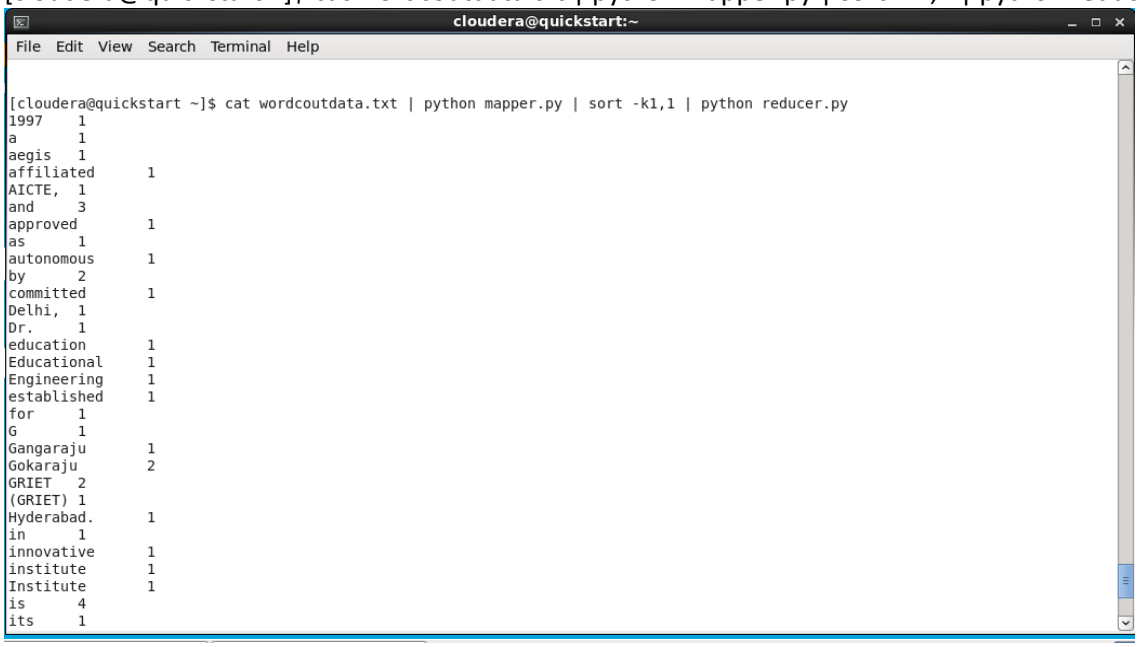


[cloudera@quickstart ~]$ cat wordcoutdata.txt | python mapper.py

Gokaraju 1
Rangaraju 1
Institute 1
of 1
Engineering 1
and 1
Technology 1
(GRIET) 1
is 1
established 1
in 1
1997 1
by 1
Dr. 1
G 1
Gangaraju 1
as 1
a 1
self-financed 1
institute 1
under 1
the 1
aegis 1
of 1
Gokaraju 1
Rangaraju 1
Educational 1
Society. 1
GRIET 1
is 1
approved 1
by 1
AICTE, 1
New 1
Delhi, 1
permanently 1
affiliated 1
to 1
and 1
autonomous 1
under 1
JNTUH, 1
Hyderabad. 1
GRIET 1
is 1
committed 1
to 1
quality 1
education 1
and 1
is 1
known 1

for	1
its	1
innovative	1
teaching	1
practices	1

[cloudera@quickstart ~]$ cat wordcoutdata.txt | python mapper.py | sort -k1,1 | python reducer.py



1997	1
a	1
aegis	1
affiliated	1
AICTE,	1
and	3
approved	1
as	1
autonomous	1
by	2
committed	1
Delhi,	1
Dr.	1
education	1
Educational	1
Engineering	1
established	1
for	1
G	1
Gangaraju	1
Gokaraju	2
GRIET	2
(GRIET)	1
Hyderabad.	1
in	1
innovative	1
institute	1
Institute	1
is	4
its	1

```
JNTUH,          1
known           1
New             1
of              2
permanently     1
practices       1
quality         1
Rangaraju       2
self-financed   1
Society.        1
teaching        1
Technology      1
the             1
to              2
under           2
```

Now make a directory Task1b in our HDFS in the root directory that will store our wordcoutdata.txt file with the below command.

```
[cloudera@quickstart ~]$ hadoop fs -mkdir task1b
```

Copy wordcoutdata.txt to this folder in our HDFS with help of copyFromLocal command.

```
[cloudera@quickstart ~]$ hadoop fs -copyFromLocal /home/cloudera/wordcoutdata.txt /user/cloudera/task1b
```

Now our data file has been sent to HDFS successfully. we can check whether it sends or not by using the below command or by manually visiting our HDFS.

```
[cloudera@quickstart ~]$ hadoop fs -ls /user/cloudera/task1b
Found 1 items
-rw-r--r--   1 cloudera cloudera        392 2023-02-04 00:52 /user/cloudera/task1b/wordcoutdata.txt
[cloudera@quickstart ~]$
```

Let’s give executable permission to our mapper.py and reducer.py with the help of below command.

```
[cloudera@quickstart ~]$ chmod 777 mapper.py reducer.py
```

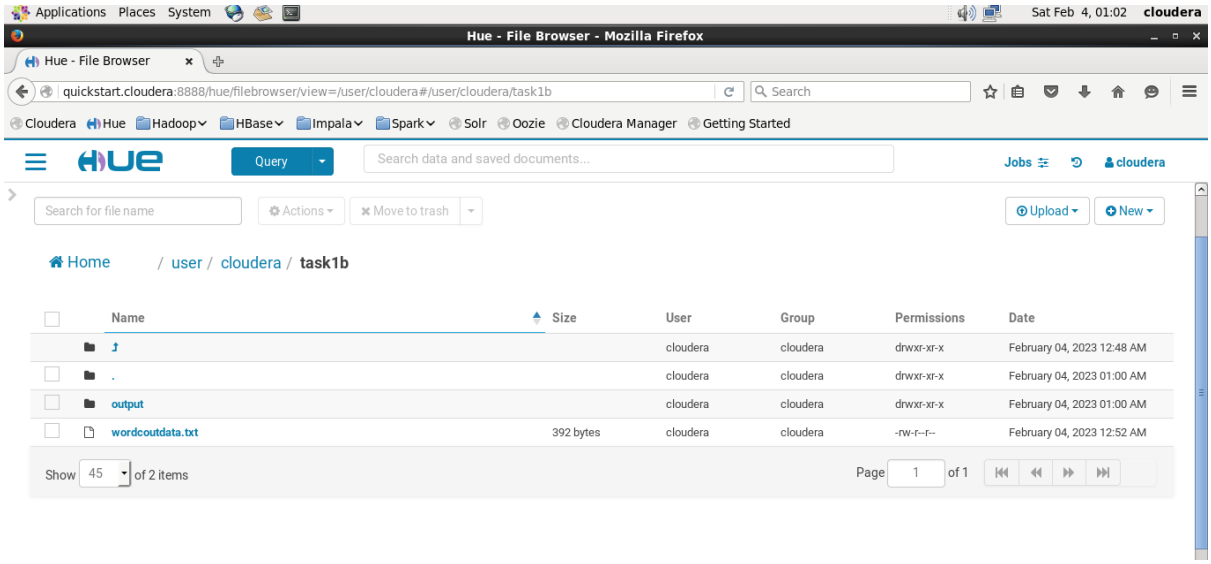
Step 5: Now download the latest hadoop-streaming jar file and then place, this Hadoop,-streaming jar file to a place from you can easily access it

```
[cloudera@quickstart ~]$ hadoop jar /home/cloudera/hadoop-streaming-2.6.0.jar \
> -input /user/cloudera/task1b/wordcoutdata.txt \
> -output /user/cloudera/task1b/output \
> -mapper /home/cloudera/mapper.py \
> -reducer /home/cloudera/reducer.py
```

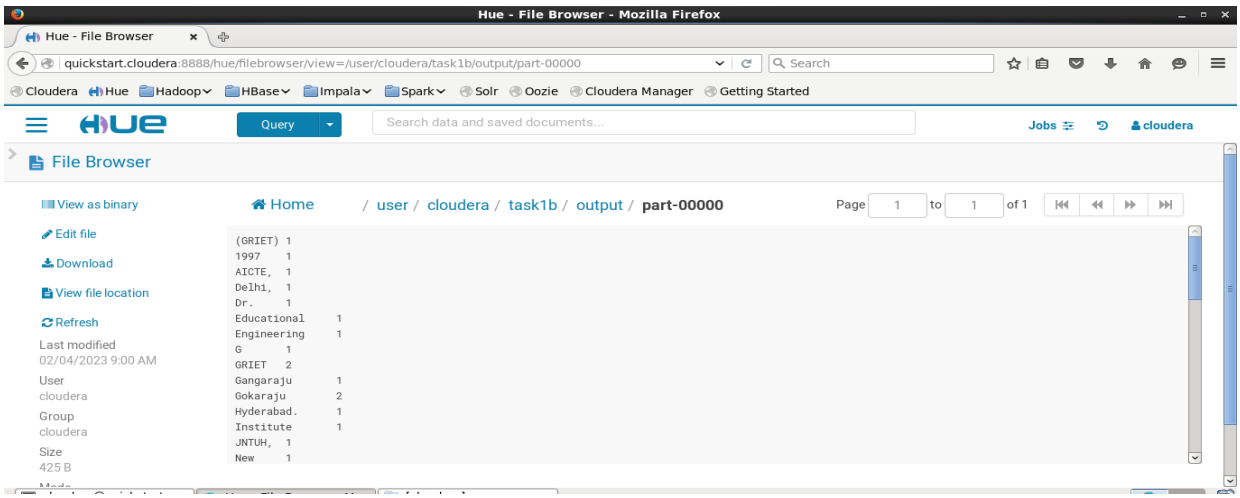
```
cloudera@quickstart:~
File  Edit  View  Search  Terminal  Help
[cloudera@quickstart ~]$ hadoop jar /home/cloudera/hadoop-streaming-2.6.0.jar \
> -input /user/cloudera/task1b/wordcoutdata.txt \
> -output /user/cloudera/task1b/output \
> -mapper /home/cloudera/mapper.py \
> -reducer /home/cloudera/reducer.py
packageJobJar: [] [/usr/lib/hadoop-mapreduce/hadoop-streaming-2.6.0-cdh5.13.0.jar] /tmp/streamjob6311890801674592475.jar tmpDir=null
23/02/04 01:00:12 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
23/02/04 01:00:12 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
23/02/04 01:00:13 INFO mapred.FileInputFormat: Total input paths to process : 1
23/02/04 01:00:13 INFO mapreduce.JobSubmitter: number of splits:2
23/02/04 01:00:13 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1675487406087_0003
23/02/04 01:00:13 INFO impl.YarnClientImpl: Submitted application application_1675487406087_0003
23/02/04 01:00:13 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/application_1675487406087_0003/
23/02/04 01:00:13 INFO mapreduce.Job: Running job: job_1675487406087_0003
23/02/04 01:00:23 INFO mapreduce.Job: Job job_1675487406087_0003 running in uber mode : false
23/02/04 01:00:23 INFO mapreduce.Job:  map 0% reduce 0%
23/02/04 01:00:40 INFO mapreduce.Job:  map 50% reduce 0%
23/02/04 01:00:41 INFO mapreduce.Job:  map 100% reduce 0%
23/02/04 01:00:46 INFO mapreduce.Job:  map 100% reduce 100%
23/02/04 01:00:46 INFO mapreduce.Job: Job job_1675487406087_0003 completed successfully
23/02/04 01:00:46 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=626
                FILE: Number of bytes written=438447
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=830
                HDFS: Number of bytes written=425
                HDFS: Number of read operations=9
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
```

```
cloudera@quickstart:~
File  Edit  View  Search  Terminal  Help
        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)=31535
                Total time spent by all reduces in occupied slots (ms)=3484
                Total time spent by all map tasks (ms)=31535
                Total time spent by all reduce tasks (ms)=3484
                Total vcore-milliseconds taken by all map tasks=31535
                Total vcore-milliseconds taken by all reduce tasks=3484
                Total megabyte-milliseconds taken by all map tasks=32291840
                Total megabyte-milliseconds taken by all reduce tasks=3567616
        Map-Reduce Framework
                Map input records=1
                Map output records=57
                Map output bytes=506
                Map output materialized bytes=632
                Input split bytes=242
                Combine input records=0
                Combine output records=0
                Reduce input groups=45
                Reduce shuffle bytes=632
                Reduce input records=57
                Reduce output records=45
                Spilled Records=114
                Shuffled Maps =2
                Failed Shuffles=0
                Merged Map outputs=2
                GC time elapsed (ms)=444
                CPU time spent (ms)=3800
                Physical memory (bytes) snapshot=669650944
                Virtual memory (bytes) snapshot=4523814912
                Total committed heap usage (bytes)=584261632
```

```
cloudera@quickstart:~
File  Edit  View  Search  Terminal  Help
                Map input records=1
                Map output records=57
                Map output bytes=506
                Map output materialized bytes=632
                Input split bytes=242
                Combine input records=0
                Combine output records=0
                Reduce input groups=45
                Reduce shuffle bytes=632
                Reduce input records=57
                Reduce output records=45
                Spilled Records=114
                Shuffled Maps =2
                Failed Shuffles=0
                Merged Map outputs=2
                GC time elapsed (ms)=444
                CPU time spent (ms)=3800
                Physical memory (bytes) snapshot=669650944
                Virtual memory (bytes) snapshot=4523814912
                Total committed heap usage (bytes)=584261632
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=588
        File Output Format Counters
                Bytes Written=425
23/02/04 01:00:46 INFO streaming.StreamJob: Output directory: /user/cloudera/task1b/output
[cloudera@quickstart ~]$
```



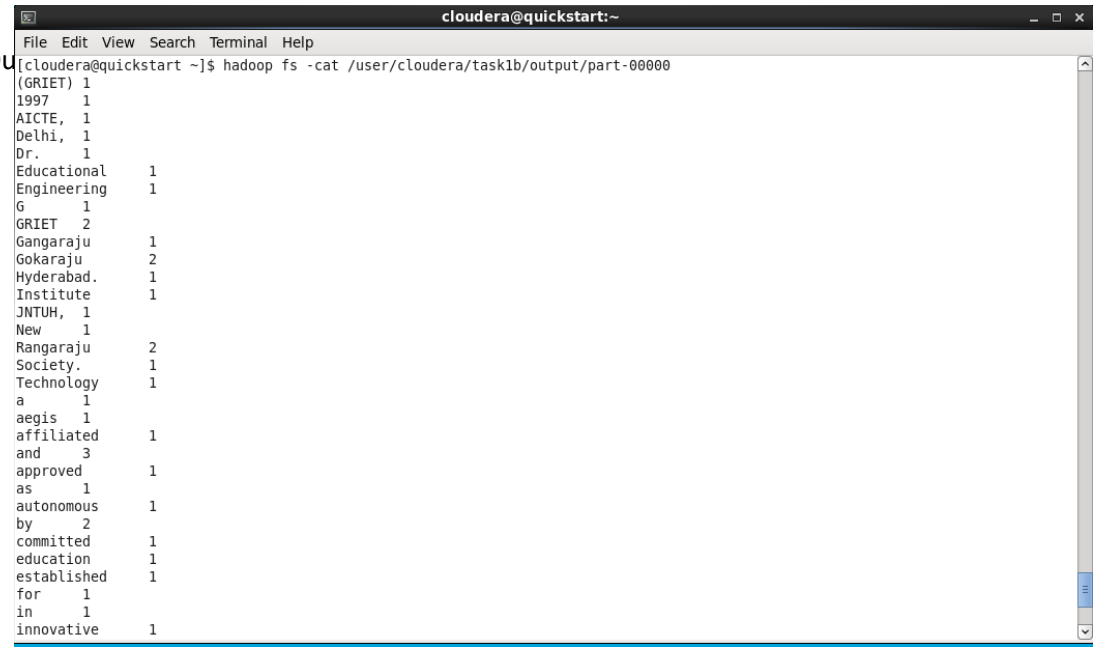
In the above command in -output, we will specify the location in HDFS where we want our output to be stored. So let's check our output in output file at location /task1b/output/part-00000 in my case. We can check results by manually vising the location in HDFS or with the help of cat command as shown below.



Or

[cloudera@quickstart ~]$ hadoop fs -cat /user/cloudera/task1b/output/part-00000

Ou



TASK 2: Write a Map Reduce program that mines weather data

Aim: To calculate max and min temperature in weather data

Procedure:

Create two programs for max and min temperature python programs and to place the 1800.csv data se in cloudera/home .

Dataset:1800.csv

Min.py

```
from mrjob.job import MRJob

class MRMinTemperature(MRJob):
    def MakeFahrenheit(self, tenthsOfCelsius):
        celsius = float(tenthsOfCelsius) / 10.0
        fahrenheit = celsius * 1.8 + 32.0
        return fahrenheit
    def mapper(self, _, line):
        (location, date, type, data, x, y, z, w) = line.split(',')
        if (type == 'TMIN'):
            temperature = self.MakeFahrenheit(data)
            yield location, temperature
    def reducer(self, location, temps):
        yield location, min(temps)
if __name__ == '__main__':
    MRMinTemperature.run()
from mrjob.job import MRJob
```

Execution:

```
[cloudera@localhost~]$cat 1800.csv | python Min.py
```

Output:

Max.py

```
class MRMaxTemperature(MRJob):
    def MakeFahrenheit(self, tenthsOfCelsius):
        celsius = float(tenthsOfCelsius) / 10.0
        fahrenheit = celsius * 1.8 + 32.0
        return fahrenheit
    def mapper(self, _, line):
        (location, date, type, data, x, y, z, w) = line.split(',')
        if (type == 'TMAX'):
            temperature = self.MakeFahrenheit(data)
            yield location, temperature
    def reducer(self, location, temps):
        yield location, min(temps)
if __name__ == '__main__':
    MRMaxTemperature.run()
```

Execution:

```
[cloudera@localhost~]$cat 1800.csv | python Max.py
```

Output:

TASK 3: Implement matrix multiplication with Hadoop Map Reduce in Python
Step 1.
Create a mat directory in HDFS
[cloudera@quickstart ~]$ hadoop fs -mkdir mat
Step2:
place the two matrices M1(2x3) and M2(3x2) in mat directory
i.e M1 and M2 are the input matrices,create M1 text file and enter the data ,similarly M2
M1= 1 2 3
4 5 6
M2= 7 8
9 10
11 12
[cloudera@quickstart ~]$ hadoop fs -put /home/cloudera/M1 /user/cloudera/mat
[cloudera@quickstart ~]$ hadoop fs -put /home/cloudera/M2 /user/cloudera/mat
Step4: write python code for Matrix Multiplication
1.Mapper.py

```
#!/usr/bin/env python
import sys
m_r=2
m_c=3
n_r=3
n_c=2

i=0
for line in sys.stdin:
        el=map(int,line.split())
        if(i<m_r):
                for j in range(len(el)):
                        for k in range(n_c): print "{0}\t{1}\t{2}\t{3}".format(i, k, j, el[j])
        else:
                for j in range(len(el)):
                        for k in range(m_r): print "{0}\t{1}\t{2}\t{3}".format(k, j, i-m_r, el[j])
        i=i+1
```

2.Reducer.py

```
#!/usr/bin/env python
import sys
m_r=2
m_c=3
n_r=3
n_c=2

matrix=[]
for row in range(m_r):
```

```
        r=[]
        for col in range(n_c):
                s=0
                for el in range(m_c):
                        mul=1
                        for num in range(2):
                                line=sys.stdin.readline()
                                n=map(int,line.split('\t'))[-1]
                                mul*=n
                        s+=mul
                r.append(s)
        matrix.append(r)
print('\n'.join([str(x) for x in matrix]))
```

Step4:Running the Map-Reduce Job on Hadoop

```
[cloudera@quickstart ~]$ chmod +x /home/cloudera/Mapper.py
[cloudera@quickstart ~]$ chmod +x /home/cloudera/Reducer.py
[cloudera@quickstart ~]$ hadoop jar /home/cloudera/hadoop-streaming-2.6.0.jar \
> -input /user/cloudera/mat/ \
> -output /user/cloudera/mat/output \
> -mapper /home/cloudera/Mapper.py \
> -reducer /home/cloudera/Reducer.py
.
.
.
.
.
23/02/11 01:30:58 INFO streaming.StreamJob: Output directory: /user/cloudera/mat/output
```

Step5:Output

```
[cloudera@quickstart ~]$ hadoop fs -cat  /user/cloudera/mat/output/part-00000
[14, 245]
[313, 77]
```

TASK 4

Program: Working with files in Hadoop file system: Reading, Writing and Copying

Aim: To implement the reading,writing and copying the file form local file system to HDFS

Reading and Writing the file content from local file system to Hadoop file system using appendToFile and cat command:

Create a directory called TASK4 in HDFS

[cloudera@quickstart ~]$ hadoop fs -mkdir TASK4

Create empty text file called source.txt in local file system and put this file in HDFS directory called TASK4

[cloudera@quickstart ~]$ hadoop fs -put /home/cloudera/Desktop/source.txt /user/cloudera/TASK4/

Create text file called Dest.txt in local file system along with the some text.

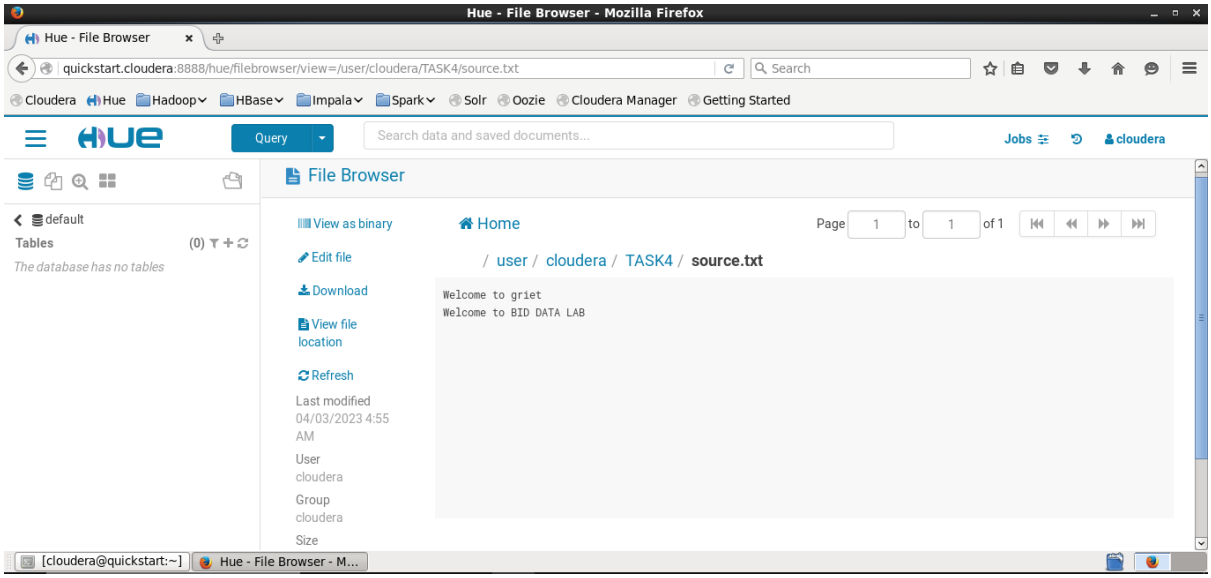
[cloudera@quickstart ~]$ hadoop fs -appendToFile /home/cloudera/Desktop/Dest.txt /user/cloudera/TASK4/source.txt

Output :

[cloudera@quickstart ~]$ hadoop fs -cat /user/cloudera/TASK4/source.txt

Welcome to griet

Welcome to BID DATA LAB



Copy files in different folders in Hadoop

Step1: Create one directory in local system is called DATA, it consist of set files

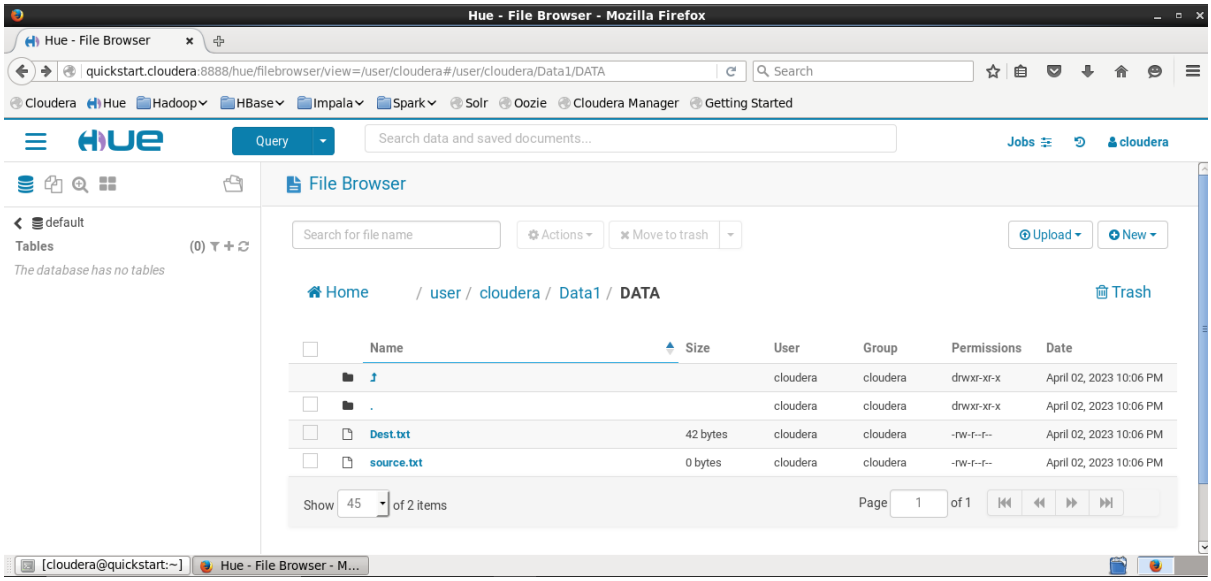
Step 2: Create directory in HDFS called Data1.

Now, copy the local directory files to HDFS files directory.

[cloudera@quickstart ~]$ hadoop fs -put /home/cloudera/Desktop/DATA/ /user/cloudera/Data1/

Output:

[cloudera@quickstart ~]$ hadoop fs -ls /user/cloudera/Data1/DATA
Found 2 items
-rw-r--r-- 1 cloudera cloudera 42 2023-04-02 22:06 /user/cloudera/Data1/DATA/Dest.txt
-rw-r--r-- 1 cloudera cloudera 0 2023-04-02 22:06 /user/cloudera/Data1/DATA/source.txt


Hue - File Browser - Mozilla Firefox
Hue - File Browser
quickstart.cloudera:8888/hue/filebrowser/view=/user/cloudera#/user/cloudera/Data1/DATA
Search
Cloudera
Hue
Hadoop
HBase
Impala
Spark
Solr
Oozie
Cloudera Manager
Getting Started
Query
Search data and saved documents...
Jobs
cloudera
File Browser
default
Tables (0)
The database has no tables
Search for file name
Actions
Move to trash
Upload
New
Home / user / cloudera / Data1 / DATA
Trash
Name
Size
User
Group
Permissions
Date
cloudera cloudera drwxr-xr-x April 02, 2023 10:06 PM
cloudera cloudera drwxr-xr-x April 02, 2023 10:06 PM
Dest.txt 42 bytes cloudera cloudera -rw-r--r-- April 02, 2023 10:06 PM
source.txt 0 bytes cloudera cloudera -rw-r--r-- April 02, 2023 10:06 PM
Show 45 of 2 items
Page 1 of 1
[cloudera@quickstart:~]
Hue - File Browser - M...

Task-5

TASK-5: Write Pig Latin scripts sort, group, join, project, and filter your data.

Step1:

Create a data the fallowing file first.txt and second.txt

first.txt

Attributes(user,url,id,)

User,url,id

Srinivas,google,g123
Srikar,yahoo,y12,
Sreyas,flipcart,f34
Lohitha,linkedin,l46
Syam,myblog,

second.txt
Attributes (url,rating)

google,8
yahoo,7
flipcart,4
linkedin,5
myblog,3

creat a directory called task 5 in HDFS and place first.txt and second.txt in task5.

LOAD:

LOAD operator is used to load data from the file system or HDFS storage into a Pig relation.

grunt>loading1 = load '/user/cloudera/task5/first.txt' using PigStorage(',') as (user:chararray,url:chararray,id:int);

grunt>loading2 = load '/user/cloudera/task5/first.txt' using PigStorage(',') as (user:chararray,url:chararray,id:int);

FOREACH:

This operator generates data transformations based on columns of data. It is used to add or remove fields from a relation. Use FOREACH-GENERATE operation to work with columns of data.

```
grunt> for_each = foreach loading1 generate url,id;
grunt> dump for_each;
```

FOREACH Result:

```
(cnn,8)
(crap,8)
(myblog,10)
(flickr,10)
(cnn,12)
grunt>
```

FILTER:

This operator selects tuples from a relation based on a condition.

In this example, we are filtering the record from 'loading1' when the condition 'id' is greater than 8.

```
grunt> filter_command = filter loading1 by id>8;
grunt> dump filter_command;
```

FILTER Result:

```
(amr,myblog,10)
(amr,flickr,10)
(fred,cnn,12)
grunt>
```

JOIN:

JOIN operator is used to perform an inner, equijoin join of two or more relations based on common field values. The JOIN operator always performs an inner join. Inner joins ignore null keys, so it makes sense to filter them out before the join.

In this example, join the two relations based on the column 'url' from 'loading1' and 'loading2'.

```
grunt> join_command = join loading1 by url,loading2 by url;
grunt> dump join_command;
```

JOIN Result:

```
(amr,cnn,8,cnn,9)
(fred,cnn,12,cnn,9)
(amr,crap,8,crap,2)
(amr,flickr,10,flickr,9)
(amr,myblog,10,myblog,7)
grunt>
```

ORDER BY:

Order By is used to sort a relation based on one or more fields. You can do sorting in ascending or descending order using ASC and DESC keywords.

In below example, we are sorting data in loading2 in ascending order on ratings field.

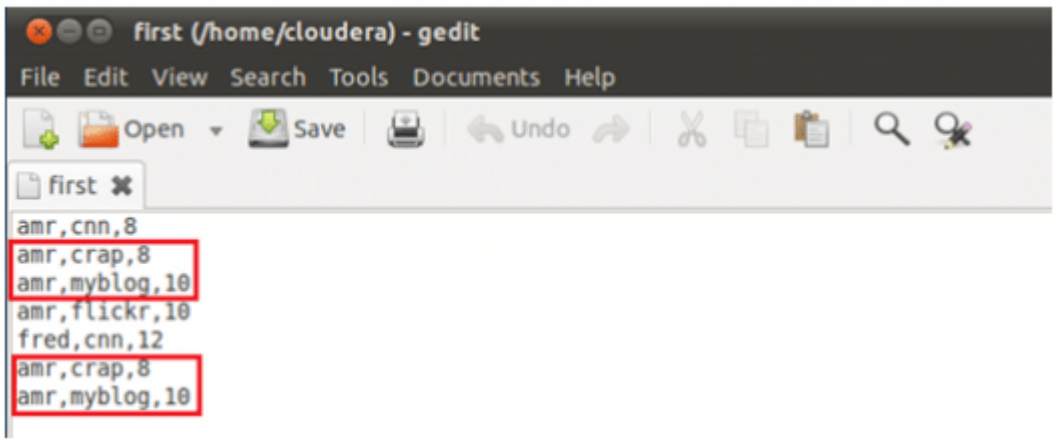
```
grunt> loading4 = ORDER loading2 by rating ASC;
grunt> dump loading4;
```

ORDER BY Result:

```
(crap,2)
(myblog,7)
(cnn,9)
(flickr,9)
grunt>
```

DISTINCT:

Distinct removes duplicate tuples in a relation.Lets take an input file as below, which has **amr,crap,8** and **amr,myblog,10** twice in the file. When we apply distinct on the data in this file, duplicate entries are removed.




```
grunt> loading1 = load '/first' using PigStorage(',') as (user:chararray,url:chararray,id:int);
grunt> loading3 = DISTINCT loading1;
grunt> dump loading3;
```

DISTINCT Result:

```
(amr,cnn,8)
(amr,crap,8)
(amr,flickr,10)
(amr,myblog,10)
(fred,cnn,12)
grunt>
```

STORE:

Store is used to save results to the file system.

Here we are saving **loading3** data into a file named **storing** on HDFS.

```
grunt> store loading3 into '/storing';
```

Apache Pig Tutorial – Project and Manipulate Columns

Goal of this tutorial is to learn Apache Pig concepts in a fast pace. So don't except lengthy posts. All posts will be short and sweet. Most posts will have (very short) "see it in action" video.

In this post, we saw different variations of loading a dataset in Pig. In this post we will see how to project and manipulate columns.

Load Dataset With Column Names and Types

```
grunt> stocks = LOAD '/user/hirw/input/stocks' USING PigStorage(',') as
(exchange:chararray, symbol:chararray, date:datetime, open:float,
high:float, low:float, close:float, volume:int, adj_close:float);
```

Use FOREACH..GENERATE operator for projecting the columns in the dataset.

```
grunt> projection = FOREACH stocks GENERATE symbol, SUBSTRING(exchange,
0, 1) as sub_exch, close - open as up_or_down;
```

Now projection will have 3 columns – symbol, sub_exch which is a result of a substring operation on the exchange column and up_or_down which is a result of close price minus the opening price.

```
grunt> top100 = LIMIT projection 100;

grunt> DUMP top100;
```

Load Dataset With Out Column Names and Types

```
grunt> stocks1 = LOAD '/user/hirw/input/stocks' USING PigStorage(',');

grunt> DESCRIBE stocks1;

Schema for stocks1 unknown.
```

With schema unknown for stocks1 relation how can we project and manipulate column? Very simple, we can use the column position starting with index 0 for first column.

```
grunt> projection = FOREACH stocks1 GENERATE $1 as symbol,
SUBSTRING($0, 0, 1) as sub_exch, $6 - $3 as up_or_down;
```

How does Pig figure out the datatypes? Pig is smart, it looks at the operation you are tying to perform on the columns and cast the type from the default bytearrary to appropriate type. For example since we are trying to do a substring operation on the exchange column ($0), Pig assumes exchange column in string and will cast bytearray to chararray.

Similarly close ($6) and open ($3) are converted to double (casting to integer or float might lose precision) since these columns are involved in numerical calculation.

```
grunt> DESCRIBE projection;

2015-12-09 12:41:55,172 [main] WARN org.apache.pig.PigServer -
Encountered Warning USING_OVERLOADED_FUNCTION 1 time(s).

2015-12-09 12:41:55,172 [main] WARN org.apache.pig.PigServer -
Encountered Warning IMPLICIT_CAST_TO_CHARARRAY 1 time(s).

2015-12-09 12:41:55,172 [main] WARN org.apache.pig.PigServer -
Encountered Warning IMPLICIT_CAST_TO_DOUBLE 2 time(s).

projection: {symbol: bytearray,sub_exch: chararray,up_or_down: double}
```

Finally, display the results.

```
grunt> top100 = LIMIT projection 100;
```

```
grunt> DUMP top100;
```

TASK 6: Run the Pig Latin Scripts to find Word Count and max. temp for each and every year.

Word Count using Pig Latin

Step1:

1. **Create a text file having few lines of text and save it as bd.txt.**
2. **Create on directory in hdfs named wc**
3. **Store bd..txt file from local file system to hdfs file system directory wc**

Step2:

```
inputline = load '/user/cloudera/wc/bd.txt' using PigStorage('\t') as (data:chararray);

words = FOREACH inputline GENERATE FLATTEN(TOKENIZE(data)) AS word;

filtered_words = FILTER words BY word MATCHES '\\w+';

word_groups = GROUP filtered_words BY word;

word_count = FOREACH word_groups GENERATE COUNT(filtered_words) AS count, group AS word;

ordered_word_count = ORDER word_count BY count DESC;

DUMP ordered_word_count;
```

You can use the below command to save the result in HDFS.

```
grunt> store ordered_word_count; into '/user/cloudera/wc/output/';
```

Find the Maximum Year in a given dataset using PIG

Option1:

```
A = LOAD 'input' USING PigStorage() AS (Year:int,Temp:int);
B = GROUP A ALL;
C = FOREACH B GENERATE MAX(A.Temp);
DUMP C;
```

or

Option2: Using (ORDER and LIMIT)

```
A = LOAD 'input' USING PigStorage() AS (Year:int,Temp:int);
B = ORDER A BY Temp DESC;
C = LIMIT B 1;
D = FOREACH C GENERATE Temp;
DUMP D;
```

TASK-7

Program: Writing User Defined Functions/Eval functions for filtering unwanted data in Pig

Aim: To implement the Filtering unwanted data in pig

loudera@quickstart ~]$ hadoop fs -mkdir task7

[cloudera@quickstart ~]$ hadoop fs -put /home/cloudera/Desktop/phonedata.txt /user/cloudera/task7

[cloudera@quickstart ~]$ pig

Step 1: Load the file

Load the text file data in pig, use below command. Change the location as per your environment.

grunt> data = load '/user/cloudera/task7/phonedata.txt' using PigStorage('|') as(id:chararray,contype:chararray);

grunt> dump data

(1A58252,POSTP)
(73VS543,)
(HD52836,PREP)
(8SH356H,POSTP)
(L62HJS5,PEND)
(Q672821,PREP)
(M672GS6,CLS)
(DGSW522,POSTP)
(RAR1729,PEND)
(YE52863,)
(GHS5281,)
(SYY8373,CLS)

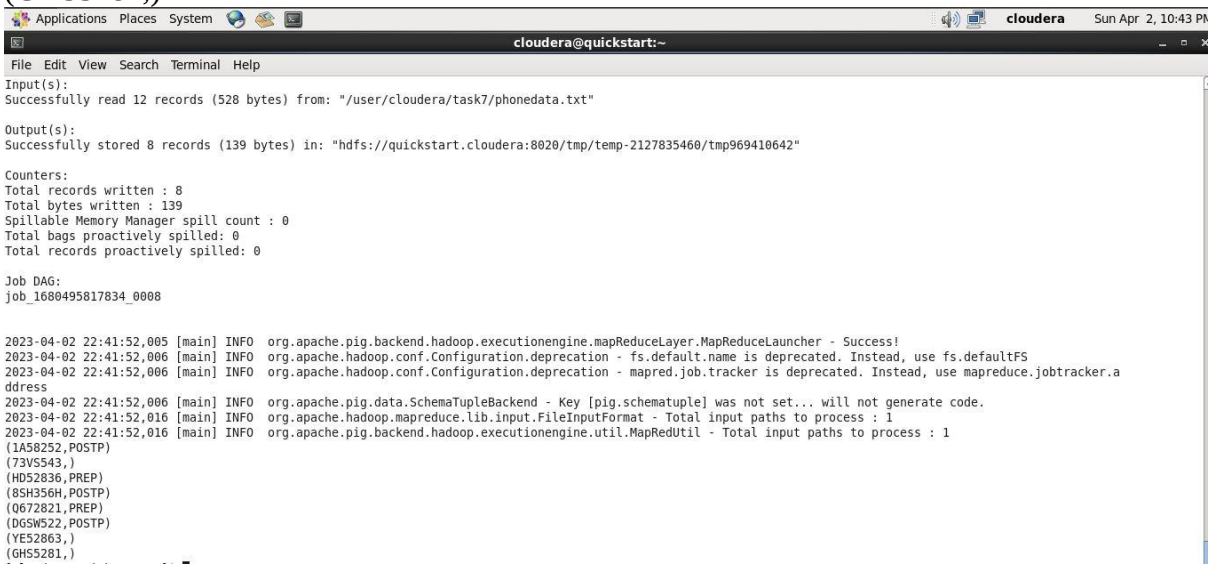
```
Successfully stored 12 records (209 bytes) in: "hdfs://quickstart.cloudera:8020/tmp/temp-2127835460/tmp-1003900899"

Counters:
Total records written : 12
Total bytes written : 209
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_1680495817834_0007


2023-04-02 22:36:25,104 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2023-04-02 22:36:25,106 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2023-04-02 22:36:25,106 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.a
ddress
2023-04-02 22:36:25,107 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2023-04-02 22:36:25,119 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2023-04-02 22:36:25,119 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(1A58252,POSTP)
(73VS543,)
(HD52836,PREP)
(8SH356H,POSTP)
(L62HJS5,PEND)
(Q672821,PREP)
(M672GS6,CLS)
(DGSW522,POSTP)
(RAR1729,PEND)
(YE52863,)
(GHS5281,)
(SYY8373,CLS)
```

Step 2: Filter the record as per the requirement

grunt> RECORDS = FILTER data BY (contype == 'POSTP' OR contype == 'PREP' OR contype IS NULL OR contype == ' ');

grunt> dump RECORDS

(1A58252,POSTP)
(73VS543,)
(HD52836,PREP)
(8SH356H,POSTP)
(Q672821,PREP)
(DGSW522,POSTP)
(YE52863,)
(GHS5281,)

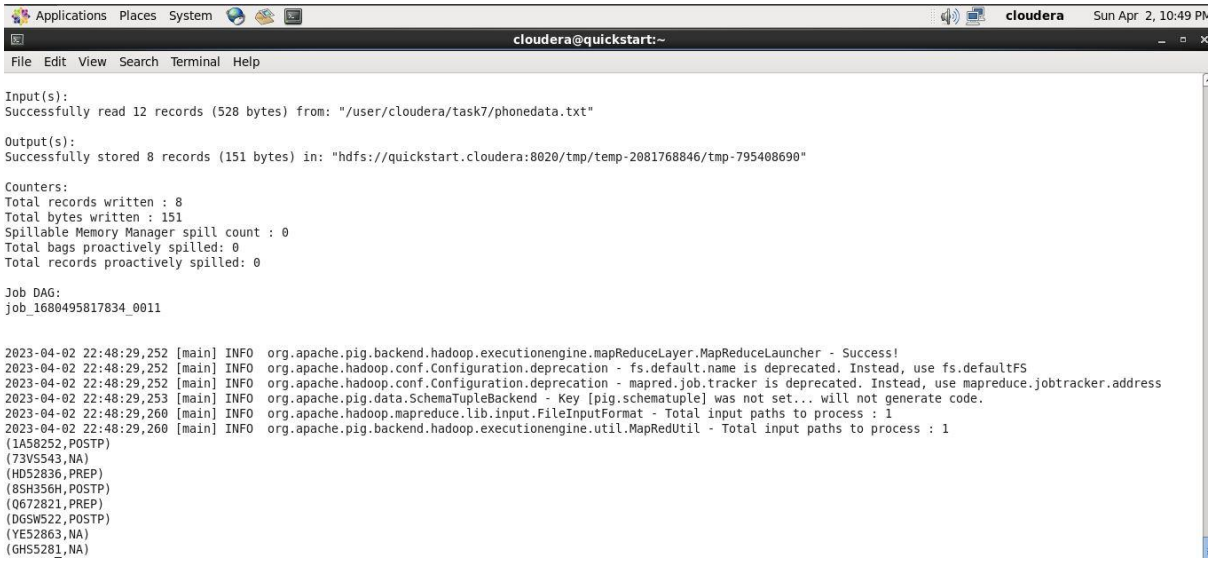


Step 3: Output

grunt> OUTPUT_RECORDS = FOREACH RECORDS GENERATE id,((contype IS NULL OR contype == '')?'NA' :contype) AS contype;

grunt> dump OUTPUT_RECORDS

(1A58252,POSTP)
(73VS543,NA)
(HD52836,PREP)
(8SH356H,POSTP)
(Q672821,PREP)
(DGSW522,POSTP)
(YE52863,NA)
(GHS5281,NA)


Applications Places System
cloudera Sun Apr 2, 10:49 PM
cloudera@quickstart:~
File Edit View Search Terminal Help
Input(s):
Successfully read 12 records (528 bytes) from: "/user/cloudera/task7/phonedata.txt"
Output(s):
Successfully stored 8 records (151 bytes) in: "hdfs://quickstart.cloudera:8020/tmp/temp-2081768846/tmp-795408690"
Counters:
Total records written : 8
Total bytes written : 151
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0
Job DAG:
job_1680495817834_0011
2023-04-02 22:48:29,252 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2023-04-02 22:48:29,252 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2023-04-02 22:48:29,252 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2023-04-02 22:48:29,253 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2023-04-02 22:48:29,260 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2023-04-02 22:48:29,260 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(1A58252,POSTP)
(73VS543,NA)
(HD52836,PREP)
(8SH356H,POSTP)
(Q672821,PREP)
(DGSW522,POSTP)
(YE52863,NA)
(GHS5281,NA)

TASK-8: Working with Hive QL, Use Hive to create, alter, and drop databases, tables, views, functions, and indexes

Sales1.csv

SalesExecutive	Name	Gender	Age	Location	SalesinRs
1	Mahesh	Male	25	Hyderabad	50
2	Suresh	Male	22	Hyderabad	75
3	Vijay	Male	20	Bangalore	11
4	Ramesh	Male	27	Bangalore	77
5	Jagadish	Male	28	Mumbai	45
6	Harish	Male	24	Hyderabad	52
7	Karan	Male	24	Bangalore	26
8	Raju	Male	23	Mumbai	24
9	Maya	Female	24	Mumbai	28
10	Rani	Female	30	Mumbai	31
11	Lakshmi	Female	19	Bangalore	36
12	Roopa	Female	24	Hyderabad	72
13	Sravya	Female	26	Hyderabad	69
14	Sruthi	Female	26	Hyderabad	51
15	Swetha	Female	21	Bangalore	34
16	Aishwarya	Female	24	Bangalore	40
17	Madhavi	Female	29	Mumbai	18
18	Rajini	Female	27	Mumbai	35
19	Haritha	Female	24	Hyderabad	29
20	Swathi	Female	25	Hyderabad	68

Step1 load the Dataset sales1.csv in hdfs

[cloudera@quickstart ~]$ hadoop fs -put /home/cloudera/Desktop/sales1.csv /user/cloudera/sample1/

1. Connect to hive form cloudera quickstart terminal by typing ‘hive’.
2. hive> SHOW databases;
3. hive> CREATE DATABASE sales1;
4. hive>USE sales1;
5. hive> CREATE TABLE sales1(SalesExecutive INT,Name STRING,Gender STRING,Age INT,Location STRING,SalesRs INT)ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' STORED AS TEXTFILE;
6. Load data from hdfs into hive table,i.e. sales1
7. hive> LOAD DATA INPATH '/user/cloudera/sample1/sales1.csv' into TABLE sales1;
8. Perform EDA using the following hive queries:
   hive> set hive.map.aggr = true;
   hive> set hive.execution.engine=tez;

hive> select *from sales1;

OK

NULL	Name	Gender	NULL	Location	NULL
1	Mahesh	Male	25	Hyderabad	50
2	Suresh	Male	22	Hyderabad	75
3	Vijay	Male	20	Bangalore	11
4	Ramesh	Male	27	Bangalore	77
5	Jagadish	Male	28	Mumbai	45
6	Harish	Male	24	Hyderabad	52
7	Karan	Male	24	Bangalore	26
8	Raju	Male	23	Mumbai	24
9	Maya	Female	24	Mumbai	28
10	Rani	Female	30	Mumbai	31
11	Lakshmi	Female	19	Bangalore	36
12	Roopa	Female	24	Hyderabad	72
13	Sravya	Female	26	Hyderabad	69
14	Sruthi	Female	26	Hyderabad	51
15	Swetha	Female	21	Bangalore	34
16	Aishwarya	Female	24	Bangalore	40
17	Madhavi	Female	29	Mumbai	18
18	Rajini	Female	27	Mumbai	35
19	Haritha	Female	24	Hyderabad	29
20	Swathi	Female	25	Hyderabad	68

Time taken: 0.1 seconds, Fetched: 21 row(s)

1. **Find out total sales of the company**

   SELECT SUM(SalesRs) **FROM** sales1;
   MapReduce Jobs Launched:
   Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 1.88 sec HDFS Read: 8470 HDFS Write: 4 SUCCESS
   Total MapReduce CPU Time Spent: 1 seconds 880 msec
   OK

871
Time taken: 27.869 seconds, Fetched: 1 row(s)

2. **Find out the max and min sales of the company**

SELECT MAX(SalesRs) FROM sales1;

MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 1.87 sec HDFS Read: 8541 HDFS Write: 3 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 870 msec
OK
77
Time taken: 22.769 seconds, Fetched: 1 row(s)

SELECT MIN(SalesRs) FROM sales1;

MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 1.79 sec HDFS Read: 8541 HDFS Write: 3 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 790 msec
OK
11
Time taken: 21.428 seconds, Fetched: 1 row(s)

3. **Find out total sales by location**

SELECT SUM(SalesRs) **FROM** sales1 **GROUP BY Location**;

MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 1.82 sec HDFS Read: 8922 HDFS Write: 15 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 820 msec
OK
224
466
NULL
181
Time taken: 21.779 seconds, Fetched: 4 row(s)

4. **Find out the average sales by location**

SELECT AVG(SalesRs) **FROM** sales1 **GROUP BY Location**;

MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 1.91 sec HDFS Read: 9380 HDFS Write: 47 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 910 msec
OK
37.333333333333336
58.25
NULL
30.166666666666668
Time taken: 21.657 seconds, Fetched: 4 row(s)

5. **Count the no. of sales person by location**

SELECT COUNT(SalesExecutive) **FROM** sales1

MapReduce Jobs Launched:

Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 1.96 sec HDFS Read: 8573 HDFS Write: 3 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 960 msec
OK
20
Time taken: 22.81 seconds, Fetched: 1 row(s)

Alter command;

1. Renaming Table Name

ALTER TABLE <current_table_name> RENAME TO <new_table_name>;

hive> ALTER table sales1 rename to sales2;

hive> select *from sales2;

NULL	Name	Gender	NULL	Location	NULL
1	Mahesh	Male	25	Hyderabad	50
2	Suresh	Male	22	Hyderabad	75
3	Vijay	Male	20	Bangalore	11
4	Ramesh	Male	27	Bangalore	77
5	Jagadish	Male	28	Mumbai	45
6	Harish	Male	24	Hyderabad	52
7	Karan	Male	24	Bangalore	26
8	Raju	Male	23	Mumbai	24
9	Maya	Female	24	Mumbai	28
10	Rani	Female	30	Mumbai	31
11	Lakshmi	Female	19	Bangalore	36
12	Roopa	Female	24	Hyderabad	72
13	Sravya	Female	26	Hyderabad	69
14	Sruthi	Female	26	Hyderabad	51
15	Swetha	Female	21	Bangalore	34
16	Aishwarya	Female	24	Bangalore	40
17	Madhavi	Female	29	Mumbai	18
18	Rajini	Female	27	Mumbai	35

19	Haritha	Female	24	Hyderabad	29
20	Swathi	Female	25	Hyderabad	68

2. ADD Columns

ALTER TABLE <table_name> ADD COLUMNS (<col-name> <data-type> COMMENT ”, <col-name> <data-type> COMMENT ”, …..)

hive> alter table sales2 add columns(phonenumber int);

OK

Time taken: 0.159 seconds

hive> describe sales2;

OK

salesexecutive int

name string

gender string

age int

location string

salesrs int

phonenumber int

Time taken: 0.061 seconds, Fetched: 7 row(s)

3. CHANGE Column

ALTER TABLE <table_name> CHANGE <column_name> <new_column_name> <new_data_type>;

hive> ALTER TABLE sales2 CHANGE Name Sname STRING;

OK

Time taken: 0.07 seconds

hive> describe sale2;

hive> describe sales2;

OK

salesexecutive int

sname string

gender string

age int

location string

salesrs int

phonenumber int

Time taken: 0.064 seconds, Fetched: 7 row(s)

4.REPLACE Column

ALTER TABLE <table_name> REPLACE COLUMNS (<attribute_name> <data_type>,<attribute_name> <data_type>,

..

);

hive> ALTER TABLE sales2 REPLACE COLUMNS (Sname STRING);

OK

Time taken: 0.071 seconds

Drop Table

DROP TABLE [IF EXISTS] table_name [PURGE];

hive> create table std(name string, age int);

OK

Time taken: 0.085 seconds

hive> drop table std;

OK

Time taken: 0.42 seconds

hive>

HiveQL – Functions

Employee_data.csv

Id	Name	Salary
1	K Krihsna	98000
2	G Gopal	90900
3	K Srinivas	110000
4	G Ramesh	120000
5	H.hema	110000
6	D.Srinivasa	60000
7	K Swapna	50000
8	R Rama Rao	55000

Step1:Load the employee_data.csv in hdfs

[cloudera@quickstart ~]$ hadoop fs -put /home/cloudera/Desktop/employee_data.csv /user/cloudera/sample1/

1. hive> create table employee_data (Id int, Name string , Salary float) row format deli mited fields terminated by ',' ;
2. **Now, load the data into the table.**

   hive> load data inpath '/user/cloudera/sample1/employee_data.csv' into tabl e employee_data;

hive> select * from employee_data;

```
OK
NULL  Name  NULL
1     K Krihsna     98000.0
2     G Gopal       90900.0
3     K Srinivas    110000.0
4     G Ramesh      120000.0
5     H.hema        110000.0
6     D.Srinivasa   60000.0
7     K Swapna      50000.0
8     R Rama Rao    55000.0
Time taken: 0.03 seconds, Fetched: 9 row(s)
```

hive> select Id, Name, sqrt(Salary) from employee_data;

```
OK
NULL   Name   NULL
```

1 K Krihsna 313.04951684997053
2 G Gopal 301.4962686336267
3 K Srinivas 331.66247903554
4 G Ramesh 346.41016151377545
5 H.hema 331.66247903554
6 D.Srinivasa 244.94897427831782
7 K Swapna 223.60679774997897
8 R Rama Rao 234.5207879911715
Time taken: 0.087 seconds, Fetched: 9 row(s)

hive> select max(Salary) from employee_data;

MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 1.73 sec HDFS Read: 7800 HDFS Write: 9 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 730 msec
OK
120000.0
Time taken: 21.901 seconds, Fetched: 1 row(s)

hive> select min(Salary) from employee_data;
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 1.94 sec HDFS Read: 7863 HDFS Write: 8 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 940 msec
OK
50000.0
Time taken: 21.604 seconds, Fetched: 1 row(s)

hive> select Id, upper(Name) from employee_data;
OK
NULL NAME
1 K KRIHSNA
2 G GOPAL
3 K SRINIVAS
4 G RAMESH

5 H.HEMA

6 D.SRINIVASA

7 K SWAPNA

8 R RAMA RAO

Time taken: 0.047 seconds, Fetched: 9 row(s)

Hive - View and Indexes

Creating a View

hive> CREATE VIEW employee_data1 AS SELECT * FROM employee_data WHERE salary>3000;

OK

Time taken: 0.122 seconds

hive> select *from employee_data1;

OK

1 K Krihsna 98000.0

2 G Gopal 90900.0

3 K Srinivas 110000.0

4 G Ramesh 120000.0

5 H.hema 110000.0

6 D.Srinivasa 60000.0

7 K Swapna 50000.0

8 R Rama Rao 55000.0

Time taken: 0.07 seconds, Fetched: 8 row(s)

Dropping a View

Time taken: 0.07 seconds, Fetched: 8 row(s)
hive> DROP VIEW employee_data1;
OK
Time taken: 0.08 seconds

Creating an Index

hive> CREATE INDEX inedx_salary ON TABLE employee_data(salary) AS
'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler' WITH DEFERRED REBUILD ;

OK

Time taken: 0.285 seconds

hive>

hive> SHOW INDEX ON employee_data;

OK

inedx_salary employee_data salary
sales1__employee_data_inedx_salary__ compact

Time taken: 0.135 seconds, Fetched: 1 row(s)

Dropping an Index

hive> DROP INDEX inedx_salary ON employee_data;

OK

Time taken: 0.157 seconds

hive>

TASK 9: Writing User Defined Functions in Hive

Aim: To implement Hive UDF using python

Procedure:

Step1: Create UDF function using python dcry.py ,encry.py and save in cloudera home

vi decry.py

```
import base64
import sys

try:
    for line in sys.stdin:
        line=line.strip()
        (a,b)=line.split('\t')
        print('\t'.join([a,b,base64.b64decode(bytes(a.replace('\n',''),encoding='utf-8')).decode()]))
except:
    print(sys.exc_info())
```

vi encry.py

```
import base64
import sys

try:
    for line in sys.stdin:
        line=line.strip()
        (a,b)=line.split('\t')
        print('\t'.join([a,b,base64.b64encode(bytes(a.replace('\n',''),encoding='utf-8')).decode()]))
except:
    print(sys.exc_info())
```

Step1 2:

Connect t Hive and do the fallowing steps:

Testing on Database

```
Hive>create database test;
  Hive>use test;
 Hive> create table abc (id integer,name string);
  Hive>insert into abc values(1,'vijay');
 Hive> insert into abc values(2,'vaasista');
  Hive>add FILE /hadoop/encry.py;
  Hive>add FILE /hadoop/decry.py;
  Hive>select transform(id) using 'python3 encry.py' as id from abc;
  Hive>select transform(id,name) using 'python3 encry.py' as id,name,ct from abc;

Hive>select base64(encode(regexp_replace('My test \n String','\\n',''), 'UTF-8'))
```

Result:

```
TXkgdGVzdCAgU3RyaW5n
```

Decode:

```
Hive>select decode(unbase64('TXkgdGVzdCAgU3RyaW5n'), 'UTF-8')
```

Result:

```
My test  String
```

TASK 10: Understanding the processing of large dataset on Spark framework.

Aim: To understanding about creating spark application and evaluate data dataset

Procedure:

Dataset: **Air Quality in Madrid**

- **Moving HDFS** (Hadoop Distributed File System) **files** using Python.
- **Loading Data from HDFS** into a Data Structure like a **Spark** or **pandas** DataFrame in order to make calculations.
- **Write** the **results** of an analysis back to HDFS**.**

Step1: PySpark and findspark installation

First of all, install findspark, a library that will help you to integrate Spark into your Python workflow, and also pyspark in case you are working in a local computer and not in a proper Hadoop cluster.

Step2: Spark setup with findspark

Code for both **local** and **cluster** mode is provided here, uncomment the line you need and adapt paths depending on your particular infrastructure and library versions (cloudera Spark path should be pretty similar to the one provided here):

```
import findspark
# Local Spark
findspark.init('/home/cloudera/miniconda3/envs/<your_environment_name>/lib/python3.7/site-packages/pyspark/')
# Cloudera Cluster Spark
findspark.init(spark_home='/opt/cloudera/parcels/SPARK2–2.3.0.cloudera4–1.cdh5.13.3.p0.611179/lib/spark2/')
```

Step3: Creating a Spark application

Once Spark is initialized, we have to create a Spark application, execute the following code, and make sure you specify the master you need, like 'yarn' in the case of a proper Hadoop cluster, or 'local[*]' in the case of a fully local setup:

```
from pyspark.sql import SparkSession
```

spark = SparkSession.builder.appName(‘example_app’).master(‘yarn’).getOrCreate()

Step4: PySpark Recipes and Use Cases

Listing Hive databases

```
spark.sql("show databases").show()
+------------+
|databaseName|
+------------+
|         db1|
|     default|
|     fhadoop|
+------------+
```

Transform pandas DataFrame into a Spark DataFrame

First, let’s load a pandas DataFrame. This one is about Air Quality in Madrid (just to satisfy your curiosity, but not important with regards to moving data from one place to another one). You can download it here. Make sure you install the library pytables to read hdf5 formatted data.

```
import pandas as pd
air_quality_df = pd.read_hdf(‘data/air_quality/air-quality-madrid/madrid.h5’, key=’28079008')
air_quality_df.head()
```

This data is a time series for many well known pollutants like NOX, Ozone, and more:

date	BEN	CH4	CO	...
2001-07-01 01:00:00	30.65	NaN	6.91	...
2001-07-01 02:00:00	29.59	NaN	2.59	...
2001-07-01 03:00:00	4.69	NaN	0.76	...
2001-07-01 04:00:00	4.46	NaN	0.74	...
2001-07-01 05:00:00	2.18	NaN	0.57	...

Let’s make some changes to this DataFrame, like resetting datetime index to avoid losing information when loading into Spark. Datetime column will also be transformed to string as Spark has some issues working with dates (related to system locale, timezones, and so on) unless further configuration depending on your locale.

```
air_quality_df.reset_index(inplace=True)
air_quality_df[‘date’] = air_quality_df[‘date’].dt.strftime(‘%Y-%m-%d %H:%M:%S’)
```

We can simply load from pandas to Spark with createDataFrame:

```
air_quality_sdf = spark.createDataFrame(air_quality_df)
```

Once DataFrame is loaded into Spark (as air_quality_sdf here), can be manipulated easily using PySpark DataFrame API:

```
air_quality_sdf.select('date', 'NOx').show(5)
```

Output

```
+— — — — — — — — — -+ — — — — — — — — — +
|               date|               NOx|
+ — — — — — — — — — + — — — — — — — — — +
|2001–07–01 01:00:00|            1017.0|
|2001–07–01 02:00:00| 409.20001220703125|
|2001–07–01 03:00:00| 143.39999389648438|
|2001–07–01 04:00:00|  149.3000030517578|
|2001–07–01 05:00:00| 124.80000305175781|
+ — — — — — — — — — + — — — — — — — — — +
only showing top 5 rows
```

Create Hive table from Spark DataFrame

To persist a Spark DataFrame into HDFS, where it can be queried using default Hadoop SQL engine (Hive), one straightforward strategy (not the only one) is to create a temporal view from that DataFrame:

```
air_quality_sdf.createOrReplaceTempView("air_quality_sdf")
```

Once the temporal view is created, it can be used from Spark SQL engine to create a real table using create table as select. Before creating this table, I will create a new database called analytics to store it:

```
sql_create_database = """
create database if not exists analytics
location '/user/cloudera/analytics/'
"""
result_create_db = spark.sql(sql_create_database)
```

Then, we can create a new table there:

```
sql_create_table = """
create table if not exists analytics.pandas_spark_hive
using parquet as select
to_timestamp(date) as date_parsed, *
from air_quality_sdf
"""
result_create_table = spark.sql(sql_create_table)
```

Reading data from Hive table using PySpark

Once we have created our Hive table, can check results using Spark SQL engine to load results back, for example to select ozone pollutant concentration over time:

```
spark.sql("select * from analytics.pandas_spark_hive") \ .select("date_parsed", "O_3").show(5)
```

Output :

```
+ — — — — — — — — — — + — — — — — — — — — — +
| date_parsed       |               O_3|
+ — — — — — — — — — — + — — — — — — — — — — +
|2001–07–01 01:00:00|  9.010000228881836|
|2001–07–01 02:00:00|  23.81999969482422|
|2001–07–01 03:00:00| 31.059999465942383|
|2001–07–01 04:00:00| 23.780000686645508|
|2001–07–01 05:00:00| 29.530000686645508|
+ — — — — — — — — — — + — — — — — — — — — — +
only showing top 5 rows
```

TASK 11: Ingesting structured and unstructured data using Sqoop, Flume

Aim: To implement data ingestion in sqoop and Flume

Procedure:

Sqoop: Import and Export

Sqoop – IMPORT Command

Import command is used to importing a table from relational databases to HDFS. In our case, we are going to import tables from MySQL databases to HDFS.

```
mysql> use employees;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables
    -> ;
+----------------------+
| Tables_in_employees  |
+----------------------+
| current_dept_emp     |
| departments          |
| dept_emp             |
| dept_emp_latest_date |
| dept_manager         |
| employees            |
| expected_values      |
| found_values         |
| salaries             |
| tchecksum            |
| titles               |
+----------------------+
```

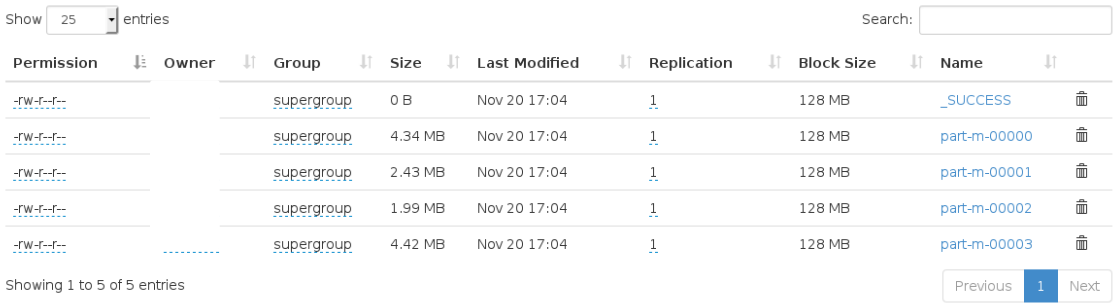
The command for importing table is:

[cloudera@localhost~]$sqoop import --connect jdbc:mysql://localhost/employees --username cloudera --table employees

As you can see in the below image, after executing this command Map tasks will be executed at the back end.

```
17/11/20 17:04:48 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 of write 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
```

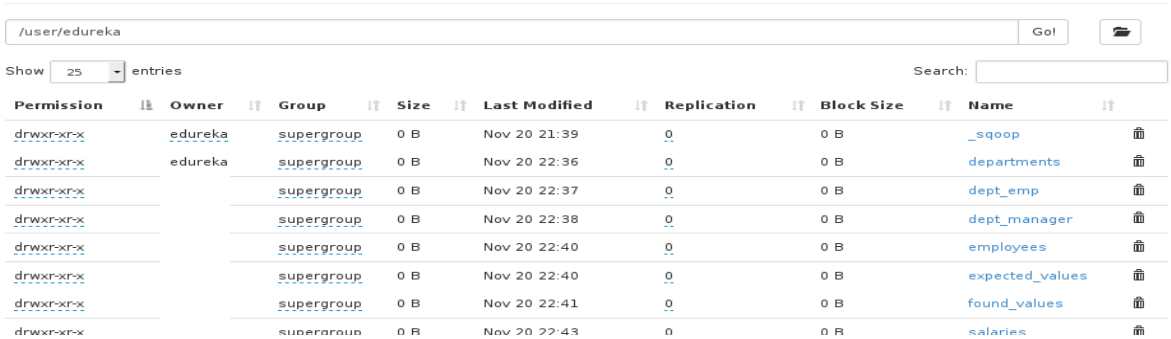
After the code is executed, you can check the Web UI of HDFS i.e. localhost:50070 where the data is imported.



Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--		supergroup	0 B	Nov 20 17:04	1	128 MB	_SUCCESS
-rw-r--r--		supergroup	4.34 MB	Nov 20 17:04	1	128 MB	part-m-00000
-rw-r--r--		supergroup	2.43 MB	Nov 20 17:04	1	128 MB	part-m-00001
-rw-r--r--		supergroup	1.99 MB	Nov 20 17:04	1	128 MB	part-m-00002
-rw-r--r--		supergroup	4.42 MB	Nov 20 17:04	1	128 MB	part-m-00003

Sqoop – Import All Tables

[cloudera@localhost~]$sqoop import-all-tables --connect jdbc:mysql://localhost/employees --username cloudera



Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	edureka	supergroup	0 B	Nov 20 21:39	0	0 B	_sqoop
drwxr-xr-x	edureka	supergroup	0 B	Nov 20 22:36	0	0 B	departments
drwxr-xr-x		supergroup	0 B	Nov 20 22:37	0	0 B	dept_emp
drwxr-xr-x		supergroup	0 B	Nov 20 22:38	0	0 B	dept_manager
drwxr-xr-x		supergroup	0 B	Nov 20 22:40	0	0 B	employees
drwxr-xr-x		supergroup	0 B	Nov 20 22:40	0	0 B	expected_values
drwxr-xr-x		supergroup	0 B	Nov 20 22:41	0	0 B	found_values
drwxr-xr-x		supergroup	0 B	Nov 20 22:43	0	0 B	salaries

Sqoop – Expo

first we are creating an empty table, where we will export our data.

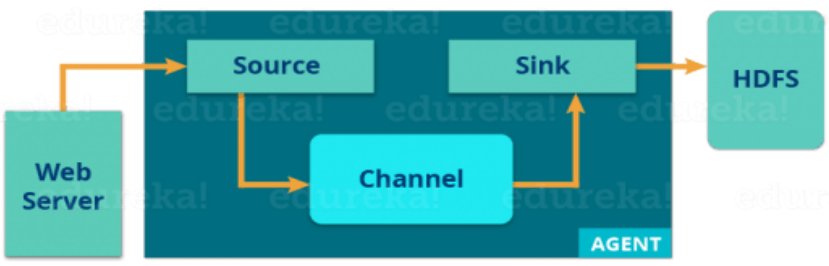
```
mysql> CREATE TABLE emp(id INT NOT NULL PRIMARY KEY, birth_date date, f_name var
char(14), l_name varchar(16), gender varchar(3), hire_date date);
Query OK, 0 rows affected (0.10 sec)
```

The command to export data from HDFS to the relational database is:

[cloudera@localhost~]$sqoop export --connect jdbc:mysql://localhost/employees --username edureka --table emp --export-dir /user/cloudera/employees

```
mysql> select count(*) from emp;
+-----------+
| count(*)  |
+-----------+
|    300025 |
+-----------+
1 row in set (0.13 sec)
```

Apache Flume



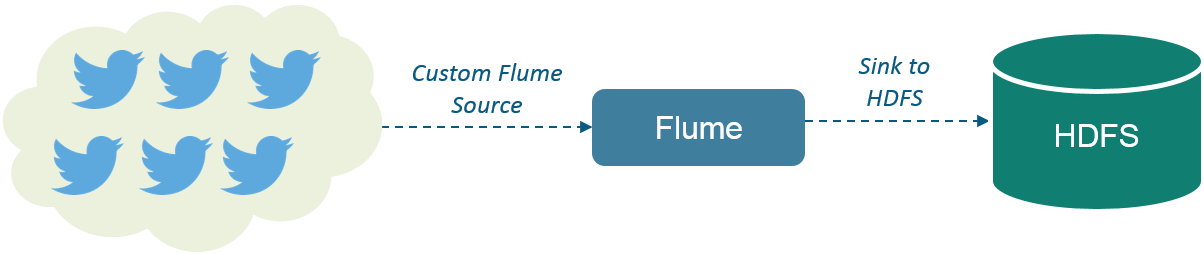
The flume agent has 3 components: source, sink and channel.

Source: It accepts the data from the incoming streamline and stores the data in the channel.

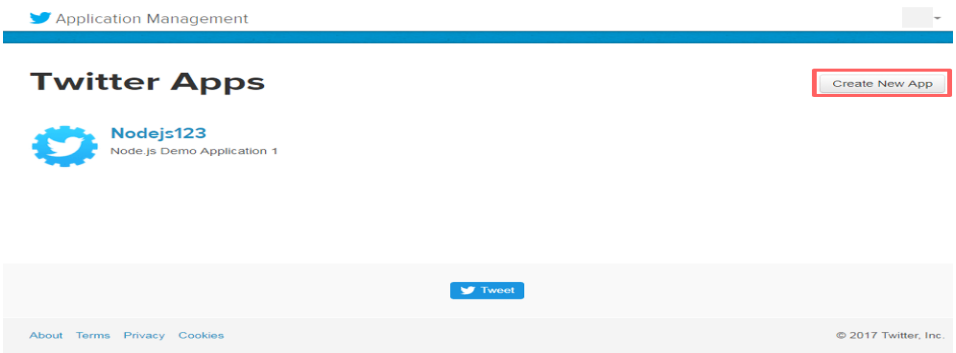
Channel: In general, the reading speed is faster than the writing speed. Thus, we need some buffer to match the read & write speed difference. Basically, the buffer acts as a intermediary storage that stores the data being transferred temporarily and therefore prevents data loss. Similarly, channel acts as the local storage or a temporary storage between the source of data and persistent data in the HDFS.

Sink: Then, our last component i.e. Sink, collects the data from the channel and commits or writes the data in the HDFS permanently.

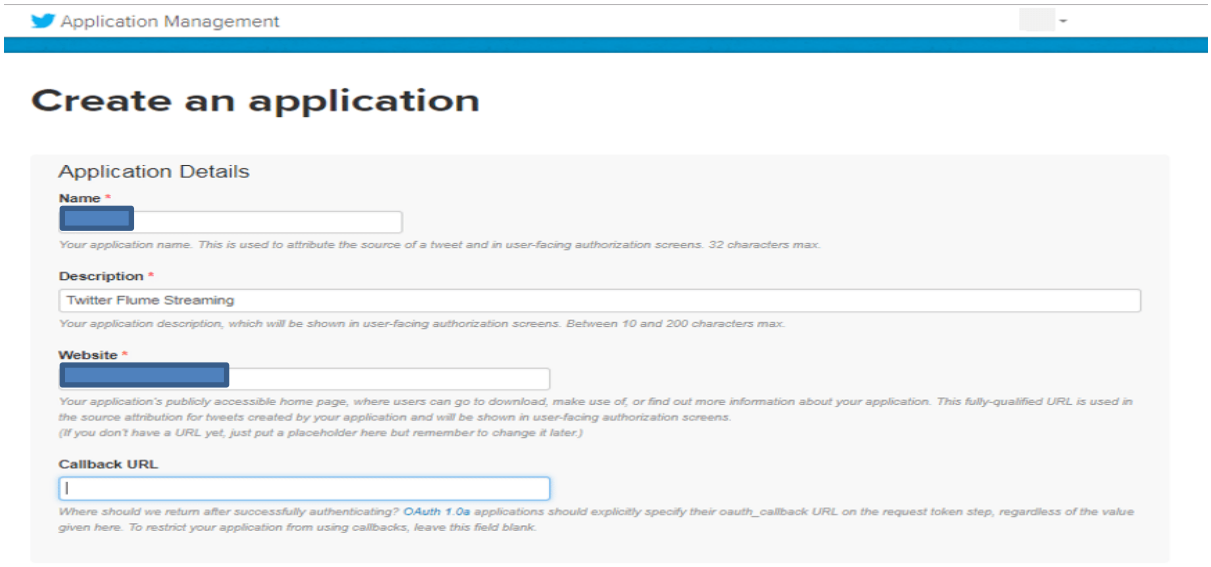
Apache Flume : Streaming Twitter Data



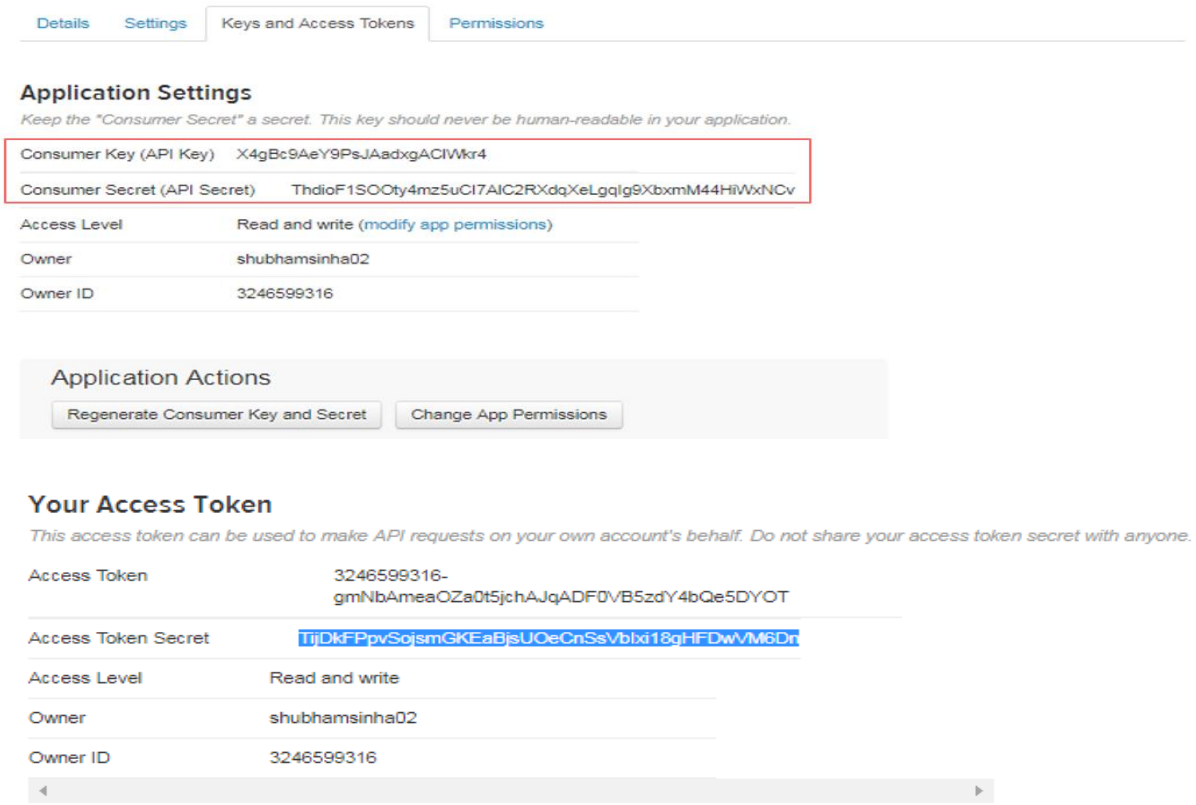
The first step is to create a Twitter application. For this, you first have to go to this url: https://apps.twitter.com/ and sign in to your Twitter account. Go to create application tab as shown in the below image.



Then, create an application as shown in the below image.



After creating this application, you will find Key & Access token. Copy the key and the access token. We will pass these tokens in our Flume configuration file to connect to this application.



Now create a flume.conf file in the flume's root directory as shown in the below image. As we discussed, in the Flume's Architecture, we will configure our Source, Sink and Channel. Our Source is Twitter, from where we are streaming the data and our Sink is HDFS, where we are writing the data.

Now create a flume.conf file in the flume's root directory as shown in the below image. As we discussed, in the Flume's Architecture, we will configure our Source, Sink and Channel. Our Source is Twitter, from where we are streaming the data and our Sink is HDFS, where we are writing the data.

```
[[redacted]@localhost ~]$ cd $FLUME_HOME
[[redacted]@localhost apache-flume-1.7.0-bin]$ sudo gedit flume.conf
```

In source configuration we are passing the Twitter source type as org.apache.flume.source.twitter.TwitterSource. Then, we are passing all the four tokens which we received from Twitter. At last in source configuration we are passing the keywords on which we are going to fetch the tweets.

In the Sink configuration we are going to configure HDFS properties. We will set HDFS path, write format, file type, batch size etc. At last we are going to set memory channel as shown in the below image.

```
TwitterAgent.sources = Twitter
TwitterAgent.channels = MemChannel
TwitterAgent.sinks = HDFS

TwitterAgent.sources.Twitter.type = org.apache.flume.source.twitter.TwitterSource
TwitterAgent.sources.Twitter.channels = MemChannel
TwitterAgent.sources.Twitter.consumerKey = X4gBc9AeY9PsJAadxgAClWkr4
TwitterAgent.sources.Twitter.consumerSecret =   ThdioF1SOOty4mz5uCI7AlC2RXdqXeLgqIg9XbxmM44HiWxNCv
TwitterAgent.sources.Twitter.accessToken = 3246599316-gmNbAmeaOZa0t5jchAJqADF0VB5zdY4bQe5DYOT
TwitterAgent.sources.Twitter.accessTokenSecret = TijDkFPpvSojsmGKEaBjsUOeCnSsVbIxi18gHFDwVM6Dn
TwitterAgent.sources.Twitter.keywords = hadoop, big data, analytics, bigdata, cloudera, data science, data scientist, business intelligence,
mapreduce, data warehouse, data warehousing, mahout, hbase, nosql, newsql, businessintelligence, cloudcomputing

TwitterAgent.sinks.HDFS.channel = MemChannel
TwitterAgent.sinks.HDFS.type = hdfs
TwitterAgent.sinks.HDFS.hdfs.path = hdfs://localhost:9000/user/flume/tweets/
TwitterAgent.sinks.HDFS.hdfs.fileType = DataStream
TwitterAgent.sinks.HDFS.hdfs.writeFormat = Text
TwitterAgent.sinks.HDFS.hdfs.batchSize = 1000
TwitterAgent.sinks.HDFS.hdfs.rollSize = 0
TwitterAgent.sinks.HDFS.hdfs.rollCount = 10000
TwitterAgent.sinks.HDFS.hdfs.rollInterval = 600

TwitterAgent.channels.MemChannel.type = memory
TwitterAgent.channels.MemChannel.capacity = 10000
TwitterAgent.channels.MemChannel.transactionCapacity = 100
```

Source Properties

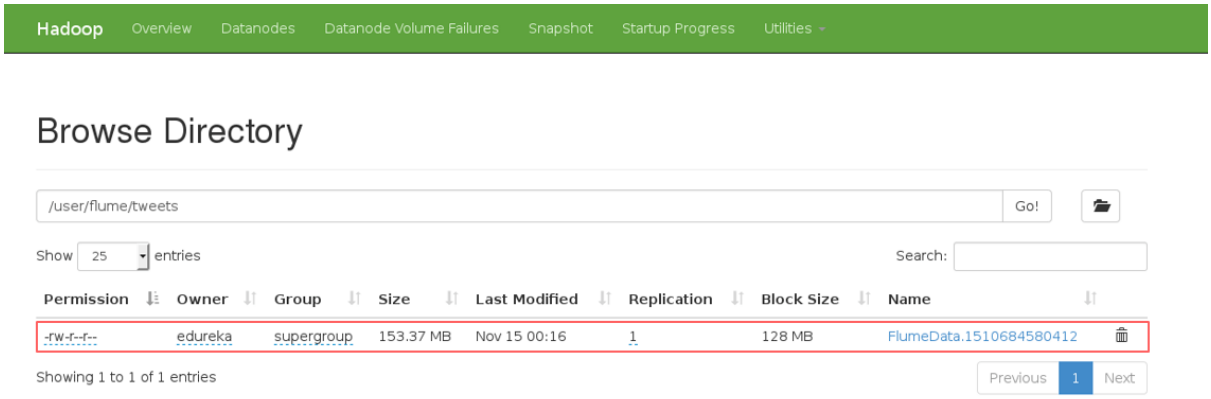
Sink Properties

Channel Properties

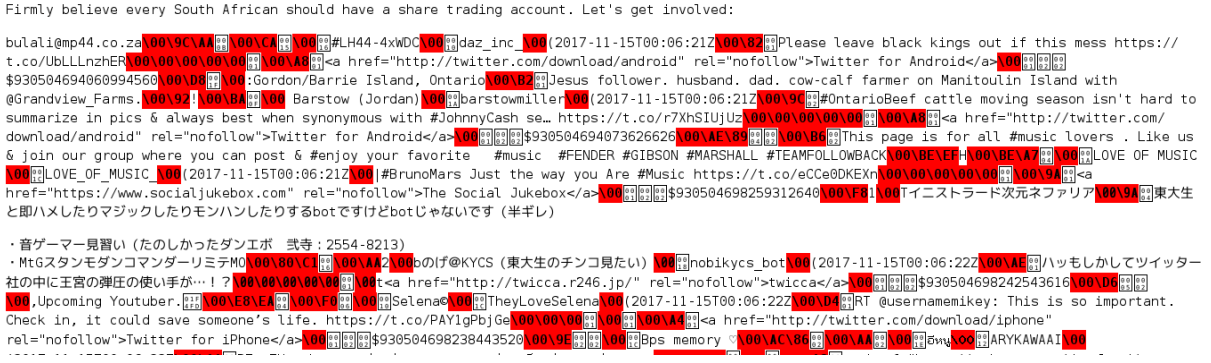
Now we are all set for execution. Let us go ahead and execute this command:

[cloudera@localhost~]$FLUME_HOME/bin/flume-ng agent --conf ./conf/ -f $FLUME_HOME/flume.conf

After executing this command for a while, and then you can exit the terminal using CTRL+C. Then you can go ahead in your Hadoop directory and check the mentioned path, whether the file is created or not.



Download the file and open it. You will get something as shown in the below image.



TASK 12: Integrating Hadoop with other data analytic framework like R

Aim: To integrate the Hadoop with R

Procedure: Integration of Hadoop and R Programming

The Main Motive behind R and Hadoop Integration :

No suspicion, that R is the most picked programming language for statistical computing, graphical analysis of data, data analytics, and data visualization. On the other hand, Hadoop is a powerful Bigdata framework that is capable to deal with a large amount of data. In all the processing and analysis of data the distributed file system(HDFS) of Hadoop plays a vital role, It applies the map-reduce processing approach during data processing(provides by rmr package of R Hadoop), Which make the data analyzing process more efficient and easier.

What would happen, if both collaborated with each other? Obviously, the efficiency of the data management and analyzing process will get increase multiple times. So, in order to have efficiency in the process of data analytics and visualization process, we have to combine R with Hadoop.

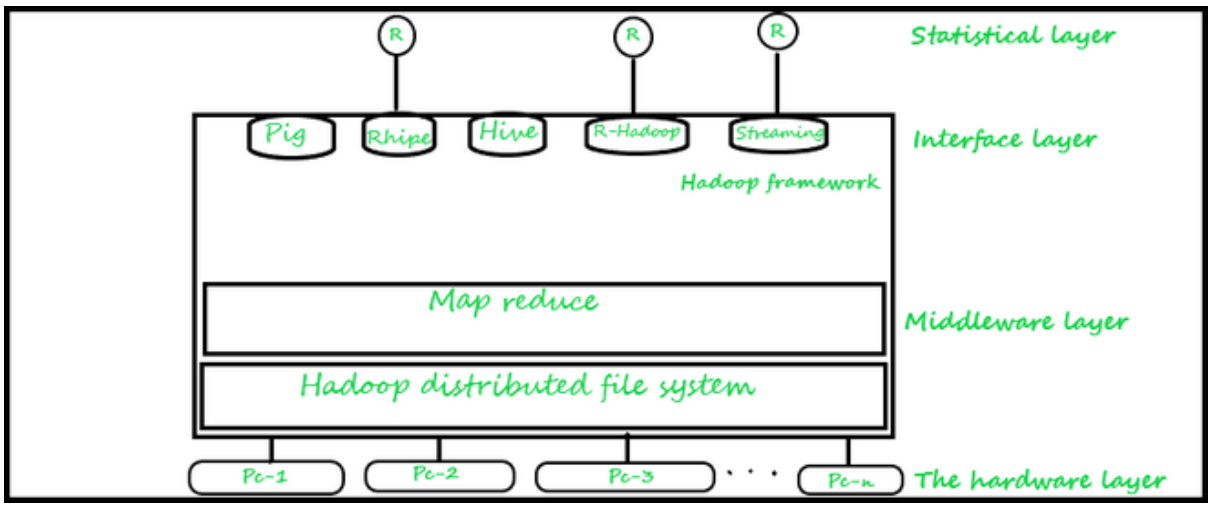
After joining these two technologies, R's statistical computing power becomes increases, then we enable to :

Use Hadoop for the execution of the R codes.

Use R for accessing the data stored in Hadoop.

Several ways using which One can Integrate both R and Hadoop:

The most popular and frequently picked methods are shown below but there are some other RODBC/RJDBC that could be used but are not popular as the below methods are. The general architecture of the analytics tools integrated with Hadoop is shown below along with its different layered structure as follows.



a) R Hadoop: R Hadoop method includes four packages, which are as follows:

The rmr package –rmr package provides Hadoop MapReduce functionality in R. So, the R programmer only has to do just divide the logic and idea of their application into the map and reduce phases associates and just submit it with the rmr methods. After that, The rmr package makes a call to the Hadoop streaming and the MapReduce API through multiple job parameters as input directory, output directory, reducer, mapper, and so on, to perform the R MapReduce job over Hadoop cluster(most of the components are similar as Hadoop streaming).

The rhbase package –Allows R developer to connect Hadoop HBASE to R using Thrift Server. It also offers functionality like (read, write, and modify tables stored in HBase from R).

A script that utilizes the RHadoop functionality looks like below as follows.

```
library(rmr)

map<-function(k,v){...}

reduce<-function(k,vv){...}

mapreduce(

input = "data.txt",

output ="output",

textinputformat = rawtextinputformat,

map = map,

reduce = reduce

)
```

The rhdfs package –It provides HDFS file management in R, because data itself stores in Hadoop file system. Functions of this package are as given as follows. File Manipulations -(hdfs.delete, hdfs.rm, hdfs.del, hdfs.chown, hdfs.put, hdfs.get etc), File Read/Write -(hdfs.flush, hdfs.read, hdfs.seek, hdfs.tell, hdfs.line.reader etc), Directory -hdfs.dircreate, hdfs.mkdir, Initialization: hdfs.init, hdfs.defaults.

The dplyr package –It provides functionality likes data manipulation, summaries of the output result, performing set operations(union, intersection, subtraction, merge, unique).

b) RHIPE: Rhipe is used in R to do an intricate analysis of the large collection of data sets via Hadoop is an integrated programming environment tool that is brought by the Divide and Recombine (D & R) to analyze the huge amount of data.

RHIPE = R and Hadoop Integrated Programming Environment

RHIPE is a package of R that enables the use of API in Hadoop. Thus, this way we can read, save the complete data that is created using RHIPE MapReduce. RHIPE is deployed with many features that help us to effectively interact with HDFS. An individual can also use various languages like Perl, Java, or Python to read data sets in RHIPE. The general structure of the R script that uses Rhipe is shown below as follows.

```
library(Rhipe)

rhint(TRUE, TRUE);
map < -expression({lapply(map.values, function(mapper)...)})
reduce < -expression(
pre={...},
reduce={...},
post={...}, }
x < - rhmr(
map=map, reduce=reduce,
ifolder=inputPath,
ofolder=outputPath,
inout=c('text', 'text'),
jobname='a job name'))
rhex(z)
```

```
$ ${HADOOP_HOME}/bin/Hadoop jar
```

```
$ {HADOOP_HOME}/contrib/streaming/*.jar\
-inputformat org.apache.hadoop.mapred.TextInputFormat \
-input input_data.txt \
-output output \
-mapper /home/tst/src/map.R \
-reducer /home/tst/src/reduce.R \
-file /home/ts/src/map.R \
-file /home/tst/src/reduce.R
```