Apache PIG

Apache Pig

- **Pig is an abstraction on top of Hadoop**
  - Provides high level programming language designed for data processing
  - Converted into MapReduce and executed on Hadoop Clusters
- Originally implemented at Yahoo! to allow analysts to access data
- Pig is widely accepted and used
  - Yahoo!, Twitter, Netflix, etc...
- **Top Level Apache Project**
  - http://pig.apache.org

Pig's Use Cases

- **Extract Transform Load (ETL)**
  - Ex: Processing large amounts of log data
    - clean bad entries, join with other data-sets
- **Research of "raw" information**
  - Ex. User Audit Logs
  - Schema maybe unknown or inconsistent
  - Data Scientists and Analysts may like Pig's data transformation paradigm

Pig Components

- **Pig Latin**
  - Command based language
  - Designed specifically for data transformation and flow expression
- **Execution Environment**
  - The environment in which Pig Latin commands are executed
  - Currently there is support for Local and Hadoop modes
- **Pig compiler converts Pig Latin to MapReduce**
  - Compiler strives to optimize execution
  - You automatically get optimization improvements with Pig updates

Running Pig

- **Script**
  - Execute commands in a file
  - $pig scriptFile.pig
- **Grunt**
  - Interactive Shell for executing Pig Commands
  - Started when script file is NOT provided
  - Can execute scripts from Grunt via run or exec commands
- **Embedded**
  - Execute Pig commands using PigServer class
    - Just like JDBC to execute SQL
  - Can have programmatic access to Grunt via PigRunne

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

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

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

Schema Data Types

Type	Description	Example
	Simple	
int	Signed 32-bit integer	10
long	Signed 64-bit integer	10L or 10l
float	32-bit floating point	10.5F or 10.5f
double	64-bit floating point	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	(19,2)
bag	An collection of tuples	{(19,2), (18,1)}
map	An collection of tuples	[open#apache]

Diagnostic Tools

- **Display the structure of the Bag**
  - grunt> DESCRIBE <bag_name>;
- **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>;

Grouping

```
grunt> chars = LOAD '/training/playArea/pig/b.txt' AS
(c:chararray);
grunt> describe chars;
chars: {c: chararray}
grunt> dump chars;
(a)
(k)
...
...
(k)
(c)
(k)
grunt> charGroup = GROUP chars by c;
grunt> describe charGroup;
charGroup: {group: chararray,chars: {(c: chararray)}}
grunt> dump charGroup;
(a,{(a),(a),(a)})
(c,{(c),(c)})
(i,{(i),(i),(i)})
(k,{(k),(k),(k),(k)})
(l,{(l),(l)})
```

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

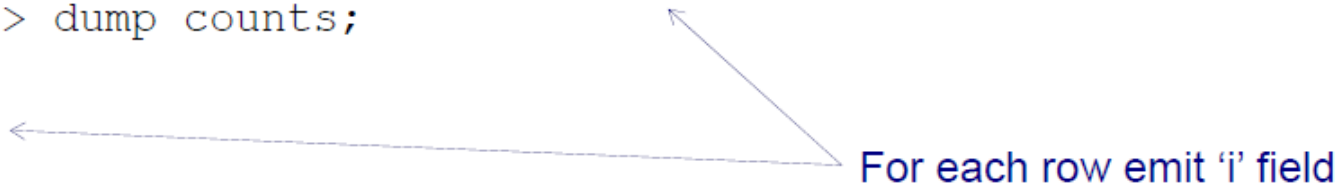
The chars bag is grouped by "c"; therefore 'group' element will contain unique values

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

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)
```



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;
(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)})
({(yet),(another),(line),(of),(text)})
({(third),(line),(of),(words)})
grunt> describe tokenBag;
tokenBag: {bag_of_tokenTuples: {tuple_of_tokens: (token: chararray)}}
```

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

Each row is a bag of words produced by TOKENIZE function

STORE and DUMP commands

- 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)

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

```
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

Joins in Pig

- **Pigs supports**
  - Inner Joins
  - Outer Joins
  - Full Joins

- **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**
  - Rows are joined where the keys match
  - Rows that do not have matches are not included in the result

Simple Inner Join Example

```
--InnerJoin.pig
posts = load '/training/data/user-posts.txt' using PigStorage(',')
       as (user:chararray,post:chararray,date:long);

likes = load '/training/data/user-likes.txt' using PigStorage(',')
       as (user:chararray,likes:int,date:long);


userInfo = join posts by user, likes by user;


dump userInfo;
```

1: Load records into a bag from input #1

Use comma as a separator

1:Load records into a bag from input #2

3: Join the 2 data-sets

When a key is equal in both data-sets then the rows are joined into a new single row; In this case when user name is equal

User Defined Functions (UDFs)

- There are times when Pig's built in operators and functions will not suffice
- **Pig provides ability to implement your own**
  - Filter
    - Ex: res = FILTER bag BY udfFilter(post);
  - Load Function
    - Ex: res = load 'file.txt' using udfLoad();
  - Eval
    - Ex: res = FOREACH bag GENERATE udfEval($1)
- **Choice between several programming languages**
  - Java, Python, Javascript

Thank You