**List filesystem in root directory**

$hadoop fs –ls /

$hdfs dfs –ls /

Hadoop fs –ls hdfs://quickstart.coudera:8020/

**Create a directory**

$hadoop fs –mkdir /aptraining

**Gives local filesystem detail**(use Hadoop fs if you want to use haddop filesystem)

$ ls

**Create a file:**

$ cat > sample.txt 🡪 hit enter and type content at end cntl+z.

$ cat sample.txt 🡪 to view content of file.

Here sample.txt is created in local file system

**Move file from local fs to Hadoop fs.**

$ hdfs dfs –copyFromLocal sample.txt / aptraining

$ hdfs dfs –put sample.txt /aptraining

$ hdfs dfs –put –f sample.txt /aptraining 🡪 to overwrite.

You can copy using put command also.it is like write file to hdfs. Difference in copyToLocal and put is , copy TO Local is for local file system where put can be used for remote file system.

$ hdsf dfs –ls /aptraining

**-get or coptToLocal to copy file from hdfs to loacal.**

**-mv and –cp to move and copy within hdfs**

Before running hadoop program we have to move data to hdfs first, before analysis.

Open file in editor.

**Nano file.txt**

**Pig:**

* High level data flow language
* Abstraction over MR
* Used to analyze larger sets of data representing them as data flow
* To analyze data, pig script is needed using PigLatin and script is being converted to MR job.
* Not complex programing
* Uses multi-query approach.
* Built-in support for data operation like join,filter,group,order.
* Built in data structure like tuple,bad,map.

**Disadvantages:**

Slow compared to MR

Limited optimization.

**Pig has two components:**

1. Pig Latin: is made up of series of operations or transformations, that are applied to input data to produce output.
2. Execution environment: grunt shell, script and embedded.

**Pig runs in two modes:**

1. Map-reduce mode , is default mode to work on HDFS
2. Local mode. To work on local FS

**$pig 🡪** will run pig in default mode. Type quit to come out.

**$pig –x local** 🡪 is to run pig in local mode.

Both the mode runs in Grunt shell. Grunt shell is native shell provided by pig to execute pig script.grunt shell, parsed, optimize and compile(create MR job here) the script

Load is loading the file from hdfs into pig and always return and store in bag(collection of tuple).Bag is also called as relation.

All pig script is terminated by ‘;’.

**Grunt> policydata = LOAD ‘hdfs://quickstart.cloudera:8020/piginputs/policydata.txt’ USING Pigstorage(‘,’);**

If you run load in $ shell then give only file path like /piginput.

Here policydata is a bag or relation and Pigstorage is bult-in function to sue demlimiter.by default LOAD commands load all fields as bytearray.

While loading you can specify data type.

**Grunt> policydata = LOAD ‘hdfs://quickstart.cloudera:8020/piginputs/policydata.txt’ USING Pigstorage(‘,’) as (policyNo:int,policyHolderN:chararray,lob:chararray,issueDate:chararray,policyMode:chararray,prrmium:double,state:chararray,remark:chararray);**

To see data in relation:

**Grunt>DUMP policydata**

This will create and submit a MR job to retrieve data.parse and optimizer prepare only a logical plan.

To view schema of a relation:

**Grunt> DESCRIBE policydata;**

Filter:

Grunt> filteredPolicy = FILTER policydata BY state == ‘IL’;

Grunt> DUMP filteredPolicy;

Logical operator can also be used. Like BY state == ‘IL’ AND lob == ‘A’;

To see History of commands executed so far.

Grunt> HISTORY;

Group by:

Grunt> groupedPolicy = GROUP policydata BY state;

Grunt> DUMP groupedPolicy;

Grouping on multiple fields use like BY (state,lob);

Grunt> sortedgroups = ORDER groupedPolicy BY group;

Here group is from groupedPolicy(see schema using DESCRIBE)

Select:

Grunt> selectedfields = FOREACH policdata GENERATE policyNo,lob,policyMode,state;

You can also append information;

Grunt> selectedfields = FOREACH policdata GENERATE policyNo,lob,policyMode,state, ‘addeddata’;

You can also store this in HDFS using STORE:

Grunt> STORE = selectedFields INTO ‘/pigoutputs’ USING PigStorage(‘:’);

Check now $ shell for file

$hadoop fs –ls /pigoutputs

$hadoop fs –cat /pigoutputs/part-m-00000

**03/08**

Join two datasets , using common fields.

Policydata = LOAD ‘hdfs://quickstart.cloudera:8020/piginputs/policydata.txt’

>> USING pigStorage(‘,’)

>> AS (policy:int,

>> policyHoldername:chararray,

>> ……. . . . . .);

Coveragesdata = LOAD ‘hdfs://quickstart.cloudera:8020/piginputs/coveragesdata.txt’ >> USING pigStorage(‘,’)

>> AS (policyno:int,coveragetype:chararray,minamount:double,insuredamount:double);

If you don’t specify schema then you can use $0,$1,$2…etc for field 1,field 2 ..and so on.

Grunt> joineddata = JOIN policydata BY policyno,coveragesdata BY policyno;

Grunt>DUMP joineddata;

LIMIT: to get specified no of rows.

Grunt> limiteddata = LIMIT joineddata 10;

Grunt> DUMP limiteddate;

EXPLAIN: gives logical plan of relation.

Grunt> EXPLAIN limiteddata;

ILLUSTRATE : gives sample data on how it will process relations.

Pig script: you can use nano or gedit or vieditor to create file

$nano statewise.pib

//script

--load data

Policydata = LOAD ‘hdfs://…… . . . . .’ USING pifStorage(‘,’) AS

(….. . . . . .);

--group by State

Policydata\_group = GROUP policydata BY state;

--generate transformation

Statewise\_policydata = FORWACH policydata\_group GENERATE group,SUM(policydata.premium) as TotalPremium,

COUNT(policydata) as PolicyCount;

DUMP statewise\_policydata;

STORE statewise\_policydata INTO ‘hdfs://. . . ./statewiseoutput’ USING PigStorage(‘,’);

Cntrl + x to save in nano editor.

Now you can execute this script using grunt shell(using EXEC or RUN) and $ shell(using pig) also.

$pig statewise.pig

$hadoop fs –ls /statewiseoutput

$hadoop fs –cat part-m-00000