

List of Experiments

B.E. (Comp.) Sem VII

(July-Oct 2021)

Subject: CSL704: Computational Lab-1 (BDA)

Expt No.	Experiments	Page number	Date
1	Experiment of HDFS Commands	3	11-08-2021
2	Use of Sqoop to transfer data between Hadoop and Relational Database servers	7	18-08-2021
3	Programming exercises in HBASE	15	25-08-2021
4	Experiment for Word Counting using Hadoop Map-Reduce	19	01-09-2021
5	Implementing algorithms for page rank using Java	23	08-10-2021
6	Experiment on pig	27	15-09-2021
7	Create HIVE Database and Descriptive analytics (basic statistics)	47	22-09-2021
8	Write a java program to calculate distance using following distance formulae - L1 norm (Manhattan distance), L2 norm (Euclidean distance), L_∞ norm, Jaccard distance, Cosine distance, Edit distance, Hamming Distance	52	29-09-2021
9	Experiment on k-means clustering using Hadoop Map- Reduce	60	06-10-2021
10	Case study of Big Data Solution for Business Analytics	65	13-10-2021
	Written ASSIGNMENTS		
11	Write briefly on big data characteristics, Hadoop architecture, Hadoop Ecosystem	69	15-09-2021
12	Mining Social-Network Graphs <ul style="list-style-type: none">clustering of social graph using Nirvan-Newman algorithmDirect discovery of communities in social graph using Clique percolation method(CPM)	78	29-09-2021

Experiment No. 1

Aim: Experiment on HDFS commands

Theory:

The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets. HDFS relaxes a few POSIX requirements to enable streaming access to file system data. HDFS was originally built as infrastructure for the Apache Nutch web search engine project. HDFS is now an Apache Hadoop subproject.

HDFS Commands:

ls:

This command is used to list all the files. Use lsr for the recursive approach. It is useful when we want a hierarchy of a folder.

df:

You can check the free space in an HDFS directory with a couple of commands. The -df command shows the configured capacity, available free space and used space of a file system in HDFS.

count:

Count the number of directories, files and bytes under the paths that match the specified file pattern.

fsck:

The fsck Hadoop command is used to check the health of the HDFS.

mkdir:

Takes path URI's as argument and creates directories. The -p option behaviour is much like Unix mkdir -p, creating parent directories along the path.

put:

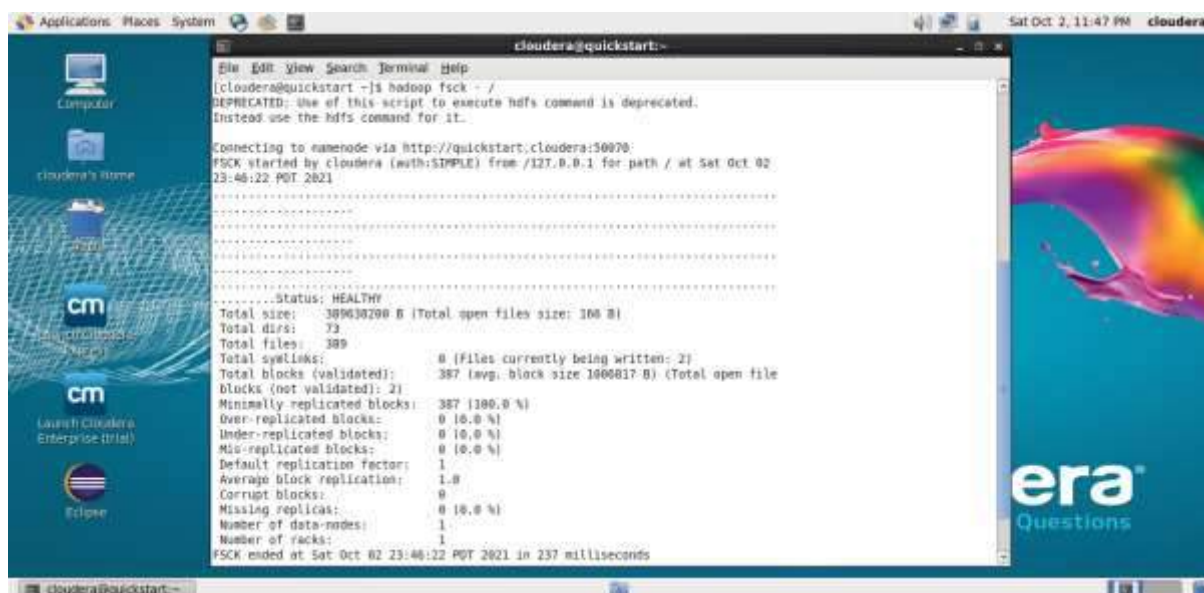
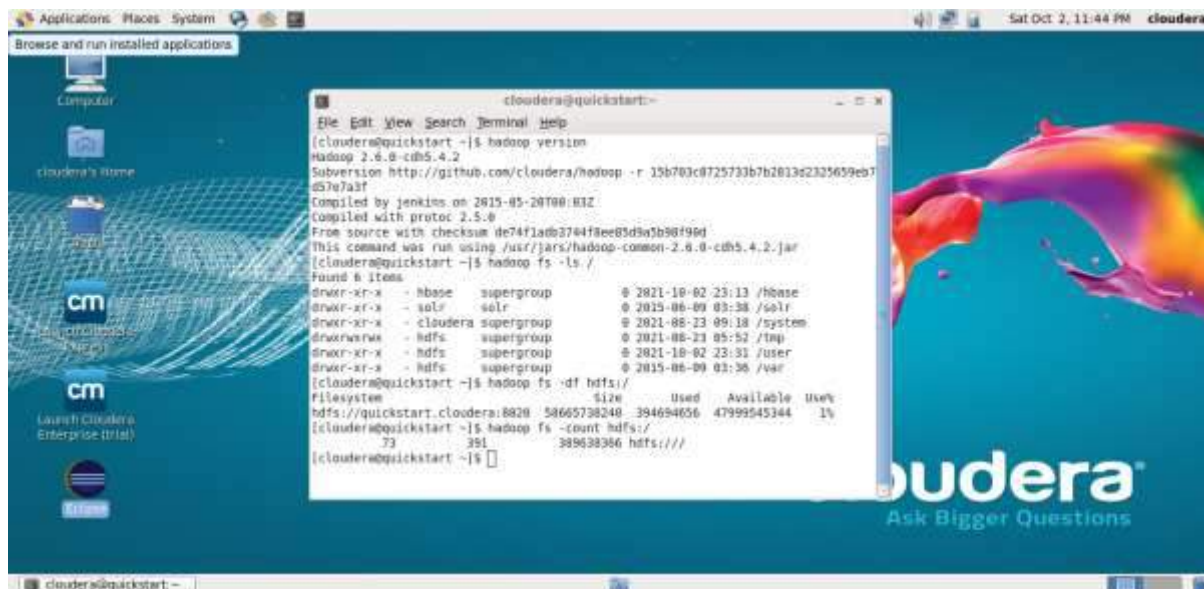
Copy single src, or multiple srcs from the local file system to the destination file system. Also reads input from stdin and writes to the destination file system if the source is set to "-"

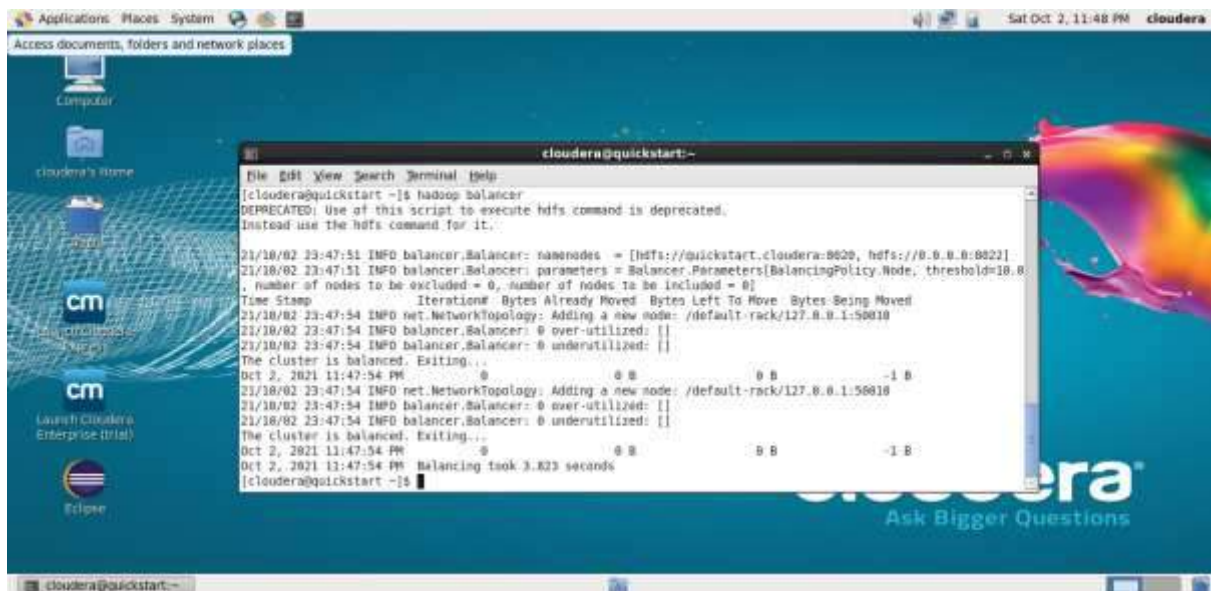
Copying fails if the file already exists unless the -f flag is given.

balancer:

Runs a cluster balancing utility. An administrator can simply press Ctrl-C to stop the rebalancing process. See Balancer for more details.

Cerate user -





```

cloudera@quickstart:~$ su
Password:
[root@quickstart cloudera]# sudo -u hdfs hadoop fs -chown root:root /user/jeet/training/hadoop/sample.txt
[root@quickstart cloudera]# sudo -u hdfs hadoop fs -chgrp training /user/jeet/training/hadoop/sample.txt
[root@quickstart cloudera]# hadoop fs -ls /user/jeet/training/hadoop/sample.txt
-rw-r--r-- 1 root training      48 2021-10-03 00:04 /user/jeet/training/hadoop/sample.txt

```

```
[root@quickstart ~]# hadoop fs -setrep -w 2 apache_hadoop/sample.txt
Application 2 set: apache_hadoop/sample.txt
Waiting for apache_hadoop/sample.txt ... done
[root@quickstart ~]# hadoop fs -mv apache_hadoop /user/training/hadoop
[root@quickstart ~]# hadoop fs -copyToLocal /user/training/hadoop/sample.txt/home/
copyToLocal: '/user/training/hadoop/sample.txt/home/': No such file or directory
[root@quickstart ~]# hadoop fs -copyToLocal /user/training/hadoop/sample.txt /home/
[root@quickstart ~]# sudo nano /home/sample.txt
```

```
[root@quickstart ~]# hadoop fs -copyFromLocal /home/sample.txt /user/training/hadoop/
[root@quickstart ~]# hadoop fs -cat /user/training/hadoop/sample.txt
Hello
How are you

[root@quickstart ~]# hadoop fs -tail /user/training/hadoop/sample.txt
Hello
How are you
```

Conclusion: Implemented all the HDFS commands successfully.

Experiment No. 2

Aim: Using Sqoop tool to transfer data between Hadoop and MySQL

Theory:

Sqoop

Sqoop is a tool designed to transfer data between Hadoop and relational databases servers. It is used to import data from relational databases such as MySQL, Oracle to Hadoop HDFS, and export from the Hadoop file system to relational databases.

Sqoop Commands:

1. List Table

This command lists the particular table of the database in MYSQL server.

```
sqoop list -tables --connect jdbc:mysql://localhost/<dbname> --username <username>
```

2. Import in target directory.

This command import table in a specific directory in HDFS. -m denotes mapper argument. They have an integer value.

```
sqoop import --connect jdbc:mysql://youripaddress:3306/<dbname> --username root --password cloudera  
--table <table_name> --targetdir=<target_directory> -m 1
```

3. Sqoop eval

This command runs quickly SQL queries of the respective database.

```
sqoop eval --connect --query "SQLQuery"
```

4. Sqoop version

This displays the installed version of sqoop.

```
sqoop version sqoop {revnumber}
```

5. Sqoop-job.

This command allows us to create a job, the parameters that are created can be invoked at any time. They take options like (-create, -delete, -show, -exit).

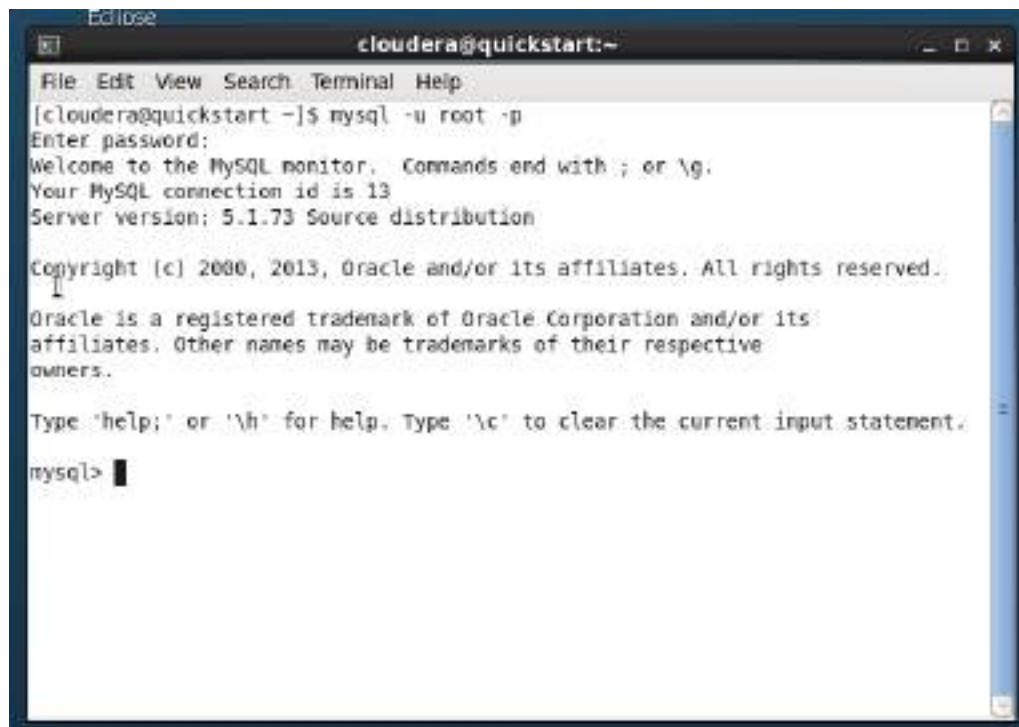
```
sqoop job --create --import --connect --table
```

Implementation

•MySQL

i.) Open the Cloudera Terminal and execute the following command in order to startMySQL server. (Note: The default password is **cloudera** for root user)

```
mysql -u root -p
```



```
cloudera@quickstart:~  
File Edit View Search Terminal Help  
[cloudera@quickstart ~]$ mysql -u root -p  
Enter password:  
Welcome to the MySQL monitor.  Commands end with ; or \g.  
Your MySQL connection id is 13  
Server version: 5.1.73 Source distribution  
  
Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.  
  
Oracle is a registered trademark of Oracle Corporation and/or its  
affiliates. Other names may be trademarks of their respective  
owners.  
  
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.  
mysql>
```

ii.) Creating a database

create database student

```
mysql> create database student;
Query OK, 1 row affected (0.01 sec)

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| cm |
| firehose |
| hue |
| metastore |
| mysql |
| nav |
| navms |
| pozie |
| retail_db |
| rman |
| sentry |
| student |
+-----+
13 rows in set (0.11 sec)
```

iii.) Creating a Table

(Note: The database must be in use before you create a table.)

```
mysql> use student;
Database changed
mysql> create table sinfo(RollNo int, fname varchar(20), lname varchar(20));
Query OK, 0 rows affected (0.09 sec)

mysql> describe sinfo
-> ;
+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+
| RollNo | int(11)       | YES  |     | NULL    |       |
| fname  | varchar(20)   | YES  |     | NULL    |       |
| lname  | varchar(20)   | YES  |     | NULL    |       |
+-----+
3 rows in set (0.00 sec)

mysql> █
```


iv.) Insert values

```
mysql> insert into sinfo values (1, 'aab', 'bba');
Query OK, 1 row affected (0.06 sec)

mysql> insert into sinfo values (2, 'pqr', 'xyz');
Query OK, 1 row affected (0.05 sec)

mysql> insert into sinfo values (3, 'mno', 'abc');
Query OK, 1 row affected (0.00 sec)

mysql> insert into sinfo values (4, 'zzz', 'bbb');
Query OK, 1 row affected (0.00 sec)

mysql> exit;
Bye
[cloudera@quickstart ~]$
```

• Cloudera

After you exit MySQL, create a folder in the Cloudera file system to import the above MySQL table which was created.

(In the following steps, 'myfirstdata' folder is created in /home/cloudera.) i.) Importing

the table using Sqoop

```
sqoop import connect jdbc:mysql://youripaddress:3306/<database_name> --
--username root --password cloudera
--table <table_name>
--target-dir=<target_directory> -m 1
```

Here,

-m specifies the number of mappers
3306 is the default port for MySQL

In our case:

```
[cloudera@quickstart ~]$ sqoop import --connect jdbc:mysql://localhost:3306/student --
username root --password cloudera --table sinfo --target-dir=/home/cloudera/myfirstdata -
m 1
```

```
cloudera@quickstart:~  
File Edit View Search Terminal Help  
21/09/08 07:38:46 INFO mapreduce.Job: Counters: 30  
File System Counters  
  FILE: Number of bytes read=0  
  FILE: Number of bytes written=151415  
  FILE: Number of read operations=0  
  FILE: Number of large read operations=0  
  FILE: Number of write operations=0  
  HDFS: Number of bytes read=0  
  HDFS: Number of bytes written=40  
  HDFS: Number of read operations=4  
  HDFS: Number of large read operations=0  
  HDFS: Number of write operations=2  
Job Counters  
  Launched map tasks=1  
  Other local map tasks=1  
  Total time spent by all maps in occupied slots (ms)=13831  
  Total time spent by all reduces in occupied slots (ms)=0  
  Total time spent by all map tasks (ms)=13831  
  Total vcore-milliseconds taken by all map tasks=13831  
  Total megabyte-milliseconds taken by all map tasks=14162944  
Map-Reduce Framework  
  Map input records=4  
  Map output records=4  
  Input split bytes=87  
  Spilled Records=0  
  Failed Shuffles=0  
  Merged Map outputs=0  
  GC time elapsed (ms)=75  
  CPU time spent (ms)=2400  
  Physical memory (bytes) snapshot=197148672  
  Virtual memory (bytes) snapshot=1567649792  
  Total committed heap usage (bytes)=160432128  
File Input Format Counters  
  Bytes Read=0  
File Output Format Counters  
  Bytes Written=40  
21/09/08 07:38:46 INFO mapreduce.ImportJobBase: Transferred 40 bytes in 59.9925 seconds (0.6668 bytes/sec)  
21/09/08 07:38:46 INFO mapreduce.ImportJobBase: Retrieved 4 records.  
[cloudera@quickstart ~]$
```

ii.) Displaying the contents in HDFS

```
hadoop fs -ls /home/cloudera/myfirstdata
```

```
hadoop fs -cat /home/cloudera/myfirstdata/part-m-00000
```

```
[cloudera@quickstart ~]$ hadoop fs -ls /home/cloudera/myfirstdata  
Found 2 items  
-rw-r--r-- 1 cloudera supergroup 0 2021-09-08 07:38 /home/cloudera/myfirstdata/ SUCCESS  
-rw-r--r-- 1 cloudera supergroup 40 2021-09-08 07:38 /home/cloudera/myfirstdata/part-m-00000  
[cloudera@quickstart ~]$ hadoop fs -cat /home/cloudera/myfirstdata/part-m-00000  
1,aab,bba  
2,pqr,xyz  
3,mno,abc  
4,zzz,kkk  
[cloudera@quickstart ~]$
```

- **Export Data from HDFS to MySQL**

In order to export data from HDFS to MySQL, an appropriate table has to be created in MySQL as we export data into a particular table. In our case, we will be exporting the contents in the 'myfirstdata' folder by creating a table 'myTableCopy' in the 'myTutorial' database. The table which we will be creating needs to have the same structure as the 'myTable' table which we created earlier.

i.) Creating the table

```
mysql> use student
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> create table sinfoCopy(RollNo int, fname varchar(20), lname varchar(20));
Query OK, 0 rows affected (0.03 sec)

mysql> describe sinfoCopy
-> ;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| RollNo | int(11)       | YES  |     | NULL    |       |
| fname  | varchar(20)   | YES  |     | NULL    |       |
| lname  | varchar(20)   | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

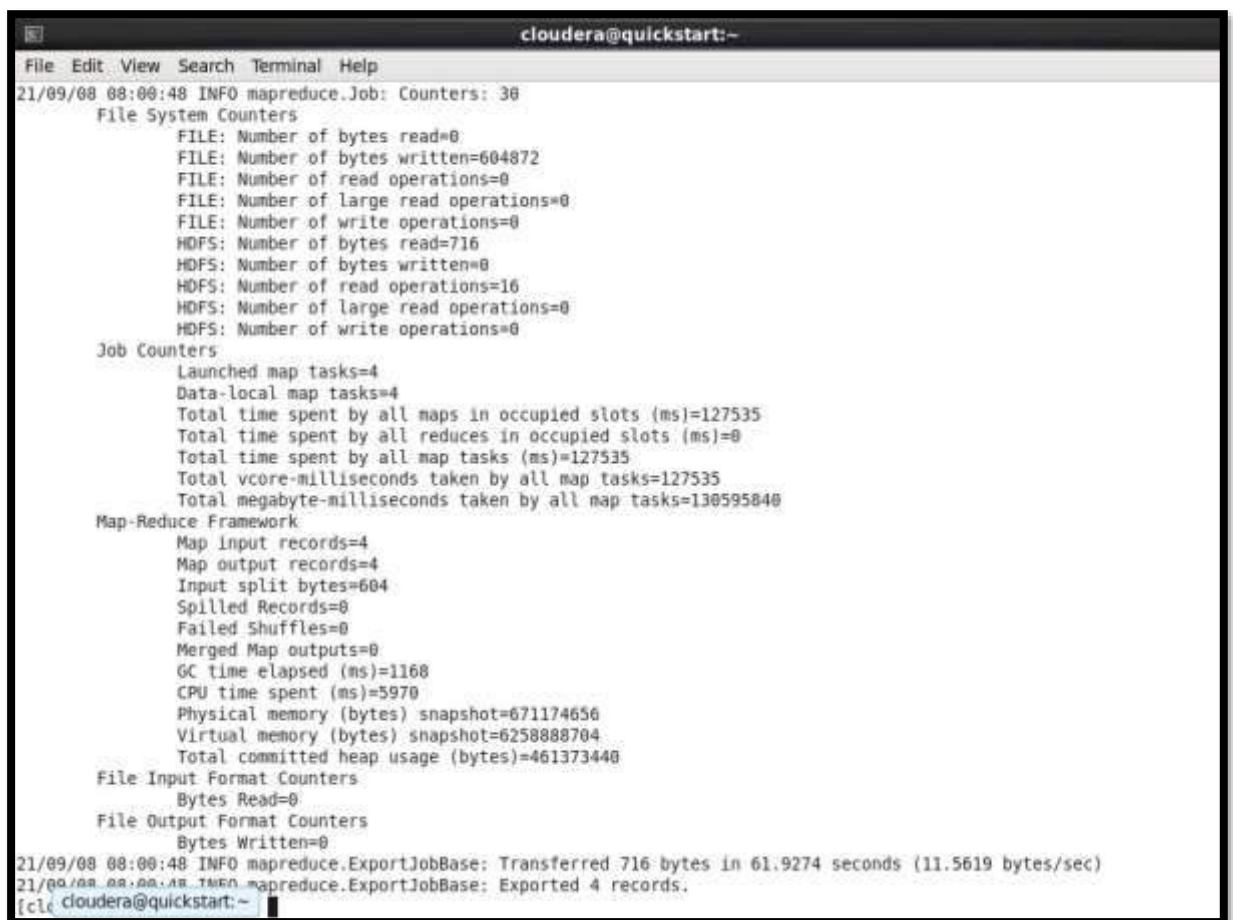
mysql> █
```

ii.) Exporting data from HDFS to MySQLSyntax:

```
sqoop export --connect jdbc:mysql://localhost/db
--username root --table <table_name>
--export-dir <directory>
```

In our case,

```
[cloudera@quickstart ~]$ sqoop export --connect jdbc:mysql://localhost/student --
username root --password cloudera --tablesinfoCopy --export-dir
/home/cloudera/myfirstdata
```



```
cloudera@quickstart:~
File Edit View Search Terminal Help
21/09/08 08:00:48 INFO mapreduce.Job: Counters: 38
  File System Counters
    FILE: Number of bytes read=0
    FILE: Number of bytes written=604872
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
    HDFS: Number of bytes read=716
    HDFS: Number of bytes written=0
    HDFS: Number of read operations=16
    HDFS: Number of large read operations=0
    HDFS: Number of write operations=0
  Job Counters
    Launched map tasks=4
    Data-local map tasks=4
    Total time spent by all maps in occupied slots (ms)=127535
    Total time spent by all reduces in occupied slots (ms)=0
    Total time spent by all map tasks (ms)=127535
    Total vcore-milliseconds taken by all map tasks=127535
    Total megabyte-milliseconds taken by all map tasks=130595840
  Map-Reduce Framework
    Map input records=4
    Map output records=4
    Input split bytes=604
    Spilled Records=0
    Failed Shuffles=0
    Merged Map outputs=0
    GC time elapsed (ms)=1168
    CPU time spent (ms)=5970
    Physical memory (bytes) snapshot=671174656
    Virtual memory (bytes) snapshot=6258888704
    Total committed heap usage (bytes)=461373440
  File Input Format Counters
    Bytes Read=0
  File Output Format Counters
    Bytes Written=0
21/09/08 08:00:48 INFO mapreduce.ExportJobBase: Transferred 716 bytes in 61.9274 seconds (11.5619 bytes/sec)
21/09/08 08:00:48 INFO mapreduce.ExportJobBase: Exported 4 records.
[cloudera@quickstart:~]
```

iii.) Verifying in MySQL

We can see that the data is exported successfully into 'myTableCopy'.

```
mysql> use student
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> select * from sinfoCopy
-> ;
+-----+-----+-----+
| RollNo | fname | lname |
+-----+-----+-----+
|      1 | aab   | bba   |
|      3 | mno   | abc   |
|      4 | zzz   | bbb   |
|      2 | pqr   | xyz   |
+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> █
```

Conclusion: Successfully transferred data between Hadoop and MySql using scoop tool.

Experiment No. 3

Aim: Programming exercises in HBASE.

Theory:

HBase is the Hadoop database, which provides random, real-time read/write access to very large data. The HBase Shell is a ruby script that helps in interacting with the HBase system using a command-line interface. This shell supports creating, deleting and altering tables and also performing other operations like inserting, listing, deleting data and interacting with HBase

NoSQL:

A NoSQL originally referring to non-SQL or non-relational is a database that provides a mechanism for storage and retrieval of data. This data is modelled in means other than the tabular relations used in relational databases. Such databases came into existence in the late 1960s but did not obtain the NoSQL moniker until a surge of popularity in the early twenty-first century. NoSQL databases are used in real-time web applications and big data and their use are increasing over time. NoSQL systems are also sometimes called Not only SQL to emphasize the fact that they may support SQL-like query languages.

A NoSQL database includes simplicity of design, simpler horizontal scaling to clusters of machines and finer control over availability. The data structures used by NoSQL databases are different from those used by default in relational databases which makes some operations faster in NoSQL. The suitability of a given NoSQL database depends on the problem it should solve. Data structures used by NoSQL databases are sometimes also viewed as more flexible than relational database tables.

Examples: MongoDB, Cassandra etc.

General Commands

1. status - Provides the status of HBase, for example, the number of servers.
2. version - Provides the version of HBase being used.
3. table_help - Provides help for table-reference commands.
4. whoami - Provides information about the user.

Data Definition Language

These are the commands that operate on the tables in HBase.

1. create - Creates a table.

Example: hbase> create 't1', {NAME => 'f1', VERSIONS => 5}, METADATA => { 'mykey' => 'myvalue' }

2. list - Lists all the tables in HBase.
3. disable - Disables a table.

Example: hbase> disable 't1'

4. is_disabled - Verifies whether a table is disabled.
5. enable - Enables a table.

Example: hbase> enable 't1'

6. is_enabled - Verifies whether a table is enabled.
7. describe - Provides the description of a table.

Example: hbase> describe 't1'

8. alter - Alters a table.
9. exists - Verifies whether a table exists.

Example: hbase> exists 't1'

10. drop - Drops a table from HBase.

Example: hbase> drop 't1'

11. drop_all - Drops the tables matching the 'regex' given in the command.

OUTPUT:

```
[cloudera@quickstart ~]$ hbase shell
2021-08-27 01:15:13,766 INFO [main] Configuration.deprecation: hadoop.native.lib is deprecated.
```

Instead, use io.native.lib.available

HBase Shell; enter 'help<RETURN>' for list of supported commands.

Type "exit<RETURN>" to leave the HBase Shell

Version 1.0.0-cdh5.4.2, rUnknown, Tue May 19 17:07:29 PDT 2015

```
hbase(main):001:0> create 'emp','personal data','professional data'
```

0 row(s) in 4.1170 seconds

```
=> Hbase::Table - emp
```

```
hbase(main):002:0> list
```

TABLE

emp

1 row(s) in 0.0850 seconds

```
=> ["emp"]
```

```
hbase(main):003:0> disable 'emp'
```

0 row(s) in 1.4740 seconds

```
hbase(main):004:0> is_disabled 'emp'
```

true

0 row(s) in 0.0440 seconds

```
hbase(main):005:0> enable 'emp'
```

0 row(s) in 0.5810 seconds

```
hbase(main):006:0> describe 'emp'
```

Table emp is ENABLED

emp

COLUMN FAMILIES DESCRIPTION

{NAME => 'personal data', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSIONS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TT

L => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}

{NAME => 'professional data', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSIONS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0'

, TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}

2 row(s) in 0.0810 seconds

```
hbase(main):007:0> alter 'emp',NAME =>'personal data',VERSIONS =>5
```

Updating all regions with the new schema...

0/1 regions updated.

1/1 regions updated.

Done.

0 row(s) in 2.9250 seconds

hbase(main):008:0> exists 'emp'

Table emp does exist

0 row(s) in 0.0450 seconds

hbase(main):009:0> disable 'emp'

0 row(s) in 1.3520 seconds

hbase(main):010:0> drop 'emp'

0 row(s) in 0.7870 seconds

hbase(main):011:0> create 'emp','personal data','professional data'

0 row(s) in 0.4970 seconds

=> Hbase::Table - emp

hbase(main):012:0> put 'emp','1','personal data:name','Gaurav'

0 row(s) in 0.5000 seconds

hbase(main):013:0> put 'emp','1','personal data:city','Mumbai'

0 row(s) in 0.5560 seconds

hbase(main):014:0> put 'emp','1','professional data:designation','MD'

0 row(s) in 0.0720 seconds

hbase(main):015:0> scan 'emp'

ROW

COLUMN+CELL

1

column=personal data:city, timestamp=1630052508843, value=Mumbai

1

column=personal data:name, timestamp=1630052485906, value=Gaurav

1

column=professional data:designation, timestamp=1630052539283,

value=MD

1 row(s) in 0.1060 seconds

hbase(main):016:0> put 'emp','1','personal data:city','Pune'

0 row(s) in 0.0120 seconds

hbase(main):017:0> scan 'emp'

ROW

COLUMN+CELL

1

column=personal data:city, timestamp=1630052586572, value=Pune

1

column=personal data:name, timestamp=1630052485906, value=Gaurav

1

column=professional data:designation, timestamp=1630052539283,

value=MD

1 row(s) in 0.0540 seconds

hbase(main):018:0> get 'emp','1',{COLUMN => 'personal data:name'}

COLUMN

CELL

personal data:name

timestamp=1630052485906, value=Gaurav

1 row(s) in 0.0550 seconds

hbase(main):019:0> delete 'emp','1','personal data:name',1630052485906

0 row(s) in 0.0690 seconds

hbase(main):020:0> count 'emp'

1 row(s) in 0.1060 seconds


```
=> 1
hbase(main):021:0> truncate 'emp'
Truncating 'emp' table (it may take a while):
- Disabling table...
- Truncating table...
0 row(s) in 1.8040 seconds

hbase(main):022:0>
```

Conclusion: Successfully Implemented Programming exercises in HBase.

EXPERIMENT - 4

AIM: Experiment for Word Counting using Hadoop Map-Reduce

THEORY:

MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce the task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes. Under the MapReduce model, the data processing primitives are called mappers and reducers. Decomposing a data processing application into mappers and reducers is sometimes nontrivial. But, once we write an application in the MapReduce form, scaling the application to run over hundreds, thousands, or even tens of thousands of machines in a cluster is merely a configuration change. This simple scalability is what has attracted many programmers to use the MapReduce model.

Word Count Operation:

Hadoop WordCount operation occurs in 3 stages –

- Mapper Phase
- Shuffle Phase
- Reducer Phase

1)WordCount.java:

```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.Job;
public class WordCount {
    public static void main(String[] args)throws Exception{
        if(args.length != 2){
            System.out.printf("Usage: WordCount <input dir> <output dir>\n");
            System.exit(-1);
        }
        Job job = new Job();
        job.setJarByClass(WordCount.class);
        job.setJobName("WordCount");
        FileInputFormat.setInputPaths(job,new Path(args[0]));
        FileOutputFormat.setOutputPath(job,new Path(args[1]));
        job.setMapperClass(WordMapper.class);
        job.setReducerClass(WordReducer.class);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        boolean success = job.waitForCompletion(true);
        System.exit(success ? 0 : 1);
    }
}
```

2)WordMapper.java:

```
import java.io.IOException;
```

```

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
public class WordMapper extends Mapper<LongWritable,Text,Text,IntWritable> {
    @Override
    public void map(LongWritable key,Text value,Context context)throws
IOException,InterruptedException{
        String line = value.toString();
        for(String word : line.split("\\W+")){
            if(word.length()>0){
                context.write(new Text(word),new IntWritable(1));
            }
        }
    }
}

```

3)WordReducer.java:

```

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

public class WordReducer extends Reducer <Text,IntWritable,Text,IntWritable> {
    @Override
    public void reduce(Text key,Iterable<IntWritable> values,Context context)throws
IOException,InterruptedException{
        int wordCount = 0;
        for(IntWritable value : values){
            wordCount += value.get();
        }
        context.write(key,new IntWritable(wordCount));
    }
}

```

4)Sample File Creation and Execution:

```

[cloudera@quickstart ~]$ hadoop fs -mkdir /inputFolder1
[cloudera@quickstart ~]$ hadoop fs -put /home/cloudera/Desktop/wc.txt/outputFolder1put:
`/home/cloudera/Desktop/wc.txt/outputFolder1': No such file or directory
[cloudera@quickstart ~]$ hadoop fs -put /home/cloudera/Desktop/wc.txt /outputFolder1
[cloudera@quickstart ~]$ hadoop fs -put /home/cloudera/Desktop/wc.txt /inputFolder1
[cloudera@quickstart ~]$ hadoop fs -cat /inputFolder1/wc.txt
hi
jeet
prajapati
[cloudera@quickstart ~]$ hadoop jar /home/cloudera/Desktop/WordCount.jar WordCount
/inputFolder1/wc.txt/outputFolder
21/08/30 04:22:55 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
21/08/30 04:23:18 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing
not performed. Implement the Tool interface and execute your application with ToolRunner to
remedy this.
21/08/30 04:23:21 INFO input.FileInputFormat: Total input paths to process : 1
21/08/30 04:23:44 INFO mapreduce.JobSubmitter: number of splits:1
21/08/30 04:23:45 INFO mapreduce.JobSubmitter: Submitting tokens for job:
job_1630053095738_0001

```

21/08/30 04:24:26 INFO impl.YarnClientImpl: Submitted application
 application_1630053095738_0001
 21/08/30 04:24:28 INFO mapreduce.Job: The url to track the job:
http://quickstart.cloudera:8088/proxy/application_1630053095738_0001/
 21/08/30 04:24:28 INFO mapreduce.Job: Running job: job_1630053095738_0001
 21/08/30 04:27:01 INFO mapreduce.Job: Job job_1630053095738_0001 running in uber mode : false
 21/08/30 04:27:08 INFO mapreduce.Job: map 0% reduce 0%
 21/08/30 04:28:27 INFO mapreduce.Job: map 100% reduce 0%
 21/08/30 04:28:41 INFO mapreduce.Job: map 100% reduce 100%
 21/08/30 04:28:43 INFO mapreduce.Job: Job job_1630053095738_0001 completed successfully
 21/08/30 04:28:44 INFO mapreduce.Job: Counters: 49

File System Counters

FILE: Number of bytes read=38
 FILE: Number of bytes written=250361
 FILE: Number of read operations=0
 FILE: Number of large read operations=0
 FILE: Number of write operations=0
 HDFS: Number of bytes read=131
 HDFS: Number of bytes written=20
 HDFS: Number of read operations=6
 HDFS: Number of large read operations=0
 HDFS: Number of write operations=2

Job Counters

Launched map tasks=1
 Launched reduce tasks=1
 Data-local map tasks=1
 Total time spent by all maps in occupied slots (ms)=71347
 Total time spent by all reduces in occupied slots (ms)=11742
 Total time spent by all map tasks (ms)=71347
 Total time spent by all reduce tasks (ms)=11742
 Total vcore-milliseconds taken by all map tasks=71347
 Total vcore-milliseconds taken by all reduce tasks=11742
 Total megabyte-milliseconds taken by all map tasks=73059328
 Total megabyte-milliseconds taken by all reduce tasks=12023808

Map-Reduce Framework

Map input records=3
 Map output records=3
 Map output bytes=26
 Map output materialized bytes=38
 Input split bytes=116
 Combine input records=0
 Combine output records=0
 Reduce input groups=3
 Reduce shuffle bytes=38
 Reduce input records=3
 Reduce output records=3
 Spilled Records=6
 Shuffled Maps =1
 Failed Shuffles=0
 Merged Map outputs=1
 GC time elapsed (ms)=325
 CPU time spent (ms)=4740
 Physical memory (bytes) snapshot=455786496
 Virtual memory (bytes) snapshot=3128238080
 Total committed heap usage (bytes)=287834112

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=15
File Output Format Counters
Bytes Written=20

5)Wc.txt:

hi
gaurav
poojary

6)Checking the Output File:

[cloudera@quickstart ~]\$ hadoop fs -ls /outputFolder

Found 2 items

```
-rw-r--r--  1 cloudera supergroup      0 2021-08-30 04:28 /outputFolder/_SUCCESS
-rw-r--r--  1 cloudera supergroup    20 2021-08-30 04:28 /outputFolder/part-r-00000
```

7)Displaying the Output:

[cloudera@quickstart ~]\$ hadoop fs -cat /outputFolder/part-r-00000

gaurav 1
hi 1
poojary 1

Experiment no:5

Aim: Implementing algorithms for page rank using Java.

Theory:

PageRank (PR) is an algorithm used by Google Search to rank websites in their search engine results. PageRank was named after Larry Page, one of the founders of Google. PageRank is a way of measuring the importance of website pages.

It is not the only algorithm used by Google to order search engine results, but it is the first algorithm that was used by the company, and it is the best-known.

The above centrality measure is not implemented for multi-graphs.

Algorithm

The PageRank algorithm outputs a probability distribution used to represent the likelihood that a person randomly clicking on links will arrive at any particular page. PageRank can be calculated for collections of documents of any size. It is assumed in several research papers that the distribution is evenly divided among all documents in the collection at the beginning of the computational process. The PageRank computations require several passes, called “iterations”, through the collection to adjust approximate PageRank values to more closely reflect the theoretical true value.

Simplified algorithm

Assume a small universe of four web pages: A, B, C and D. Links from a page to itself, or multiple outbound links from one single page to another single page, are ignored. PageRank is initialized to the same value for all pages. In the original form of PageRank, the sum of PageRank over all pages was the total number of pages on the web at that time, so each page in this example would have an initial value of 1. However, later versions of PageRank, and the remainder of this section, assume a probability distribution between 0 and 1. Hence the initial value for each page in this example is 0.25. The PageRank transferred from a given page to the targets of its outbound links upon the next iteration is divided equally among all outbound links.

If the only links in the system were from pages B, C, and D to A, each link would transfer 0.25 PageRank to A upon the next iteration, for a total of 0.75.

$$PR(A) = PR(B) + PR(C) + PR(D).$$

Suppose instead that page B had a link to pages C and A, page C had a link to page A, and page D had links to all three pages. Thus, upon the first iteration, page B would transfer half of its existing value, or 0.125, to page A and the other half, or 0.125, to page C. Page C would transfer all of its existing value, 0.25, to the only page it links to, A. Since D had three outbound links, it would transfer one third of its existing value, or approximately 0.083, to A. At the completion of this iteration, page A will have a PageRank of approximately 0.458.

$$PR(A) = \frac{PR(B)}{2} + \frac{PR(C)}{1} + \frac{PR(D)}{3}.$$

In other words, the PageRank conferred by an outbound link is equal to the document's own PageRank score divided by the number of outbound links $L()$.

$PR(A) = \frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)}$. In the general case, the PageRank value for any page u can be expressed as:

$$PR(A) = \frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)}.$$

i.e. the PageRank value for a page u is dependent on the PageRank values for each page v contained in the set B_u (the set containing all pages linking to page u), divided by the number $L(v)$ of links from page v. The algorithm involves a damping factor for the calculation of the pagerank. It is like the income tax which the govt extracts from one despite paying him itself.

Code:

```

import java.util.*;
import java.io.*;
public class PageRank {

    public int path[][] = new int[10][10];
    public double pagerank[] = new double[10];

    public void calc(double totalNodes) {

        double InitialPageRank;
        double OutgoingLinks = 0;
        double DampingFactor = 0.85;
        double TempPageRank[] = new double[10];
        int ExternalNodeNumber;
        int InternalNodeNumber;
        int k = 1; // For Traversing
        int ITERATION_STEP = 1;
        InitialPageRank = 1 / totalNodes;
        System.out.printf("Total Number of Nodes : " + totalNodes + "\t Initial PageRank of All Nodes : " + InitialPageRank + "\n");

        // 0th ITERATION _ OR _ INITIALIZATION PHASE //

        for (k = 1; k <= totalNodes; k++) {
            this.pagerank[k] = InitialPageRank;
        }

        System.out.printf("\n Initial PageRank Values , 0th Step \n");
        for (k = 1; k <= totalNodes; k++) {
            System.out.printf("Page Rank of " + k + " is :\t" + this.pagerank[k] + "\n");
        }

        while (ITERATION_STEP <= 2) // Iterations
        {
            // Store the PageRank for All Nodes in Temporary Array
            for (k = 1; k <= totalNodes; k++) {
                TempPageRank[k] = this.pagerank[k];
                this.pagerank[k] = 0;
            }

            for (InternalNodeNumber = 1; InternalNodeNumber <= totalNodes; InternalNodeNumber++) {
                for (ExternalNodeNumber = 1; ExternalNodeNumber <= totalNodes; ExternalNodeNumber++) {
                    if (this.path[ExternalNodeNumber][InternalNodeNumber] == 1) {
                        k = 1;
                        OutgoingLinks = 0; // Count the Number of Outgoing Links for each ExternalNodeNumber
                        while (k <= totalNodes) {
                            if (this.path[ExternalNodeNumber][k] == 1) {
                                OutgoingLinks = OutgoingLinks + 1; // Counter for Outgoing Links

```

```

    }
    k = k + 1;
}
// Calculate PageRank
this.pagerank[InternalNodeNumber] += TempPageRank[ExternalNodeNumber] * (1 / OutgoingLinks);
}
}
}

System.out.printf("\n After Step " + ITERATION_STEP + " \n");

for (k = 1; k <= totalNodes; k++)
    System.out.printf(" Page Rank of " + k + " is :\t" + this.pagerank[k] + "\n");

ITERATION_STEP = ITERATION_STEP + 1;
}
// Add the Damping Factor to PageRank
for (k = 1; k <= totalNodes; k++) {
    this.pagerank[k] = (1 - DampingFactor) + DampingFactor * this.pagerank[k];
}

// Display PageRank
System.out.printf("\n Final Page Rank : \n");
for (k = 1; k <= totalNodes; k++) {
    System.out.printf(" Page Rank of " + k + " is :\t" + this.pagerank[k] + "\n");
}
}

public static void main(String args[]) {
    int nodes, i, j, cost;
    Scanner in = new Scanner(System.in);
    System.out.println("Enter the Number of WebPages \n");
    nodes = in.nextInt();
    PageRank p = new PageRank();
    System.out.println("Enter the Adjacency Matrix with 1->PATH & 0-
>NO PATH Between two WebPages:");
    for (i = 1; i <= nodes; i++)
        for (j = 1; j <= nodes; j++) {
            p.path[i][j] = in.nextInt();
            if (j == i)
                p.path[i][j] = 0;
        }
    p.calc(nodes);
}
}

```


Output:

Enter the Number of WebPages

4

Enter the Adjacency Matrix with 1->PATH & 0->NO PATH Between two WebPages:

1 0 1 1

0 0 1 1

0 1 1 0

0 0 0 0

Total Number of Nodes :4.0 Initial PageRank of All Nodes :0.25

Initial PageRank Values , 0th Step

Page Rank of 1 is : 0.25

Page Rank of 2 is : 0.25

Page Rank of 3 is : 0.25

Page Rank of 4 is : 0.25

After Step 1

Page Rank of 1 is : 0.0

Page Rank of 2 is : 0.25

Page Rank of 3 is : 0.25

Page Rank of 4 is : 0.25

After Step 2

Page Rank of 1 is : 0.0

Page Rank of 2 is : 0.25

Page Rank of 3 is : 0.125

Page Rank of 4 is : 0.125

Final Page Rank :

Page Rank of 1 is : 0.15000000000000002

Page Rank of 2 is : 0.36250000000000004

Page Rank of 3 is : 0.25625000000000003

Page Rank of 4 is : 0.25625000000000003

Conclusion: Implementation of PageRank algorithm is successful.

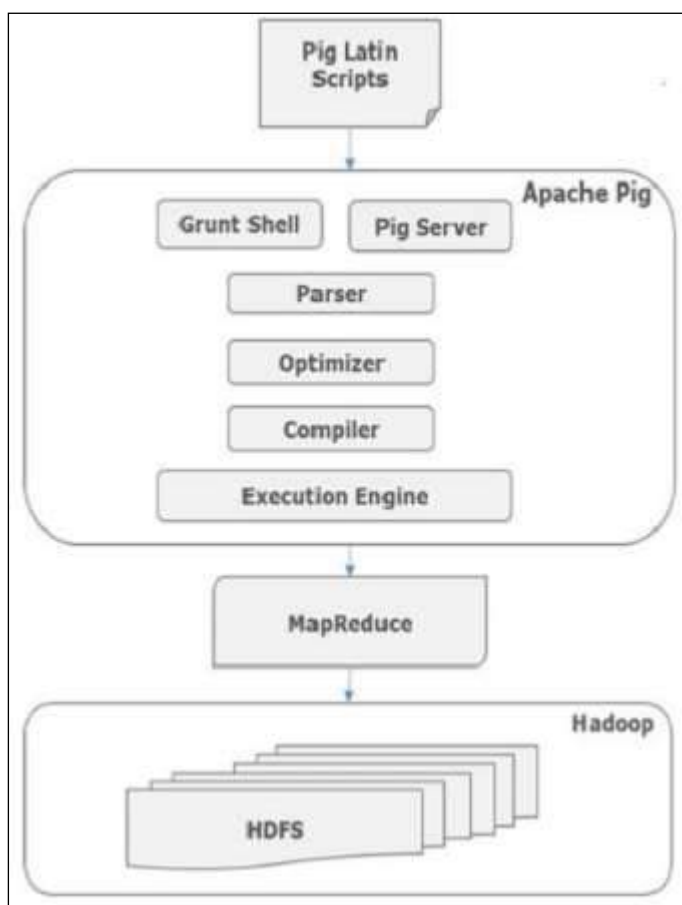
Experiment No.-6

Aim: Implementation of PIG commands

Theory:

Pig is a high-level platform or tool which is used to process the large datasets. It provides a high-level of abstraction for processing over the MapReduce. It provides a high-level scripting language, known as Pig Latin which is used to develop the data analysis codes. First, to process the data which is stored in the HDFS, the programmers will write the scripts using the Pig Latin Language. Internally Pig Engine (a component of Apache Pig) converted all these scripts into a specific map and reduce task. But these are not visible to the programmers to provide a high-level of abstraction. Pig Latin and Pig Engine are the two main components of the Apache Pig tool. The result of Pig always stored in the HDFS.

● PIG Architecture:



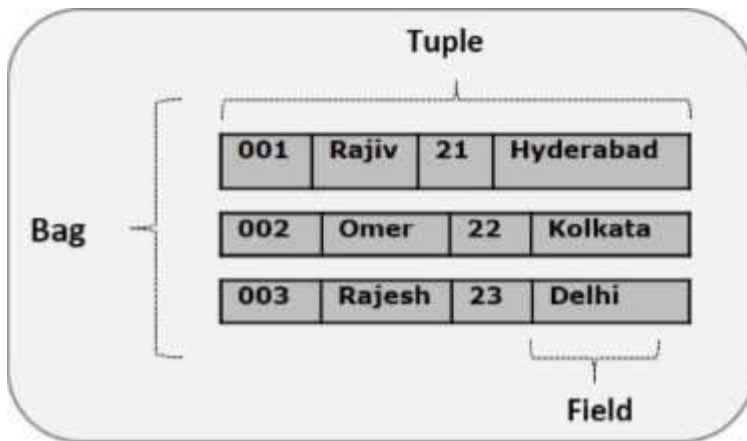
• Apache Pig Components

As shown in the figure, there are various components in the Apache Pig framework. Following are the major components.

- **Parser:** Initially the Pig Scripts are handled by the Parser. It checks the syntax of the script, does type checking, and other miscellaneous checks. The output of the parser will be a DAG (directed acyclic graph), which represents the Pig Latin statements and logical operators.

In the DAG, the logical operators of the script are represented as the nodes and the data flows are represented as edges.

- **Optimizer:** The logical plan (DAG) is passed to the logical optimizer, which carries out the logical optimizations such as projection and pushdown.
- **Compiler:** The compiler compiles the optimized logical plan into a series of MapReduce jobs.
- **Execution engine:** Finally, the MapReduce jobs are submitted to Hadoop in a sorted order. Finally, these MapReduce jobs are executed on Hadoop producing the desired results.
- **Pig Latin Data Model:** The data model of Pig Latin is fully nested and it allows complex non-atomic datatypes such as map and tuple. Given below is the diagrammatical representation of Pig Latin's data model.



- **Atom:** Any single value in Pig Latin, irrespective of their data, type is known as an Atom. It is stored as string and can be used as string and number. int, long, float, double, chararray, and bytearray are the atomic values of Pig. A piece of data or a simple atomic value is known as a field.

Example – 'raja' or '30'

- **Tuple:** A record that is formed by an ordered set of fields is known as a tuple, the fields can be of any type. A tuple is similar to a row in a table of RDBMS.

Example – (Raja, 30)

- **Bag:** A bag is an unordered set of tuples. In other words, a collection of tuples (non-unique) is known as a bag. Each tuple can have any number of fields (flexible schema). A bag is represented by '{}'. It is similar to a table in RDBMS, but unlike a table in RDBMS, it is not necessary that every tuple contain the same number of fields or that the fields in the same position (column) have the same type.

Example – {(Raja, 30), (Mohammad, 45)}

A bag can be a field in a relation; in that context, it is known as inner bag.

Example – {Raja, 30, {9848022338, raja@gmail.com,}}

- **Map:** A map (or data map) is a set of key-value pairs. The key needs to be of type chararray and should be unique. The value might be of any type. It is represented by '[]'

Example – [name#Raja, age#30]

- Relation: A relation is a bag of tuples. The relations in Pig Latin are unordered (there is no guarantee that tuples are processed in any particular order).

● Pig Latin:

The Pig Latin is a data flow language used by Apache Pig to analyse the data in Hadoop. It is a textual language that abstracts the programming from the Java MapReduce idiom into a notation.

● Pig Operators:

- The grunt shell can be invoked using the command *pig*.
- We can invoke the ls command of HDFS from the Grunt shell using fs command.
- After invoking the Grunt shell, you can execute a Pig script by directly entering the Pig Latin statements in it.

```
grunt> customers = LOAD 'customers.txt' USING PigStorage(',');
```

- Following is a Pig Latin statement, which loads data to Apache Pig.

```
grunt> Student_data = LOAD 'student_data.txt' USING PigStorage(',')as
```

```
( id:int, firstname:chararray, lastname:chararray, phone:chararray, city:chararray );
```

• Pig Latin – Relational Operations

The following table describes some of the relational operators of Pig Latin.

Operator	Description
LOAD	To Load the data from the file system (local/HDFS) into a relation.
FILTER	To remove unwanted rows from a relation.
DISTINCT	To remove duplicate rows from a relation.
FOREACH, GENERATE	To generate data transformations based on columns of data.

JOIN	To join two or more relations.
COGROUP	To group the data in two or more relations.
GROUP	To group the data in a single relation.
CROSS	To create the cross product of two or more relations.

PIG COMMANDS:

1.Run PIG:

```
[cloudera@quickstart ~]$ pig -x local
```

```
log4j:WARN No appenders could be found for logger (org.apache.hadoop.util.Shell).
```

```
log4j:WARN Please initialize the log4j system properly.
```

```
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
```

```
2021-10-03 01:52:17,597 [main] INFO org.apache.pig.Main - Apache Pig version 0.12.0-cdh5.12.0
(rexported) compiled Jun 29 2017, 04:34:31
```

```
2021-10-03 01:52:17,599 [main] INFO org.apache.pig.Main - Logging error messages to:
/home/cloudera/pig_1632559937448.log
```

```
2021-10-03 01:52:17,721 [main] INFO org.apache.pig.impl.util.Utils - Default bootup file
/home/cloudera/.pigbootup not found
```

```
2021-10-03 01:52:18,149 [main] INFO org.apache.hadoop.conf.Configuration.deprecation -
fs.default.name is deprecated. Instead, use fs.defaultFS
```

```
2021-10-03 01:52:18,150 [main] INFO org.apache.hadoop.conf.Configuration.deprecation -
mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
```

```
2021-10-03 01:52:18,153 [main] INFO
org.apache.pig.backend.hadoop.executionengine.HExecutionEngine - Connecting to hadoop file
system at: file:///
```

```
2021-10-03 01:52:19,376 [main] INFO org.apache.hadoop.conf.Configuration.deprecation -
io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
```

```
2021-10-03 01:52:19,671 [main] INFO org.apache.hadoop.conf.Configuration.deprecation -
fs.default.name is deprecated. Instead, use fs.defaultFS
```

```
2021-10-03 01:52:19,689 [main] INFO org.apache.hadoop.conf.Configuration.deprecation -
mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
```

```
2021-10-03 01:52:19,703 [main] INFO org.apache.hadoop.conf.Configuration.deprecation -
io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
```

```
2021-10-03 01:52:20,052 [main] INFO org.apache.hadoop.conf.Configuration.deprecation -
fs.default.name is deprecated. Instead, use fs.defaultFS
```

2021-10-03 01:52:20,062 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address

2021-10-03 01:52:20,064 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum

2021-10-03 01:52:20,253 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS

2021-10-03 01:52:20,257 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address

2021-10-03 01:52:20,258 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum

2021-10-03 01:52:20,417 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS

2021-10-03 01:52:20,420 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address

2021-10-03 01:52:20,422 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum

2021-10-03 01:52:20,536 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS

2021-10-03 01:52:20,544 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address

2021-10-03 01:52:20,545 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum

2021-10-03 01:52:20,661 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS

2021-10-03 01:52:20,664 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address

2021-10-03 01:52:20,665 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum

2021-10-03 01:52:20,793 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS

2021-10-03 01:52:20,795 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address

2021-10-03 01:52:20,796 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum

2021-10-03 01:52:20,924 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS

2021-10-03 01:52:20,929 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address

2021-10-03 01:52:20,932 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum

2.fs -ls -grunt:

```
grunt> fs -ls
```

Found 1 items

```
-rw-r--r-- 1 cloudera cloudera      0 2021-08-12 22:05 sample.txt
```

3. Loading data:

```
grunt> cat /home/cloudera/Desktop/student.txt
```

1,Arpit,102,F,Male

2,Ravi,104,F,Male

3,Mann,120,F,Male

4,Pradnesh,89,F,Male

5,Gaurav,80,F,Male

6,Jeet,78,F,Male

7,Pulkit,60,F,Male

8,Saurav,50,F,Male

9,Dhrumil,45,F,Male

4.Loading the data into a Table (student):

```
grunt> student = load '/home/cloudera/Dekstop/student.txt' USING PigStorage(',') as  
(id:int,name:chararray,sal:float,sex:chararray);
```

```
2021-10-03 02:01:48,045 [main] INFO org.apache.hadoop.conf.Configuration.deprecation -  
fs.default.name is deprecated. Instead, use fs.defaultFS
```

```
2021-10-03 02:01:48,045 [main] INFO org.apache.hadoop.conf.Configuration.deprecation -  
mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
```

```
2021-10-03 02:01:48,045 [main] INFO org.apache.hadoop.conf.Configuration.deprecation -  
io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
```

```
grunt> student = load '/home/cloudera/Desktop/student.txt' USING PigStorage(',') as  
(id:int,name:chararray,sal:float,sex:chararray);
```

```
grunt> dump student;
```

```
2021-10-03 02:02:08,807 [main] INFO org.apache.pig.tools.pigstats.ScriptState - Pig features used in  
the script: UNKNOWN
```

```
2021-10-03 02:02:08,846 [main] INFO  
org.apache.pig.newplan.logical.optimizer.LogicalPlanOptimizer -  
{ RULES_ENABLED=[AddForEach, ColumnMapKeyPrune, DuplicateForEachColumnRewrite,  
GroupByConstParallelSetter, ImplicitSplitInserter, LimitOptimizer, LoadTypeCastInserter,  
MergeFilter, MergeForEach, NewPartitionFilterOptimizer, PushDownForEachFlatten, PushUpFilter,  
SplitFilter, StreamTypeCastInserter], RULES_DISABLED=[FilterLogicExpressionSimplifier,  
PartitionFilterOptimizer]}
```

2021-10-03 02:02:08,916 [main] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MRCompiler - File concatenation threshold: 100 optimistic? false

2021-10-03 02:02:08,918 [main] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MultiQueryOptimizer - MR plan size before optimization: 1

2021-10-03 02:02:08,918 [main] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MultiQueryOptimizer - MR plan size after optimization: 1

2021-10-03 02:02:08,920 [main] INFO org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized

2021-10-03 02:02:08,933 [main] INFO org.apache.pig.tools.pigstats.ScriptState - Pig script settings are added to the job

2021-10-03 02:02:09,177 [main] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.JobControlCompiler - mapred.job.reduce.markreset.buffer.percent is not set, set to default 0.3

2021-10-03 02:02:09,242 [main] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.JobControlCompiler - Setting up single store job

2021-10-03 02:02:09,247 [main] INFO org.apache.pig.data.SchemaTupleFrontend - Key [pig.schematuple] is false, will not generate code.

2021-10-03 02:02:09,247 [main] INFO org.apache.pig.data.SchemaTupleFrontend - Starting process to move generated code to distributed cache

2021-10-03 02:02:09,248 [main] INFO org.apache.pig.data.SchemaTupleFrontend - Distributed cache not supported or needed in local mode. Setting key [pig.schematuple.local.dir] with code temp directory: /tmp/1632560529243-0

2021-10-03 02:02:09,327 [main] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - 1 map-reduce job(s) waiting for submission.

2021-10-03 02:02:09,598 [JobControl] INFO org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized

2021-10-03 02:02:09,816 [JobControl] WARN org.apache.hadoop.mapreduce.JobResourceUploader - No job jar file set. User classes may not be found. See Job or Job#setJar(String).

2021-10-03 02:02:10,076 [JobControl] INFO
org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1

2021-10-03 02:02:10,076 [JobControl] INFO
org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1

2021-10-03 02:02:10,280 [JobControl] INFO
org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths (combined) to process : 1

2021-10-03 02:02:10,503 [JobControl] INFO org.apache.hadoop.mapreduce.JobSubmitter - number of splits:1

2021-10-03 02:02:11,780 [JobControl] INFO org.apache.hadoop.mapreduce.JobSubmitter - Submitting tokens for job: job_local1198861867_0002

2021-10-03 02:02:15,029 [JobControl] INFO org.apache.hadoop.mapreduce.Job - The url to track the job: <http://localhost:8080/>

2021-10-03 02:02:15,030 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - HadoopJobId: job_local1198861867_0002

2021-10-03 02:02:15,030 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Processing aliases student

2021-10-03 02:02:15,030 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - detailed locations: M: student[3,10],student[-1,-1] C: R:

2021-10-03 02:02:15,395 [Thread-9] INFO org.apache.hadoop.mapred.LocalJobRunner - OutputCommitter set in config null

2021-10-03 02:02:17,519 [Thread-9] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS

2021-10-03 02:02:17,562 [Thread-9] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.reduce.markreset.buffer.percent is deprecated. Instead, use mapreduce.reduce.markreset.buffer.percent

2021-10-03 02:02:17,563 [Thread-9] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address

2021-10-03 02:02:17,563 [Thread-9] INFO org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum

2021-10-03 02:02:17,669 [Thread-9] INFO org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter - File Output Committer Algorithm version is 1

2021-10-03 02:02:17,672 [Thread-9] INFO org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter - FileOutputCommitter skip cleanup _temporary folders under output directory:false, ignore cleanup failures: false

2021-10-03 02:02:17,674 [Thread-9] INFO org.apache.hadoop.mapred.LocalJobRunner - OutputCommitter is org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.PigOutputCommitter

2021-10-03 02:02:19,110 [Thread-9] INFO org.apache.hadoop.mapred.LocalJobRunner - Waiting for map tasks

2021-10-03 02:02:19,122 [LocalJobRunner Map Task Executor #0] INFO org.apache.hadoop.mapred.LocalJobRunner - Starting task: attempt_local1198861867_0002_m_000000_0

2021-10-03 02:02:21,664 [LocalJobRunner Map Task Executor #0] INFO org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter - File Output Committer Algorithm version is 1

2021-10-03 02:02:21,665 [LocalJobRunner Map Task Executor #0] INFO

org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter - FileOutputCommitter skip cleanup
_temporary folders under output directory:false, ignore cleanup failures: false

2021-10-03 02:02:21,854 [LocalJobRunner Map Task Executor #0] INFO
org.apache.hadoop.mapred.Task - Using ResourceCalculatorProcessTree : []

2021-10-03 02:02:21,880 [LocalJobRunner Map Task Executor #0] INFO
org.apache.hadoop.mapred.MapTask - Processing split: Number of splits :1

Total Length = 171

Input split[0]:

Length = 171

Locations:

2021-10-03 02:02:22,025 [LocalJobRunner Map Task Executor #0] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.PigRecordReader - Current split
being processed <file:/home/cloudera/Desktop/student.txt:0+171>

2021-10-03 02:02:22,112 [LocalJobRunner Map Task Executor #0] INFO
org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter - File Output Committer Algorithm
version is 1

2021-10-03 02:02:22,112 [LocalJobRunner Map Task Executor #0] INFO
org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter - FileOutputCommitter skip cleanup
_temporary folders under output directory:false, ignore cleanup failures: false

2021-10-03 02:02:22,235 [LocalJobRunner Map Task Executor #0] INFO
org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate
code.

2021-10-03 02:02:22,321 [LocalJobRunner Map Task Executor #0] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.PigMapOnly\$Map - Aliases being
processed per job phase (AliasName[line,offset]): M: student[3,10],student[-1,-1] C: R:

2021-10-03 02:02:22,412 [LocalJobRunner Map Task Executor #0] INFO
org.apache.hadoop.mapred.LocalJobRunner -

2021-10-03 02:02:22,414 [LocalJobRunner Map Task Executor #0] INFO
org.apache.hadoop.mapred.Task - Task:attempt_local1198861867_0002_m_000000_0 is done. And is
in the process of committing

2021-10-03 02:02:22,472 [LocalJobRunner Map Task Executor #0] INFO
org.apache.hadoop.mapred.LocalJobRunner -

2021-10-03 02:02:22,472 [LocalJobRunner Map Task Executor #0] INFO
org.apache.hadoop.mapred.Task - Task attempt_local1198861867_0002_m_000000_0 is allowed to
commit now

2021-10-03 02:02:22,482 [LocalJobRunner Map Task Executor #0] INFO
org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter - Saved output of task

'attempt_local1198861867_0002_m_000000_0' to
file:/tmp/temp1185102156/tmp1818553402/ temporary/0/task local1198861867_0002_m_000000_0

2021-10-03 02:02:22,486 [LocalJobRunner Map Task Executor #0] INFO
org.apache.hadoop.mapred.LocalJobRunner - map

2021-10-03 02:02:22,486 [LocalJobRunner Map Task Executor #0] INFO
org.apache.hadoop.mapred.Task - Task 'attempt_local1198861867_0002_m_000000_0' done.

2021-10-03 02:02:22,488 [LocalJobRunner Map Task Executor #0] INFO
org.apache.hadoop.mapred.LocalJobRunner - Finishing task:
attempt_local1198861867_0002_m_000000_0

2021-10-03 02:02:22,488 [Thread-9] INFO org.apache.hadoop.mapred.LocalJobRunner - map task
executor complete.

2021-10-03 02:02:25,498 [main] INFO org.apache.hadoop.conf.Configuration.deprecation -
mapred.map.tasks is deprecated. Instead, use mapreduce.job.maps

2021-10-03 02:02:25,498 [main] INFO org.apache.hadoop.conf.Configuration.deprecation -
mapred.reduce.tasks is deprecated. Instead, use mapreduce.job.reduces

2021-10-03 02:02:31,499 [main] WARN org.apache.pig.tools.pigstats.PigStatsUtil - Failed to get
RunningJob for job job_local1198861867_0002

2021-10-03 02:02:31,499 [main] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - 100%
complete

2021-10-03 02:02:31,499 [main] INFO org.apache.pig.tools.pigstats.SimplePigStats - Detected Local
mode. Stats reported below may be incomplete

2021-10-03 02:02:31,506 [main] INFO org.apache.pig.tools.pigstats.SimplePigStats - Script
Statistics:

HadoopVersion	PigVersion	UserId	StartedAt	FinishedAt	Features
2.6.0-cdh5.12.0	0.12.0-cdh5.12.0	cloudera	2021-10-03 02:02:08	2021-10-03	
02:02:31	UNKNOWN				

Success!

Job Stats (time in seconds):

JobId	Alias	Feature	Outputs
job_local1198861867_0002	student	MAP_ONLY	file:/tmp/temp1185102156/tmp1818553402 ,

Input(s):

Successfully read records from: "/home/cloudera/Desktop/student.txt"

Output(s):

Successfully stored records in: "<file:/tmp/temp1185102156/tmp1818553402>"

Job DAG:

job_local1198861867_0002

2021-10-03 02:02:37,508 [main] INFO

org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!

2021-10-03 02:02:37,509 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS

2021-10-03 02:02:37,510 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address

2021-10-03 02:02:37,510 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum

2021-10-03 02:02:37,510 [main] WARN org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend has already been initialized

2021-10-03 02:02:37,606 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1

2021-10-03 02:02:37,606 [main] INFO

org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1

(1,Arpit,102.0,F)

(2,Ravi,104.0,F)

(3,Mann,120.0,F)

(4,Pradnesh,89.0,F)

(5,Gaurav,80.0,F)

(6,Jeet,78.0,F)

(7,Pulkit,60.0,F)

(8,Saurav,50.0,F)

(9,Dhruvil,45.0,F)

5.Loading the data into a Table (job):

grunt> job = load '/home/cloudera/Desktop/sample2.txt' USING PigStorage(',') as

(name:chararray,position:chararray);

grunt> dump job;

2021-10-03 02:32:02,477 [main] INFO org.apache.pig.tools.pigstats.ScriptState - Pig features used in the script: UNKNOWN

2021-10-03 02:32:02,481 [main] INFO
org.apache.pig.newplan.logical.optimizer.LogicalPlanOptimizer -
{ RULES_ENABLED=[AddForEach, ColumnMapKeyPrune, DuplicateForEachColumnRewrite, GroupByConstParallelSetter, ImplicitSplitInserter, LimitOptimizer, LoadTypeCastInserter, MergeFilter, MergeForEach, NewPartitionFilterOptimizer, PushDownForEachFlatten, PushUpFilter, SplitFilter, StreamTypeCastInserter], RULES_DISABLED=[FilterLogicExpressionSimplifier, PartitionFilterOptimizer]}

2021-10-03 02:32:02,484 [main] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MRCompiler - File concatenation threshold: 100 optimistic? false

2021-10-03 02:32:02,486 [main] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MultiQueryOptimizer - MR plan size before optimization: 1

2021-10-03 02:32:02,486 [main] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MultiQueryOptimizer - MR plan size after optimization: 1

2021-10-03 02:32:02,487 [main] INFO org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized

2021-10-03 02:32:02,487 [main] INFO org.apache.pig.tools.pigstats.ScriptState - Pig script settings are added to the job

2021-10-03 02:32:02,490 [main] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.JobControlCompiler -
mapred.job.reduce.markreset.buffer.percent is not set, set to default 0.3

2021-10-03 02:32:02,503 [main] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.JobControlCompiler - Setting up single store job

2021-10-03 02:32:02,504 [main] INFO org.apache.pig.data.SchemaTupleFrontend - Key [pig.schematuple] is false, will not generate code.

2021-10-03 02:32:02,504 [main] INFO org.apache.pig.data.SchemaTupleFrontend - Starting process to move generated code to distributed cache

2021-10-03 02:32:02,504 [main] INFO org.apache.pig.data.SchemaTupleFrontend - Distributed cache not supported or needed in local mode. Setting key [pig.schematuple.local.dir] with code temp directory: /tmp/1632562322504-0

2021-10-03 02:32:02,513 [main] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - 1 map-reduce job(s) waiting for submission.

2021-10-03 02:32:02,515 [JobControl] INFO org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized

2021-10-03 02:32:02,528 [JobControl] WARN org.apache.hadoop.mapreduce.JobResourceUploader
- No job jar file set. User classes may not be found. See Job or Job#setJar(String).

2021-10-03 02:32:02,538 [JobControl] INFO
org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1

2021-10-03 02:32:02,538 [JobControl] INFO
org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1

2021-10-03 02:32:02,538 [JobControl] INFO
org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths (combined) to process : 1

2021-10-03 02:32:02,540 [JobControl] INFO org.apache.hadoop.mapreduce.JobSubmitter - number of splits:1

2021-10-03 02:32:02,619 [JobControl] INFO org.apache.hadoop.mapreduce.JobSubmitter - Submitting tokens for job: job_local437355495_0003

2021-10-03 02:32:02,793 [JobControl] INFO org.apache.hadoop.mapreduce.Job - The url to track the job: <http://localhost:8080/>

2021-10-03 02:32:02,797 [Thread-15] INFO org.apache.hadoop.mapred.LocalJobRunner - OutputCommitter set in config null

2021-10-03 02:32:02,804 [Thread-15] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS

2021-10-03 02:32:02,804 [Thread-15] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.reduce.markreset.buffer.percent is deprecated. Instead, use mapreduce.reduce.markreset.buffer.percent

2021-10-03 02:32:02,804 [Thread-15] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address

2021-10-03 02:32:02,804 [Thread-15] INFO org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum

2021-10-03 02:32:02,804 [Thread-15] INFO
org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter - File Output Committer Algorithm version is 1

2021-10-03 02:32:02,804 [Thread-15] INFO
org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter - FileOutputCommitter skip cleanup _temporary folders under output directory:false, ignore cleanup failures: false

2021-10-03 02:32:02,804 [Thread-15] INFO org.apache.hadoop.mapred.LocalJobRunner - OutputCommitter is
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.PigOutputCommitter

2021-10-03 02:32:02,810 [Thread-15] INFO org.apache.hadoop.mapred.LocalJobRunner - Waiting for map tasks

2021-10-03 02:32:02,811 [LocalJobRunner Map Task Executor #0] INFO
org.apache.hadoop.mapred.LocalJobRunner - Starting task:
attempt_local437355495_0003_m_000000_0

2021-10-03 02:32:02,875 [LocalJobRunner Map Task Executor #0] INFO

org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter - File Output Committer Algorithm version is 1

2021-10-03 02:32:02,875 [LocalJobRunner Map Task Executor #0] INFO
org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter - FileOutputCommitter skip cleanup
_temporary folders under output directory:false, ignore cleanup failures: false

2021-10-03 02:32:02,882 [LocalJobRunner Map Task Executor #0] INFO
org.apache.hadoop.mapred.Task - Using ResourceCalculatorProcessTree : []

2021-10-03 02:32:02,911 [LocalJobRunner Map Task Executor #0] INFO
org.apache.hadoop.mapred.MapTask - Processing split: Number of splits :1

Total Length = 44

Input split[0]:

Length = 44

Locations:

2021-10-03 02:32:02,917 [LocalJobRunner Map Task Executor #0] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.PigRecordReader - Current split
being processed <file:/home/cloudera/Desktop/sample2.txt:0+44>

2021-10-03 02:32:02,921 [LocalJobRunner Map Task Executor #0] INFO
org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter - File Output Committer Algorithm
version is 1

2021-10-03 02:32:02,921 [LocalJobRunner Map Task Executor #0] INFO
org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter - FileOutputCommitter skip cleanup
_temporary folders under output directory:false, ignore cleanup failures: false

2021-10-03 02:32:02,942 [LocalJobRunner Map Task Executor #0] INFO
org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate
code.

2021-10-03 02:32:02,950 [LocalJobRunner Map Task Executor #0] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.PigMapOnly\$Map - Aliases being
processed per job phase (AliasName[line,offset]): M: job[4,6],job[-1,-1] C: R:

2021-10-03 02:32:02,952 [LocalJobRunner Map Task Executor #0] INFO
org.apache.hadoop.mapred.LocalJobRunner -

2021-10-03 02:32:02,952 [LocalJobRunner Map Task Executor #0] INFO
org.apache.hadoop.mapred.Task - Task:attempt_local437355495_0003_m_000000_0 is done. And is
in the process of committing

2021-10-03 02:32:02,954 [LocalJobRunner Map Task Executor #0] INFO
org.apache.hadoop.mapred.LocalJobRunner -

2021-10-03 02:32:02,954 [LocalJobRunner Map Task Executor #0] INFO
org.apache.hadoop.mapred.Task - Task attempt_local437355495_0003_m_000000_0 is allowed to

commit now

2021-10-03 02:32:02,956 [LocalJobRunner Map Task Executor #0] INFO
org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter - Saved output of task
'attempt_local437355495_0003_m_000000_0' to file:/tmp/temp1185102156/tmp-1194264816/ temporary/0/task local437355495_0003_m_000000

2021-10-03 02:32:02,957 [LocalJobRunner Map Task Executor #0] INFO
org.apache.hadoop.mapred.LocalJobRunner - map

2021-10-03 02:32:02,957 [LocalJobRunner Map Task Executor #0] INFO
org.apache.hadoop.mapred.Task - Task 'attempt_local437355495_0003_m_000000_0' done.

2021-10-03 02:32:02,957 [LocalJobRunner Map Task Executor #0] INFO
org.apache.hadoop.mapred.LocalJobRunner - Finishing task:
attempt_local437355495_0003_m_000000_0

2021-10-03 02:32:02,957 [Thread-15] INFO org.apache.hadoop.mapred.LocalJobRunner - map task
executor complete.

2021-10-03 02:32:03,014 [main] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher -
HadoopJobId: job_local437355495_0003

2021-10-03 02:32:03,014 [main] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Processing
aliases job

2021-10-03 02:32:03,014 [main] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - detailed
locations: M: job[4,6],job[-1,-1] C: R:

2021-10-03 02:32:09,018 [main] WARN org.apache.pig.tools.pigstats.PigStatsUtil - Failed to get
RunningJob for job job_local437355495_0003

2021-10-03 02:32:09,018 [main] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - 100%
complete

2021-10-03 02:32:09,018 [main] INFO org.apache.pig.tools.pigstats.SimplePigStats - Detected Local
mode. Stats reported below may be incomplete

2021-10-03 02:32:09,018 [main] INFO org.apache.pig.tools.pigstats.SimplePigStats - Script
Statistics:

HadoopVersion	PigVersion	UserId	StartedAt	FinishedAt	Features
2.6.0-cdh5.12.0	0.12.0-cdh5.12.0	cloudera	2021-10-03 02:32:02	2021-10-03	
02:32:09	UNKNOWN				

Success!

Job Stats (time in seconds):

JobId Alias Feature Outputs

job_local437355495_0003 job MAP_ONLY <file:/tmp/temp1185102156/tmp-1194264816>,

Input(s):

Successfully read records from: "/home/cloudera/Desktop/sample2.txt"

Output(s):

Successfully stored records in: "<file:/tmp/temp1185102156/tmp-1194264816>"

Job DAG:

job_local437355495_0003

2021-10-03 02:32:15,019 [main] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!

2021-10-03 02:32:15,021 [main] INFO org.apache.hadoop.conf.Configuration.deprecation -
fs.default.name is deprecated. Instead, use fs.defaultFS

2021-10-03 02:32:15,021 [main] INFO org.apache.hadoop.conf.Configuration.deprecation -
mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address

2021-10-03 02:32:15,022 [main] INFO org.apache.hadoop.conf.Configuration.deprecation -
io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum

2021-10-03 02:32:15,022 [main] WARN org.apache.pig.data.SchemaTupleBackend -
SchemaTupleBackend has already been initialized

2021-10-03 02:32:15,043 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat -
Total input paths to process : 1

2021-10-03 02:32:15,043 [main] INFO
org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1

(arpit,analyst)

(ravi,developer)

(mann,engineer)

6. Projection of table (student):

2021-10-03 02:33:43,349 [main] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!

2021-10-03 02:33:43,355 [main] INFO org.apache.hadoop.conf.Configuration.deprecation -

fs.default.name is deprecated. Instead, use fs.defaultFS

2021-10-03 02:33:43,360 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address

2021-10-03 02:33:43,360 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum

2021-10-03 02:33:43,360 [main] WARN org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend has already been initialized

2021-10-03 02:33:43,396 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1

2021-10-03 02:33:43,396 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1

(Arpit)

(Ravi)

(Mann)

(Pradnesh)

(Gaurav)

(Jeet)

(Pulkit)

(Saurav)

(Dhrumil)

7. Inner join between tables 'student' and 'job':

2021-10-03 02:38:52,429 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!

2021-10-03 02:38:52,433 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS

2021-10-03 02:38:52,433 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address

2021-10-03 02:38:52,433 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum

2021-10-03 02:38:52,434 [main] WARN org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend has already been initialized

2021-10-03 02:38:52,474 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1

2021-10-03 02:38:52,474 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1

(1,Arpit,102.0,F,Arpit,analyst)

(2,Ravi,104.0,F,Ravi,developer)

(3,Mann,120.0,F,Mann,engineer)

8. Cross product of tables 'student' and 'job':

(,,,Arpit,analyst)

(9,Dhruvil,45.0,F,,)

(9,Dhruvil,45.0,F,Mann,engineer)

(9,Dhruvil,45.0,F,Ravi,developer)

(9,Dhruvil,45.0,F,Arpit,analyst)

(8,Saurav,50.0,F,,)

(8,Saurav,50.0,F,Mann,engineer)

(8,Saurav,50.0,F,Ravi,developer)

(8,Saurav,50.0,F,Arpit,analyst)

(7,Pulkit,60.0,F,,)

(7,Pulkit,60.0,F,Mann,engineer)

(7,Pulkit,60.0,F,Ravi,developer)

(7,Pulkit,60.0,F,Arpit,analyst)

(6,Jeet,78.0,F,,)

(6,Jeet,78.0,F,Mann,engineer)

(6,Jeet,78.0,F,Ravi,developer)

(6,Jeet,78.0,F,Arpit,analyst)

(5,Gaurav,80.0,F,,)

(5,Gaurav,80.0,F,Mann,engineer)

(5,Gaurav,80.0,F,Ravi,developer)

(5,Gaurav,80.0,F,Arpit,analyst)

(4,Pradnesh,89.0,F,,)

(4,Pradnesh,89.0,F,Mann,engineer)

(4,Pradnesh,89.0,F,Ravi,developer)

(4,Pradnesh,89.0,F,Arpit,analyst)

(3,Mann,120.0,F,,)

(3,Mann,120.0,F,Mann,engineer)

(3,Mann,120.0,F,Ravi,developer)

(3,Mann,120.0,F,Arpit,analyst)

(2,Ravi,104.0,F,,)

(2,Ravi,104.0,F,Mann,engineer)

(2,Ravi,104.0,F,Ravi,developer)

(2,Ravi,104.0,F,Arpit,analyst)

(1,Arpit,102.0,F,,)

(1,Arpit,102.0,F,Mann,engineer)

(1,Arpit,102.0,F,Ravi,developer)

(1,Arpit,102.0,F,Arpit,analyst)

9. Cogroup (grouping) of tables 'student' and 'job':

2021-10-03 02:41:55,178 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!

2021-10-03 02:41:55,178 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS

2021-10-03 02:41:55,178 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address

2021-10-03 02:41:55,178 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum

2021-10-03 02:41:55,179 [main] WARN org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend has already been initialized

2021-10-03 02:41:55,209 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1

2021-10-03 02:41:55,209 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1

(Mann,{(3,Mann,120.0,F)},{(Mann,engineer)})

(Ravi,{(2,Ravi,104.0,F)},{(Ravi,developer)})

(Arpit,{(1,Arpit,102.0,F)},{(Arpit,analyst)})

10. Nested Projection of the above cogroup:

2021-10-03 02:44:38,081 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!

2021-10-03 02:44:38,090 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS

2021-10-03 02:44:38,090 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address

2021-10-03 02:44:38,091 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum

2021-10-03 02:44:38,091 [main] WARN org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend has already been initialized

2021-10-03 02:44:38,166 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1

2021-10-03 02:44:38,166 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1

(Mann,{(Mann,120.0)},{(engineer)})

(Ravi,{(Ravi,104.0)},{(developer)})

(Arpit,{(Arpit,102.0)},{(analyst)})

Student.txt:

1,Arpit,102,F,Male

2,Ravi,104,F,Male

3,Mann,120,F,Male

4,Pradnesh,89,F,Male

5,Gaurav,80,F,Male

6,Jeet,78,F,Male

7,Pulkit,60,F,Male

8,Saurav,50,F,Male

9,Dhrumil,45,F,Male

Sample2.txt:

Arpit,analyst

Ravi,developer

Mann,engineer

Conclusion: Successfully executed the pig commands.

Experiment No. 7

Aim: Implementation of HIVE commands and Importing MYSQL database to HIVE and exporting HIVE database to MYSQL.

Theory:

HIVE: What is HIVE?

Hive is a data warehouse infrastructure tool to process structured data in Hadoop. It resides on top of Hadoop to summarize Big Data and makes querying and analysing easy.

Initially Hive was developed by Facebook, later the Apache Software Foundation took it up and developed it as an open source under the name Apache Hive. For ex, Amazon uses it in Amazon Elastic MapReduce.

HQL: What is HQL.

Hive defines a simple SQL-like query language to querying and managing large datasets called Hive-QL (HQL). It's easy to use if you're familiar with SQL Language. Hive allows programmers who are familiar with the language to write the custom MapReduce framework to perform more sophisticated analysis.

Uses of Hive

1. The Apache Hive distributed storage.
2. Hive provides tools to enable easy data extract/transform/load (ETL)
3. It provides the structure on a variety of data formats.
4. By using Hive, we can access files stored in Hadoop Distributed File System (HDFS is used to querying and managing large datasets residing in) or in other data storage systems such as Apache HBase.

Components of HIVE:

The five major components of Hive and its interaction with the Hadoop is described below:

- **User Interface (UI)** –
User interface provide an interface between user and hive. It enables user to submit queries and other operations to the system. Hive web UI, Hive command line, and Hive HD Insight (In windows server) are supported by the user interface.
- **Driver** –
Queries of the user after the interface are received by the driver within the Hive. Concept of session handles is implemented by driver. Execution and Fetching of APIs modelled on JDBC/ODBC interfaces is provided by the user.
- **Compiler** –
Queries are parses, semantic analysis on the different query blocks and query expression is done by the compiler. Execution plan with the help of the table in the database and partition metadata observed from the metastore are generated by the compiler eventually.
- **Metastore** –
All the structured data or information of the different tables and partition in the warehouse containing attributes and attributes level information are stored in the metastore. Sequences or de-sequences necessary to read and write data and the corresponding HDFS files where the data is stored. Hive selects corresponding database servers to stock the schema or Metadata of databases, tables, attributes in a table, data types of databases, and HDFS mapping.


```
[cloudera@quickstart ~]$ hive
```

Logging initialized using configuration in file:/etc/hive/conf.dist/hive-log4j.properties

WARNING: Hive CLI is deprecated and migration to Beeline is recommended.

```
hive> show databases;
```

```
OK
```

```
default
```

```
Time taken: 2.687 seconds, Fetched: 1 row(s)
```

```
hive> show tables;
```

```
OK
```

```
Time taken: 2.16 seconds
```

```
hive> use dbname;
```

```
FAILED: SemanticException [Error 10072]: Database does not exist: dbname
```

```
hive> create database retail;
```

```
OK
```

```
Time taken: 25.163 seconds
```

```
hive> use retail;
```

```
OK
```

```
Time taken: 1.153 seconds
```

```
hive> create table emp(id INT,name STRING,sal DOUBLE)row format delimited fields terminated by  
' ' stored as textfile;
```

```
OK
```

```
Time taken: 4.305 seconds
```

```
hive> use retail;
```

```
OK
```

```
Time taken: 0.348 seconds
```

```
hive> describe emp;
```

```
OK
```

id	int
name	string
sal	double

```
Time taken: 2.349 seconds, Fetched: 3 row(s)
```

```
hive> cat /home/workspace/training/demo.txt
```

```
> [cloudera@quickstart ~]$ cat /home/workspace/training/demo.txt
```

```
cat: /home/workspace/training/demo.txt: No such file or directory
```

```
[cloudera@quickstart ~]$ hive
```

Logging initialized using configuration in file:/etc/hive/conf.dist/hive-log4j.properties

WARNING: Hive CLI is deprecated and migration to Beeline is recommended.

```
hive> load data local inpath '/home/workspace/training/demo.txt' into table emp;
```

```
FAILED: SemanticException [Error 10001]: Line 1:70 Table not found 'emp'
```

```
hive> select * from emp;
```

```
FAILED: SemanticException [Error 10001]: Line 1:14 Table not found 'emp'
```

```
hive> describe emp
```

```
> describe emp;
```

```
FAILED: ParseException line 2:9 extraneous input 'emp' expecting EOF near '<EOF>'
```

```
hive> use retail;
```

```
OK
```

```
Time taken: 0.276 seconds
```

```
hive> describe emp;
```

```
OK
```



```

id          int
name        string
sal         double
Time taken: 1.98 seconds, Fetched: 3 row(s)
hive> load data local inpath '/home/workspace/training/demo.txt' into table emp;
FAILED: SemanticException Line 1:23 Invalid path "/home/workspace/training/demo.txt": No files
matching path file:/home/workspace/training/demo.txt
hive> load data local inpath '/home/workspace/training/demo.txt' into table emp;
FAILED: SemanticException Line 1:23 Invalid path "/home/workspace/training/demo.txt": No files
matching path file:/home/workspace/training/demo.txt
hive> load data local inpath '/home/cloudera/workspace/training/demo.txt' into table emp;
Loading data to table retail.emp
Failed with exception org.apache.hadoop.ipc.RemoteException(java.io.IOException): File
/user/hive/warehouse/retail.db/emp/demo.txt could only be replicated to 0 nodes instead of
minReplication (=1). There are 1 datanode(s) running and 1 node(s) are excluded in this operation.
    at
org.apache.hadoop.hdfs.server.blockmanagement.BlockManager.chooseTarget4NewBlock(BlockMa
nager.java:1716)
    at
org.apache.hadoop.hdfs.server.namenode.FSNamesystem.getAdditionalBlock(FSNamesystem.java:3
385)
    at
org.apache.hadoop.hdfs.server.namenode.NameNodeRpcServer.addBlock(NameNodeRpcServer.jav
a:683)
    at
org.apache.hadoop.hdfs.server.namenode.AuthorizationProviderProxyClientProtocol.addBlock(Auth
orizationProviderProxyClientProtocol.java:214)
    at
org.apache.hadoop.hdfs.protocolPB.ClientNamenodeProtocolServerSideTranslatorPB.addBlock(Clien
tNamenodeProtocolServerSideTranslatorPB.java:495)
    at
org.apache.hadoop.hdfs.protocol.proto.ClientNamenodeProtocolProtos$ClientNamenodeProtocol$
2.callBlockingMethod(ClientNamenodeProtocolProtos.java)
    at
org.apache.hadoop.ipc.ProtobufRpcEngine$Server$ProtoBufRpcInvoker.call(ProtobufRpcEngine.java
:617)
        at org.apache.hadoop.ipc.RPC$Server.call(RPC.java:1073)
        at org.apache.hadoop.ipc.Server$Handler$1.run(Server.java:2217)
        at org.apache.hadoop.ipc.Server$Handler$1.run(Server.java:2213)
        at java.security.AccessController.doPrivileged(Native Method)
        at javax.security.auth.Subject.doAs(Subject.java:415)
        at
org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1917)
        at org.apache.hadoop.ipc.Server$Handler.run(Server.java:2211)

FAILED: Execution Error, return code 1 from org.apache.hadoop.hive ql.exec.MoveTask
hive> load data local inpath '/home/cloudera/workspace/training/demo.txt' into table emp;
Loading data to table retail.emp
Table retail.emp stats: [numFiles=2, totalSize=22]
OK
Time taken: 6.873 seconds

```

```

hive> select * from emp;
OK
1      abc      2000.0
2      pqr      4500.0
Time taken: 1.638 seconds, Fetched: 2 row(s)
hive> use retail;
OK
Time taken: 0.03 seconds
hive> alter table emp rename to emp_sal;
OK
Time taken: 1.025 seconds
hive> select * from emp_sal where id=1;
OK
1      abc      2000.0
Time taken: 1.218 seconds, Fetched: 1 row(s)
hive> select count(*) from emp_sal;
Query ID = cloudera_20210917011515_c598ad2c-be76-4481-b780-27672582162b
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1631861277051_0001, Tracking URL =
http://quickstart.cloudera:8088/proxy/application_1631861277051_0001/
Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job_1631861277051_0001
Hadoop job information for Stage-1: number of mappers: 2; number of reducers: 1
2021-09-17 01:18:25,016 Stage-1 map = 0%, reduce = 0%
2021-09-17 01:19:41,193 Stage-1 map = 0%, reduce = 0%
2021-09-17 01:21:00,292 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 5.45 sec
2021-09-17 01:21:36,660 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 7.68 sec
MapReduce Total cumulative CPU time: 7 seconds 680 msec
Ended Job = job_1631861277051_0001
MapReduce Jobs Launched:
Stage-Stage-1: Map: 2 Reduce: 1 Cumulative CPU: 7.68 sec HDFS Read: 10319 HDFS Write: 2
SUCCESS
Total MapReduce CPU Time Spent: 7 seconds 680 msec
OK
2
Time taken: 346.462 seconds, Fetched: 1 row(s)
hive> select count(*) from emp_sal;
Query ID = cloudera_20210917012222_da67f150-a685-4ba3-8ec6-96c0e2439402
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:

```

```

set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
set mapreduce.job.reduces=<number>
Starting Job = job_1631861277051_0002, Tracking URL =
http://quickstart.cloudera:8088/proxy/application_1631861277051_0002/
Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job_1631861277051_0002
Hadoop job information for Stage-1: number of mappers: 2; number of reducers: 1
2021-09-17 01:22:48,712 Stage-1 map = 0%, reduce = 0%
2021-09-17 01:24:41,209 Stage-1 map = 0%, reduce = 0%
2021-09-17 01:24:49,199 Stage-1 map = 50%, reduce = 0%, Cumulative CPU 4.14 sec
2021-09-17 01:24:52,372 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 5.83 sec
2021-09-17 01:25:55,876 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 5.83 sec
2021-09-17 01:25:57,892 Stage-1 map = 100%, reduce = 33%, Cumulative CPU 6.76 sec
2021-09-17 01:26:05,437 Stage-1 map = 100%, reduce = 67%, Cumulative CPU 7.59 sec
2021-09-17 01:26:10,368 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 8.74 sec
MapReduce Total cumulative CPU time: 10 seconds 510 msec
Ended Job = job_1631861277051_0002
MapReduce Jobs Launched:
Stage-Stage-1: Map: 2 Reduce: 1 Cumulative CPU: 10.51 sec HDFS Read: 10510 HDFS Write: 2
SUCCESS
Total MapReduce CPU Time Spent: 10 seconds 510 msec
OK
2
Time taken: 256.919 seconds, Fetched: 1 row(s)
hive> drop table emp_sal;
OK
Time taken: 85.768 seconds
hive> exit;
WARN: The method class org.apache.commons.logging.impl.SLF4JLogFactory#release() was invoked.
WARN: Please see http://www.slf4j.org/codes.html#release for an explanation.
[cloudera@quickstart ~]$

```

Conclusion: Successfully imported data from MySQL to Hive and export from Hive to MySQL using sqoop.

Exp-8 Distance Formulae

Aim: Write a java program to calculate distance using following distance formulae - L1 norm (Manhattan distance) , L2 norm (Euclidean distance), L_∞ norm ,Jaccard distance, Cosine distance, Edit distance, Hamming Distance

Theory:

Manhattan Distance:

The Manhattan distance estimates the distance between two real-valued vectors or points. It is calculated as the sum of the absolute differences of their Cartesian coordinates.

Mathematically:

Given two points p and q in a Euclidean space of dimension n, with coordinates (p_1, p_2, \dots, p_n) and (q_1, q_2, \dots, q_n) , the Manhattan distance between them is defined as:

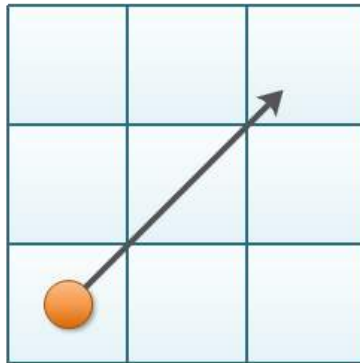
$$|p_1 - q_1| + |p_2 - q_2| + \dots + |p_n - q_n|$$

Euclidean Distance:

Euclidean distance is the shortest distance between two points in an N dimensional space also known as Euclidean space. It is used as a common metric to measure the similarity between two data points and used in various fields such as geometry, data mining, deep learning and others.

It is, also, known as Euclidean norm, Euclidean metric, L2 norm, L2 metric and Pythagorean metric. The concept of Euclidean distance is captured by this image:

Euclidean Distance



$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

L-infinity norm:

Gives the largest magnitude among each element of a vector.

Having the vector $X = [-6, 4, 2]$, the L-infinity norm is 6.

In L-infinity norm, only the largest element has any effect. So, for example, if your vector represents the cost of constructing a building, by minimizing L-infinity norm we are reducing the cost of the most expensive building.

Jaccard Distance:

The Jaccard Index and the Jaccard Distance between the two sets can be calculated by using the formula:

Here, A and B are two sets of strings.

$$JaccardIndex = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

$$JaccardDistance = 1 - JaccardIndex$$

Cosine Similarity and Cosine Distance:

Cosine similarity says that to find the similarity between two points or vectors we need to find Angle between them.

Formula to find the Cosine Similarity and Distance is as below:

$$1 - \text{Cosine_Similarity} = \text{Cosine_Distance}$$

$$\text{Cosine_Similarity} = \cos \theta(\text{theta})$$

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}},$$

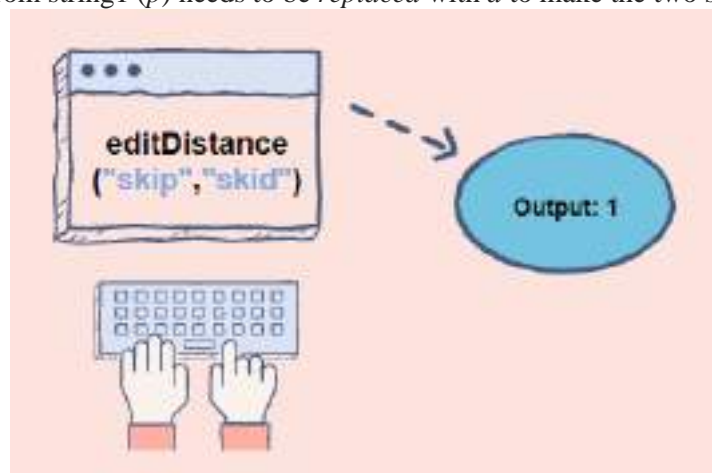
Here A=Point P1,B=Point P2 (in our example)

Edit Distance:

The edit distance is the minimum number of changes required to convert string1 to string2. This conversion can be brought on by:

- Insertion
- Removal
- Replacement

The time complexity for all three of these processes is $O(1)$. As shown in the illustration below, only the last word from string1 (*p*) needs to be *replaced* with *d* to make the two strings equal.



Dynamic programming is the best approach to use when solving this problem. To begin, traverse both strings from either end. If both of the characters are the same, then move to the next iteration; else, compute the minimum cost for all three processes (i.e., insert, remove, and replace) and choose the option with the lowest cost.

Hamming distance:

Hamming distance is a metric for comparing two binary data strings. While comparing two binary strings of equal length, Hamming distance is the number of bit positions in which the two bits are different.

The Hamming distance between two strings, a and b is denoted as $d(a,b)$.

It is used for error detection or error correction when data is transmitted over computer networks. It is also using in coding theory for comparing equal length data words.

Calculation of Hamming Distance

In order to calculate the Hamming distance between two strings, and , we perform their XOR operation, $(a \oplus b)$, and then count the total number of 1s in the resultant string.

Example

Suppose there are two strings 1101 1001 and 1001 1101.

$11011001 \oplus 10011101 = 01000100$. Since, this contains two 1s, the Hamming distance, $d(11011001, 10011101) = 2$.

Program:

```
import java.io.*;
import java.util.*;
import java.util.Random;
import java.util.function.*;

import static java.lang.Math.*;
import static java.lang.System.*;

class Main {

    public static void printVectors(int[][] V, boolean isHamming) {
        if(isHamming) {out.println("Boolean Vectors");}
        else {out.println("Integer Vectors");}
        for(int i = 0; i < 2; i++) {
            out.print("Vector " + (i + 1) + ": ");
            for(int j = 0; j < V[i].length; j++) {
                out.print(V[i][j] + " ");
            }
            out.println("\n");
        }
    }

    public static void printStrings(String[] strings) {
        for(int i = 0; i < 2; i++) {
            out.println("String " + (i + 1) + ": " + strings[i]);
        }
    }

    public static void main(String args[]) {

        Functions f = new Functions();
        DataGenerator dg = new DataGenerator(5);

        int[][] vectors = dg.generateVectorPair(false);
        int[][] boolVectors = dg.generateVectorPair(true);
        String[] strings = dg.generateStringPair();

        double L1 = f.norm(1, vectors);
        double L2 = f.norm(2, vectors);
        double LMax = f.maxNorm(vectors);
```

```

double jaccardDistance = f.jaccardDistance(vectors);
double cosineDistance = f.cosineDistance(vectors);
int editDistance = f.editDistance(strings);
int hammingDistance = f.hammingDistance(boolVectors);

printVectors(vectors, false);
out.println(
    "L1 Norm: " + L1 + "\n" +
    "L2 Norm: " + L2 + "\n" +
    "L(inf) Norm: " + LMax
);
out.println(
    "Jaccard Distance: " + jaccardDistance + "\n" +
    "Cosine Distance: " + cosineDistance + "\n"
);
printStrings(strings);
out.println("Edit Distance: " + editDistance + "\n");
printVectors(boolVectors, true);
out.println("Hamming Distance: " + hammingDistance);
}

}

class Functions {

    public void printDP(int[][] dp) {
        for(int i = 0; i < dp.length; i++) {
            for(int j = 0; j < dp.length; j++) {
                out.print(dp[i][j] + " ");
            }
            out.print("\n");
        }
    }

    public double norm(int p, int[][] V) {
        int sum = 0;
        for(int i = 0; i < V[0].length; i++) {
            sum += pow(abs(V[0][i] - V[1][i]), p);
        }
        return pow(sum, 1.0 / p);
    }

    public double maxNorm(int[][] V) {
        double _max = 0;
        for(int i = 0; i < V[0].length; i++) {
            _max = max(_max, abs(V[0][i] - V[1][i]));
        }
        return _max;
    }

    public double jaccardDistance(int[][] V) {
        double intersection = 0, union = 0;
        for(int i = 0; i < V[0].length; i++) {
            for(int j = 0; j < V[0].length; j++) {
                if(V[0][i] == V[1][j]) {intersection++;}
            }
        }
        union = V[0].length + V[1].length - intersection;
        return intersection / union;
    }
}

```

```

    }
}
union = 2 * V[0].length - intersection;
return 1.0 - (intersection / union);
}

public double cosineDistance(int[][] V) {
    double dotProduct = 0;
    for(int i = 0; i < V[0].length; i++) {
        dotProduct += V[0][i] * V[1][i];
    }
    int[][] aux = new int[2][V[0].length];
    aux[0] = V[0];
    double norm1 = norm(2, aux);
    aux[0] = V[1];
    double norm2 = norm(2, aux);
    return 1.0 - (dotProduct / (norm1 * norm2));
}

public int editDistance(String[] s) {
    /* Wagner-Fischer Algorithm */
    int subCost = 0;
    int length = s[0].length();
    int[][] dp = new int[length + 1][length + 1];
    for(int i = 1; i <= length; i++) {dp[i][0] = i;}
    for(int i = 1; i <= length; i++) {dp[0][i] = i;}
    for(int j = 1; j <= length; j++) {
        for(int i = 1; i <= length; i++) {
            if(s[0].charAt(i - 1) == s[1].charAt(j - 1)) {subCost = 0;}
            else {subCost = 1;}
            dp[i][j] = min(
                min(dp[i - 1][j] + 1,
                    dp[i][j - 1] + 1),
                dp[i - 1][j - 1] + subCost
            );
        }
    }
    return dp[length][length];
}

public int hammingDistance(int[][] V) {
    int sum = 0;
    for(int i = 0; i < V[0].length; i++) {
        sum += V[0][i] ^ V[1][i];
    }
    return sum;
}

}

class DataGenerator {

    int length;

    public DataGenerator(int length) {

```



```

        this.length = length;
    }

    public int[][] generateVectorPair(boolean isHamming) {
        ArrayList<Integer> vector1 = new ArrayList<Integer>();
        ArrayList<Integer> vector2 = new ArrayList<Integer>();
        Random rand = new Random();
        int range = 10;
        if(isHamming) {
            range = 2;
            for(int i = 0; i < length; i++) {
                vector1.add(rand.nextInt(range));
                vector2.add(rand.nextInt(range));
            }
        } else {
            for(int i = 0; i < length * 2; i++) {
                vector1.add(i);
                vector2.add(i);
            }
        }
        Collections.shuffle(vector1);
        Collections.shuffle(vector2);
        vector1 = new ArrayList<Integer>(vector1.subList(0, length));
        vector2 = new ArrayList<Integer>(vector2.subList(0, length));
        int[][] vectors = new int[2][length];
        vectors[0] = vector1.stream().mapToInt(i -> i).toArray();
        vectors[1] = vector2.stream().mapToInt(i -> i).toArray();
        return vectors;
    }

    public String[] generateStringPair() {
        String characters = "abcdefghi";
        char[] string1 = new char[length];
        char[] string2 = new char[length];
        Random rand = new Random();
        for(int i = 0; i < length; i++) {
            string1[i] = characters.charAt(
                rand.nextInt(characters.length())
            );
            string2[i] = characters.charAt(
                rand.nextInt(characters.length())
            );
        }
        String[] strings = new String[2];
        strings[0] = new String(string1);
        strings[1] = new String(string2);
        return strings;
    }
}

```

Output:

Integer Vectors

Vector 1: 9 8 4 1 0

Vector 2: 8 7 0 3 5

L1 Norm: 13.0
L2 Norm: 6.855654600401044
L(inf) Norm: 5.0
Jaccard Distance: 0.75
Cosine Distance: 0.1511027610989515

String 1: cfdch
String 2: eiahh
Edit Distance: 4

Boolean Vectors
Vector 1: 1 1 1 0 1
Vector 2: 0 1 1 0 0
Hamming Distance: 2

Conclusion: Succesfully implemented program to calculate distance between two sets using given methods.

Experiment No.:9

Aim: Implement k-means clustering using Hadoop Map-Reduce.

Theory:

Kmeans Algorithm

Kmeans algorithm is an iterative algorithm that tries to partition the dataset into K pre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to **only one group**. It tries to make the intra-cluster data points as similar as possible while also keeping the clusters as different (far) as possible. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid (arithmetic mean of all the data points that belong to that cluster) is at the minimum. The less variation we have within clusters, the more homogeneous (similar) the data points are within the same cluster.

The way kmeans algorithm works is as follows:

1. Specify number of clusters K .
2. Initialize centroids by first shuffling the dataset and then randomly selecting K data points for the centroids without replacement.
3. Keep iterating until there is no change to the centroids. i.e assignment of data points to clusters isn't changing.
 - Compute the sum of the squared distance between data points and all centroids.
 - Assign each data point to the closest cluster (centroid).
 - Compute the centroids for the clusters by taking the average of the all data points that belong to each cluster.

The approach kmeans follows to solve the problem is called **Expectation-Maximization**. The E-step is assigning the data points to the closest cluster. The M-step is computing the centroid of each cluster. Below is a break down of how we can solve it mathematically (feel free to skip it).

The objective function is:

$$J = \sum_{i=1}^m \sum_{k=1}^K w_{ik} \|x^i - \mu_k\|^2 \quad (1)$$

where $w_{ik}=1$ for data point x_i if it belongs to cluster k ; otherwise, $w_{ik}=0$. Also, μ_k is the centroid of x_i 's cluster.

It's a minimization problem of two parts. We first minimize J w.r.t. w_{ik} and treat μ_k fixed. Then we minimize J w.r.t. μ_k and treat w_{ik} fixed. Technically speaking, we differentiate J w.r.t. w_{ik} first and update cluster assignments (*E-step*). Then we differentiate J w.r.t. μ_k and recompute the centroids after the cluster assignments from previous step (*M-step*). Therefore, E-step is:

$$\begin{aligned} \frac{\partial J}{\partial w_{ik}} &= \sum_{i=1}^m \sum_{k=1}^K \|x^i - \mu_k\|^2 \\ \Rightarrow w_{ik} &= \begin{cases} 1 & \text{if } k = \operatorname{argmin}_j \|x^i - \mu_j\|^2 \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \quad (2)$$

In other words, assign the data point x_i to the closest cluster judged by its sum of squared distance from cluster's centroid.

And M-step is:

$$\frac{\partial J}{\partial \mu_k} = 2 \sum_{i=1}^m w_{ik} (x^i - \mu_k) = 0$$

$$\Rightarrow \mu_k = \frac{\sum_{i=1}^m w_{ik} x^i}{\sum_{i=1}^m w_{ik}} \quad (3)$$

Which translates to recomputing the centroid of each cluster to reflect the new assignments.

Few things to note here:

- Since clustering algorithms including kmeans use distance-based measurements to determine the similarity between data points, it's recommended to standardize the data to have a mean of zero and a standard deviation of one since almost always the features in any dataset would have different units of measurements such as age vs income.
- Given kmeans iterative nature and the random initialization of centroids at the start of the algorithm, different initializations may lead to different clusters since kmeans algorithm may *stuck in a local optimum and may not converge to global optimum*. Therefore, it's recommended to run the algorithm using different initializations of centroids and pick the results of the run that yielded the lower sum of squared distance.
- Assignment of examples isn't changing is the same thing as no change in within-cluster variation:

$$\frac{1}{m_k} \sum_{i=1}^{m_k} \|x^i - \mu_{c^k}\|^2 \quad (4)$$

Code:

```
import java.io.IOException;
import java.util.*;
import java.io.*;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.filecache.DistributedCache;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.mapred.Reducer;

@SuppressWarnings("deprecation")
public class KMeans {
    public static String OUT = "outfile";
    public static String IN = "inputlarger";
    public static String CENTROID_FILE_NAME = "/centroid.txt";
    public static String OUTPUT_FILE_NAME = "/part-00000";
    public static String DATA_FILE_NAME = "/data.txt";
    public static String JOB_NAME = "KMeans";
    public static String SPLITTER = "\\t ";
    public static List<Double> mCenters = new ArrayList<Double>();

    /*
     * In Mapper class we are overriding configure function. In this we are
     * reading file from Distributed Cache and then storing that into instance
     */
}
```

```

    * variable "mCenters"
    */
    public static class Map extends MapReduceBase implements
        Mapper<LongWritable, Text, DoubleWritable, DoubleWritable> {
        @Override
        public void configure(JobConf job) {
            try {
                // Fetch the file from Distributed Cache Read it and store the
                // centroid in the ArrayList
                Path[] cacheFiles = DistributedCache.getLocalCacheFiles(job);
                if (cacheFiles != null && cacheFiles.length > 0) {
                    String line;
                    mCenters.clear();
                    BufferedReader cacheReader = new BufferedReader(
                        new FileReader(cacheFiles[0].toString()));
                    try {
                        // Read the file split by the splitter and store it in
                        // the list
                        while ((line = cacheReader.readLine()) != null) {
                            String[] temp = line.split(SPLITTER);
                            mCenters.add(Double.parseDouble(temp[0]));
                        }
                    } finally {
                        cacheReader.close();
                    }
                }
            } catch (IOException e) {
                System.err.println("Exception reading DistribtuedCache: " + e);
            }
        }

        /*
        * Map function will find the minimum center of the point and emit it to
        * the reducer
        */
        @Override
        public void map(LongWritable key, Text value,
            OutputCollector<DoubleWritable, DoubleWritable> output,
            Reporter reporter) throws IOException {
            String line = value.toString();
            double point = Double.parseDouble(line);
            double min1, min2 = Double.MAX_VALUE, nearest_center = mCenters
                .get(0);
            // Find the minimum center from a point
            for (double c : mCenters) {
                min1 = c - point;
                if (Math.abs(min1) < Math.abs(min2)) {
                    nearest_center = c;
                }
            }
            output.collect(nearest_center, key);
        }
    }
}

```

```

        min2 = min1;
    }
}
// Emit the nearest center and the point
output.collect(new DoubleWritable(nearest_center),
    new DoubleWritable(point));
}
}

public static class Reduce extends MapReduceBase implements
    Reducer<DoubleWritable, DoubleWritable, DoubleWritable, Text> {

    /*
     * Reduce function will emit all the points to that center and calculate
     * the next center for these points
     */
    @Override
    public void reduce(DoubleWritable key, Iterator<DoubleWritable> values,
        OutputCollector<DoubleWritable, Text> output, Reporter reporter)
        throws IOException {
        double newCenter;
        double sum = 0;
        int no_elements = 0;
        String points = "";
        while (values.hasNext()) {
            double d = values.next().get();
            points = points + " " + Double.toString(d);
            sum = sum + d;
            ++no_elements;
        }

        // We have new center now
        newCenter = sum / no_elements;

        // Emit new center and point
        output.collect(new DoubleWritable(newCenter), new Text(points));
    }
}

public static void main(String[] args) throws Exception {
    run(args);
}

public static void run(String[] args) throws Exception {
    IN = args[0];
    OUT = args[1];
    String input = IN;
    String output = OUT + System.nanoTime();
}

```

```

String again_input = output;

// Reiterating till the convergence
int iteration = 0;
boolean isdone = false;
while (isdone == false) {
    JobConf conf = new JobConf(KMeans.class);
    if (iteration == 0) {
        Path hdfsPath = new Path(input + CENTROID_FILE_NAME);
        // upload the file to hdfs. Overwrite any existing copy.
        DistributedCache.addCacheFile(hdfsPath.toUri(), conf);
    } else {
        Path hdfsPath = new Path(again_input + OUTPUT_FIE_NAME);
        // upload the file to hdfs. Overwrite any existing copy.
        DistributedCache.addCacheFile(hdfsPath.toUri(), conf);
    }

    conf.setJobName(JOB_NAME);
    conf.setMapOutputKeyClass(DoubleWritable.class);
    conf.setMapOutputValueClass(DoubleWritable.class);
    conf.setOutputKeyClass(DoubleWritable.class);
    conf.setOutputValueClass(Text.class);
    conf.setMapperClass(Map.class);
    conf.setReducerClass(Reduce.class);
    conf.setInputFormat(TextInputFormat.class);
    conf.setOutputFormat(TextOutputFormat.class);

    FileInputFormat.setInputPaths(conf,
        new Path(input + DATA_FILE_NAME));
    FileOutputFormat.setOutputPath(conf, new Path(output));

    JobClient.runJob(conf);

    Path ofile = new Path(output + OUTPUT_FIE_NAME);
    FileSystem fs = FileSystem.get(new Configuration());
    BufferedReader br = new BufferedReader(new InputStreamReader(
        fs.open(ofile)));
    List<Double> centers_next = new ArrayList<Double>();
    String line = br.readLine();
    while (line != null) {
        String[] sp = line.split("\\t| ");
        double c = Double.parseDouble(sp[0]);
        centers_next.add(c);
        line = br.readLine();
    }
    br.close();

    String prev;

```

```

if (iteration == 0) {
    prev = input + CENTROID_FILE_NAME;
} else {
    prev = again_input + OUTPUT_FILE_NAME;
}
Path prevfile = new Path(prev);
FileSystem fs1 = FileSystem.get(new Configuration());
BufferedReader br1 = new BufferedReader(new InputStreamReader(
    fs1.open(prevfile)));
List<Double> centers_prev = new ArrayList<Double>();
String l = br1.readLine();
while (l != null) {
    String[] sp1 = l.split(SPLITTER);
    double d = Double.parseDouble(sp1[0]);
    centers_prev.add(d);
    l = br1.readLine();
}
br1.close();

// Sort the old centroid and new centroid and check for convergence
// condition
Collections.sort(centers_next);
Collections.sort(centers_prev);

Iterator<Double> it = centers_prev.iterator();
for (double d : centers_next) {
    double temp = it.next();
    if (Math.abs(temp - d) <= 0.1) {
        isdone = true;
    } else {
        isdone = false;
        break;
    }
}
++iteration;
again_input = output;
output = OUT + System.nanoTime();
}
}

```

Conclusion: Kmeans clustering using Mapreduce implemented successfully.

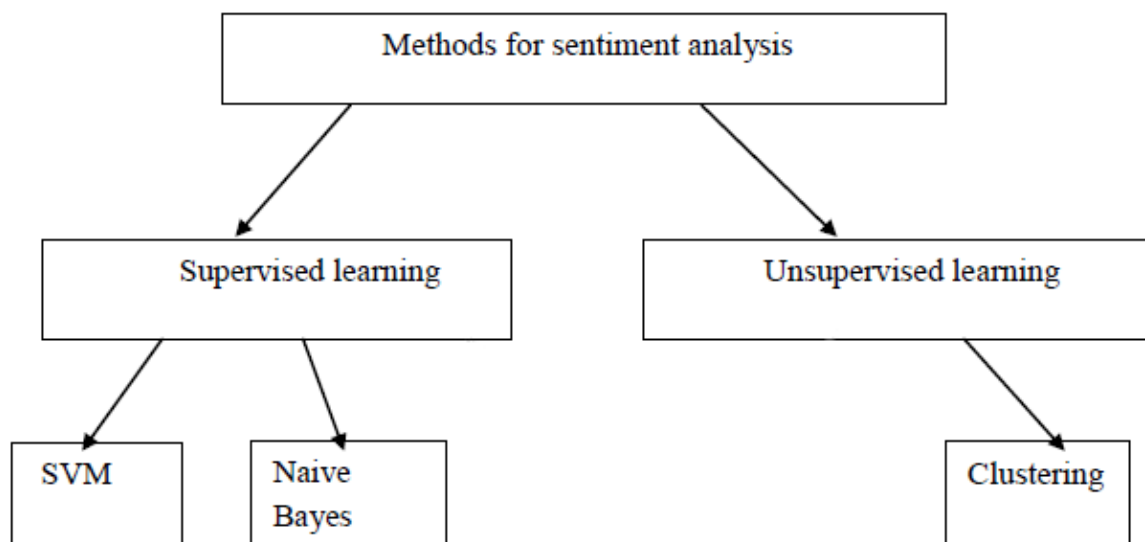
Experiment 10

Twitter Sentiment Analysis

Social media has gained a lot of light in the past few years. Twitter has become an important platform which people are taking up to express their views and opinions about any topic. Sentiment analysis has made it possible to analyse the moods of a person. It can help us to decide the positive, negative or neutral views of a person based on his attitude on a given topic. Previously, it was used for lexical or syntax feature extraction, assigning a polarity label to each given document. These days, sites like Twitter show the influence that surroundings have on online users. It is tough to process big data using traditional techniques. The current Analytics tools and models used that are available in the market are not sufficient to manage big data. Therefore, there is a need to use a cloud storage for such type of applications

To analyse such an enormous amount of data, we make use of Apache Hadoop and its functionalities on the Mahout framework. Hadoop is a framework that performs computations over large datasets. Cloud computing like Hadoop helps to perform operations on distributed data in an efficient manner. Hadoop has an internal framework called MapReduce to perform its functionality. Queries are divided among different nodes, to be performed in parallel. This is known as the Map stage. Then results are combined in the reduce stage to give an output. This provides faster query execution and faster result provision.

Methods used for Sentiment Analysis-



Supervised Machine Learning

This technique mainly contains Support vector machine, Naive Bayes methods. These are classifiers and classify the opinions in two classes. These classifiers need the training set data and test set. Based on these inputs, these methods classify the reviews.

Unsupervised Machine Learning

This technique contains clustering and lexical based analysis and it does not need training set data. Results of polarity of sentiments may vary from one technique or method to another technique and accuracy may differ.

Support vector machine

In support vector machine, each data point is considered as k-dimensional vector which is a list of k numbers. It is checked that whether these points can be separated with k-1 dimensional hyperplane. It is known as linear classifier. Data is classified by these hyperplanes.

Naive bayes

Naive Bayes gave an effective method to carry out the study of classification. It works on the following principles. Suppose there are n possible classes $X=\{x_1, x_2, \dots, x_m\}$ for a domain of documents $Y=\{y_1, y_2, \dots, y_n\}$. Let $W=\{w_1, w_2, \dots, w_n\}$ be a set of words that appear in the documents of Y.

Then

$$P(x|y) = \frac{P(x)P(y|x)}{P(y)}$$

Naive Bayes assumed that each word in W appears independently. Thus, this formula takes the form as shown below:

$$P(x|y) = P(x) \prod_{n=1}^{i_d} P(w_k | x)^{t_k}$$

where i_d is the number of unique words in document d, t_k is the frequency of each word w_k .

By applying Naive Bayes classifier, we can estimate $P(x)$ and $P(w_k | x)$ as:

$$\hat{P}(x) = N_x / N \quad \hat{P}(w_k | x) = \frac{N_{wk}}{\sum_{w \in W} N_{wi}}$$

N is the total number of documents, N_x is the number of documents in class x. N_{wi} is the frequency of word w_i in class x.

Clustering

In clustering technique, terms having similar features belong to one cluster and so there are two clusters are created. One cluster is for the opinions having positive feature and another for the opinions having negative features. And neutral opinions do not belong to any of the cluster. In sentiment analysis, k-means algorithm is used for clustering. In clustering, there are centroids and for each term distance is

calculated from all the centroids and term belongs to that cluster from which its distance is minimum.

Fuzzy c-means clustering

Fuzzy c-means clustering comprises of two processes: in the first phase cluster is calculated and in the next phase, points are assigned to centres by using a formula of distance. This is done till cluster centres get stabilized. The algorithm similar to k-means clustering in many ways but it gives data items a membership value between 0 to 1. Thus it includes a concept of partial membership and provides overlapping clusters to support it. A fuzzification parameter m in the range $[1, \infty]$ is needed to determine the degree of fuzziness in clusters. When m tends to 1, the algorithm functions like a crisp partitioning algorithm. For larger values of m the overlapping in clusters becomes more. The algorithm calculates the membership value μ with the formula,

$$\mu_j(x_i) = \frac{\left(\frac{1}{d_{ji}}\right)^{\frac{1}{m-1}}}{\sum_{k=1}^p \left(\frac{1}{d_{ki}}\right)^{\frac{1}{m-1}}}$$

where

$\mu_j(x_i)$: membership of x_i in cluster j
 d_{ji} : distance of x_i from cluster c_j
 m : fuzzification parameter
 p : number of clusters

d_{ki} : distance of x_i in cluster C_k

The new cluster centers are calculated with these membership values

$$c_j = \frac{\sum_i [\mu_j(x_i)]^m x_i}{\sum_i [\mu_j(x_i)]^m}$$

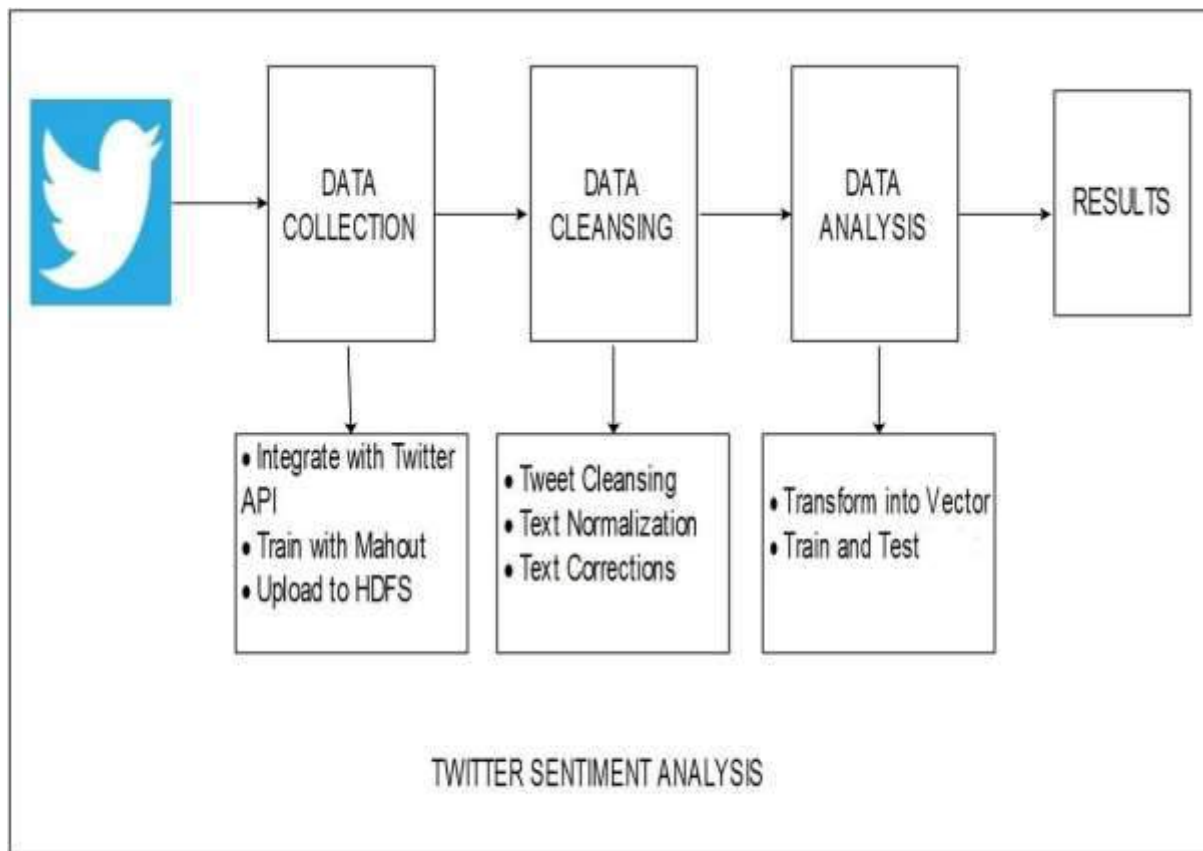
where

C_j : is the center of the j th cluster

x_i : is the i th data point

μ_j : the function which returns the membership

m : is the fuzzification parameter



Data Collection

Twitter API v1.1 does the collection with the help of consumer keys/secrets and access token key/secrets. Finally, we prepare a training set by creating three folders- positive, negative, neutral. Tweets are stored as values in files and names of documents are tweet ids.

PREPROCESSING

So, to remove unnecessary words which are not required for sentiment analysis, preprocessing of data is done which includes following steps which are described in

So, initially dataset is collected and then tokenization is done and output of the tokenization is terms. On these terms stemming and lemmatization is done and after that POS tagging is done. After all these steps pre-processing is finished.

Now, all the steps are described in detail:-

Dataset

Initially dataset is created by collecting data from social media and other sources.

Tokenization

It is a process of breaking the sentence into its constituent parts. And there is not only one process or method to do tokenization. Different algorithms are available for this for different apps. It is very important in the field of sentiment analysis.

.

White space tokenizer

The whitespace tokenizer simply breaks the sentences and divides the text on any sequence of whitespace, tab, or newline characters. For example: "I saw a movie, it is nice and songs are very good". After applying white space tokenizer output is I, saw, a, movie, it, is, nice, and, songs, are, very, good.

Additional Punctuation

Punctuation must be considered at the tokenization stage. So, the goal for tokenizing is to properly differentiate between various senses for individual punctuation marks. The basic strategy for handling punctuation is by trying to identify all the word- internal marks or symbols first, so all others can be tokenized as separate elements.

Stemming and lemmatization

Due to grammatical reasons, sentences may contain the different forms of a word like decorate, decorates and decorating. In addition to these, there are many derivationally related word which possess the similar meaning like automata, automatic and automatically. While doing sentiment analysis, these types of words can be analysed as same word. So the aim of both lemmatization and stemming is to reduce these types of forms and reduce the word from its derivational form to its corresponding base form.

For example: am, are, is be girl, girls, girl's, girls' girl

The result of this mapping of text will be something like:

the girl's cars are different colors the girl car be differ color, whereas, the two words are used in different senses.

Assignment - 1

Q. Write briefly on big data characteristics, Hadoop architecture, Hadoop ecosystem.

Ans. Big data characteristics.

Volume: Huge amount of data is generated during big data applications.

(i) The amount of data generated as well as storage volume is very big in size.

(ii) Velocity: For time critical applications the faster processing is very important.

(iii) The huge amount of data is generated and stored requires higher processing speed of processing data.

(iv) The amount of digital data will be doubled in every 18 months and it repeats many times in less time in future.

(v) Variety: The type of nature of data is having great variety.

Veracity: (i) The data captured is not in certain format.

(ii) Data captured is very greater.

(iii) The accuracy of analysis depends on variety of source data.

DATE / /
PAGE
1 Programmable : It is possible given big data to explore all types of by programming logic.

(1) Programming can be used to perform any kind of exploration because of the scale of data.

(2) Data driven : The data driven approach is possible for scientists as data collected is huge amounts.

(3) Multi attributes : It is possible to deal with many gigabytes of data that consists of thousands of attributes, as all data iterations are happening on a larger scale.

(4) Iterative : The more computing power can iterate on your models until as get them as per your own requirements.

1. Hadoop Architecture

(1) Running hadoop means running a set of resident programs. These resident programs are also known as daemons.

(2) These daemons may be running on same or different servers in the network.

(3) All these daemons have specific functionalities assigned to them following are the daemons.

a NameNode: a NameNode is also known as the master of HDFS.

- (i) NameNode has file trackers which keep track of files distributed to datanodes.
- (ii) NameNode directs Datanodes regarding how to do tasks.
- (iii) NameNode is the only single point of failure component.

b Datanode (i) Datanode is also known as the slave of HDFS.

(ii) Datanode takes client block address from NameNode.

(iii) Using this address, client communicates directly with datanode.

(iv) For replication of data, a datanode may communicate with other datanodes.

(v) Datanode periodically informs local storage updates to NameNode.

(vi) To create, move or delete blocks datanode receives instructions from local disk.

c Secondary NameNode (SNN) (i) State monitoring of cluster HDFS is done by SNN.

(ii) Every cluster has an SNN.

(iii) SNN resides on its own machine also.

(iv) On the same server any other datanode as test tracks daemons running on it.

DATE / /
PAGE

⑥ Job Tracker ③ job tracker determines lineage, process, made assignments for different tasks, task monitoring etc.

⑦ only one jobtracker element per hadoop cluster is allowed.

⑧ job tracker runs on a server as master node of cluster.

⑨ task tracker ① Individual tasks assigned by job tracker are executed by task tracker.

② There is single task tracker per slave node.

③ Task trackers may handle multiple tasks parallel by using multiple JVMs.

④ Task tracker constantly communicates with job tracker. Within specified amount of time if task tracker fails to respond to job tracker then it is assumed that task tracker has crashed & responsibility of corresponding tasks are done to other nodes in cluster.

a HDFS (i) Hadoop distributed file system is the primary component of Hadoop ecosystem and is responsible for storing large data sets of structured and unstructured data across various nodes and thereby maintaining the metadata in form of log files.

(i) HDFS consists of two or more components i.e. Name Node and Data Node. Name Node is the primary node which contains metadata, storing comparatively fewer resources than the data nodes that store the actual data. The data nodes are commodity hardware in the distributed environment.

(ii) HDFS maintains all the coordination between the clusters and hardware, thus working at the heart of the system.

b yarn: (i) Yet Another Resource Negotiation is the one who helps to manage the resources across the clusters. It performs scheduling and resource allocation for Hadoop.

(i) It consists of two major components:

- Resource manager has the knowledge of allocating resource for the application in a system.

- Node manager works on the allocation of resources as CPU, memory bandwidth to the machine and later acknowledge the

resource and manager.

- Application manages works as an interface between resource manager and node manager and performs negotiation as per requirement of the two.

○ Map Reduce: (i) By making the use of distributed and parallel algorithms, Map Reduce makes it possible to carry out processing logic and help to write applications which transform big data sets into a manageable one.

(ii) Map() performs sorting and filtering of data and thereby grouping them in form of group.

Map generates a key-value pair. The result which is later on processed by Reduce() method.

- Reduce() does the summarization by aggregating the mapped data. Reduce() takes by aggregating the mapped data. Reduce takes the output generated by Map() as input combines those tuples into smaller set of tuples.

○ PIG (i) PIG was developed by Yahoo which works on pigatin language which is query based language like SQL.

(ii) It enables the creation of scripts for queries and reports which are compiled into Map Reduce jobs.

HIVE (i) HIVE declarative query language which is from a data warehouse used for ad-hoc queries and simplified report generation.

DATE / /
PAGE

(ii) Hadoop fetches data from plans and files on HBase.

(3) Mahout: It is library for ML and data mining.

(4) HBase: It is NoSQL database which supports all kinds of data and has capability of handling anything of Hadoop Database.

(5) At times, when we need to search or retrieve the occurrence of something small in huge database, request must be processed within a short span of time. At such times, HBase comes handy as it gives an tolerant way of storing limited data.

(6) Zookeepers: It is a library of modules for implementing coordination and synchronization services in Hadoop cluster.

(7) Oozie: Work flows can be described and automated using oozie by considering its dependences between individual jobs.

(8) Sqoop: It is used for loading large data volumes from relational data bases into HDFS and vice versa.

(9) Flume: It is suited for importing data streams, such as web logs or other log data into HDFS.

① Avro: CD It serves for serializing structured data.

(i) Structured data is converted into bit strings & efficiently deposited in HDFS in compact format.
(ii) The serialized data contains information of original data schema.

① By means of NOSQL databases. HBase, Cassandra & Accumulo, large tables can be stored and accessed efficiently.

① Chukwa: It maintains large Hadoop environments.
① Logging data is collected, processed and visualized.

① Ambari: It maintains large Hadoop environments.
① Logging data is collected, processed and visualized.

(i) Cluster installations and up can be conducted fully automated.
(ii) This increases availability and accelerates recovery.

① Spark: using spark, we can load data from HDFS or HBase into memory of cluster nodes for faster processing.

Assignment-2

Mining social network graphs

Clustering of social graph using Girvan-Newman algorithm

In order to find out between edges, we need to calculate shortest paths from going through each of the edges.

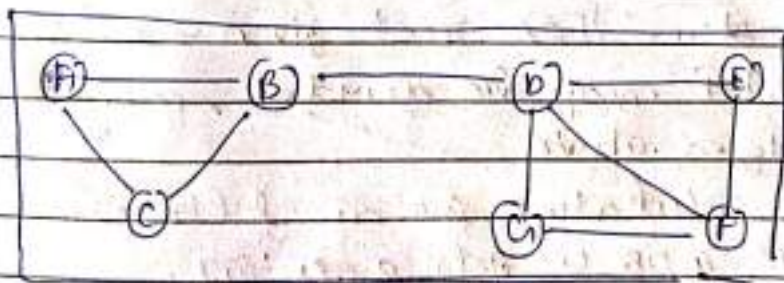
1) Girvan-Newman algorithm visits each node once and computes the no. of shortest paths from x to each other of the nodes that go through each of the edges.

2) The algo begins by performing BFS of the graph, starting at node x .

3) The edges that go between node at the same level can never be a part of the shortest path from x .

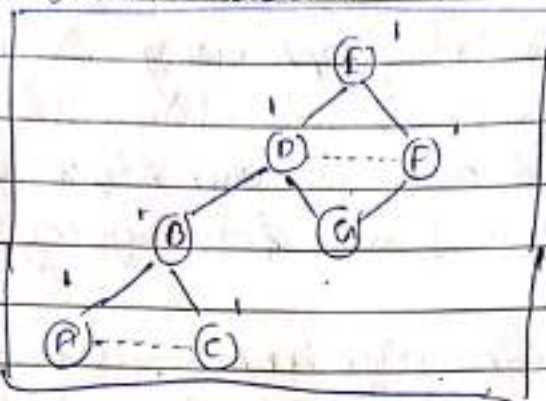
4) edges that will be a part of at least one shortest path from root x .

5) considers the graph as follows.



Sample graph

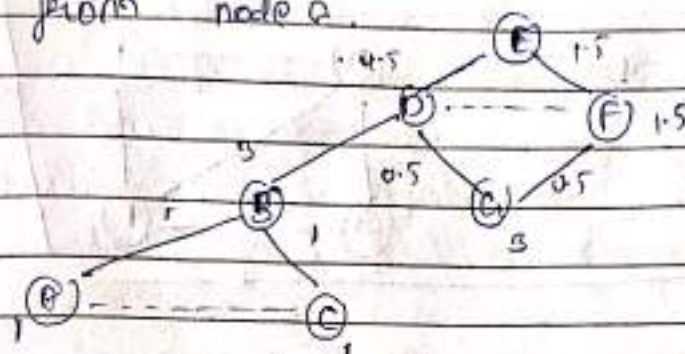
- ④ Following fig shows the BFS representation starting at node c. Solid edges are DPG edges and dashed edges connect nodes at same level.



- ① The second stage of the algo is to label each node by the no of shortest path that reach it from root.
- ② start by labelling the root 1, then from root label each node y by sum of labels of its parents. The labeling of node is shown in above figure.
- ③ The final step is to calculate for each edge e the sum over nodes y of the fraction of shortest paths from the root x to y that go through e .
- ④ each node other than root given a credit of 1. This credit may be divided among nodes and edges above.
- ⑤ The rule for calculation are as follows.
- ① each leaf node gets a credit.
 - ② each non leaf node gets a credit equal of 1 plus the sum of credits of DAG edges from that node up to the level.
 - ③ A DAG edge entering node z from the level above is given a share of credit $1/z$.

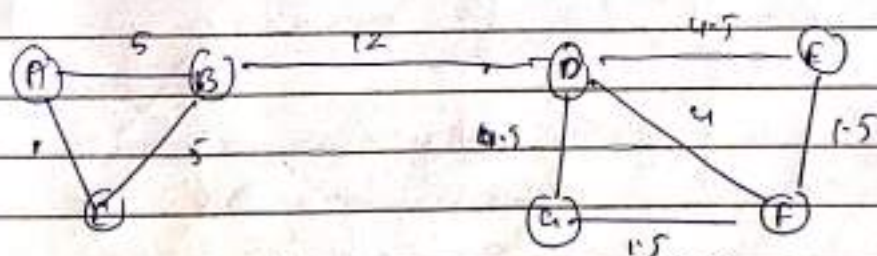
proportional to the fraction of shortest paths from the root to z that go through x .

- (13) after performing the credit calculation with each node as the roots we sum the credits.
- (14) As each shortest path will have been discovered twice we must divide the credit for each edge by 2.
- (15) following graph shows the calculation for DAC starting from node a.



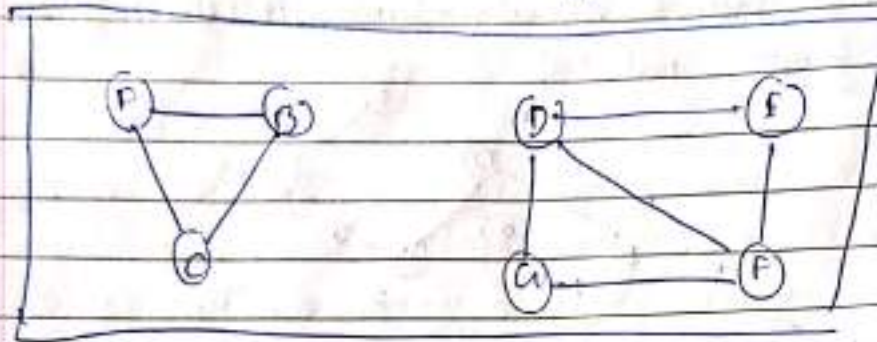
Calculation in DAC.

- (16) To complete the betweenness calculation, we have to repeat the calculation for every node as the root and sum the contribution.
- (17) After calculation, final betweenness are shown in the graph.



graph with betweenness values

- (16) We can cluster by taking the nodes to in series
- (17) between them and add them to graph at a time,
- (18) we can remove edges with highest values to cluster the graph.
- (19) In the eg graph, we remove edge BD to get two communities as follows.



Communities by clustering

Q.2 Direct discovery of communities in social graph using clique percolation method (CPM)

- Ans) The assumption based on which clique percolation method works is that a community comprises of overlapping sets of fully connected subgraphs.
- 1) This algo detects communities by searching for adjacent cliques.
 - 2) It begins by exploring all k -cliques in networks when all k -cliques have been found a new graph
 - 3) commonly referred as clique graph is constructed as adjacent if two share $(k-1)$ members.
 - 4) Each connected component in clique-graph represents a community.

Algorithm:

I/P : Two networks, G and clique size k
 O/P : Community structure C

Step 1: All k -cliques present in network G are identified.

Step 2: A new network referred as clique-graph G_c is formed where each node represents an identified clique and two nodes in network G_c are connected by edge, if they share $k-1$ members.

Step 3: Connected components in G_c are identified.

Step 4: Each connected component in G_c represents a community. Set of communities from the identified community structure for network G .

① A k -clique at $k=3$ is equivalent to a triangle.
Two k -cliques are considered adjacent if they share $k-1$ nodes.

② A community is defined as maximal union of k -cliques that can be reached from each other through a series of adjacent k -cliques.

③ Such communities can be best interpreted with help of k -clique template.

④ Such a template can be placed into any k -clique in the graph and rolled to an adjacent k -clique by relocating one of its nodes keeping its others $k-1$ fixed.

⑤ Since even small networks can contain a vast no. of k -cliques, the implementation of this approach is based on locating all maximal cliques rather than individual k -cliques.

⑥ This inevitably requires finding the graph's maximum clique which is an NP-hard problem.

⑦ The worst case runtime complexity is exponential in the number of nodes.