**Experiment-9**

**Install and Running Pig then write Pig Latin scripts to sort and group your data.**

**Experiment-9.2**

**Aim: To write Pig Latin scripts to sort and group data.**

1. **Pig Latin Concepts**

* 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) }
* Similar 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

1. **Simple** Pig Latin Example

$ pig (Start Grunt with default MapReduce mode)

grunt> cat /training/playArea/pig/a.txt a 1 (Grunt supports file system commands)

d 4

c 9

k 6

Load contents of text files into a Bag named records

grunt> records = LOAD '/training/playArea/pig/a.txt' as (letter:chararray, count:int);

grunt> dump records; (Display records bag to the screen)

Results of the bag named records are printed to the screen

(a,1) (d,4) (c,9) (k,6)

1. **DUMP** and STORE statements
   * + No action is taken until DUMP or STORE commands are encountered

* Pig will parse, validate and analyze statements but not execute them
  + - DUMP – displays the results to the screen
    - STORE – saves results (typically to a file)

records = LOAD '/training/playArea/pig/a.txt' as (letter:chararray, count:int);

Nothing is executed; Pig will optimize this entire chunk of scrip

DUMP final\_bag;

1. **Large** Data
   * + Hadoop data is usually quite large and it doesn’t 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

grunt> records = LOAD '/training/playArea/pig/excite-small.log' AS (userId:chararray, timestamp:long, query:chararray);

grunt> toPrint = LIMIT records 5;

grunt> DUMP toPrint; (Only 5 records will be displayed)

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’)
* The delimiter can be customized using regular expressions

AS – assign a schema to incoming data

– Assigns names to fields

– Declares types to fields

LOAD Command Example

records = LOAD '/training/playArea/pig/excite-small.log' USING PigStorage() AS (userId:chararray, timestamp:long, query:chararray);

1. **Schema** Data Types



1. **Pig Latin – Diagnostic Tools**

• Display the structure of the Bag

– grunt> DESCRIBE ;

Display Execution Plan

– Produces Various reports

• Logical Plan

• MapReduce Plan

– grunt> EXPLAIN <bag\_name>;

Illustrate how Pig engine transforms the data

– grunt> ILLUSTRATE <bag\_name>;

1. **Pig Latin - Grouping**

grunt> chars = LOAD '/training/playArea/pig/b.txt' AS (c:chararray);

grunt> describe chars;

chars: {c: chararray}

grunt> dump chars;

Creates a new bag with element named group and element named chars

(a)

(k)

...

...

(k)

(c)

(k)

The chars bag is grouped by “c”; therefore ‘group’ element will contain unique values

grunt> charGroup = GROUP chars by c;

grunt> describe charGroup;

charGroup:{group: chararray,chars: {(c: chararray)}}

grunt> dump charGroup;

‘chars’ element is a bag itself and contains all tuples from ‘chars’ bag that match the value form ‘c’

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

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

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

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

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

1. **ILUSTRATE Command**



1. **Inner vs. Outer Bag**

grunt> chars = LOAD ‘/training/playArea/pig/b.txt' AS (c:chararray);

grunt> charGroup = GROUP chars by c;

grunt> **ILLUSTRATE charGroup**;



grunt> chars = LOAD '/training/playArea/pig/b.txt' AS

(c:chararray);

grunt> charGroup = GROUP chars by c;

grunt> **dump charGroup;**



1. **Pig Latin – FOREACH**

**FOREACH <bag> GENERATE <data>**

– Iterate over each element in the bag and produce a result

– Ex: grunt> result = FOREACH bag GENERATE f1;

grunt> records = LOAD 'data/a.txt' AS (c:chararray, i:int);

grunt> dump records;

(a,1)

(d,4)

(c,9)

(k,6)

grunt> **counts = foreach records generate i;**

grunt> dump counts;

(1)

(4)

(9)

(6)

Note: For each row emit ‘i’ field

1. **FOREACH with Functions**

**FOREACH B GENERATE group, FUNCTION(A);**

– Pig comes with many functions including COUNT, FLATTEN, CONCAT, etc...

– Can implement a custom function

grunt> chars = LOAD 'data/b.txt' AS (c:chararray);

grunt> charGroup = GROUP chars by c;

grunt> dump charGroup;

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

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

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

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

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

grunt> describe charGroup;

charGroup: {group: chararray,chars: {(c: chararray)}}

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

For each row in ‘charGroup’ bag, emit group field and count the number of items in ‘chars’ bag

grunt> dump counts;

(a,3)

(c,2)

(i,3)

(k,4)

(l,2)

1. **TOKENIZE Function**

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

– Separators are: space, double quote("), coma(,) parenthesis(()), star(\*)

grunt> linesOfText = LOAD 'data/c.txt' AS (line:chararray);

grunt> dump linesOfText;

Split each row line by space and return a bag of tokens

(this is a line of text)

(yet another line of text)

(third line of words)

grunt> **tokenBag = FOREACH linesOfText GENERATE TOKENIZE(line);**

grunt> dump tokenBag;

({(this),(is),(a),(line),(of),(text)})

Each row is a bag of words produced by TOKENIZE function

({(yet),(another),(line),(of),(text)})

({(third),(line),(of),(words)})

grunt> describe tokenBag;

tokenBag: {bag\_of\_tokenTuples: {tuple\_of\_tokens: (token: chararray)}}

1. **FLATTEN Operator**

• **Flattens nested bags and data types**

• **FLATTEN is not a function, it’s an operator**

– Re-arranges output



1. **Sorting Data**

* Relations are unordered!
* sort order not guaranteed to be retained

=> sort just before output

a = LOAD 'input/pig/sort/A' AS (first:int, second:int);

b = ORDER a BY $0, $1 DESC;

DUMP b;

c = LIMIT b 2;

DUMP c;

1. **Conventions and Case Sensitivity**



Example

Find out maximum temperature by year

Note: For simplicity, the program assumes that the input is tab-delimited text, with each line having just year, temperature, and quality fields.

Year Temp Quality

1974 48 0

1974 48 1

1991 56 2

1983 89 3

1993 91 4

1938 41 5

1938 56 6

1941 93 9

1983 87 3

pig -x local

records = LOAD path/sample.txt' AS (year:chararray, temperature:int, quality:int);

DUMP records;

DESCRIBE records;

filtered\_records = FILTER records BY temperature != 9999 AND quality IN (0,1,4,5,9);

DUMP filtered\_records;

DESCRIBE filtered\_records;

grouped\_records = GROUP filtered\_records BY year;

DUMP grouped\_records;

DESCRIBE grouped\_records;

max\_temp = FOREACH grouped\_records GENERATE group, MAX(filtered\_records.temperature);

DUMP max\_temp;

DESCRIBE max\_temp;