Apache Pig

CMSC 491
Hadoop-Based Distributed Computing
Spring 2016
Adam Shook

Objectives

- Develop understanding of Pig’s data model
- Understand basics of PigLatin

What Is Pig?

- Developed by Yahoo! and a top level Apache project
- Immediately makes data on a cluster available to non-Java programmers via Pig Latin – a dataflow language
- Interprets Pig Latin and generates MapReduce jobs that run on the cluster
- Enables easy data summarization, ad-hoc reporting and querying, and analysis of large volumes of data
- Pig interpreter runs on a client machine – no administrative overhead required

Pig Terms

- All data in Pig one of four types:
  - An Atom is a simple data value - stored as a string but can be used as either a string or a number
  - A Tuple is a data record consisting of a sequence of "fields"
    - Each field is a piece of data of any type (atom, tuple or bag)
  - A Bag is a set of tuples (also referred to as a ‘Relation’)
    - The concept of a table
  - A Map is a map from keys that are string literals to values that can be any data type
    - The concept of a hash map

Pig Capabilities

- Support for
  - Grouping
  - Joins
  - Filtering
  - Aggregation
- Extensibility
  - Support for User Defined Functions (UDF's)
- Leverages the same massive parallelism as native MapReduce

Pig Basics

- Pig is a client application
  - No cluster software is required
- Interprets Pig Latin scripts to MapReduce jobs
  - Parses Pig Latin scripts
  - Performs optimization
  - Creates execution plan
- Submits MapReduce jobs to the cluster

Execution Modes

- Pig has two execution modes
  - Local Mode - all files are installed and run using your local host and file system
  - MapReduce Mode - all files are installed and run on a Hadoop cluster and HDFS installation
- Interactive
  - By using the Grunt shell by invoking Pig on the command line

```
$ pig
grunt>
```

- Batch
  - Run Pig in batch mode using Pig Scripts and the "pig" command

```
$ pig –f id.pig –p <param>=<value> ...
```

Pig Latin

- Pig Latin scripts are generally organized as follows
  - A LOAD statement reads data
  - A series of “transformation” statements process the data
  - A STORE statement writes the output to the filesystem
    - A DUMP statement displays output on the screen
- Logical vs. physical plans:
  - All statements are stored and validated as a logical plan
  - Once a STORE or DUMP statement is found the logical plan is executed

Example Pig Script

```
-- Load the content of a file into a pig bag named ‘input_lines’
input_lines = LOAD 'CHANGES.txt' AS (line:chararray);

-- Extract words from each line and put them into a pig bag named ‘words’
words = FOREACH input_lines GENERATE FLATTEN(TOKENIZE(line)) AS word;

-- filter out any words that are just white spaces
filtered_words = FILTER words BY word MATCHES '\\w+';

-- create a group for each word
word_groups = GROUP filtered_words BY word;

-- count the entries in each group
word_count = FOREACH word_groups GENERATE COUNT(filtered_words) AS count, group AS word;

-- order the records by count
ordered_word_count = ORDER word_count BY count DESC;

-- Store the results ( executes the pig script )
STORE ordered_word_count INTO 'output';
```

Basic “grunt” Shell Commands

- Help is available

```
$ pig -h
```

- Pig supports HDFS commands

```
grunt> pwd
```

  – put, get, cp, ls, mkdir, rm, mv, etc.

About Pig Scripts

- Pig Latin statements grouped together in a file
- Can be run from the command line or the shell
- Support parameter passing
- Comments are supported
  - Inline comments '--'
  - Block comments /* */

Simple Data Types

Type	Description
int	4-byte integer
long	8-byte integer
float	4-byte (single precision) floating point
double	8-byte (double precision) floating point
bytearray	Array of bytes; blob
chararray	String (“hello world”)
boolean	True/False (case insensitive)
datetime	A date and time
biginteger	Java BigInteger
bigdecimal	Java BigDecimal

Complex Data Types

Type	Description
Tuple	Ordered set of fields (a “row / record”)
Bag	Collection of tuples (a “resultset / table”)
Map	A set of key-value pairs Keys must be of type chararray

Pig Data Formats

- BinStorage
  - Loads and stores data in machine-readable (binary) format
- PigStorage
  - Loads and stores data as structured, field delimited text files
- TextLoader
  - Loads unstructured data in UTF-8 format
- PigDump
  - Stores data in UTF-8 format
- YourOwnFormat!
  - via UDFs

Loading Data Into Pig

- Loads data from an HDFS file

```
var = LOAD 'employees.txt';
var = LOAD 'employees.txt' AS (id, name,
  salary);
var = LOAD 'employees.txt' using PigStorage()
       AS (id, name, salary);
```

- Each LOAD statement defines a new bag
  - Each bag can have multiple elements (atoms)
  - Each element can be referenced by name or position ($*n*)
- A bag is immutable
- A bag can be aliased and referenced later

Input And Output

- STORE
  - Writes output to an HDFS file in a specified directory

    ```
    grunt> STORE processed INTO 'processed_txt';
    ```

    - Fails if directory exists
    - Writes output files, part-[m|r]-xxxxx, to the directory
  - PigStorage can be used to specify a field delimiter
- DUMP
  - Write output to screen

    ```
    grunt> DUMP processed;
    ```

Relational Operators

- FOREACH
  - Applies expressions to every record in a bag
- FILTER
  - Filters by expression
- GROUP
  - Collect records with the same key
- ORDER BY
  - Sorting
- DISTINCT
  - Removes duplicates

FOREACH . . .GENERATE

- Use the FOREACH …GENERATE operator to work with rows of data, call functions, etc.
- Basic syntax:

```
alias2 = FOREACH alias1 GENERATE expression;
```

- Example:

```
DUMP alias1;
(1,2,3) (4,2,1) (8,3,4) (4,3,3) (7,2,5) (8,4,3)
alias2 = FOREACH alias1 GENERATE col1, col2;
DUMP alias2;
(1,2) (4,2) (8,3) (4,3) (7,2) (8,4)
```

FILTER. . .BY

- Use the FILTER operator to restrict tuples or rows of data
- Basic syntax:

```
alias2 = FILTER alias1 BY expression;
```

- Example:

```
DUMP alias1;
(1,2,3) (4,2,1) (8,3,4) (4,3,3) (7,2,5) (8,4,3)
alias2 = FILTER alias1 BY (col1 == 8) OR (NOT
   (col2+col3 > col1));
DUMP alias2;
(4,2,1) (8,3,4) (7,2,5) (8,4,3)
```

GROUP. . .ALL

- Use the GROUP…ALL operator to group data
  - Use GROUP when only one relation is involved
  - Use COGROUP with multiple relations are involved
- Basic syntax:

```
alias2 = GROUP alias1 ALL;
```

- Example:

```
DUMP alias1;
(John,18,4.0F) (Mary,19,3.8F) (Bill,20,3.9F) (Joe,
   18,3.8F)
alias2 = GROUP alias1 BY col2;
DUMP alias2;
(18,{(John,18,4.0F),(Joe,18,3.8F)})
(19,{(Mary,19,3.8F)})
(20,{(Bill,20,3.9F)})
```

ORDER. . .BY

- Use the ORDER…BY operator to sort a relation based on one or more fields
- Basic syntax:

```
alias = ORDER alias BY field_alias [ASC|DESC];
```

- Example:

```
DUMP alias1;
(1,2,3) (4,2,1) (8,3,4) (4,3,3) (7,2,5) (8,4,3)
alias2 = ORDER alias1 BY col3 DESC;
DUMP alias2;
(7,2,5) (8,3,4) (1,2,3) (4,3,3) (8,4,3) (4,2,1)
```

DISTINCT. . .

- Use the DISTINCT operator to remove duplicate tuples in a relation.
- Basic syntax:

```
alias2 = DISTINCT alias1;
```

- Example:

```
DUMP alias1;
(8,3,4) (1,2,3) (4,3,3) (4,3,3) (1,2,3)
alias2= DISTINCT alias1;
DUMP alias2;
(8,3,4) (1,2,3) (4,3,3)
```

Relational Operators

- FLATTEN
  - Used to un-nest tuples as well as bags
- INNER JOIN
  - Used to perform an inner join of two or more relations based on common field values
- OUTER JOIN
  - Used to perform left, right or full outer joins
- SPLIT
  - Used to partition the contents of a relation into two or more relations
- SAMPLE
  - Used to select a random data sample with the stated sample size

INNER JOIN. . .

- Use the JOIN operator to perform an inner, equi-join join of two or more relations based on common field values
- The JOIN operator always performs an inner join
- Inner joins ignore null keys
  - Filter null keys before the join
- JOIN and COGROUP operators perform similar functions
  - JOIN creates a flat set of output records
  - COGROUP creates a nested set of output records

INNER JOIN Example

```
DUMP Alias1;
    (1,2,3)
    (4,2,1)
    (8,3,4)
    (4,3,3)
    (7,2,5)
    (8,4,3)
DUMP Alias2;
    (2,4)
    (8,9)
    (1,3)
    (2,7)
    (2,9)
    (4,6)
    (4,9)
```

```
Join Alias1 by Col1 to
Alias2 by Col1
    Alias3 = JOIN Alias1 BY
    Col1, Alias2 BY Col1;

Dump Alias3;
    (1,2,3,1,3)
    (4,2,1,4,6)
    (4,3,3,4,6)
    (4,2,1,4,9)
    (4,3,3,4,9)
    (8,3,4,8,9)
    (8,4,3,8,9)
```

OUTER JOIN. . .

- Use the OUTER JOIN operator to perform left, right, or full outer joins
  - Pig Latin syntax closely adheres to the SQL standard
- The keyword OUTER is optional
  - keywords LEFT, RIGHT and FULL will imply left outer, right outer and full outer joins respectively
- Outer joins will only work provided the relations which need to produce nulls (in the case of non-matching keys) have schemas
- Outer joins will only work for two-way joins
  - To perform a multi-way outer join perform multiple two-way outer join statements

OUTER JOIN Examples

- Left Outer Join
  - A = LOAD 'a.txt' AS (n:chararray, a:int);
  - B = LOAD 'b.txt' AS (n:chararray, m:chararray);
  - C = JOIN A by $0 LEFT OUTER, B BY $0;
- Full Outer Join
  - A = LOAD 'a.txt' AS (n:chararray, a:int);
  - B = LOAD 'b.txt' AS (n:chararray, m:chararray);
  - C = JOIN A BY $0 FULL OUTER, B BY $0;

User-Defined Functions

- Natively written in Java, packaged as a jar file
  - Other languages include Jython, JavaScript, Ruby, Groovy, and Python
- Register the jar with the REGISTER statement
- Optionally, alias it with the DEFINE statement

```
REGISTER /src/myfunc.jar;
A = LOAD 'students';
B = FOREACH A GENERATE myfunc.MyEvalFunc($0);
```

DEFINE

- DEFINE can be used to work with UDFs and also streaming commands
  - Useful when dealing with complex input/output formats

```
/* read and write comma-delimited data */
DEFINE Y 'stream.pl' INPUT(stdin USING PigStreaming(','))
    OUTPUT(stdout USING PigStreaming(','));
A = STREAM X THROUGH Y;

/* Define UDFs to a more readable format */
DEFINE MAXNUM org.apache.pig.piggybank.evaluation.math.MAX;
A = LOAD ‘student_data’ AS (name:chararray, gpa1:float, gpa2:double);
B = FOREACH A GENERATE name, MAXNUM(gpa1, gpa2);
DUMP B;
```

References

- http://pig.apache.org