An Easier Way?

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
 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';
```

Apache Hadoop (Java)

2. Map

(Writable<key,value>)

4. Shuffle

6. Reduce

7. Output / Input (Java)
Pig: Local Mode vs. MapReduce Mode

Three Ways to Execute Pig: (i) Grunt

Three Ways to Execute Pig: (ii) Script

Three Ways to Execute Pig: (iii) Embedded

APACHE PIG: LIDER EXAMPLE

Pig: Products by Hour

transact.txt

<table>
<thead>
<tr>
<th>Customer</th>
<th>Product</th>
<th>Date</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer412</td>
<td>1L_leche</td>
<td>2014-03-31 08:47:57</td>
<td>$900</td>
</tr>
<tr>
<td>customer412</td>
<td>Nescafe</td>
<td>2014-03-31 08:47:57</td>
<td>$2,000</td>
</tr>
<tr>
<td>customer412</td>
<td>400g_zanahoria</td>
<td>2014-03-31 08:48:03</td>
<td>$1,240</td>
</tr>
<tr>
<td>customer413</td>
<td>El_mercurio</td>
<td>2014-03-31 08:48:03</td>
<td>$500</td>
</tr>
<tr>
<td>customer413</td>
<td>Gillette_mach3</td>
<td>2014-03-31 08:48:03</td>
<td>$8,250</td>
</tr>
<tr>
<td>customer413</td>
<td>Santo_domingo</td>
<td>2014-03-31 08:48:03</td>
<td>$2,450</td>
</tr>
<tr>
<td>customer413</td>
<td>Nescafe</td>
<td>2014-03-31 08:48:03</td>
<td>$2,000</td>
</tr>
<tr>
<td>customer414</td>
<td>300g_frutillas</td>
<td>2014-03-31 08:48:24</td>
<td>$9,230</td>
</tr>
<tr>
<td>customer414</td>
<td>Chocolates</td>
<td>2014-03-31 08:48:24</td>
<td>$7,000</td>
</tr>
<tr>
<td>customer414</td>
<td>Rosas</td>
<td>2014-03-31 08:48:24</td>
<td>$2,200</td>
</tr>
</tbody>
</table>

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

View data as a (streaming) relation with fields (cust, item, time, price) ...
Pig: Products by Hour

```pig
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);

<table>
<thead>
<tr>
<th>item, hour</th>
<th>cust</th>
<th>item</th>
<th>hour</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Nescafe,08]</td>
<td>customer412</td>
<td>Nescafe</td>
<td>08</td>
<td>$2.000</td>
</tr>
<tr>
<td>[Nescafe,08]</td>
<td>customer413</td>
<td>Nescafe</td>
<td>08</td>
<td>$2.000</td>
</tr>
<tr>
<td>[Nescafe,08]</td>
<td>customer415</td>
<td>Nescafe</td>
<td>08</td>
<td>$2.000</td>
</tr>
<tr>
<td>[400g_Zanahoria,08]</td>
<td>customer413</td>
<td>400g_Zanahoria</td>
<td>08</td>
<td>$1.240</td>
</tr>
</tbody>
</table>
```

Pig: Products by Hour

```pig
grunt> hrItemCntSorted = hrItemCnt;

<table>
<thead>
<tr>
<th>item, hour</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Nescafe,08]</td>
<td>3</td>
</tr>
<tr>
<td>[400g_Zanahoria,08]</td>
<td>1</td>
</tr>
</tbody>
</table>
```

Pig: Products by Hour

```pig
grunt> hrItemCntSorted = PRECINCT hrItemCntSorted BY (item, hour);

<table>
<thead>
<tr>
<th>item, hour</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Nescafe,08]</td>
<td>3</td>
</tr>
<tr>
<td>[400g_Zanahoria,08]</td>
<td>1</td>
</tr>
</tbody>
</table>
```

Pig: Products by Hour

```pig
grunt> hrItemCntSorted = ORDER hrItemCntSorted BY (item, hour), (hour, count DESC);

<table>
<thead>
<tr>
<th>item, hour</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Nescafe,08]</td>
<td>3</td>
</tr>
<tr>
<td>[400g_Zanahoria,08]</td>
<td>1</td>
</tr>
</tbody>
</table>
```
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 the same type
  - Relations are by default unordered
- **Pig Schema**: Names for fields, etc.
  - `AS (cust, item, time, price);`

<table>
<thead>
<tr>
<th>cust</th>
<th>item</th>
<th>time</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer412</td>
<td>1L_Leche</td>
<td>2014-03-31T08:47:57Z</td>
<td>$900</td>
</tr>
<tr>
<td>customer412</td>
<td>Nescafe</td>
<td>2014-03-31T08:47:57Z</td>
<td>$2,000</td>
</tr>
<tr>
<td>customer413</td>
<td>400g_Zanahoria</td>
<td>2014-03-31T08:48:03Z</td>
<td>$1,240</td>
</tr>
</tbody>
</table>

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);`

<table>
<thead>
<tr>
<th>cust</th>
<th>item</th>
<th>time</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer412</td>
<td>1L_Leche</td>
<td>2014-03-31T08:47:57Z</td>
<td>$900</td>
</tr>
<tr>
<td>customer412</td>
<td>Nescafe</td>
<td>2014-03-31T08:47:57Z</td>
<td>$2,000</td>
</tr>
<tr>
<td>customer413</td>
<td>400g_Zanahoria</td>
<td>2014-03-31T08:48:03Z</td>
<td>$1,240</td>
</tr>
</tbody>
</table>

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), byarray (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 (t1:tuple(t1a:int, t1b:int, t1c:int), t2:tuple(t2a:int, t2b:int, t2c:int));
DUMP A;
```

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;
```

```
t1 t2
3 4 5 6
4 5 6 7
2 5 8 9
```

A:
### Pig Complex Types: Tuple

**Load Data**

```
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))
```

**Generate X**

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

**Dump X**

```
(3,4)
(1,3)
(2,9)
```

### Pig Complex Types: Bag

**Load Data**

```
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) })
```

### Pig Complex Types: Map

**Load Prices**

```
A = LOAD 'prices' AS (M:map);[
  {Nescafe="#$2.000"},
  {Gillette_Mach3="#$8.250"},
]
```

### Pig Complex Types: Summary

- **Tuple**: A row in a table / a list of fields
  - e.g., (customer=412, Nescafe, 08, $2.000)

- **Bag**: A set of tuples (allows duplicates)
  - e.g., { (cust=412, Nescafe, 08, $2.000), (cust=413, Gillette_Mach3, 08, $8.250) }

- **Map**: A set of key–value pairs
  - e.g., [Nescafe#$2.000]
Pig Atomic Operators
- Comparison
  \(==, \neq, <, >, \leq, \geq\), matches (regex)
- Arithmetic
  ++, --, *, /
- Reference
  tuple.field, map#value
- Boolean
  AND, OR, NOT
- Casting

Pig Conditionals
- Ternary operator:
  \[7 \text{pm} \text{ or } 7 \text{am}
- Cases:
  \[
  x \begin{cases} \text{FOREACH } \text{GENERATE} & \text{hour}\%12, \text{"pm"}, \text{"am"}; \\
  \text{CASE} &  \text{WHEN} \text{hour}>12 & \text{"pm"}, \\
  \text{ELSE} & \text{"am"}; \\
  \text{END} \\
  \end{cases}
  \]

Pig Aggregate Operators
- Grouping:
  \[\text{GROUP: group on a single relation} \]
  \[\text{GROUP: group multiple relations} \]
  \[\text{COGROUP: COGROUP considered more readable for multiple items} \]
- Aggregate Operations:
  \[\text{AVG, MIN, MAX, SUM, COUNT, SIZE, CONCAT} \]

Pig Joins
- Inner join: As shown (default)
- Self join: Copy an alias and join with that
- Outer joins:
  \[\text{LEFT} / \text{RIGHT} / \text{FULL} \]
- Cross product:
  \[\text{CROSS} \]

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

Pig: Split

```
cat = LOAD "transact.txt" USING PigStorage('	') AS (cust, item, time, price);
numeric = FOREACH cat GENERATE cust item org.udf.RemoveDollarSign(price) AS price;
numeric::price

<table>
<thead>
<tr>
<th>cust</th>
<th>item</th>
<th>time</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer412</td>
<td>1L_Leche</td>
<td>2014-03-31T08:47:57Z</td>
<td>900</td>
</tr>
<tr>
<td>customer413</td>
<td>400g_Zanahoria</td>
<td>2014-03-31T08:48:03Z</td>
<td>1240</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
```

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

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 ...

JUST TO MENTION ...
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

Questions

?