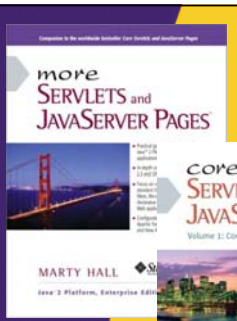
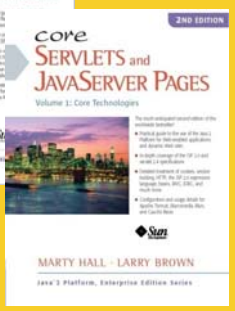
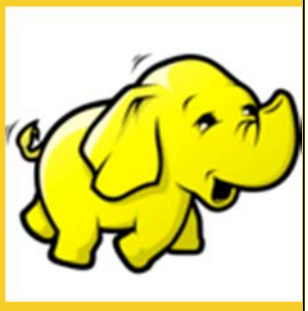
Apache Pig

Originals of Slides and Source Code for Examples:
http://www.coreservlets.com/hadoop-tutorial/

© 2012 coreservlets.com and Dima May







For live Hadoop training, please see courses at http://courses.coreservlets.com/.

Taught by the author of this Hadoop tutorial. Available at public venues, or customized versions can be held on-site at your organization.

- Courses developed and taught by Marty Hall
  - JSF 2, PrimeFaces, servlets/JSP, Ajax, jQuery, Android development, Java 6 or 7 programming, custom mix of topics
  - Ajax courses can concentrate on 1 library (jQuery, Prototype/Scriptaculous, Ext-JS, Dojo, etc.) or survey several
- Courses developed and taught by coreservlets.com experts (edited by Marty)
  - **Hadoop,** Spring, Hibernate/JPA, GWT, SOAP-based and RESTful Web Services

Contact hall@coreservlets.com for details

Agenda

- **Pig Overview**
- **Execution Modes**
- **Installation**
- **Pig Latin Basics**
- **Developing Pig Script**
  - Most Occurred Start Letter
- **Resources**

Pig

"is a platform for analyzing large data sets that consists of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs. "

- **Top Level Apache Project**
  - http://pig.apache.org
- **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
- **Pig is widely accepted and used**
  - Yahoo!, Twitter, Netflix, etc...

Pig and MapReduce

- **MapReduce requires programmers**
  - Must think in terms of map and reduce functions
  - More than likely will require Java programmers
- **Pig provides high-level language that can be used by**
  - Analysts
  - Data Scientists
  - Statisticians
  - Etc...
- **Originally implemented at Yahoo! to allow analysts to access data**

Pig's Features

- **Join Datasets**
- **Sort Datasets**
- **Filter**
- **Data Types**
- **Group By**
- **User Defined Functions**
- **Etc..**

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

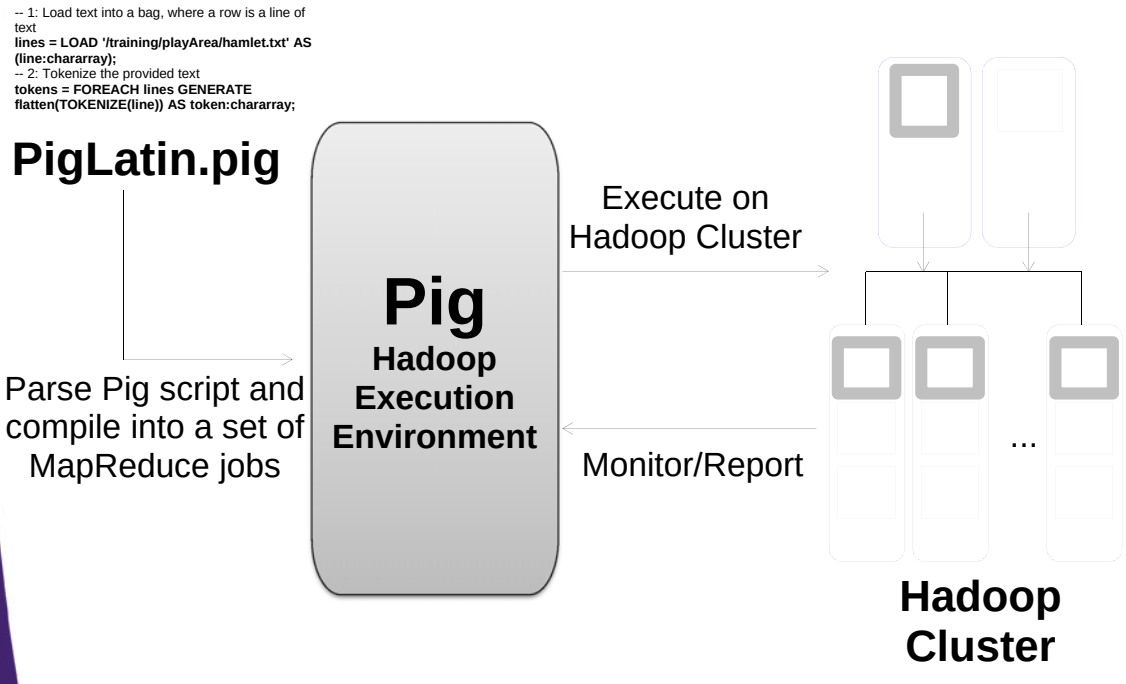
Pig Components

- **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

Execution Modes

- **Local**
  - Executes in a single JVM
  - Works exclusively with local file system
  - Great for development, experimentation and prototyping
- **Hadoop Mode**
  - Also known as MapReduce mode
  - Pig renders Pig Latin into MapReduce jobs and executes them on the cluster
  - Can execute against semi-distributed or fully-distributed hadoop installation
    - We will run on semi-distributed cluster

Hadoop Mode



Installation Prerequisites

- **Java 6**
  - With $JAVA_HOME environment variable properly set
- **Cygwin on Windows**

Installation

- **Add pig script to path**
  - export PIG_HOME=$CDH_HOME/pig-0.9.2-cdh4.0.0
  - export PATH=$PATH:$PIG_HOME/bin
- **$ pig -help**
- **That's all we need to run in local mode**
  - Think of Pig as a 'Pig Latin' compiler, development tool and executor
  - Not tightly coupled with Hadoop clusters

Pig Installation for Hadoop Mode

- **Make sure Pig compiles with Hadoop**
  - Not a problem when using a distribution such as Cloudera Distribution for Hadoop (CDH)
- **Pig will utilize $HADOOP_HOME and $HADOOP_CONF_DIR variables to locate Hadoop configuration**
  - We already set these properties during MapReduce installation
  - Pig will use these properties to locate Namenode and Resource Manager

Running Modes

- **Can manually override the default mode via '-x' or '-exectype' options**
  - $pig -x local
  - $pig -x mapreduce

$ pig
2012-07-14 13:38:58,139 [main] INFO org.apache.pig.Main - Logging error messages to: /home/hadoop/Training/play_area/pig/pig_1342287538128.log
2012-07-14 13:38:58,458 [main] INFO org.apache.pig.backend.hadoop.executionengine.HExecutionEngine - Connecting to hadoop file system at: **hdfs://localhost:8020**

$ pig -x local
2012-07-14 13:39:31,029 [main] INFO org.apache.pig.Main - Logging error messages to: /home/hadoop/Training/play_area/pig/pig_1342287571019.log
2012-07-14 13:39:31,232 [main] INFO org.apache.pig.backend.hadoop.executionengine.HExecutionEngine - Connecting to hadoop file system at: **file:///**

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 PigRunner class

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

Simple Pig Latin Example

```
$ pig
grunt> cat /training/playArea/pig/a.txt
a       1
d       4
c       9
k       6
grunt> records = LOAD '/training/playArea/pig/a.txt' as
(letter:chararray, count:int);
grunt> dump records;
...
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer
.MapReduceLauncher - 50% complete
2012-07-14 17:36:22,040 [main] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer
.MapReduceLauncher - 100% complete
...
(a,1)
(d,4)
(c,9)
(k,6)
grunt>
```

Start Grunt with default MapReduce mode

Grunt supports file system commands

Load contents of text files into a Bag named *records*

Display records bag to the screen

Results of the bag named records are printed to the screen

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)**

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

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

The fun begins here

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;
```

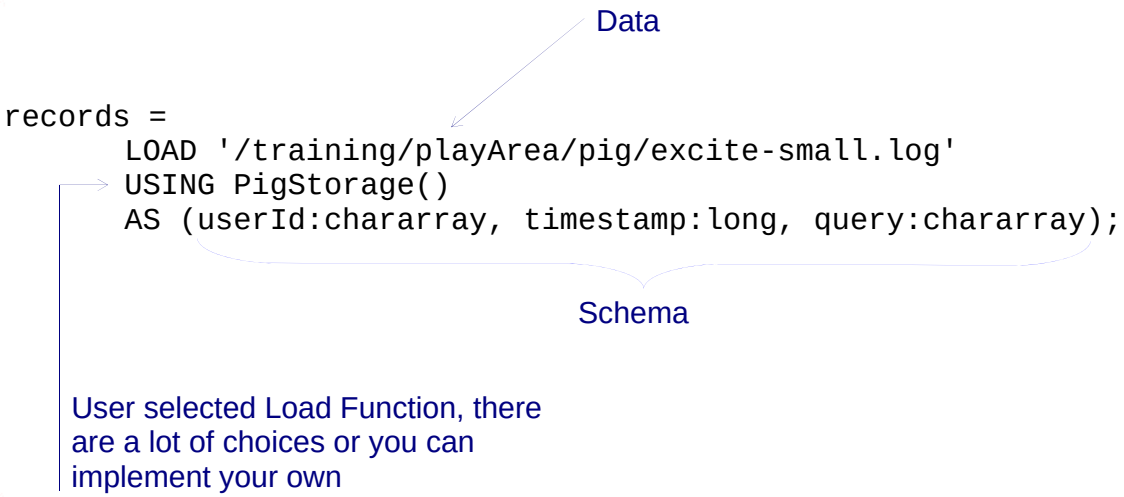
Only 5 records will be displayed

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



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]

Source: Apache Pig Documentation 0.9.2; “Pig Latin Basics”. 2012

Pig Latin – 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>;

Pig Latin - 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’

ILUSTRATE Command

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

chars	c:chararray
	c
	c

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

Inner vs. Outer Bag

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

chars	c:chararray
	c
	c

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

Inner Bag

Outer Bag

Inner vs. Outer Bag

```
grunt> chars = LOAD '/training/playArea/pig/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)})
```

Inner Bag

Outer Bag

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)
```

For each row emit 'i' field

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);
grunt> dump counts;
(a,3)
(c,2)
(i,3)
(k,4)
(l,2)
```

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

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

FLATTEN Operator

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

```
grunt> dump tokenBag;
({(this),(is),(a),(line),(of),(text)})
({(yet),(another),(line),(of),(text)})
({(third),(line),(of),(words)})
grunt> flatBag = FOREACH tokenBag GENERATE flatten($0);
grunt> dump flatBag;
(this)
(is)
(a)
...
...
(text)
(third)
(line)
(of)
(words)
```

Nested structure: bag of bags of tuples

Each row is flatten resulting in a bag of simple tokens

Elements in a bag can be referenced by index

Conventions and Case Sensitivity

- **Case Sensitive**
  - Alias names
  - Pig Latin Functions
- **Case Insensitive**
  - Pig Latin Keywords

```
counts = FOREACH charGroup GENERATE group, COUNT(c);
```

Alias Case Sensitive

Function Case Sensitive

Alias Case Sensitive

Keywords Case Insensitive

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

Problem: Locate Most Occurred Start Letter

- Calculate number of occurrences of each letter in the provided body of text
- Traverse each letter comparing occurrence count
- Produce start letter that has the most occurrences

(For so this side of our known world esteem'd him)
Did slay this Fortinbras; who, by a seal'd compact,
Well ratified by law and heraldry,
Did forfeit, with his life, all those his lands
Which he stood seiz'd of, to the conqueror;
Against the which a moiety competent
Was gaged by our king; which had return'd
To the inheritance of Fortinbras,

A	89530
B	3920
..	
..	
Z	876

T	495959

'Most Occurred Start Letter' Pig Way

1. Load text into a bag (named 'lines')
2. Tokenize the text in the 'lines' bag
3. Retain first letter of each token
4. Group by letter
5. Count the number of occurrences in each group
6. Descending order the group by the count
7. Grab the first element => Most occurring letter
8. Persist result on a file system

1: Load Text Into a Bag

```
grunt> lines = LOAD '/training/data/hamlet.txt'
AS (line:chararray);
```

Load text file into a bag, stick entire line into element ‘line’ of type ‘chararray’

INSPECT lines bag:

```
grunt> describe lines;
lines: {line: chararray}
grunt> toDisplay = LIMIT lines 5;
grunt> dump toDisplay;
(This Etext file is presented by Project Gutenberg, in)
(This etext is a typo-corrected version of Shakespeare's Hamlet,)
(cooperation with World Library, Inc., from their Library of the)
(*This Etext has certain copyright implications you should read!*
(Future and Shakespeare CDROMS.  Project Gutenberg often releases
```

Each row is a line of text

2: Tokenize the Text in the ‘Lines’ Bag

```
grunt> tokens = FOREACH lines GENERATE
flatten(TOKENIZE(line)) AS token:chararray;
```

For each line of text (1) tokenize that line
(2) flatten the structure to produce 1 word per row

INSPECT tokens bag:

```
grunt> describe tokens
tokens: {token: chararray}
grunt> toDisplay = LIMIT tokens 5;
grunt> dump toDisplay;
(a)
(is)
(of)
(This)
(etext)
```

Each row is now a token

3: Retain First Letter of Each Token

```
grunt> letters = FOREACH tokens GENERATE
SUBSTRING(token,0,1) AS letter:chararray;
```

For each token grab the first letter; utilize SUBSTRING function

INSPECT letters bag:

```
grunt> describe letters;
letters: {letter: chararray}
grunt> toDisplay = LIMIT letters 5;
grunt> dump toDisplay;
(a)
(i)
(T)
(e)
(t)
```

What we have no is 1 character per row

4: Group by Letter

```
grunt> letterGroup = GROUP letters BY letter;
```

Create a bag for each unique character; the "grouped" bag will contain the same character for each occurrence of that character

INSPECT letterGroup bag:

```
grunt> describe letterGroup;
letterGroup: {group: chararray,letters: {(letter: chararray)}}
grunt> toDisplay = LIMIT letterGroup 5;
grunt> dump toDisplay;
(0,{(0),(0),(0)})
(a,{(a),(a)})
(2,{(2),(2),(2),(2),(2)})
(3,{(3),(3),(3)})
(b,{(b)})
```

Next we'll need to convert characters occurrences into counts; Note this display was modified as there were too many characters to fit on the screen

5: Count the Number of Occurrences in Each Group

```
grunt> countPerLetter = FOREACH letterGroup
GENERATE group, COUNT(letters);
```

For each row, count occurrence of the letter

INSPECT countPerLetter bag:

```
grunt> describe countPerLetter;
countPerLetter: {group: chararray,long}
grunt> toDisplay = LIMIT countPerLetter 5;
grunt> dump toDisplay;
(A,728)
(B,325)
(C,291)
(D,194)
(E,264)
```

Each row now has the character and the number of times it was found to start a word. All we have to do is find the maximum

6: Descending Order the Group by the Count

```
grunt> orderedCountPerLetter = ORDER
countPerLetter BY $1 DESC;
```

Simply order the bag by the first element, a number of occurrences for that element

INSPECT orderedCountPerLetter bag:

```
grunt> describe orderedCountPerLetter;
orderedCountPerLetter: {group: chararray,long}
grunt> toDisplay = LIMIT orderedCountPerLetter 5;
grunt> dump toDisplay;
(t,3711)
(a,2379)
(s,1938)
(m,1787)
(h,1725)
```

All we have to do now is just grab the first element

7: Grab the First Element

```
grunt> result = LIMIT orderedCountPerLetter 1;
```

The rows were already ordered in descending order, so simply limiting to one element gives us the result

INSPECT `orderedCountPerLetter` bag:

```
grunt> describe result;
result: {group: chararray,long}
grunt> dump result;
(t,3711)
```

There it is

8: Persist Result on a File System

```
grunt> STORE result INTO
'/training/playArea/pig/mostSeenLetterOutput';
```

Result is saved under the provided directory

INSPECT result

```
$ hdfs dfs -cat
/training/playArea/pig/mostSeenLetterOutput/part-r-00000
t	3711
```

result

Notice that result was stored int part-r-0000, the regular artifact of a MapReduce reducer; Pig compiles Pig Latin into MapReduce code and executes.

MostSeenStartLetter.pig Script

```
-- 1: Load text into a bag, where a row is a line of text
lines = LOAD '/training/data/hamlet.txt' AS (line:chararray);
-- 2: Tokenize the provided text
tokens = FOREACH lines GENERATE flatten(TOKENIZE(line)) AS token:chararray;
-- 3: Retain first letter of each token
letters = FOREACH tokens GENERATE SUBSTRING(token,0,1) AS letter:chararray;
-- 4: Group by letter
letterGroup = GROUP letters BY letter;
-- 5: Count the number of occurrences in each group
countPerLetter = FOREACH letterGroup GENERATE group, COUNT(letters);
-- 6: Descending order the group by the count
orderedCountPerLetter = ORDER countPerLetter BY $1 DESC;
-- 7: Grab the first element => Most occurring letter
result = LIMIT orderedCountPerLetter 1;
-- 8: Persist result on a file system
STORE result INTO '/training/playArea/pig/mostSeenLetterOutput';
```

- **Execute the script:**
  - $ pig MostSeenStartLetter.pig

Pig Tools

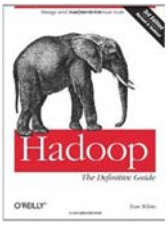
- **Community has developed several tools to support Pig**
  - https://cwiki.apache.org/confluence/display/PIG/PigTools
- **We have PigPen Eclipse Plugin installed:**
  - Download the latest jar release at https://issues.apache.org/jira/browse/PIG-366
    - As of writing org.apache.pig.pigpen_0.7.5.jar
  - Place jar in eclupse/plugins/
  - Restart eclipse

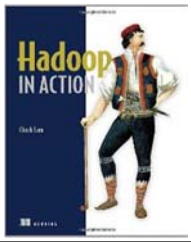
Pig Resources

- **Apache Pig Documentation**
  - http://pig.apache.org

Programming Pig
Alan Gates (Author)
O'Reilly Media; 1st Edition (October, 2011)

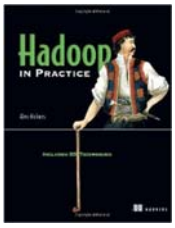
Chapter About Pig
Hadoop: The Definitive Guide
Tom White (Author)
O'Reilly Media; 3rd Edition (May6, 2012)





Chapter About Pig
Hadoop in Action
Chuck Lam (Author)
Manning Publications; 1st Edition (December, 2010)

Pig Resources



Hadoop in Practice
Alex Holmes (Author)
Manning Publications; (October 10, 2012)

Wrap-Up

Summary

- **We learned about**
  - Pig Overview
  - Execution Modes
  - Installation
  - Pig Latin Basics
  - Resources
- **We developed Pig Script to locate "Most Occurred Start Letter"**

Questions?