Pig

Arend Hintze

“this is so cumbersome!”

- Instead of programming everything in java MapReduce or streaming: wouldn’t it we wonderful to have a simpler interface?
- Problem: break down complex MapReduce tasks into simple commands
- pig does that!

approach

- Pig Latin -> high level language to program tasks
    - takes input and cues commands on that input to create output
- Pig compiler -> takes a script, creates jobs, runs them locally or distributed on HDFS

interacting with pig

- grunt shell -> enter commands directly (start: pig -x local OR: pig -X mapreduce)
- pig pigScript.pig -> executes a pig script
- put your commands inside your java program (import org.apache.pig.PigServer)
- OR: use the graphical web interface!

typical workflow

- Load data into an alias
    - *alias* = LOAD *filename* AS (…)
- Manipulate the alias using relational operators or functions
    - *new_alias* = pig_command(*old_alias*)
- Dump alias to shell, output, or file in a HDFS directory

Pig Lating Commands Categories

- Read-Write from/to HDFS
- Diagnostics
- Data types
- Expression and functions
- Relational operators

Read-Write Operators

Table 10.2 Data read/write operators in Pig Latin

`LOAD`	`alias = LOAD 'file' [USING function] [AS schema];` Load data from a file into a relation. Uses the `PigStorage` load function as default unless specified otherwise with the `USING` option. The data can be given a schema using the `AS` option.
`LIMIT`	`alias = LIMIT alias n;` Limit the number of tuples to *n*. When used right after `alias` was processed by an `ORDER` operator, `LIMIT` returns the first *n* tuples. Otherwise there's no guarantee which tuples are returned. The `LIMIT` operator defies categorization because it's certainly not a read/write operator but it's not a true relational operator either. We include it here for the practical reason that a reader looking up the `DUMP` operator, explained later, will remember to use the `LIMIT` operator right before it.
`DUMP`	`DUMP alias;` Display the content of a relation. Use mainly for debugging. The relation should be small enough for printing on screen. You can apply the `LIMIT` operation on an alias to make sure it's small enough for display.
`STORE`	`STORE alias INTO 'directory' [USING function];` Store data from a relation into a directory. The directory must not exist when this command is executed. Pig will create the directory and store the relation in files named `part-`*`nnnnn`* in it. Uses the `PigStorage` store function as default unless specified otherwise with the `USING` option.

example

```
data = LOAD ‘movie_short.csv’ using PigStorage(‘,’) AS
(id,name,year,rating,score);
res = FILTER data BY (float)rating>2.0;
DUMP res;
```

LOADING

```
alias = LOAD ‘file’ [USING function] [AS schema];
```

- Default: assumes data is tab-delimeted
  - if data has different spacers use PigStorage(‘delimiter’)
  - the schema can have types

```
data = LOAD ‘movie_short.csv’ using PigStorage(‘,’) AS
(id:int,name:chararray,year:int,rating:float,score:float);
```

SAVING

- DUMP just dumps the output to the command line or display
- STORE saves the content to a file
- LIMIT allows you to specify the number of tuples to be returned

saving

```
data = LOAD ‘movie_short.csv’ using PigStorage(‘,’) AS
(id,name,year,rating,score);
res = FILTER data BY (float)rating>2.0;
STORE res INTO ‘output’;
```

uses tabs as delimiter

```
data = LOAD ‘movie_short.csv’ using PigStorage(‘,’) AS
(id,name,year,rating,score);
res = FILTER data BY (float)rating>2.0;
STORE res INTO ‘output’ using PigStorage(‘,’);
```

uses other delimiter

LIMIT

```
data = LOAD ‘movie_short.csv’ using PigStorage(‘,’) AS
(id,name,year,rating,score);
res = FILTER data BY (float)rating>2.0;
res_limit = LIMIT res 10;
STORE res_limit INTO ‘output’ using PigStorage(‘,’);
```

DIAGNOSTIC

`DESCRIBE`	`DESCRIBE alias;` Display the schema of a relation.
`EXPLAIN`	`EXPLAIN [-out path] [-brief] [-dot] [-param ...] [-param_file ...] alias;` Display the execution plan used to compute a relation. When used with a script name, for example, `EXPLAIN myscript.pig`, it will show the execution plan of the script.
`ILLUSTRATE`	`ILLUSTRATE alias;` Display step-by-step how data is transformed, starting with a load command, to arrive at the resulting relation. To keep the display and processing manageable, only a (not completely random) sample of the input data is used to simulate the execution. In the unfortunate case where none of Pig's initial sample will survive the script to generate meaningful data, Pig will "fake" some similar initial data that will survive to generate data for alias.

Atomic Data Types

Table 10.4 Atomic data types in Pig Latin

`int`	Signed 32-bit integer
`long`	Signed 64-bit integer
`float`	32-bit floating point
`double`	64-bit floating point
`chararray`	Character array (string) in Unicode UTF-8
`bytearray`	Byte array (binary object)

```
data = LOAD ‘movie_short.csv’ using PigStorage(‘,’) AS
(id:int,name:chararray,year:int,rating:float,score:float);
```

Complex Data Types

Table 10.5 Complex data types in Pig Latin

`Tuple`	`(12.5,hello world,-2)` A tuple is an ordered set of fields. It's most often used as a row in a relation. It's represented by fields separated by commas, all enclosed by parentheses.
`Bag`	`{(12.5,hello world,-2),(2.87,bye world,10)}` A bag is an unordered collection of tuples. A relation is a special kind of bag, sometimes called an outer bag. An inner bag is a bag that is a field within some complex type. A bag is represented by tuples separated by commas, all enclosed by curly brackets. Tuples in a bag aren't required to have the same schema or even have the same number of fields. It's a good idea to do this though, unless you're handling semistructured or unstructured data.
`Map`	`[key#value]` A map is a set of key/value pairs. Keys must be unique and be a string (`chararray`). The value can be any type.

Data Types

- A field in a tuple or a value in a map can be null or any atomic/complex type (NESTING)
  - (John , {(48, Jolly Rd, Okemos),(10, Grand,Lansing)})
- Defining a schema
  - if you leave out the field type Pig will default to byte array
  - if you leave out the name a field would be unnamed and you can reference it by it's position ($0, $1, $2 … and so on)

Loading complex data types

```
cat data;
(3,8,9)  (4,5,6)
(1,4,7)  (3,7,5)
(2,5,8)  (9,5,8)

A = LOAD 'data' AS (t1:tuple(t1a:int, t1b:int,t1c:int),
                    t2:tuple(t2a:int,t2b:int,t2c:int));
DUMP A;
((3,8,9),(4,5,6))
((1,4,7),(3,7,5))
((2,5,8),(9,5,8))

X = FOREACH A GENERATE t1.t1a,t2.$0;

DUMP X;
(3,4)
(1,3)
(2,9)
```

tuples are tab delimited

Expressions 1

- expressions are used in FILTER, FOREACH, GROUP and SPLIT as well as in eval functions

Constant	`12, 19.2, 'hello world'`	Constant values such as 19 or “hello world.” Numeric values without decimal point are treated as `int` unless l or L is appended to the number to make it a `long`. Numeric values with a decimal point are treated as `double` unless f or F is appended to the number to make it a `float`.
Basic arithmetic	`+,-,*,/`	Plus, minus, multiply, and divide.
Sign	`+x, -x`	Negation (-) changes the sign of a number.
Cast	`(t)x`	Convert the value of *x* into type *t*.
Modulo	`x % y`	The remainder of x divided by *y*.

Expressions 2

Conditional	`(x ? y : z)`	Returns *y* if x is true, z otherwise. This expression must be enclosed in parentheses.
Comparison	`==, !=, <, >, <=, >=`	Equals to, not equals to, greater than, less than, etc.
Pattern matching	`x matches regex`	Regular expression matching of string x. Uses Java's regular expression format (under the `java.util.regex.Pattern` class) to specify `regex`.
Null	`x is null`, `x is not null`	Check if x is null (or not).
Boolean	`x and y`, `x or y`, `not x`	Boolean and, or, not.

Built-In Functions

AVG	Calculate the average of numeric values in a single-column bag.
CONCAT	Concatenate two strings (chararray) or two bytearrays.
COUNT	Calculate the number of tuples in a bag. See SIZE for other data types.
DIFF	Compare two fields in a tuple. If the two fields are bags, it will return tuples that are in one bag but not the other. If the two fields are values, it will emit tuples where the values don't match.
MAX	Calculate the maximum value in a single-column bag. The column must be a numeric type or a chararray.
MIN	Calculate the minimum value in a single-column bag. The column must be a numeric type or a chararray.
SIZE	Calculate the number of elements. For a bag it counts the number of tuples. For a tuple it counts the number of elements. For a chararray it counts the number of characters. For a bytearray it counts the number of bytes. For numeric scalars it always returns 1.
SUM	Calculate the sum of numeric values in a single-column bag.
TOKENIZE	Split a string (chararray) into a bag of words (each word is a tuple in the bag). Word separators are space, double quote ("), comma, parentheses, and asterisk (*).
IsEmpty	Check if a bag or map is empty.

case sensitive!