

# Big Data and Cloud Computing

## Ejecución paralela de flujos de datos en Hadoop: Pig y Pig Latin

# Sobre el uso de esta presentación

- En algunas páginas de esta presentación, aparece el texto `<username>`, que debe ser reemplazado por el nombre de usuario de cada usuario, a saber, `mdat` seguido del DNI del usuario.

# ¿Qué es Pig?

- Es un lenguaje para flujos de datos.
- Permite al usuario especificar cómo leer cada flujo de datos, cómo procesar cada uno y cómo almacenarlos (separada o conjuntamente).
- Es útil para:
  - ETL (*Extract, Transform, Load*) flujos de datos
  - analizar datos en bruto, y
  - procesamiento iterativo.

# ¿Qué es Pig?

- Se basa en cuatro claves:
  - *Pigs eat anything* (puede trabajar con cualquier flujo de datos -relacional, anidado o no estructurado-)
  - *Pigs live anywhere* (se ha implementado para Hadoop pero no funciona exclusivamente sobre él)
  - *Pigs are domestic animals* (controlable y modificable por el usuario)
  - *Pigs fly* (es rápido)

# Wordcount con MapReduce

```
package org.myorg;

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordCount {

    public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                context.write(word, one);
            }
        }
    }
}
```

# Wordcount con MapReduce

```
public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {  
  
    public void reduce(Text key, Iterable<IntWritable> values, Context context)  
        throws IOException, InterruptedException {  
        int sum = 0;  
        for (IntWritable val : values) {  
            sum += val.get();  
        }  
        context.write(key, new IntWritable(sum));  
    }  
}
```

```
public static void main(String[] args) throws Exception {  
    Configuration conf = new Configuration();  
  
    Job job = new Job(conf, "wordcount");  
  
    job.setOutputKeyClass(Text.class);  
    job.setOutputValueClass(IntWritable.class);  
  
    job.setMapperClass(Map.class);  
    job.setReducerClass(Reduce.class);  
  
    job.setInputFormatClass(TextInputFormat.class);  
    job.setOutputFormatClass(TextOutputFormat.class);  
  
    FileInputFormat.addInputPath(job, new Path(args[0]));  
    FileOutputFormat.setOutputPath(job, new Path(args[1]));  
  
    job.waitForCompletion(true);  
}
```

# Wordcount con Pig

```
a = load '/user/hue/word_count_text.txt';  
b = foreach a generate  
    flatten(TOKENIZE((chararray)$0)) as  
    word;  
c = group b by word;  
d = foreach c generate COUNT(b), group;  
store d into '/user/hue/pig_wordcount';
```

# Algunos *tips* sobre la instalación

- Máquinas virtuales pre-configuradas con Hadoop y Pig: <http://cloudera.com>
- Incluidos en las distros de Linux o en repos (p. e. Cloudera)



# Para acceder a Pig en consola

- Conecta a la máquina para la parte práctica:  
`ssh mdat<DNI>@hadoop.ugr.es`
- Crear los directorios de trabajo de entrada y de salida:  
`mkdir -p pig/input`  
`mkdir pig/pigResults`
- Copiar el fichero de datos de entrada al directorio de entrada:  
`cp /var/tmp/materialPig/airQualityEs.csv`  
`pig/input`
- Comprobar que la copia se ha realizado correctamente:  
`ls pig/input`

# Para acceder a Pig en consola

- Cambiar al directorio de trabajo:

```
cd ~/pig
```

- Entrar en Pig en modo local:

```
pig -x local
```

# Modelo de datos de Pig: Tipos

- Escalares:
  - `int`
  - `long`
  - `float`
  - `double`
  - `chararray`
  - `bytearray`

# Modelo de datos de Pig: Tipos

- Tipos complejos:
  - map: estructura de pares (chararray, tipo)
  - tuple: secuencia fija y ordenada de elementos de Pig  
(`<valor 1>`, `<valor 2>`)
  - bag: colección desordenada de tuplas  
{(`<valor 1>`, `<valor 2>`), ...}
  - null: valor desconocido o no aplicable en cualquier tipo de datos escalar o complejo

# Modelo de datos de Pig: esquemas

- *Pigs eat anything*
- Si hay esquema, se le especifica durante la carga de datos:

```
measure = load 'input/airQualityEs.csv' using  
  PigStorage(';') AS (date:chararray,  
    co:float, no:float, no2:float, o3:float,  
    pm10:float, sh2:float, pm25:float,  
    pst:float, so2:float, province:chararray,  
    station:chararray);
```

# Modelo de datos de Pig: esquemas

- Si no se especifica el tipo de datos, Pig escoge `bytearray`.
- Pig es capaz de cargar el esquema desde fuentes externas (base de datos Hcatalog, ficheros JSON, ...)
- Permite forzar tipos (*casts*)

# El lenguaje de consulta Pig Latin

- No sensible a casos en las palabras reservadas.
- Comentarios:
  - En línea: --
  - Varias líneas: /\* ... \*/

# El lenguaje de consulta Pig Latin

- Carga de flujo:

```
grunt> measure = load
    'input/airQualityEs.csv' using
    PigStorage(';') AS (date:chararray,
    co:float, no:float, no2:float, o3:float,
    pm10:float, sh2:float, pm25:float,
    pst:float, so2:float, province:chararray,
    station:chararray);
```



# El lenguaje de consulta Pig Latin

- Volcado de resultados:

```
grunt> store measure into  
      'pigResults/AirQualityProcessed';
```

- Comprueba el resultado (<espacio> para avanzar y <q> para salir) desde la ventana de la *shell* de sistema operativo:

```
$ less pigResults/AirQualityProcessed/part-m-  
00000
```

# El lenguaje de consulta Pig Latin

- Volcado de resultados con separadores:

```
grunt> store measure into  
      'pigResults/AirQualityProcessed2' using  
      PigStorage (',');
```

- Comprueba el resultado desde la ventana de *shell* del sistema operativo:

```
$ less pigResults/AirQualityProcessed2/part-  
m-00000
```

# El lenguaje de consulta Pig Latin

- Muestra de datos en pantalla:

```
grunt> dump measure;
```

# El lenguaje de consulta Pig Latin

- Operador relacional foreach:

```
<relación> = foreach <relación> generate  
    <expresión>
```

donde <expresión> puede ser:

- <proyección en tipo simple>[, <expresión>]
- ...

# El lenguaje de consulta Pig Latin: foreach

donde <proyección en tipo simple> es:

- \*
- <campo>{, <campo>}\*
- ..o3
- co..o3
- co...

# El lenguaje de consulta Pig Latin: foreach

...

```
grunt> Localizacion = foreach measure  
      generate province, station;  
grunt> dump Localizacion;
```

- ... 0:

...

```
grunt> Localizacion = foreach measure  
      generate $10, $11;  
grunt> dump Localizacion;
```

# El lenguaje de consulta Pig Latin: foreach

donde <expresión> puede ser:

- <proyección en tipo complejo>[, <expresión>]
- ...

# El lenguaje de consulta Pig Latin: foreach

donde <proyección en tipo complejo> es:

- <map># '<nombre de la clave>'
- <tupla>.<campo>
- <bag>.<campo> (esto crea un nuevo bag con un único campo)
- <bag>.( <campo>, <campo>, ... ) (esto crea un nuevo bag con varios campos)



# El lenguaje de consulta Pig Latin: `foreach`

donde `<expresión>` puede ser:

- `<función>[ , <expresión>]`
- ...

# El lenguaje de consulta Pig Latin: foreach

Eval	Load/Store	Math	String	Bag and Tuple
AVG	Handling Compression	ABS	INDEXOF	TOBAG
CONCAT	BinStorage	ACOS, ASIN, ATAN	LAST_INDEX_OF	TOP
COUNT	PigDump	CBRT	LCFIRST	TOTUPLE
COUNT_STAR	PigStorage	CEIL, FLOOR, ROUND	LOWER, UPPER	
DIFF	TextLoader	COS, SIN, TAN	REGEX_EXTRACT	
IsEmpty		COSH, SINH, TANH	REGEX_EXTRACT _ALL	
MAX		EXP	REPLACE	
MIN		LOG	STRSPLIT	
SIZE		LOG10	SUBSTRING	
SUM		RANDOM	TRIM	
TOKENIZE		SQRT	UCFIRST	

# El lenguaje de consulta Pig Latin: foreach

- Funciones en biblioteca externa:

- Java:

```
register '<path>/piggybank.jar';
```

```
...
```

```
B = foreach A generate  
    org.apache.pig.piggybank.evaluation.string.Reverse (A.y);
```

- Python:

```
register 'eudfs.py' using jython as eudfs;
```

```
...
```

```
B = foreach A generate eudfs.myfunction  
    (A.y);
```

# El lenguaje de consulta Pig Latin: `foreach`

- Referencia a una expresión en una lista:
  - Por posición: `$0`, `$1`, ...
  - Por nombre: `<expresión> as <nombre>`

# El lenguaje de consulta Pig Latin

- Operador relacional `filter`:

`<relación> = filter <relación> by <condición>`

donde `<condición>` puede ser:

- `<expresión> matches <patrón> [and <condición>]`
- `<expresión> matches <patrón> [or <condición>]`
- ...

donde `<patrón>` puede ser un literal de chararray que incluya comodines como `*`.

# El lenguaje de consulta Pig Latin: `filter`

donde `<condición>` puede ser:

- `<expresión> <operador> <expresión> [and <condición>]`
- `<expresión> <operador> <expresión> [or <condición>]`
- ...

donde `<operador>` puede ser: `==`, `!=`, `<`, `<=`, `>`, `>=`.

# El lenguaje de consulta Pig Latin: `filter`

donde `<condición>` puede ser:

- `<llamada a función de filtrado> [and <condición>]`
- `<llamada a función de filtrado> [and <condición>]`

donde `<operador>` puede ser: `==`, `!=`, `<`, `<=`, `>`, `>=`.

# El lenguaje de consulta Pig Latin

- Operador relacional group:

```
<relación> = group <relación> by  
    <agrupamiento>;
```

donde <agrupamiento> puede ser:

- <campo>
- (<campo>{, <campo>}+)

La relación resultante tiene dos campos, un campo clave (con los valores concretos de los campos del agrupamiento) y un bag con todos los registros de <relación> con esos valores concretos.



# Un ejemplo

```
grunt> measure = load 'input/airQualityEs.csv' using
  PigStorage(';') AS (date:chararray, co:float,
    no:float, no2:float, o3:float, pm10:float,
    sh2:float, pm25:float, pst:float, so2:float,
    province:chararray, station:chararray);

grunt> filter_measure = filter measure by date !=
  'DIA';

grunt> measure_by_province = group filter_measure by
  province;

grunt> num_measures_by_province = foreach
  measure_by_province generate group,
  AVG(filter_measure.co) as measure;

grunt> store num_measures_by_province into
  'pigResults/AirQualityProcessed3';
```

# El lenguaje de consulta Pig Latin

- Comprueba el resultado desde la ventana de *shell* del sistema operativo:

```
$ less pigResults/AirQualityProcessed3/part-  
r-00000
```

# El lenguaje de consulta Pig Latin

- Operador relacional order:

`<relación> = order <relación> by <criterio>;`

donde `<criterio>` puede ser:

- `<campo> [desc]{, <campo> [desc]}*`

Ordena toda los datos de la relación, incluso si está particionada. No se pueden ordenar tipos complejos.

# El lenguaje de consulta Pig Latin

- Operador relacional `distinct`:

`<relación> = distinct <relación>;`

elimina registros completos repetidos.

# El lenguaje de consulta Pig Latin

- Operador relacional join:

```
<relación> = join <relación> by <lista  
campos> [{left | right | full} outer],  
<relación> by <lista campos>;
```

que crea una relación con registros formados por parejas de registros de las dos relaciones con valores iguales para los campos de la lista de campos.

# El lenguaje de consulta Pig Latin: `join`

- Varias tablas (sólo *inner*):

```
<relación> = join <relación> by <lista  
campos>{, <relación> by <lista campos>}+;
```

que crea una relación con registros formados por combinaciones de registros de las relaciones involucradas con valores iguales para los campos de las correspondientes listas de campos.

# El lenguaje de consulta Pig Latin

- Operador relacional `limit`:

```
<relación> = limit <relación> <int>;
```

que crea una relación con el número de registros especificado.

# El lenguaje de consulta Pig Latin

- Operador relacional `sample`:

`<relación> = sample <relación> <real entre 0 y 1>;`

que crea una relación que incluye una muestra que contiene un `<real entre 0 y 1> * 100` por ciento de los registros.



# El lenguaje de consulta Pig Latin

- Operadores que fuerzan una fase *Reduce*:
  - group
  - order
  - distinct
  - join
  - limit
  - **cogroup**
  - **cross**

# El lenguaje de consulta Pig Latin

- Los operadores que fuerzan una fase *Reduce* pueden ir acompañados de la cláusula:

```
... parallel <int>
```

que fuerza a usar un número específico de *Reducers*.

Se puede usar un valor por defecto con:

```
set default_parallel <int>;
```

aunque puede cambiarse localmente con la cláusula `parallel`.

# El lenguaje de consulta Pig Latin avanzado

## Operador `flatten`:

- Aparece en la expresión de una proyección `foreach` como una función que se aplica a un campo de tipo `bag` o tupla:

```
B = foreach A generate A.x, flatten (A.y);
```

# El lenguaje de consulta Pig Latin avanzado

## Operador cogroup:

- Genera grupos a partir de datos de distintas fuentes:

`C = cogroup A by x, B by y;`

La relación resultante tiene tres campos, un campo clave (con los valores concretos de los campos del agrupamiento), un bag con todos los registros de A y un bag con todos los registros de B con esos valores concretos para la clave.

# El lenguaje de consulta Pig Latin avanzado

Operador `union`:

- Une varios conjuntos de datos en uno:

```
C = union A, B;
```

Puede ocurrir:

- A y B tienen el mismo esquema, C tiene el mismo esquema.
- A y B tienen esquemas convertibles por *casting*, C tiene el esquema del *casting*.
- A y B no tienen el mismo esquema, C no tiene esquema.

# El lenguaje de consulta Pig Latin avanzado

Operador `cross`:

```
C = cross A, B;
```

Genera una relación que combina cada registro de A con cada registro de B.

# Bibliografía

- <http://shop.oreilly.com/product/0636920018087.do>
- <http://pig.apache.org/>
- [https://pig.apache.org/docs/r0.7.0/piglatin\\_ref2.html](https://pig.apache.org/docs/r0.7.0/piglatin_ref2.html)