Experiment-1 30-07-2024

Aim: Practice on basic Linux commands.

System Info:

1. Code:

Date: Displays the Date

[cloudera@localhost ~]$ date

Output:

Mon Jul 29 22:56:01 PDT 2024.

Cal: Displays the calendar

[cloudera@localhost ~]$cal

Output:

July 2024

Su	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

Uptime: Show current uptime

[cloudera@localhost ~]$ uptime

Output:

22:59:06 up 1:08, 2 users, load average: 0.00,0.00,0.00

W: Display who is on line

[cloudera@localhost ~]$ w

Output:

22:59:06 up 1:08, 2 users, load Average: 0.00,0.00,0.00

User	TTY	From	Login@	IDLE	JCPU	PCPU
cloudera	tty1	:0	21:50	1:09m	10.22s	0.00s
cloudera	pts/0	:0.0	22:55	0.00s	0.01s	0.00s

Whoami; who you are logged in as

[cloudera@localhost ~]$ whoami

Output:

cloudera

Ps: Display currently working

[cloudera@localhost ~] $ps

PID	TTY	TIME	CMD
4668	pts/0	00:00:00	bash
4981	pts/0	00:00:00	ps

ls: Directory listing

[cloudera@localhost ~] $ls

Output:

datasets Documents eclips Music Public Videos

Desktop Downloads lib Pictures Templates workspace

Cd dir: change directory to dir

[cloudera@localhost ~] $cd dir

Output:

bash: cd: dir: No such file or directory.

mkdir: creating a directory dir

[cloudera@localhost ~] $ mkdir aids

pwd: show current working directory

[cloudera@localhost ~] $ pwd

Output:

/home/cloudera laids.

rm: remove or delete the file

[cloudera@localhost ~] $rm pic1

Output:

rm: remove regular empty file 'pic1' ?y

Cp: copying the files

[cloudera@localhost ~] $ cp file1 file2

Output:

Cp: DO you want to overwrite 'file2' ? Y

Experiment-2

6/08/2024

Aim: Implement the following file management tasks in Hadoop:

a. Adding files and Directories
b. Retrieving files
c. Deleting files
d. Copying files from local filesystem to HDFS and viceversa
e. Moving files.

Program:

```
[cloudera@localhost ~]$ mkdir 682024
[cloudera@localhost ~]$ cd 682024
[cloudera@localhost 682024]$ gedit f1
```

f1

This is the file1 that I created.

```
[cloudera@localhost 682024]$ pwd
/home1/cloudera/682024
[cloudera@localhost 682024]$ ls
```

Output:

f1

a. Adding files and Directories: creating a directory,

```
hadoop fs -mkdir f1
```

Before running hadoop program need to put data to HDFS

```
hadoop fs -mkdir/user/cloudera/f1/f2
hadoop fs- ls
```

Output:

drwxr-xr-x- cloudera cloudera cloudera 02020-08-

b. Retrieving files: Accessing the files from HDFS to local file system.

cat > f1

Helloworld

cat > l2

Hello welcome to My file

hadoop fs -put l2 f2

Output:

Hellowelcome to My file

hadoop fs -put l2 /user/douldera/file

hadoop fs -cat /user/cloudera/file/l2

Output:

Hello welcome to My file.

hadoop fs -get /user/cloudera/f1 l3

cat l3

Output:

Helloworld.

c, Deleting files: rm- Hadoop command for deleting files.

hadoop fs -rm f1

Output: Moved: 'hdfs://localhost: Local domain: 8020/user/cloudera/f1 to trash at: hdfs://localhost. localdomain: 8020 /user/cloudera/Trash/current.

d, Copying files from local file system to HDFS and Vice versa:

hadoop fs -cp /user/cloudera/file/l2 user/cloudera fs

-> from local to Hadoop

put command & get command -> Hadoop to local.

hadoop fs -put l2 /user/cloudera/file

hadoop fs -get /user/cloudera/f1 l3

e. Moving files: Moving files accross file system is not permitted.

hadoop fs -mv /user/cloudera/f1 /user/cloudera/f6

hadoop fs -cat f6

Output:

Hello Welcome to My file.

6/8

Experiment-3

13/08/20

Aim: Write driver code, mapper code, reducer code to count number of words in a given file. (Hint: WordCount Map-Reduce Program).

Word Count Mapper program:

```
import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
public class WordCountMapper extends
        Mapper<LongWritable, Text, Text, LongWritable>{
    private final static LongWritable one=new
                                    LongWritable(1);
    @Override
    protected void map(LongWritable key, Text
                        value, Context context)
        throws IOException, InterruptedException{
        String line=value.toString();
        String[] words=line.split(" ");
        for(int i=0; i<words.length; i++) {
            context.write(new Text(words[i]), one);
        }
    }
}
```

Word Count Reducer Program:

```
import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.io.Text;
public class WordCountReducer extends
    Reducer<Text, LongWritable, Text, LongWritable>
{
```

```
@Override
protected void reduce(Text key, Iterable<
        LongWritable> value; Context context)
    throws IOException, InterruptedException
    {
        long sum=0;
        while(value.iterator().hasNext())
        {
            sum+=value.iterator().next().get();
        }
        context.write(key, new LongWritable(sum)
    }
}
```

Word Count Driver Exception:

```
import org.apache.hadoop.conf.configuration,
import org.apache.hadoop.fs.path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileIn
                                        putFormat;
import org.apache.hadoop.mapreduce.lib.output.
                                    FileInputFormat;
import org.apache.hadoop.util.ToolRunner;
import org.apache.hadoop.util.Tool;
public class WordCountJob implements Tool {
    private Configuration conf;
    @Override
    public configuration getConf()
    {
      return conf;
    }
    @Override
    public void setConf(Configuration conf)
    {
```

```
        this.conf=conf;
    }
    @Override
    public int run(String [] args) throws Exceptior
    {
        Job wordcountjob = new Job(getConf());
        wordcountjob.setJobName("mat word Count");
        wordcountjob.setJarByClass(this.getclass());
        wordcountjob.setMapperClass(WordCountMapper
                                    class);
        wordcountjob.setReducerClass(WordCountReducer
                                     class);
        wordcountjob.setMapOutputKeyclass(Text.class)
        wordcountjob.setMapOutputValueClass(
                                    LongWritable.class);
        wordcountjob.setOutputKeyClass(Text.class);
        wordcountjob.setOutputValueClass(wordcount-
                                    job.class);
        FileInputFormat.setInputPaths(wordcountjob,
                    new Path(args[0]));
        FileOutputFormat.setOutputPath(wordcountjob,
                    new Path(args[1]));
        wordcountjob.setNumReduceTasks(2);
        return wordcountjob.waitForCompletion(true
        ==true? 0:1;
    }
    Public static void main(String [] args) throws
                                    Exception
    {
        ToolRunner.run(new Configuration(), new
                    WordCount Job(), args);
    }
}
```

Output:

```
[cloudera@localhost ~]$ hadoop fs -pwd
-pwd: Unknown command
[cloudera@localhost ~]$ hadoop fs -ls
Found 2 items
-rw-r--r--   3 cloudera cloudera         16 2024-08-05 21:18 ff1
-rw-r--r--   3 cloudera cloudera         12 2024-08-05 21:29 ff2
[cloudera@localhost ~]$ ls
datasets    eclipse     hm      Public      Videos
Desktop     ff1         lib     task1       wc.jar
Documents   ff2         Music   task2       workspace
Downloads   filename    Pictures Task2
[cloudera@localhost ~]$ hadoop jar wc.jar WordCount
                        -Job ff1 Wcout.
24/08/12 22:49:29 INFO input.FileInputFormat: Total
                        input paths to process:1
24/08/12 22:49:46 INFO mapred.JobClient: Counters:32
[cloudera@localhost ~]$ hadoop fs -ls
Found 4 items
drwx------   - cloudera cloudera          0 2024-08-12 22:49 .Staging
drwxr-xr-x   - cloudera cloudera          0 2024-08-12 22:49 Wcout
-rw-r--r--   3 cloudera cloudera         16 2024-08-12 21:18 ff1
-rw-r--r--   3 cloudera cloudera         12 2024-08-12 21:29 ff2
[cloudera@localhost ~]$ hadoop fs -ls Wcout
Found 4 items
-rw-r--r--   3 cloudera cloudera          0
drwxr-xr-x   - cloudera cloudera          0
-rw-r--r--   3 cloudera cloudera         18
-rw-r--r--   3 cloudera cloudera          6
[cloudera@localhost ~]$ hadoop fs -cat Wcout/
                        part-r-00000
```

ffi 1
$is 1
thi 1

[cloudera@localhost ~]$ hadoop fs -cat Wcout1/part-r-00001

hm/ 1

Experiment-4

Aim: Write a MapReduce program that mines weather data. Weather sensors collecting data every hour at many locations across the globe gather a large volume of log data, which is a good candidate for analysis with MapReduce, since it is semi structured and record-oriented.

Max Temperature Driver program.

```
import org.apache.hadoop.fs.path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileIn
                                    -put Format;
import org.apache.hadoop.mapreduce.lib.output.File
                                    -OutputFormat;
public class MaxTemperature{
    public static void main(String[] args) throws
                                    Exception{
        if(args.length!=2){
        System.err.println("Usage: MaxTemperature
                                <input path><output path
        System.exit(-1);
        }
        Job job=new Job();
        job.setJarByClass(MaxTemperature.class);
        job.setJobName("Max temperature");
        FileInputFormat.addInputPath(job,new
                                    Path(args[0]));
        FileOutputFormat.setOutputPath(job,new
                                    Path(args[1]));
```

```
        job.setMapperClass(MaxTemperatureMapper.
                                        class);
        job.setReducerClass(MaxTemperatureReducer.
                                        class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        System.exit(job.waitForCompletion(true)?0:1);
        }
}
```

Max Temperature Mapper Program:

```
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 MaxTemperatureMapper extends
        Mapper<LongWritable, Text, Text, IntWritable>
{
  private static final int MISSING=9999;
  @Override
  public void map(LongWritable key, Text value,
                                  Context context)
        throws IOException, InterruptedException{
    String line=value.toString();
    String year=line.substring(15,19);
    int airTemperature;
    if(line.charAt(87)=='+'){
      airTemperature=Integer.parseInt(line.substring
                                  (88,92));
    }
```

```
else{
    airTemperature=Integer.parseInt(line.substring
                                    (87,92));
}
String quality=line.substring(92,93);
if(airTemperature!=MISSING && quality.matches
                                ("[01459]")){
context.write(new Text(year),new IntWritable
                                (airTemperature));
}
}
}
```

Max Temperature Reducer program:

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
public class MaxTemperatureReducer extends
        Reducer<Text, IntWritable, Text, IntWritable>
{
@Override
public void reduce(Text key, Iterable<IntWritable>
                values, Context context)
    throws IOException, InterruptedException{
int maxValue=Integer.MIN_VALUE;
for(IntWritable value:values){
maxValue=Math.max(maxValue,value.get()
```

```
    }
    context.write(Key, new IntWritable(maxValue));
  }
}
```

Output:

```
1901    317
1902    244
1903    289
1904    256
1905    283
1906    294
1907    283
```

10/10/2018

Name:

Experiment:5

Aim: Implement Matrix Multiplication with Hadoop Map Reduce.

Description:

MapReduce is a technique in which a huge program is subdivided into small tasks and run parallelly to make computation faster, save time, and mostly used in distributed systems. It has 2 important parts:

- Mapper
- Reducer

Driver Program:

```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.
                                FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.
                                TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.
                                FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.
                                FileTextOutputFormat;
public class MatrixMultiplication{
public static void main(String[] args) throws
                                Exception{
Configuration conf = new Configuration();
conf.set("m", "2");
```

```
conf.set("n", "5");
Conf.Set("p", "3");
Job job: new Job(conf, "Matrix Multiplication");
job.setJarByClass(MatrixMultiplication.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(Text.class);
job.SetMapperClass(MatrixMapper.class);
job.set ReducerClass(Matrix Reducer.class);
job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatclass(TextOutputFormat.class);
FileInputFormat.addInputPath(job, new Path
                                    (args[0]));
FileOutputFormat.setOutputPath(job, new Path
                                    (args[1]));
job.waitForCompletion(true);
}
}
```

Mapper Program:

```
import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
public class MatrixMapper extends Mapper<
                    LongWritable, Text, Text, Text>{
public void map(LongWritable key, Text value,
Context context) throws IOException, InterruptedException{
Configuration conf= context.getConfiguration();
int m= Integer.parseInt(conf.get("m"));
```

```
int p=Integer.parseInt(conf.get("p"));
String line=value.toString();
String[] indicesAndValue=line.split(",");
Text outputKey=new Text();
Text outputValue=new Text();
if(indicesAndValue[0].equals("A")){
    for(int k=0;k<p;k++){
        outputKey.set(indicesAndValue[1]+","+k);
        outputKey.set("A,"+indicesAndValue[2]+
                ","+indicesAndValue[3]);
        context.write(outputKey,outputValue);
}
}
else{
for(int i=0;i<m;i++){
    outputKey.set(i+","+indicesAndValue[2]);
    outputValue.set("B,"+indicesAndValue[1]+","+
            indicesAndValue[3]);
    context.write(outputKey,outputValue);
}
}
}
}
```

Reducer Program:

```
import java.io.IOException;
import java.util.HashMap;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
public class MatrixReducer extends Reducer<Text,
```

```
Text, Text, Text>{
public void reduce (Text key, Iterable<Text> values,
Context context) throws IOException, InterruptedExcep
                                                -tion{
String[] value;
HashMap<Integer, Float> hashA=new HashMap
                                <Integer,Float>();
HashMap<Integer, Float> hashB=new HashMap
                                <Integer,Float>();
for (Text val: values){
value =val.toString().split(",");
if(value[0].equals("A")){
    hashA.put(Integer.parseInt(value[1]),
     Float.parseFloat(value[2]));
}
else{
    hashB.put(Integer.parseInt(value[1]),
    Float.parseFloat(value[2]));
}
}
int n= Integer.parseInt(context.getConfiguration().
                                get("n"));
float result=0.0f;
float a_ij;
float b_jk;
for(int j=0;j<n;j++){
    a_ij= hashA.containsKey(j)? hashA.get(j) :0.0f;
    b_jk= hashB.containsKey(j)?hashB.get(j):0.0f;
```

```
        result += a_ij * b_jk;
    }
    if (result != 0.0f) {
        context.write(null, new Text(key.toString() +
            "," + Float.toString(result)));
    }
  }
}
```

Output:

```
0,0,147.0
0,1,111.0
0,2,104.0
1,0,93.0
1,1,72.0
1,2,62.0
```

16/9/24

Input:

mm.txt:

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

Aim:

Install and run Pig then write Pig Latin scripts to load, store and filter data.

Description:

Pig Running modes

We can manually override the default mode is -x or -exectype options.

```
$pig -x local
$pig -x mapreduce.
```

Building blocks:

- Field: piece of data. Ex: "abc".
- Tuple: Ordered set of fields represented with "(and")". Ex: (103, abc, 5)
- Bag: - collection of tuples represented with "{"and"}". Ex: {(103, abc, 5), (def, 12, 13.5)}

Load Command:

LOAD 'data' [USING function] [AS schema];

- data - name of the directory or file must be in single quotes.
- USING - Specifies the load function to use.
  - By default uses pig storage which parses each line into Fields using a delimiter.
  - Default delimiter is tab("\t").
- AS - assign a schema to incoming data.
  - assigns names to Fields.

LOADING DATA: Create file in local File system.

Command: cat >a.txt

25
35
45
55
65
24
12
19
27
35
24

* Copy file from local File system to hdfs

Command: hadoop fs -put a.txt.

Command: pig.

Load data to pig storage

grunt> data = load "a.txt" using PigStorage()
as (age:int);

grunt> dump data;

(25)
(35)
(45)
(55)
(65)
(24)
(12)
(19)
(27)
(35)
(24)

Pig Latin- Diagnostic Tools:

- Display the structure of the Bag.

grunt> DESCRIBE <bag-name>;
ex: DESCRIBE data;

- Display Execution plan
-Produces various reports

- Logical plan.
- MapReduce plan.

grunt > EXPLAIN < bag-Names >;

ex: EXPLAIN data;

- Illustrate how Pig engine transforms the data.

grunt> ILLUSTRATE < bag-names>;

ex: ILLUSTRATE data;

Filter data:

grunt> Filter1 = filter data by age>30;

grunt > dump filter1;

(35)
(45)
(55)
(65)
(35)

grunt> Filter2 = filter data by age<20;

grunt > dump filter 2;

(12)
(19)

15/10

Experiment-7:

Aim:

Write pig Latin scripts to perform data processing operations.

a) Grouping and joining data.

b) Sorting data

c) Combining and splitting :data.

Grouping data:

```
grunt > group1 = group data by age;
grunt > describe group1;
```

o/p: group1 : {group:int, data: {(age:int)}}

```
grunt > dump group1;
(4, {(4)})
(12, {(12)})
(18, {(18)})
(24, {(24)})
(25, {(25)})
(27, {(27)})
(35, {(35), (35)})
(55, {(55)})
(65, {(65)})
```

The data bag is grouped by 'age' therefore Group element contain Unique values.

To see how pig transforms data

```
grunt > ILLUSTRATE group1;
```

Join:

The JOIN Operator is used to combine records from two or more relations joins can be of three types:

1. self-join
2. Inner-join
3. Outer-join: left-join, right-join

Self-join:

Self-join is used to join a table with itself.

Inner-join:

Default join is inner-join. Rows are join a table with the keys match. Rows that do not have matches are not includes in the result.

Outer-join:

Records which will not join with the "Other" record-set are still included in the result.

a. Create a file in local system

cat > a1.txt

1, 2, 3
4, 2, 1
8, 3, 4
4, 3, 3
7, 2, 5
8, 4, 3

Command: cat > b.txt

2, 4
8, 9
1, 3
2, 7
2, 9
4, 6
4, 9

put line file in hadoop

>hadoop fs -put a1.txt
> hadoop fs -put b.txt.

Self-join:

We join one table to itself rather than joining two tables.

grunt> ONE = load 'a1.txt' using PigStorage(',')
as (a1:int, a2:int, a3:int);

grunt > Two = load 'a1.txt' using PigStorage
as (a1:int, a2:int, a3:int);

SELFJ = JOIN ONE by a1, Two By a1;

grunt > describe SELFJ;

SELFJ: {ONE::a1:int, ONE::a2:int, ONE::a3:int
Two::a1:int, Two::a2:int, Two::a3:int}

Equi-join:

A inner join returns rows when there is match in both tables.

grunt > A = load 'a1.txt' using PigStorage(
as (a1:int, a2:int, a3:int);

grunt > B = load 'b.txt' using PigStorage(',') as
(b1:int, b2:int, b3:int);

grunt > x = join A by a1, B by b1;

grunt > Dump x;

(1, 2, 3, 1, 3)

(4, 2, 1, 4, 6)

(4, 2, 1, 4, 9)

(4, 3, 3, 4, 6)

(4, 3, 3, 4, 9)

(8, 3, 4, 8, 9)

(8, 4, 3, 8, 9)

Left Outer join:

> A = LOAD 'A.txt' using PigStorage(',') As
(a1:int, a2:int, a3:int);

> B = LOAD 'B.txt' using PigStorage(',') AS
(b1:int, b2:int, int:b3);

> LEFTJ = JOIN A by a 'LEFT, OUTER, B by

> DUMP LEFTJ;

(1,2,3,1,3)
(4,3,3,4,9)
(4,3,3,4,6)
(4,2,1,4,9)
(4,2,1,4,6)
(7,2,5,,)
(8,4,3,8,9)
(8,3,4,8,9)

Right Outer join:

>A = LOAD 'A.txt' using PigStorage(',') As (a1:int, a2:int, a3:int);

>B= LOAD 'B.txt' using PigStorage(',') As (b1:int, b2:int);

>RIGHTJ=JOIN A by as Right OUTER B by b1;

>DUMP RIGHTJ;

(1,2,3,1,3)
(,,,2,4)
(,,,2,7)
(,,,2,9)
(4,2,1,4,6)
(4,2,1,4,9)
(4,3,3,4,9)
(4,3,3,4,9)
(8,3,4,8,9)
(8,4,3,8,9)

Full join:

>A = LOAD 'A.txt' using PigStorage(',') As (a1:txt, a2:int, a3:int);

>B= LOAD 'B.txt' using Pigstorage(',') As (b1:int, b2:int);

>FULLJ=JOIN A by a1 FULL, B by b1;

> DUMP FULLJ;

(1,2,3,1,3)

(,,,2,4)

(,,,2,7)

(,,,2,9)

(4,2,1,4,6)

(4,2,1,4,9)

(4,3,3,4,6)

(4,3,3,4,9)

(7,2,5,,,)

Sorting:

Sort by Ascending order.

grunt > sort1 = order data by age ASC;

grunt > dump sort1;

(12)

(19)

(24)

(24)

(25)

(27)

(35)

(35)

(45)

(55)

(65)

Sort by Descending order:

grunt > sort 2 = Order data by age ESC;

grunt > dump sort 2;

(65)

(55)

(45)

(35)
(35)
(27)
(25)
(24)
(24)
(19)
(12)

Union & Split:

Union combines multiple relations together whereas SPLIT partitions a relation in to multiple ones.

```
grunt> cat>a.txt
1, 2, 3
4,2,1
8,3,4
grunt> cat>b.txt
4, 3, 3
7,2,5
8,4,3
```

```
grunt> a= load 'a.txt' using PigStorage(',')
        as (a1:int, a2:int, a3:int);
grunt> b= load 'b.txt' using PigStorage(',')
        as (b1:int, b2:int, b3:int);
grunt> dump a;
(1,2,3)
(4,2,1)
(8,3,4)
grunt> dump b;
(4,3,3)
(7,2,5)
```

(8,4,3)

grunt> c= UNION a,b;

(1,2,3)

(4,2,1)

(8,3,4)

(4,3,3)

(7,2,5)

(8,4,3)

grunt> SPLIT c into sp1 if $0==4, sp2 if $0==8;

Split operation on "c" sends a tuple to sp1 if its first Field ($0) is 4, and to sp2 if it is 8.

grunt> dump sp1;

(4,3,3)

(4,2,1)

grunt> dump sp2;

(8,4,3)

(8,3,4)

FOREACH with Functions:

FOREACH:

FOREACH <bag> GENERATE <data>

iterates over each element in the bag and produce a wait-result.

grunt >records = LOAD 'std.txt' USING PigStorage(',') AS (roll:int, name:chdarray);

grunt >dump records;

(501, aaa)

(502, bbb)

(503, yyy)

(204, rrr)
(510, bbb)

grunt> stdname= foreach records generate name;

grunt> dump stdname;

(aaa)
(hhh)
(yyy)
(rrr)
(bbb)

FOREACH with Functions:

- Pig comes with many functions including COUNT, FLATTEN, CONCAT, etc,....
- can implements a custom function.

COUNT:

grunt> chars= LOAD 'char.txt' As CC:charArray);

grunt> chargrp= GROUP chars by c;

grunt > dump chargrp;

(a, {(a), (a), (a),})

(c, {(c), (c)})

(i, {(i), (i), (i)})

(k, {(k), (k), (k), (k)})

(l, {(l), (l)})

grunt> describe chargrp;

chargrp: {group: chararray, chars: {(C: chararray)}};

grunt> counts= FOREACH chargrp GENERATE group, COUNT(chars);

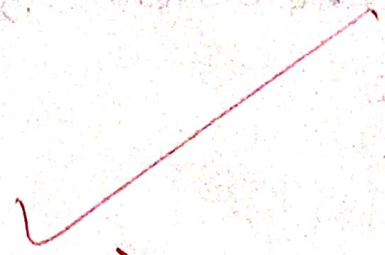
(a, 3)
(b, 2)

(C, 2)

(i, 3)

(K, 4)

(l, 2)



29/10

Experiment-9

Aim:

Install Hive and use hive to create databases and tables:

a) Create and drop databases

b) Create alter and drop tables.

c) Insert, update and delete records

Description:

Install hive: Download and install hadoop and hive on your virtual Box vm. Configure hadoop environment variables and start hadoop services

Start Hive shell: launch the Hive shell using the "hive" command.

Create Database and Table:- Execute SQL-like commands to create a database and table:-

```
>create database my_database;
> create table my_table (column1 int,
  column2 string)
  low format delimited
  fields terminated by ',';
```

Program:

Login in to hive:

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

i, Database creation:

```
hive> create database employees;
OK
```

ii, drop database:

```
hive> drop database employees;
OK
done
```

b, Create, alter and drop tables:

i) Create table:

```
hive> create table employee2(cid int, name
                    string, salary int, destination
Comment 'Employee details'
Row Format Delimited
Fields terminated by '\t'
lines terminated by '\n'
Stored as textfile;
```

ii) load the data:

a) Create text file:

Sample3.txt

```
1201    Gopal       45000    Technical-manager
1202    Manisha     45000    Proof-reader
1203    Masthan     40000    Technical-writer
1204    Kiran       40000    Hr-admin
1205    Kranthi     30000    Op-admin
```

b) load the data from local path:

```
hive> load data local inpath '/home/cloudera
        Sample3.txt' overwrite into
            table employee2;
```

Output:

```
Loading data to table default.employee2
OK
Time Taken : 0.192 seconds
hive> select * from employee2;
1201    Gopal       45000    Technical-manager
1202    Manisha     45000    Proof_reader
1203    Masthan     40000    Technical-reader
1204    Kiran       40000    Hr-admin
```

1205 kranthi 30000 Op-Admin

2) alter the table:

hive> alter table employee2 rename to emp2;

Output:

altered successfully

3) drop the table:

hive> drop table employee2;

Output:

OK

c) Insert, update and delete records:

i) hive> insert overwrite table employee2
select
case
when eid=1201 then 'Job'
else name
end as name, eid, salary, destination,
from employee2;

Output:

inserted successfully.

ii) Update:

hive> update employee2 set name='harika'
where eid=1202

Output:

updated successfully.

(or)

hive> create table temp6 (eid int, name string);
hive> insert temp6 values (1202, 'harika');
hive> insert overwrite table employee2
select * from temp6;

Output:

1201	Gopal	45000	technical_manager
1202	harika	45000	proof-reader
1203	Masthan	40000	Technical-writer
1204	Kiran	40000	Hr-admin
1205	Kranthi	30000	Op-admin.

c) delete the table:

hive > delete from employee2 where eid=120
deleted successfully.

Experiment -10

Aim: Perform data processing operations using Hive.

a. Sort and aggregation of data.

b. joins.

Sort:

Sort is used to Arrange data in order either Ascending or Descending order.

Syntax:

sort by col-name [ASC|DESC]

Ex: Ascending

from student-details select sno, sname sort by sno Asc;

>OK

5401 Suzuki

5402 doraemon

5403 tom

5404 jerry.

Ex: Descending:

from student_details select sno, sname sort Sname DESC;

>OK

5403 tom

5401 Suzuki

5404 jerry

5402 doraemon

Aggregate function:

1. Count(*) - Big data.

-It returns total no.of retrieved rows.

Ex

>Select count(*) from student_details;

OK
4

2. Sum - Double:

sum(col)

->It returns the sum of the element the group.

>Select sum(sno) from student_details;

OK

21610

3. avg - Double:

avg(col)

->It returns the average of the elements in the group

Ex: Select avg(sno) from student_details;

>OK

5405.0

4. Min - Double:

min(col)

->It returns the min value of column in the group.

Ex: Select min(sno) from student_details

>OK

5401

5. max: It returns the maximum value of Column in the group:

max(col)

Ex: Select max(sno) from student_details

>OK

5405

Joins:

→ Join is a clause that is used for combining specific fields from two tables by using values common to each one.

→ Used to combine two or more tables in the database.

→ Similar to SQL joins

There are different types of joins

→ JOIN

→ LEFT OUTER JOIN

→ RIGHT OUTER JOIN

→ FULL OUTER JOIN

```
>Select * from fruits;
Ok
apple 5
banana 2
mango 3
>Select * from icecream;
Ok
butterscotch 3
chocolate 5
strawberry 6
```

Inner Join:

The simplest kind of join is the inner where each match in the input tables result in a row in the output table it is being joined to cice-cream)

Ex: Select fruits.*, icecream.*
>from fruits join icecream on (fruits.no = icecream.no);

OK
mango 3 butterscotch 3
apple 5 chocolate 5

->The table in the from clause (fruits) is joined with the table in the join (ice-cream) using the predicate in the ON clause.

Left outer join:

Outer joins allow you to non-matches in the tables being joined.
if we change
the join type to left outer join, then the query will return a row for every row in being joined to (ice-cream)

hive>select fruits.*, icecream.*
>from fruits left outer join icecream on (fruits.no=icecream.no);

OK
banana 3 NULL NULL
mango 3 Butterscotch 3
apple 5 chocolate 5
NULL NULL

Right Outer join:

Opposite of left outer join, Rows from second table are included no matter, columns from t unmatched are set to

```
hive> Select fruits.*, icecream.*
    > from fruits right outer join icecream on
        (fruit.no=icecream.no);
    > ok
mango 3     butterscoth 3
apple 5     Chocolate 5
NULL NULL   Strawberry 6
```

Full Outer join:

→ Rows from both sides are includes for unmatched rows the columns from outer table are set to null.

→ In full outer join the output has a row for each row from both tables in the join.

```
hive> select fruits.*, icecream.*
    > from fruits full outer join icecream on
    > (fruits.no=icecream.no);
OK
banana  2   NULL        NULL
mango   3   Butterscoth  3
apple   5   Chocolate    5
NULL    NULL Strawberry  6
```

Experiment-11

Aim: Perform data processing operations using Hive.

a. Views

b. Indexes.

Views:

- A view is a sort of "virtual table" that is defined by a select statement.
- Views can be used to present data to users in a different way to the way it is actually stored on disk.
- Views may also be used as to restrict users to particular subset of tables that they are authorized to see.

→ first create table and then insert data to it

```
hive> select * from fruits;
OK
apple 5
banana 2
mango 3
```

Create View:

Syn: Create view viewname;

Ex:

```
hive> create view fruits_name as
      select name from fruits;
hive> create view first_id as
      select * from fruits where no=1;
```

• Show Views:

hive>show tables
OK

fruits
fruits-name
fruit-id

To display data in views:

Select * from fruits-name;
OK
apple.
banana
mango

Alter views: To rename view

hive>alter view first-is rename to 1st is
OK

Drop views:

To Drop views

hive>drop view 1st id;
OK.

Indexes:

-> An index is nothing but a pointer on a particular column of a table.

-> Creating on index means creating a point on a particular column of a table.

Example:

Index Creation:

hive> create index fruits-no on table
> fruits (no)
> As 'org.apache.hadoop.hive.ql.index.com
compactIndexHandler

>with Deffered rebuild;

>OK.

Alter index:

hive>alter index fruits_no on fruits r

OK

fruit_no altered

Display indexes:

hive>show indexes on fruits

OK

fruits_no

Dropping index:

Drop index fruits_no on fruits

ok.

>with Deffered rebuild;

>OK.

Alter index:

hive>alter index fruits_no on fruits rebuild

OK

fruit_no altered

Display indexes:

hive>show indexes on fruits

OK

fruits_no

Dropping index:

Drop index fruits_no on fruits

ok.