NAME:- DUBEY KARAN SANJEEV
CLASS:- B.E - 4
ROLL NO:- 04
BATCH:- A

Experiment No – 3

AIM: To execute PIG commands of Hadoop Ecosystem..

THEORY:

Pig is a high level scripting language that is used with Apache Hadoop. Pig enables data workers to write complex data transformations without knowing Java. Pig's simple SQL-like scripting language is called Pig Latin, and appeals to developers already familiar with scripting languages and SQL. Pig is complete, so one can do all required data manipulations in Apache Hadoop with Pig. Through the User Defined Functions(UDF) facility in Pig, Pig can invoke code in many languages like JRuby, Python and Java. You can also embed Pig scripts in other languages. The result is that one can use Pig as a component to build larger and more complex applications that tackle real business problems. Pig works with data from many sources, including structured and unstructured data, and store the results into the Hadoop Data File System. Pig scripts are translated into a series of MapReduce jobs that are run on the Apache Hadoop cluster.

Commands:

Apache Pig Execution Modes

You can run Apache Pig in two modes, namely, Local Mode and HDFS mode.

Local Mode

In this mode, all the files are installed and run from your local host and local file system. There is no need of Hadoop or HDFS. This mode is generally used for testing purpose.

MapReduce Mode

MapReduce mode is where we load or process the data that exists in the Hadoop File System (HDFS) using Apache Pig. In this mode, whenever we execute the Pig Latin statements to process the data, a MapReduce job is invoked in the back-end to perform a particular operation on the data that exists in the HDFS.

Apache Pig Execution Mechanisms

Apache Pig scripts can be executed in three ways, namely, interactive mode, batch mode, and embedded mode.

- **Interactive Mode** (Grunt shell) – You can run Apache Pig in interactive mode using the Grunt shell. In this shell, you can enter the Pig Latin statements and get the output (using Dump operator).
- **Batch Mode** (Script) – You can run Apache Pig in Batch mode by writing the Pig Latin script in a single file with **.pig** extension.
- **Embedded Mode** (UDF) – Apache Pig provides the provision of defining our own functions (**U**ser **D**efined **F**unctions) in programming languages such as Java, and using them in our script.

Invoking the Grunt Shell

You can invoke the Grunt shell in a desired mode (local/MapReduce) using the **–x** option as shown below.

Local mode	MapReduce mode
Command – $ ./pig –x local	**Command –** $ ./pig -x mapreduce
Output – 15/09/28 10:13:03 INFO pig.Main: Logging error messages to: /home/Hadoop/pig_1443415383991.log 2015-09-28 10:13:04,838 [main] INFO org.apache.pig.backend.hadoop.execution engine.HExecutionEngine - Connecting to hadoop file system at: file:/// grunt>	**Output –** 15/09/28 10:13:03 INFO pig.Main: Logging error messages to: /home/Hadoop/pig_1443415383991.log 2015-09-28 10:13:04,838 [main] INFO org.apache.pig.backend.hadoop.execution engine.HExecutionEngine - Connecting to hadoop file system at: file:/// grunt>

Either of these commands gives you the Grunt shell prompt as shown below.

```
grunt>
```

You can exit the Grunt shell using **'ctrl + d'.**

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(',');
```

Executing Apache Pig in Batch Mode

You can write an entire Pig Latin script in a file and execute it using the **–x command**. Let us suppose we have a Pig script in a file named **sample_script.pig** as shown below.

Sample_script.pig

```
student = LOAD 'hdfs://localhost:9000/pig_data/student.txt' USING
 PigStorage(',') as (id:int,name:chararray,city:chararray);
 Dump student;
```

Now, you can execute the script in the above file as shown below.

Local mode	MapReduce mode

$ pig -x local **Sample_script.pig**	$ pig -x mapreduce **Sample_script.pig**

CONCLUSION:

Pig Latin's ability to include user code at any point in the pipeline is useful for pipeline development. If SQL is used, data must first be imported into the database, and then the cleansing and transformation process can begin.

Program formation/ Execution/ ethical practices (06)	**Timely Submission and Documentation (02)**	**Viva Answer (02)**	**Experiment Marks (10)**	**Teacher Signature with date**

PIG commands

1) Loading the contents into pig (simple example)

Step1: In Local directory create file using vi editor.

Step2: Relocate the file into HDFS.

Step3: Check the contents.

Step4: Open pig in grunt shell by just entering Pig in terminal.

Step5: Create one variable and LOAD the contents into it.

Res = LOAD '/user/training/file_name' USING PigStorage() AS

(col_name:data_type); Step6: Display the result

dump res;

2) Grouping

Step1, Step2, Step3, Step4, Step5,Step6 will remain same .

Step 7: Group the characters

Chargroup = GROUP chars by c;

Step8: Display the results

dump Chargroup;

3) using FOREACH

Step1, Step2, Step3, Step4 will remain same

Step 5: Load contents and display result

records = LOAD '/user/training/a' USING PigStorage(',') AS

(c:chararray, i:int); dump records;

(a,1)

(b,2)

(c,3)

Step 6: use FOREACH to generate the integer value

count = FOREACH records GENERATE I;

dump count;

4) using FOREACH with functions

Step 1 to 5 will remain same from 3 rd exp

Step 7 and Step 8 remains same from 2 nd exp

Step 9: count the number of occurrence of chars in Grouped chars

count = FOREACH chargroup GENERATE group, COUNT (chars);

Step 10: Display the result

dump count;

5) Tokenize Function

Step1: create a file with some text in local directory and copy it into HDFS.

Step2: Load the contents into variable in PIG and display the results

Step3: Use TOKENIZE function to break the text into tokens

Tokenbag = FOREACH <loaded variable > GENERATE TOKENIZE

(line); * (line is the col_name used while loading data)

dump tokenbag;

6) Inner Join Example

1) Load records into a bag from input #1.

grunt> posts = load '/training/data/user-posts.txt' using PigStorage(',') as
(user:chararray, post:chararray,

date:long);

2) Load records into a bag from input #2.

grunt> likes = load '/training/data/user-likes.txt' using PigStorage(',') as (user:chararray,

likes:int, date:long);

3) Join the data sets.

grunt> userInfo = join posts by user, likes by user;

grunt> dump userInfo;

7) Left Outer Join

1) Load records into a bag from input #1.

grunt> posts = load '/training/data/user-posts.txt' using PigStorage(',') as
(user:chararray, post:chararray,

date:long);

2) Load records into a bag from input #2.
grunt> likes = load '/training/data/user-likes.txt' using PigStorage(',') as (user:chararray,

likes:int, date:long);

3) Join the data sets.

grunt> userInfo = join posts by user LEFT OUTER, likes by user;

grunt> dump userInfo;

8) Right Outer Join

1) Load records into a bag from input #1.

grunt> posts = load '/training/data/user-posts.txt' using PigStorage(',') as (user:chararray, post:chararray,

date:long);

2) Load records into a bag from input #2.

grunt> likes = load '/training/data/user-likes.txt' using PigStorage(',') as (user:chararray,

likes:int, date:long);

3) Join the data sets.

grunt> userInfo = join posts by user RIGHT OUTER, likes by user;

grunt> dump userInfo;

9) Full Outer Join

1) Load records into a bag from input #1.

grunt> posts = load '/training/data/user-posts.txt' using PigStorage(',') as (user:chararray, post:chararray,

date:long);

2) Load records into a bag from input #2.

grunt> likes = load '/training/data/user-likes.txt' using PigStorage(',') as (user:chararray,

likes:int, date:long);

3) Join the data sets.

grunt> userInfo = join posts by user FULL OUTER, likes by user;

grunt> dump userInfo;

10) WordCount in Pig

Step1: Create file in local directory.

Step2: Copy file into HDFS directory

Step3: Check contents.
Step4: LOAD the file contents.

Step5: using TOKENIZE and FLATTEN operator rearrange the

output Words = FOREACH lines GENERATE FLATTEN

(TOKENIZE(line)) as word; Step6: Group the contents

Grouped = GROUP words by word;

Step7: Perform WordCount

wordcount = FOREACH grouped GENERATE group,

COUNT(words); Step8: Display result

dump wordcount;