PIG Activity Sheet

Apache Pig is a framework/platform which can be used to analyse large content data set. People with scripting language back ground will find Apache Pig very familiar.

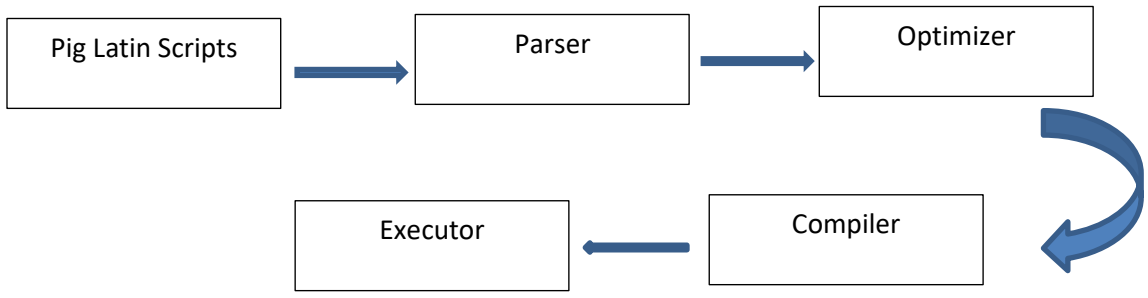
Ease of use of PIG

- People who are not familiar with writing map reduce programmes in Java, can easily adapt to write Pig scripts to accomplish their tasks.
- Do not need to write lengthy map reduce codes, same functionalities can be achieved in small amount of code.
- Support for join, filter and group like functions.

Let us see some comparison between Pig and MapReduce.

PIG	Map Reduce
Do not require to have complex programming skills	To develop map reduce programme, good understanding of programming knowledge is required.
Pig scripts, internally get converted to sequence of map reduce programmes, programmer do not have to worry about compilation process	Map reduce jobs required to be compiled before they can be run. Compilation process can be long and required a certain amount of process to follow.
Amount of coding is less	Amount of code is normally high in Map Reduce Programmes.

Flow of executing Pig Latin Scripts:



Learning Outcomes:

- Ways of Invoking Grunt Shell.
- Read Data from local file system or HDFS.
- Perform various Pig operations.
- Solve a word count problem using Pig scripts.

In this lab session, we will use the dataset (XML dataset) which we placed using flume in the lab previously.
We will process that data in various way by using PIG.

PIG Lab Session: Instructions

1. The environment in which Pig Latin commands are executed: - Currently there is support for Local and Hadoop modes

2. Pig compiler converts Pig Latin to MapReduce

3. Think of Pig as a "Pig Latin" compiler, development tool and executor

4. We can manually override the default mode via "-x" or "-exectype" options

$pig -x mapreduce

$ pig (Connects to hadoop file system at: Connecting to hadoop file system at: hdfs://bigdata)

$ pig -x local (Connects to hadoop file system at: file:///)

Grunt is an Interactive Shell for executing Pig Commands, it is started when script file is NOT provided. It can execute scripts from Grunt via exec command.

Building blocks

Field - piece of data

Tuple - ordered set of fields, represented with "(" and ")"

(10.4, 5, word, 4, field1)

Bag - collection of tuples, represented with "{" and "}"

{ (10.4, 5, word, 4, field1), (this, 1, blah) }

Analogy of Data structures in PIG to Relational Database

- Bag is a table in the database
- Tuple is a row in a table
- Bags do not require that all tuples contain the same number unlike relational table

Conventions

- Upper case is a system keyword
- Lowercase is something that you provide

Example

In this example, we will work on the xml data that is already available in HDFS.

Open Grunt shell in local mode.

```
$ pig -x local
```

`REGISTER /usr/hdp/current/pig-client/lib/piggybank.jar;` <- The Piggy Bank is a place for Pig users to share their functions. This contains the User Defined Functions that enables to auto-mate some frequent tasks. Here we use this jar to read the xml file.

Read the data from local file system.

```
> courses = LOAD '/home/manasm/reed.xml' USING
org.apache.pig.piggybank.storage.XMLLoader('COURSE') as (x:chararray);
```

Invoke PIG grunt shell in default mode.
command : $pig

courses = LOAD 'hdfs://bigdata/user/manasm/reed.xml' USING
org.apache.pig.piggybank.storage.XMLLoader('COURSE') as (x:chararray);

If you invoke grunt shell in mapreduce mode:

courses = LOAD '/user/manasm/reed.xml' USING
org.apache.pig.piggybank.storage.XMLLoader('COURSE') as (x:chararray);

Parse Data
> records = FOREACH courses GENERATE
FLATTEN(REGEX_EXTRACT_ALL(x,'<COURSE>\\s*<REG_NUM>(.*)</REG_NUM>\\s*<SUBJ>(.*)</SUBJ>\\s*<CRSE>(.*)</CRSE>\\s*<SECT>(.*)</SECT>\\s*<TITLE>(.*)</TITLE>\\s*<UNITS>(.*)</UNITS>\\s*<INSTRUCTOR>(.*)</INSTRUCTOR>\\s*<DAYS>(.*)</DAYS>\\s*<START_TIME>(.*)</START_TIME>\\s*<END_TIME>(.*)</END_TIME>\\s*<BUILDING>(.*)</BUILDING>\\s*<ROOM>(.*)</ROOM>\\s*</COURSE>'));

```
> describe records;
```
Schema for records unknown.

Apply/Define the schema to the data:

> coursesData = FOREACH records GENERATE (int)$0 AS register_number, (chararray)$1 AS subject, (int)$2 AS course, (chararray)$3 AS section, (chararray)$4 AS title, (float)$5 AS units, (chararray)$6 AS instructor, (chararray)$7 AS days, (chararray)$8 AS start_time, (chararray)$9 AS end_time, (chararray)$10 AS building, (chararray)$11 AS room;

```
> describe coursesData;
```

coursesData: {register_number: int,subject: chararray,course: int,section: chararray,title: chararray,units: float,instructor: chararray,days: chararray,start_time: chararray,end_time: chararray,building: chararray,room: chararray}

Now we can see the schema for the Bag.

Now let us try to display some records, we will restrict for only 5 records for the display of data.

```
> coursedata_lim = LIMIT coursesData 10;
> dump coursedata_lim;
```

No action is taken until DUMP or STORE commands are encountered. Pig will parse, validate and analyse statements but not execute them.

DUMP - displays the results to the screen
STORE - saves results (typically to a file)

You can store the data to hdfs by using command like given below:
> STORE records INTO '<HDFS Location>' USING PigStorage ('|');
Hadoop data is usually quite large and it does not make sense to print it to the screen. The common pattern is to persist results to Hadoop (HDFS, HBase). This is done with STORE command. For information and debugging purposes, you can print a small sub-set to the screen.

Brief Description of commands:
1. 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 PigStorage 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 and declares types to fields.

Schema Data Types
Simple
Int - Signed 32-bit integer - Ex:10
long - Signed 64-bit integer - Ex:10L or 10l
float - 32-bit floating point - Ex:10.5F or 10.5f
double - 64-bit floating point - Ex:10.5 or 10.5e2 or 10.5E2

Arrays
chararray - Character array (string) in Unicode UTF-8 hello world
bytearray - Byte array (blob)

Complex Data Types
tuple - An ordered set of fields Ex: (19,2)
bag - An collection of tuples {(19,2), (18,1)}

Pig Latin – Diagnostic Tools

Display the structure of the Bag

> DESCRIBE <bag_name>;

Display Execution Plan

- Produces Various reports
- Logical Plan
- MapReduce Plan

> EXPLAIN <bag_name>;

Pig Latin - Grouping

Example

We will use **coursesData** which we have generated in the previous example.

```
> course_group = GROUP coursesData by subject;
> course_group_lim = LIMIT course_group 5;
> dump course_group_lim;
> describe course_group;
> EXPLAIN course_group;
```

Pig Latin - FOREACH

> FOREACH <bag> GENERATE <data>

Iterate over each element in the bag and produce a result

Example

We will use **coursesData** which we have generated in the previous example

```
> faculty_name = FOREACH coursesData generate subject,instructor;
> faculty_name_lim = LIMIT faculty_name 20;
> dump faculty_name_lim;
```

FOREACH with Functions

> FOREACH B GENERATE group, FUNCTION(A);

Example

course_group - already created based course grouping

```
> courseCount = FOREACH course_group GENERATE group,
COUNT(coursesData)
> dump courseCount;
```

```
> all_rec = GROUP coursesData ALL;
> all_rec_count = FOREACH all_rec Generate
COUNT(coursesData.register_number);
> dump all_rec_count;
```

Tokenize Function

Splits a string into tokens and outputs as a bag of tokens

Example

```
> token = FOREACH coursesData generate title;
> token_limit = LIMIT token 10;
> dump token_limit;
```

"Sample Output"
(MolecularStructure and Properties)
(MolecularStructure and Properties)
(MolecularStructure and Properties)
(MolecularStructure and Properties)
(Introduction to Anthropology)
(Field Biology of Amphibians)
(Bacterial Pathogenesis)
(MolecularStructure and Properties)

Commands for TOKENIZE:

```
> token_ize = FOREACH coursesData generate TOKENIZE(title);
> token_ize_lim = LIMIT token_ize 10;
> dump token_ize_lim;
```

"Sample Output"
({(MolecularStructure),(and),(Properties)})
({(MolecularStructure),(and),(Properties)})
({(MolecularStructure),(and),(Properties)})

FLATTEN Operator

Flattens nested bags and data types
FLATTEN is not a function, it's an operator - Rearranges output

Example

```
> flat_bag = FOREACH token_ize_lim generate FLATTEN($0);
> dump flat_bag;
```

Sample output:
(MolecularStructure)
(and)
(Properties)
(MolecularStructure)
(and)
(Properties)

Joins Overview

Critical Tool for Data Processing
Will probably be used in most of your Pig scripts
Pig supports

- Inner Joins
- Outer Joins
- Full Joins

How to Join in Pig

Join Steps

1. Load records into a bag from input #1
2. Load records into a bag from input #2
3. Join the 2 data-sets (bags) by provided join key

Default Join is Inner Join

1. Rows are joined where the keys match
2. Rows that do not have matches are not included in the result

Example of running Pig Scripts from File:

Use the code in "InnerJoin.pig" script

Now let use see the code for inner join operation, create a file called InnerJoin.pig and place the below code there :

Create a directory in your local system as PigData where you will keep your data files and another directory called PigScripts where you will put your pig scripts.
Then go to the directory where you have kept your pig scripts and give the below command to run the script.

```
$ pig -x local InnerJoin.pig
```

Input for this code will be two files called user-posts.txt and user-likes.txt

User-posts.txt contains:

user1,Funny Story,1343182026191
user2,Cool Deal,1343182133839
user4,Interesting Post,1343182154633
user5,Yet Another Blog,13431839394

user-likes.txt contains:

user1,12,1343182026191
user2,7,1343182139394
user3,0,1343182154633
user4,50,1343182147364

After you create all the files run your script file.

Output will be as given below.

(user1,Funny Story,1343182026191,user1,12,1343182026191)
(user2,Cool Deal,1343182133839,user2,7,1343182139394)
(user4,Interesting Post,1343182154633,user4,50,1343182147364)

Join by Multiple Keys

- Must provide the same number of keys
- Each key must be of the same type

Example

Use the code in "InnerJoinWithMultipleKeys.pig" script

Code is given below:

```
$ pig -x local InnerJoinWithMultipleKeys.pig
```

Output is given below:

(user1,Funny Story,1343182026191,user1,12,1343182026191)

Outer Join

1. Left Outer

Records from the first data-set are included whether they have a match or not. Fields from the unmatched (second) bag are set to null.

2. Right Outer

The opposite of Left Outer Join: Records from the second data-set are included no matter what. Fields from the unmatched (first) bag are set to null.

3. Full Outer

Records from both sides are included. For unmatched records the fields from the 'other' bag are set to null.

Example

Use the code in LeftOuterJoin.pig script

Code for this script is given below.

```
$ pig -x local LeftOuterJoin.pig
```

Output is given below:

(user1,Funny Story,1343182026191,user1,12,1343182026191)
(user2,Cool Deal,1343182133839,user2,7,1343182139394)
(user4,Interesting Post,1343182154633,user4,50,1343182147364)
(user5,Yet Another Blog,13431839394,,,)

Example

See the code in RightOuterJoin.pig script

Code is given below:

```
$ pig -x local RightOuterJoin.pig
```

Output is given below:

(user1,Funny Story,1343182026191,user1,12,1343182026191)
(user2,Cool Deal,1343182133839,user2,7,1343182139394)
(,,,user3,0,1343182154633)
(user4,Interesting Post,1343182154633,user4,50,1343182147364)

Cogroup

- Joins data-sets preserving structure of both sets
- Creates tuple for each key

Matching tuples from each relationship become fields

Example

Use the code present in "Cogroup.pig" script

Code of the script given below:

```
$ pig -x local Cogroup.pig
```

Output is given below:

(user1,{(user1,Funny Story,1343182026191)},{(user1,12,1343182026191)})
(user2,{(user2,Cool Deal,1343182133839)},{(user2,7,1343182139394)})
(user3,{},{(user3,0,1343182154633)})
(user4,{(user4,Interesting Post,1343182154633)},{(user4,50,1343182147364)})
(user5,{(user5,Yet Another Blog,13431839394)},{})

Cogroup with INNER

1.Cogroup by default is an OUTER JOIN
2.You can remove empty records with empty bags by performing INNER on each bag "INNER JOIN" like functionality

Example

Use the code in script "CogroupInner.pig"

Code is given below:

$ pig -x local CogroupInner.pig

Output is given below:

(user1,{(user1,Funny Story,1343182026191)},{(user1,12,1343182026191)})
(user2,{(user2,Cool Deal,1343182133839)},{(user2,7,1343182139394)})
(user4,{(user4,Interesting Post,1343182154633)},{(user4,50,1343182147364)})

WORD COUNT PROBLEM EXAMPLE USING PIG:

Let us see an example for word count problem using PIG.
Here we will be using a file called wordcount.txt and I have given its contents below.
hello world one two three hello world
three four five six seven one one one
hello, world hello world
Now let's see how we can get the wordcounts using PIG .

Code is given below:

```
lines = LOAD '/home/manasm/wordcount.txt' AS (c:chararray);

words = FOREACH lines generate TOKENIZE(c,' ') as twords;

flat_words = FOREACH words generate FLATTEN(twords) as fwords;

grouped_words = GROUP flat_words by fwords;

wc = FOREACH grouped_words GENERATE group, COUNT(flat_words);

dump wc;
```

Output is given below:

(one,4)
(six,1)
(two,1)
(five,1)
(four,1)
(hello,3)
(seven,1)
(three,2)
(world,4)
(hello,,1)

Summary:

In this lab activity for PIG we have learned about:

- PIG basics
- PIG Architecture
- PIG Functionalities
- Comparison between PIG scripts and Map reduce

We also learned about PIG concepts like.

- Data access by PIG using LOAD and STORE
- Operator like GROUP, FLATTEN, TOKENIZE, FOREACH
- We also see many examples of joins in PIG and how to run them as pig scripts
- We have seen a sample example of word count problem also.