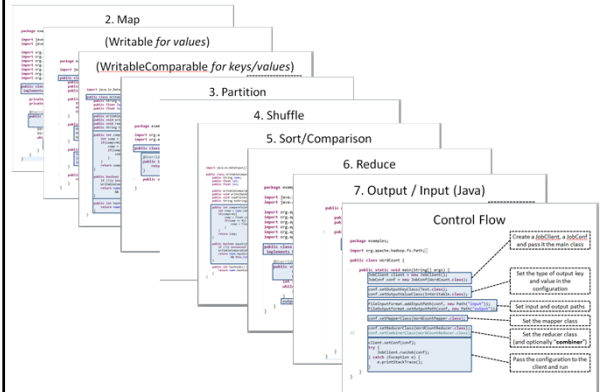


CC5212-1  
 PROCESAMIENTO MASIVO DE DATOS  
 OTOÑO 2015

Lecture 6: DFS & MapReduce III

Aidan Hogan  
 aidhog@gmail.com

Apache Hadoop (Java)



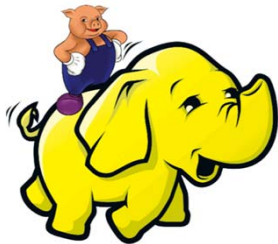
An Easier Way?



APACHE PIG: OVERVIEW

Apache Pig

- Create MapReduce programs to **run on Hadoop**
- Use a high-level “scripting” language called **Pig Latin**
- Can embed **User Defined Functions**: call a Java function (or Python, Ruby, etc.)
- Based on **Pig Relations**



Pig Latin: Hello Word Count



```
input_lines = LOAD '/tmp/book.txt' AS (line:chararray);

-- Extract words from each line and put them into a pig bag
-- datatype, then flatten the bag to get one word on each row
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;

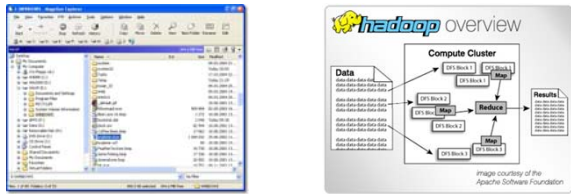
-- order the records by count
ordered_word_count = ORDER word_count BY count DESC;

STORE ordered_word_count INTO '/tmp/book-word-count.txt';
```

Map Reduce  
 Map + Reduce

Any guesses which line(s) correspond to Map, Reduce?

### Pig: Local Mode vs. MapReduce Mode



### Three Ways to Execute Pig: (i) Grunt

```

grunt> in_lines = LOAD '/tmp/book.txt' AS (line:chararray);
grunt> words = FOREACH in_lines GENERATE FLATTEN(TOKENIZE(line)) AS word;
grunt> filtered_words = FILTER words BY word MATCHES '\\w*';
grunt> ...
grunt> STORE ordered_word_count INTO '/tmp/book-word-count.txt';
    
```

### Three Ways to Execute Pig: (ii) Script

```

grunt> pig wordcount.pig
    
```

### Three Ways to Execute Pig: (iii) Embedded

```

package scratch;
import org.apache.pig.PigServer;
public class PigLatinWordCount {
    public static void main(String[] args) {
        String inputFile = args[0];
        String outputFile = args[1];
        try {
            PigServer pigServer = new PigServer("local");
            pigServer.registerQuery("in_lines = LOAD '"+ inputFile + "' AS (line:chararray);");
            pigServer.registerQuery("words = FOREACH in_lines GENERATE FLATTEN(TOKENIZE(line)) AS word;");
            // ...
            pigServer.store("ordered_word_count", outputFile);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
    
```

### APACHE PIG: LIDER EXAMPLE

### Pig: Products by Hour

transact.txt



customer412	1L_Leche	2014-03-31T08:47:57Z	\$900
customer412	Nescafe	2014-03-31T08:47:57Z	\$2,000
customer412	Nescafe	2014-03-31T08:47:57Z	\$2,000
customer413	400g_Zanahoria	2014-03-31T08:48:03Z	\$1,240
customer413	El_Mercurio	2014-03-31T08:48:03Z	\$500
customer413	Gillette_Mach3	2014-03-31T08:48:03Z	\$8,250
customer413	Santo_Domingo	2014-03-31T08:48:03Z	\$2,450
customer413	Nescafe	2014-03-31T08:48:03Z	\$2,000
customer414	Rosas	2014-03-31T08:48:24Z	\$7,000
customer414	Chocolates	2014-03-31T08:48:24Z	\$9,230
customer414	300g_Frutillas	2014-03-31T08:48:24Z	\$1,230

Your boss in Lider Headquarters tells you to find out the frequency of premium items (price>\$1,000) sold per hour counting duplicate items from each customer once ...

Pig: Products by Hour

```
grunt> REGISTER userDefinedFunctions.jar
```

User-defined-functions written in Java (or Python, Ruby, etc. ...)

```
userDefinedFunctions.jar
public class ExtractHour extends EvalFunc<String> {
    public String exec(Tuple input) throws IOException {
        if (input == null || input.size() == 0)
            return null;
        try {
            String timestamp = (String)input.get(0);
            return timestamp.substring(6, 8);
        } catch (Exception e) {
            System.err.println("ExtractHour: failed to process input: error - " + e.getMessage());
            return null;
        }
    }
}
```

Pig: Products by Hour

```
grunt> REGISTER userDefinedFunctions.jar
grunt> raw = LOAD 'transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
```

View data as a (streaming) relation with fields (cust, item, etc.) and tuples (data rows) ...

cust	item	time	price
customer412	1L_Leche	2014-03-31T08:47:57Z	\$900
customer412	Nescafe	2014-03-31T08:47:57Z	\$2.000
customer412	Nescafe	2014-03-31T08:47:57Z	\$2.000
customer413	400g_Zanahoria	2014-03-31T08:48:03Z	\$1.240

raw:

Pig: Products by Hour

```
grunt> REGISTER userDefinedFunctions.jar
grunt> raw = LOAD 'transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
grunt> premium = FILTER raw BY org.udf.MinPrice1000(price);
```

Filter tuples depending on their value for a given attribute (in this case, price < 1000)

cust	item	time	price
customer412	1L_Leche	2014-03-31T08:47:57Z	\$900
customer412	Nescafe	2014-03-31T08:47:57Z	\$2.000
customer412	Nescafe	2014-03-31T08:47:57Z	\$2.000
customer413	400g_Zanahoria	2014-03-31T08:48:03Z	\$1.240

premium:

Pig: Products by Hour

```
grunt> REGISTER userDefinedFunctions.jar
grunt> raw = LOAD 'transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
grunt> premium = FILTER raw BY org.udf.MinPrice1000(price);
grunt> hourly = FOREACH premium GENERATE cust, item, org.udf.ExtractHour(time) AS hour, price;
```

cust	item	hour	price
customer412	Nescafe	08	\$2.000
customer412	Nescafe	08	\$2.000
customer413	400g_Zanahoria	08	\$1.240
customer413	Gillette_Mach3	08	\$8.250

hourly:

Pig: Products by Hour

```
grunt> REGISTER userDefinedFunctions.jar
grunt> raw = LOAD 'transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
grunt> premium = FILTER raw BY org.udf.MinPrice1000(price);
grunt> hourly = FOREACH premium GENERATE cust, item, org.udf.ExtractHour(time) AS hour, price;
grunt> unique = DISTINCT hourly;
```

cust	item	hour	price
customer412	Nescafe	08	\$2.000
customer412	Nescafe	08	\$2.000
customer413	400g_Zanahoria	08	\$1.240
customer413	Gillette_Mach3	08	\$8.250

unique:

Pig: Products by Hour

```
grunt> REGISTER userDefinedFunctions.jar
grunt> raw = LOAD 'transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
grunt> premium = FILTER raw BY org.udf.MinPrice1000(price);
grunt> hourly = FOREACH premium GENERATE cust, item, org.udf.ExtractHour(time) AS hour, price;
grunt> unique = DISTINCT hourly;
grunt> hrlItem = GROUP unique BY (item, hour);
```

cust	item	hour	price
customer412	Nescafe	08	\$2.000
customer413	400g_Zanahoria	08	\$1.240
customer413	Gillette_Mach3	08	\$8.250
customer413	Santo_Domingo	08	\$2.450

unique:

### Pig: Products by Hour

```

grunt> REGISTER userDefinedFunctions.jar
grunt> raw = LOAD 'transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
grunt> premium = FILTER raw BY org.udf.MinPrice1000(price);
grunt> hourly = FOREACH premium GENERATE cust, item,
org.udf.ExtractHour(time) AS hour, price;
grunt> unique = DISTINCT hourly;
grunt> hrItem = GROUP unique BY (item, hour);
    
```

[item, hour]	cust	item	hour	price
[Nescafe,08]	customer412	Nescafe	08	\$2.000
	customer413	Nescafe	08	\$2.000
	customer415	Nescafe	08	\$2.000
[400g_Zanahoria,08]	customer413	400g_Zanahoria	08	\$1.240
...	...	...	...	...

hrItem:

### Pig: Products by Hour

```

grunt> REGISTER userDefinedFunctions.jar
grunt> raw = LOAD 'transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
grunt> premium = FILTER raw BY org.udf.MinPrice1000(price);
grunt> hourly = FOREACH premium GENERATE cust, item,
org.udf.ExtractHour(time) AS hour, price;
grunt> unique = DISTINCT hourly;
grunt> hrItem = GROUP unique BY (item, hour);
grunt> hrItemCnt = FOREACH hrItem GENERATE flatten($0), COUNT($1) AS count;
    
```

[item, hour]	cust	item	hour	price
[Nescafe,08]	customer412	Nescafe	08	\$2.000
	customer413	Nescafe	08	\$2.000
	customer415	Nescafe	08	\$2.000
[400g_Zanahoria,08]	customer413	400g_Zanahoria	08	\$1.240
	count			
...	...	...	...	...

hrItem:

### Pig: Products by Hour

```

grunt> REGISTER userDefinedFunctions.jar
grunt> raw = LOAD 'transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
grunt> premium = FILTER raw BY org.udf.MinPrice1000(price);
grunt> hourly = FOREACH premium GENERATE cust, item,
org.udf.ExtractHour(time) AS hour, price;
grunt> unique = DISTINCT hourly;
grunt> hrItem = GROUP unique BY (item, hour);
grunt> hrItemCnt = FOREACH hrItem GENERATE flatten($0), COUNT($1) AS count;
    
```

[item, hour]	count
[400g_Zanahoria,08]	1
[Nescafe,08]	3
...	...

hrItemCnt:

### Pig: Products by Hour

```

grunt> REGISTER userDefinedFunctions.jar
grunt> raw = LOAD 'transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
grunt> premium = FILTER raw BY org.udf.MinPrice1000(price);
grunt> hourly = FOREACH premium GENERATE cust, item,
org.udf.ExtractHour(time) AS hour, price;
grunt> unique = DISTINCT hourly;
grunt> hrItem = GROUP unique BY (item, hour);
grunt> hrItemCnt = FOREACH hrItem GENERATE flatten($0), COUNT($1) AS count;
grunt> hrItemCntSorted = ORDER hrItemCnt BY count DESC;
    
```

[item, hour]	count
[400g_Zanahoria,08]	1
[Nescafe,08]	3
...	...

hrItemCnt:

### Pig: Products by Hour

```

grunt> REGISTER userDefinedFunctions.jar
grunt> raw = LOAD 'transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
grunt> premium = FILTER raw BY org.udf.MinPrice1000(price);
grunt> hourly = FOREACH premium GENERATE cust, item,
org.udf.ExtractHour(time) AS hour, price;
grunt> unique = DISTINCT hourly;
grunt> hrItem = GROUP unique BY (item, hour);
grunt> hrItemCnt = FOREACH hrItem GENERATE flatten($0), COUNT($1) AS count;
grunt> hrItemCntSorted = ORDER hrItemCnt BY count DESC;
    
```

[item, hour]	count
[Nescafe,08]	3
[400g_Zanahoria,08]	1
...	...

hrItemCntSorted:

### Pig: Products by Hour

```

grunt> REGISTER userDefinedFunctions.jar
grunt> raw = LOAD 'transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
grunt> premium = FILTER raw BY org.udf.MinPrice1000(price);
grunt> hourly = FOREACH premium GENERATE cust, item,
org.udf.ExtractHour(time) AS hour, price;
grunt> unique = DISTINCT hourly;
grunt> hrItem = GROUP unique BY (item, hour);
grunt> hrItemCnt = FOREACH hrItem GENERATE flatten($0), COUNT($1) AS count;
grunt> hrItemCntSorted = ORDER hrItemCnt BY count DESC;
grunt> STORE hrItemCntSorted INTO 'output.txt'
    
```

[item, hour]	count
[Nescafe,08]	3
[400g_Zanahoria,08]	1
...	...

hrItemCntSorted:

# APACHE PIG: SCHEMA

## Pig Relations

- **Pig Relations:** Like relational tables
  - Except tuples can be “jagged”
  - Fields in the same column don’t need to be same type
  - Relations are by default unordered
- **Pig Schema:** Names for fields, etc.
 

```
... AS (cust, item, time, price);
```

cust	item	time	price
customer412	1L_Leche	2014-03-31T08:47:57Z	\$900
customer412	Nescafe	2014-03-31T08:47:57Z	\$2.000
customer412	Nescafe	2014-03-31T08:47:57Z	\$2.000
customer413	400g_Zanahoria	2014-03-31T08:48:03Z	\$1.240
...	...	...	...

## Pig Fields

- **Pig Fields:**
  - Reference using name
    - `premium = FILTER raw BY org.udf.MinPrice1000(price);`
  - ... or position
    - `premium = FILTER raw BY org.udf.MinPrice1000($3);`

More readable!

Starts at zero.

cust	item	time	price
customer412	1L_Leche	2014-03-31T08:47:57Z	\$900
customer412	Nescafe	2014-03-31T08:47:57Z	\$2.000
customer412	Nescafe	2014-03-31T08:47:57Z	\$2.000
customer413	400g_Zanahoria	2014-03-31T08:48:03Z	\$1.240
...	...	...	...

## Pig Simple Types

- **Pig Types:**

```
LOAD 'transact.txt' USING PigStorage('\t') AS
(cust:charArray, item:charArray, time:datetime,
price:int);
```
- `int, long, float, double, biginteger, bigdecimal, boolean, chararray (string), bytearray (blob), datetime`

## Pig Types: Duck Typing

- What happens if you omit types?
  - Fields default to `bytearray`
  - Implicit conversions if needed (~duck typing)

```
A = LOAD 'data' AS (cust, item, hour, price);
B = FOREACH A GENERATE hour + 4 % 24;
C = FOREACH A GENERATE hour + 4f % 24;
```

hour an integer  
hour a float

## Pig Complex Types: Tuple

```
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;
```

	t1			t2		
	t1a	t1b	t1c	t2a	t2b	t2c
A:	3	8	9	4	5	6
	1	4	7	3	7	5
	2	5	8	9	5	8

### Pig Complex Types: Tuple

```
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)
```

	\$0	\$1
3	3	4
1	1	3
2	2	9

### Pig Complex Types: Bag

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

A = LOAD 'data' AS (c1:int, c2:int, c3:int);
B = GROUP A BY c1;
```

	c1	c2	c3
3	3	8	9
2	2	3	6
1	1	4	7
2	2	5	8

### Pig Complex Types: Bag

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

A = LOAD 'data' AS (c1:int, c2:int, c3:int);
B = GROUP A BY c1;
DUMP B;
(1, (1,4,7))
(2, (2,5,8),(2,3,6))
(3, (3,8,9))
```

group (c1)	A		
	c1	c2	c3
3	3	8	9
2	2	3	6
1	1	4	7

### Pig Complex Types: Map

```
cat prices;
[Nescafe#$2.000"]
[Gillette_Mach3#$8.250"]

A = LOAD 'prices' AS (M:map {});
```



### Pig Complex Types: Summary

- **tuple**: A row in a table / a list of fields
  - e.g., (customer412, Nescafe, 08, \$2.000)
- **bag**: A set of tuples (allows duplicates)
  - e.g., { (cust412, Nescafe, 08, \$2.000), (cust413, Gillette\_Mach3, 08, \$8.250) }
- **map**: A set of key-value pairs
  - e.g., [Nescafe#\$2.000]

### APACHE PIG: OPERATORS

### Pig Atomic Operators

- **Comparison**  
==, !=, >, <, >=, <=, matches (regex)
- **Arithmetic**  
+, -, \*, /
- **Reference**  
tuple.field, map#value
- **Boolean**  
AND, OR, NOT
- **Casting**

### Pig Conditionals

- **Ternary operator:**  
`hr12 = FOREACH item GENERATE hour%12, (hour>12 ? 'pm' : 'am');`
- **Cases:**  
`X = FOREACH A GENERATE hour%12, (  
CASE  
WHEN hour>12 THEN 'pm'  
ELSE 'am'  
END  
);`

### Pig Aggregate Operators

Can GROUP multiple items or COGROUP single item (COGROUP considered more readable for multiple items)

- **Grouping:**
  - **GROUP:** group on a single relation
    - `GROUP premium BY (item, hour);`
  - **COGROUP:** group multiple relations
    - `COGROUP premium BY (item, hour), cheap BY (item, hour);`
- **Aggregate Operations:**
  - **AVG, MIN, MAX, SUM, COUNT, SIZE, CONCAT**

### Pig Joins

```
cat data1;
(Nescafe,08,120)
(EI_Mercurio,08,142)
(Nescafe,09,153)

cat data2;
(2000,Nescafe)
(8250,Gillette_Mach3)
(500,EI_Mercurio)

A = LOAD 'data1' AS (prod:charArray, hour:int, count:int);
B = LOAD 'data2' AS (price:int, name:charArray);
X = JOIN A BY prod, B BY name;

DUMP X;
```

prod	hour	count	price	name
Nescafe	08	120	2000	Nescafe
Nescafe	09	153	2000	Nescafe
EI_Mercurio	08	142	500	EI_Mercurio

### Pig Joins

- **Inner join:** As shown (default)
- **Self join:** Copy an alias and join with that
- **Outer joins:**
  - **LEFT / RIGHT / FULL**
- **Cross product:**
  - **CROSS**

You guys know (or remember ☺) what an INNER JOIN is versus an OUTER JOIN / LEFT / RIGHT / FULL versus a CROSS PRODUCT?

### Pig Aggregate/Join Implementations

- **Custom partitioning / number of reducers:**
  - **PARTITION BY** specifies a UDF for partitioning
  - **PARALLEL** specifies number of reducers

```
X = JOIN A BY prod, B BY name PARTITION BY org.udp.Partitioner PARALLEL 5;
```

```
X = GROUP A BY hour PARTITION BY org.udp.Partitioner PARALLEL 5;
```

### Pig: Disambiguate

```

cat data1:
(Nescafe,08,120)
(El_Mercurio,08,142)
(Nescafe,09,153)

cat data2:
(2000,Nescafe)
(8250,Gillette_Mach3)
(500,El_Mercurio)

A = LOAD 'data1' AS (prodName:charArray, hour:int, count:int);
B = LOAD 'data2' AS (price:int, prodName:charArray);
X = JOIN A BY prodName, B BY prodName;

DUMP X;
(El_Mercurio,08,142,500,El_Mercurio)
(Nescafe,08,120,2000,Nescafe)
(Nescafe,09,153,2000,Nescafe)

Y = FOREACH X GENERATE prodName;
Z = FOREACH X GENERATE A::prodName;
    
```

which prodName?

### Pig: Split

```

raw = LOAD 'transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
numeric = FOREACH raw GENERATE cust item time org.udf.RemoveDollarSign(price) AS price;
SPLIT numeric INTO cheap IF price<1000, premium IF price>=1000;
    
```

#### numeric:

cust	item	time	price
customer412	1L_Leche	2014-03-31T08:47:57Z	900
customer412	Nescafe	2014-03-31T08:47:57Z	2.000
customer412	Nescafe	2014-03-31T08:47:57Z	2.000
customer413	400g_Zanahoria	2014-03-31T08:48:03Z	1.240
...	...	...	...

#### cheap:

cust	item	time	price
customer412	1L_Leche	2014-03-31T08:47:57Z	900
...	...	...	...

#### premium:

cust	item	time	price
customer412	Nescafe	2014-03-31T08:47:57Z	2.000
customer412	Nescafe	2014-03-31T08:47:57Z	2.000
customer413	400g_Zanahoria	2014-03-31T08:48:03Z	1.240
...	...	...	...

### Pig: Other Operators

- **FILTER**: Filter tuples by an expression
- **LIMIT**: Only return a certain number of tuples
- **MAPREDUCE**: Run a native Hadoop .jar
- **ORDER BY**: Sort tuples
- **SAMPLE**: Sample tuples
- **UNION**: Concatenate two relations

JUST TO MENTION ...

### Apache Hive

- SQL-style language that compiles into MapReduce jobs in Hadoop
- Similar to Apache Pig but ...
  - Pig more procedural whilst Hive more declarative



RECAP ...





### Apache Pig (Latin)

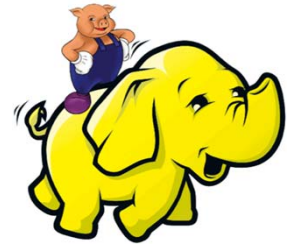
- Allows for **scripting MapReduce jobs**:
- **Procedural**, but **makes use of relational algebra**
- **Three ways to run:**
  1. Interactive command line
  2. Batch script
  3. Call from Java

### Apache Pig (Latin)

- Schema based on relations:
  - A bit like databases
- Some basic programming operators:
  - arithmetic, boolean, conditions, casting
- Some relational algebra:
  - joins, groups, count, avg, sum, filter, limit, order by, etc.
- For everything else, there's user-defined functions

### More reading

[https://pig.apache.org/docs/r0.7.0/piglatin\\_ref2.html](https://pig.apache.org/docs/r0.7.0/piglatin_ref2.html)



### Questions

