

CC5212-1

PROCESAMIENTO MASIVO DE DATOS
OTOÑO 2017

Lecture 5: Hadoop III / PIG

Aidan Hogan
aidhog@gmail.com

HADOOP: WRAPPING UP

0. Reading/Writing to HDFS

```
public class HDFSHelloWorld {  
  
    public static final String theFilename = "hello.txt";  
    public static final String message = "Hello, world!\n";  
  
    public static void main (String [] args) throws IOException {  
  
        Configuration conf = new Configuration();  
        FileSystem fs = FileSystem.get(conf);  
  
        Path filenamePath = new Path(theFilename);  
  
        try {  
            if (fs.exists(filenamePath)) {  
                // remove the file first  
                fs.delete(filenamePath, false);  
            }  
  
            FSDatOutputStream out = fs.create(filenamePath);  
            out.writeUTF(message);  
            out.close();  
  
            FSDatInputStream in = fs.open(filenamePath);  
            String messageIn = in.readUTF();  
            System.out.print(messageIn);  
            in.close();  
        } catch (IOException ioe) {  
            System.err.println("IOException during operation: " + ioe.toString());  
            System.exit(1);  
        }  
    }  
}
```

Creates a file system for default configuration

Check if the file exists; if so delete

Create file and write a message

Open and read back

Hadoop: Citation Example

cited	citing
Why Eppley and Hannah's Experiment Isn't	$10 = 6 + 4$
$10 = 6 + 4$	A Fly in the SOUP
Decapitating Tadpoles	A Fly in the SOUP
The axis of evil	Local Pancake Defeats Axis of Evil
Waking the Colored Plasma	Nuttier Bubbles
$10 = 6 + 4$	Nutty Bubbles
Brane New World	Nutty Bubbles
Nuttier Bubbles	Nutty Bubbles
Deconstructing Noncommutativity with a Giant Fuzzy Moose	The axis of evil
The axis of evil	The axis of evil
Brane New World	Unhiggsing the del Pezzo
Higgs Pain? Take a Preon!	Unhiggsing the del Pezzo
...	...

1. $10 = 6 + 4$

Author: Frank Tony Smith

arXiv: hep-th/9908205

Go!



Compute total citations per paper?

2. $\log(M_{\text{Pl}}/m_{3/2})$

Authors: Oscar Loaiza-Brito, Johannes Martin, Hans Peter Nilles, Michael Ratz

Recent Comments

paper	cites
$10 = 6 + 4$	2
Brane New World	2
...	...

1. Input (Java)

InputFormat:	Description:	Key:	Value:
TextInputFormat	Default format; reads lines of text files	The byte offset of the line	The line contents
KeyValueInputFormat	Parses lines into key, val pairs	Everything up to the first tab character	The remainder of the line
SequenceFileInputFormat	A Hadoop-specific high-performance binary format	user-defined	user-defined

2. Map

Mapper<InputKey, InputValue, MapKey, MapValue>

```
public static class CitationCountMapper extends Mapper<Object, Text, Text, IntWritable>{  
  
    private final IntWritable one = new IntWritable(1);  
    private Text paperTitle = new Text();  
  
    /**  
     * @throws InterruptedException  
     *  
     */  
    @Override  
    public void map(Object key, Text value, Context output)  
        throws IOException, InterruptedException {  
        String line = value.toString();  
        String[] paperCitedByPaper = line.split(SPLIT_REGEX);  
        paperTitle.set(paperCitedByPaper[0]);  
        output.write(paperTitle, one);  
    }  
}
```

(input) key: file offset.
(input) value: line of the file.
output: for output and logging.

Emit output

(Writable for values)

```
package ejemplo;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

import org.apache.hadoop.io.Writable;

public class WritableCitation implements Writable {
    public String citingPaper;
    public String citingVenue;
    public int mentions;

    public WritableCitation(String citingPaper, String citingVenue, int mentions) {
        this.citingPaper = citingPaper;
        this.citingVenue = citingVenue;
        this.mentions = mentions;
    }

    public void write(DataOutput out) throws IOException {
        out.writeUTF(citingPaper);
        out.writeUTF(citingVenue);
        out.writeInt(mentions);
    }

    public void readFields(DataInput in) throws IOException {
        citingPaper = in.readUTF();
        citingVenue = in.readUTF();
        mentions = in.readInt();
    }

    public String toString() {
        return citingPaper + "\t" + citingVenue + "\t" + mentions;
    }
}
```

Same order

(not needed in the running example)

(WritableComparable for keys/values)

```
public class WritableComparableCitation implements WritableComparable<WritableComparableCitation> {  
    public String citingPaper;  
    public String citingVenue;  
    public int mentions;  
  
    public WritableComparableCitation(String citingPaper, String citingVenue, int mentions) {}  
    public void write(DataOutput out) throws IOException {}  
    public void readFields(DataInput in) throws IOException {}  
    public String toString() {}  
  
    public int compareTo(WritableComparableCitation other) {  
        int comp = citingPaper.compareTo(other.citingPaper);  
        if(comp==0){  
            comp = citingVenue.compareTo(other.citingVenue);  
            if(comp == 0){  
                comp = Integer.compare(mentions, other.mentions);  
            }  
        }  
        return comp;  
    }  
  
    public boolean equals(Object o) {  
        if(o==null) return false;  
        if(o==this) return true;  
        if (!(o instanceof WritableComparableCitation)) return false;  
        WritableComparableCitation wcp = (WritableComparableCitation)o;  
        return citingPaper.equals(wcp.citingPaper) && this.citingVenue.equals(wcp.citingVenue)  
            && this.mentions == wcp.mentions;  
    }  
  
    public int hashCode() {  
        return citingPaper.hashCode() ^ citingVenue.hashCode() ^ mentions;  
    }  
}
```

New Interface

Same as before

Needed to sort keys

Needed for default partition function

(not needed in the running example)

3. Partition

Partitioner interface

```
package ejemplo;  
  
import org.apache.hadoop.mapred.JobConf;  
  
public class PartitionCites<E> implements Partitioner<WritableComparableCitation, E> {  
  
    @Override  
    public int getPartition(WritableComparableCitation key, E val, int machines) {  
        return Math.abs(key.hashCode() % machines);  
    }  
}
```

(This happens to be the default partition method, but we can put anything we like)

(not needed in the running example)

4. Shuffle



5. Sort/Comparison

```
public class WritableComparableCitation implements WritableComparable<WritableComparableCitation> {  
    public String citingPaper;  
    public String citingVenue;  
    public int mentions;  
  
    public WritableComparableCitation(String citingPaper, String citingVenue, int mentions) {}  
    public void write(DataOutput out) throws IOException {}  
    public void readFields(DataInput in) throws IOException {}  
    public String toString() {}  
  
    public int compareTo(WritableComparableCitation other) {  
        int comp = citingPaper.compareTo(other.citingPaper);  
        if(comp==0){  
            comp = citingVenue.compareTo(other.citingVenue);  
            if(comp == 0){  
                comp = Integer.compare(mentions, other.mentions);  
            }  
        }  
        return comp;  
    }  
  
    public boolean equals(Object o) {  
        if(o==null) return false;  
        if(o==this) return true;  
        if (!(o instanceof WritableComparableCitation)) return false;  
        WritableComparableCitation wcp = (WritableComparableCitation)o;  
        return citingPaper.equals(wcp.citingPaper) && this.citingVenue.equals(wcp.citingVenue)  
            && this.mentions == wcp.mentions;  
    }  
  
    public int hashCode() {  
        return citingPaper.hashCode() ^ citingVenue.hashCode() ^ mentions;  
    }  
}
```



Methods in
WritableComparable

(not needed in the
running example)

6. Reduce

Reducer<MapKey, MapValue, OutputKey, OutputValue>

```
public static class CitationCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {  
  
    /**  
     * @throws InterruptedException  
     */  
    @Override  
    public void reduce(Text key, Iterable<IntWritable> values,  
                      Context output) throws IOException, InterruptedException {  
        int sum = 0;  
        for(IntWritable value: values) {  
            sum += value.get();  
        }  
        output.getCounter("citations", key.toString().substring(0, 1)).increment(1);  
        output.write(key, new IntWritable(sum));  
    }  
}
```

key: as emitted from the previous map

values: iterator over all values for that key output

Write to output

7. Output / Input (Java)

```
public class HDFSHelloWorld {  
  
    public static final String theFilename = "hello.txt";  
    public static final String message = "Hello, world!\n";  
  
    public static void main (String [] args) throws IOException {  
  
        Configuration conf = new Configuration();  
        FileSystem fs = FileSystem.get(conf);  
  
        Path filenamePath = new Path(theFilename);  
  
        try {  
            if (fs.exists(filenamePath)) {  
                // remove the file first  
                fs.delete(filenamePath, false);  
            }  
  
            FSDatOutputStream out = fs.create(filenamePath);  
            out.writeUTF(message);  
            out.close();  
  
            FSDatInputStream in = fs.open(filenamePath);  
            String messageIn = in.readUTF();  
            System.out.print(messageIn);  
            in.close();  
        } catch (IOException ioe) {  
            System.err.println("IOException during operation: " + ioe.toString());  
            System.exit(1);  
        }  
    }  
}
```

Creates a file system for default configuration

Check if the file exists; if so delete

Create file and write a message

Open and read back

7. Output (Java)

OutputFormat:	Description
TextOutputFormat	Default; writes lines in "key \t value" form
SequenceFileOutputFormat	Writes binary files suitable for reading into subsequent MapReduce jobs
NullOutputFormat	Disregards its inputs

Control Flow

```
public static void main(String[] args) throws Exception {  
    Configuration conf = new Configuration();  
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainderArgs();  
    if (otherArgs.length != 2) {  
        System.err.println("Usage: CitationCount <in> <out>");  
        System.exit(2);  
    }  
    String inputLocation = otherArgs[0];  
    String outputLocation = otherArgs[1];  
  
    Job job = Job.getInstance(new Configuration());
```

```
    FileInputFormat.setInputPaths(job, new Path(inputLocation));  
    FileOutputFormat.setOutputPath(job, new Path(outputLocation));
```

```
    job.setOutputKeyClass(Text.class);  
    job.setOutputValueClass(IntWritable.class);  
    job.setMapOutputKeyClass(Text.class);  
    job.setMapOutputValueClass(IntWritable.class);
```

```
    job.setMapperClass(CitationCountMapper.class);  
    job.setCombinerClass(CitationCountReducer.class);  
    job.setReducerClass(CitationCountReducer.class);
```

```
    job.setJarByClass(CitationCount.class);  
    job.waitForCompletion(true);  
}
```

Create a job with default configuration (custom configs can be read from XML files on the master)

Set input and output paths

Set the type of map and output keys and values in the configuration

Set the mapper class

Set the reducer class (and optionally combiner)

Run and wait for job to complete.

More in Hadoop: Counters

```
public static class CitationCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

    /**
     * @throws InterruptedException
     */
    @Override
    public void reduce(Text key, Iterable<IntWritable> values,
                      Context output) throws IOException, InterruptedException {
        int sum = 0;
        for(IntWritable value: values) {
            sum += value.get();
        }
        output.getCounter("citations", key.toString().substring(0, 1)).increment(1);
        output.write(key, new IntWritable(sum));
    }
}
```

Context has a group of maps of counters

Hadoop: Supermarket Example

ReceiptItems		ReceiptTimes		ItemDetails		
RECEIPT ID	ITEM ID	RECEIPT ID	TIME	ITEM ID	NAME	PRICE (\$)
R1401	I306	R1403	19:00	I306	Zanahoria 500g	500
R1401	I306	R1401	18:59	I504	CocaCola 3L	1400
R1401	I504	R1402	19:01	I007	Comfort	1200
R1402	I007
R1402	I306					
R1403	I306					
R1403	I504					
...	...					

Compute total sales per hour of the day?



Output	
HOUR	TOTAL
...	...
18:00–18:59	\$2400
19:00–19:59	\$3600
...	...

More in Hadoop: Multiple Inputs

```
public class RevenuePerHour {  
    public static void main(String[] args) throws Exception {  
        Configuration conf = new Configuration();  
        String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();  
        if (otherArgs.length != 4) {  
            System.err.println("Usage: WordCount <in1> <in2> <in3> <tmp1> <tmp2> <out>");  
            System.exit(2);  
        }  
  
        Job job1 = Job.getInstance(new Configuration());  
        MultipleInputs.addInputPath(job1, new Path(otherArgs[0]),  
            TextInputFormat.class, ReceiptItemsMapper.class);  
        MultipleInputs.addInputPath(job1, new Path(otherArgs[1]),  
            TextInputFormat.class, ReceiptTimesMapper.class);  
        FileOutputFormat.setOutputPath(job1, new Path(otherArgs[3]));  
  
        job1.setReducerClass(ItemsTimesReducer.class);  
        job1.setMapOutputKeyClass(Text.class);  
        job1.setMapOutputValueClass(Text.class);  
        job1.setOutputKeyClass(Text.class);  
        job1.setOutputValueClass(Text.class);  
        job1.waitForCompletion(true);  
    }  
}
```

Multiple inputs, different map for each

One reducer

More in Hadoop: Chaining Jobs

```
public class RevenuePerHour {  
    public static void main(String[] args) throws Exception {  
        Configuration conf = new Configuration();  
        String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();  
        if (otherArgs.length != 4) {  
            System.err.println("Usage: WordCount <in1> <in2> <in3> <tmp1> <tmp2> <out>");  
            System.exit(2);  
        }  
  
        Job job1 = Job.getInstance(new Configuration());  
        MultipleInputs.addInputPath(job1, new Path(otherArgs[0]),  
            TextInputFormat.class, ReceiptItemsMapper.class);  
        MultipleInputs.addInputPath(job1, new Path(otherArgs[1]),  
            TextInputFormat.class, ReceiptTimesMapper.class);  
        FileOutputFormat.setOutputPath(job1, new Path(otherArgs[3]));
```

```
job1.setReducerClass(ItemsTimesReducer.class);  
job1.setMapOutputKeyClass(Text.class);  
job1.setMapOutputValueClass(Text.class);  
job1.setOutputKeyClass(Text.class);  
job1.setOutputValueClass(Text.class);  
job1.waitForCompletion(true);
```

Run and wait

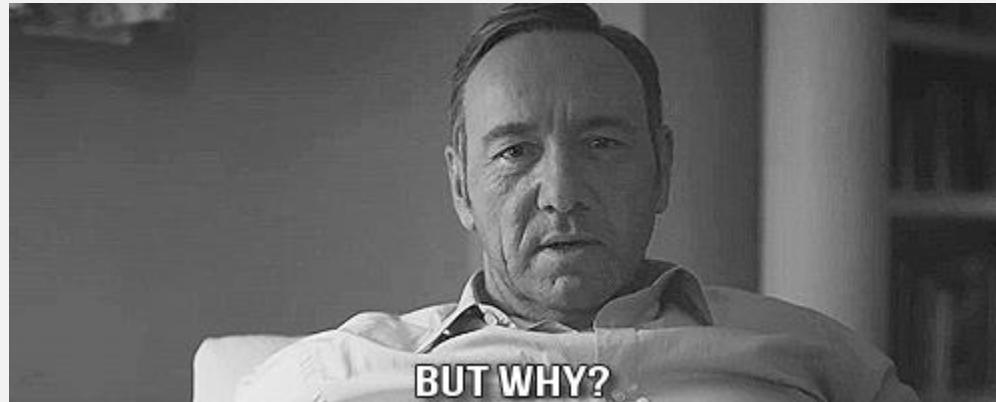
```
Job job2 = Job.getInstance(new Configuration());  
MultipleInputs.addInputPath(job2, new Path(otherArgs[2]),  
    TextInputFormat.class, ItemsMapper.class);  
MultipleInputs.addInputPath(job2, new Path(otherArgs[3]),  
    TextInputFormat.class, ItemsPricesMapper.class);  
FileOutputFormat.setOutputPath(job2, new Path(otherArgs[4]));  
  
job2.setReducerClass(TimesPricesReducer.class);  
job2.setMapOutputKeyClass(LongWritable.class);  
job2.setMapOutputValueClass(Text.class);
```

Output of Job1 set to
Input of Job2

More in Hadoop: Number of Reducers

```
job.setNumReduceTasks(1);
```

Set number of parallel reducer tasks for the job



Why would we ask for 1 reduce task?



Output requires a merge on one machine
(for example, sorting, top-k)



Hadoop: Filtered Supermarket Example

ReceiptItems		ReceiptTimes		ItemDetails		
RECEIPT ID	ITEM ID	RECEIPT ID	TIME	ITEM ID	NAME	PRICE (\$)
R1401	I306	R1403	19:00	I306	Zanahoria 500g	500
R1401	I306	R1401	18:59	I504	CocaCola 3L	1400
R1401	I504	R1402	19:01	I007	Comfort	1200
R1402	I007
R1402	I306					
R1403	I306					
R1403	I504					
...	...					

Compute total sales per hour of the day ...
but exclude certain item IDs passed as an input file?

Output	
HOUR	TOTAL
...	...
18:00–18:59	\$1400
19:00–19:59	\$2600
...	...

More in Hadoop: Distributed Cache

- Some tasks need “global knowledge”
 - Hopefully not too much though
- Use a distributed cache:
 - Makes global data available locally to all nodes
 - On the local hard-disk of each machine

How might we use this?



Make the filtered products global and read them (into memory?) when processing items



Hadoop: (ಠ_ಠ)

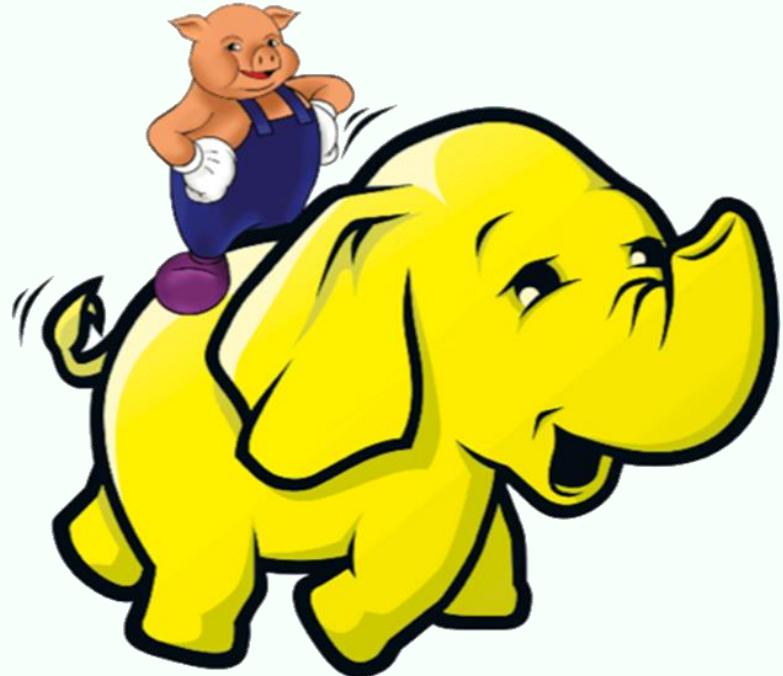
```
public class RevenuePerHour {  
    public static void main(String[] args) throws Exception {  
        Configuration conf = new Configuration();  
        String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();  
        if (otherArgs.length != 4) {  
            System.err.println("Usage: WordCount <in1> <in2> <in3> <tmp1> <tmp2> <out>");  
            System.exit(2);  
        }  
  
        Job job1 = Job.getInstance(new Configuration());  
        MultipleInputs.addInputPath(job1, new Path(otherArgs[0]),  
            TextInputFormat.class, ReceiptItemsMapper.class);  
        MultipleInputs.addInputPath(job1, new Path(otherArgs[1]),  
            TextInputFormat.class, ReceiptTimesMapper.class);  
        FileOutputFormat.setOutputPath(job1, new Path(otherArgs[3]));  
  
        job1.setReducerClass(ItemsTimesReducer.class);  
        job1.setMapOutputKeyClass(Text.class);  
        job1.setMapOutputValueClass(Text.class);  
        job1.setOutputKeyClass(Text.class);  
        job1.setOutputValueClass(Text.class);  
        job1.waitForCompletion(true);  
  
        Job job2 = Job.getInstance(new Configuration());  
        MultipleInputs.addInputPath(job2, new Path(otherArgs[2]),  
            TextInputFormat.class, ItemsTimesMapper.class);  
        MultipleInputs.addInputPath(job2, new Path(otherArgs[3]),  
            TextInputFormat.class, ItemsPricesMapper.class);  
        FileOutputFormat.setOutputPath(job2, new Path(otherArgs[4]));  
  
        job2.setReducerClass(TimesPricesReducer.class);  
        job2.setMapOutputKeyClass(LongWritable.class);  
        job2.setMapOutputValueClass(Text.class);
```



APACHE PIG: OVERVIEW

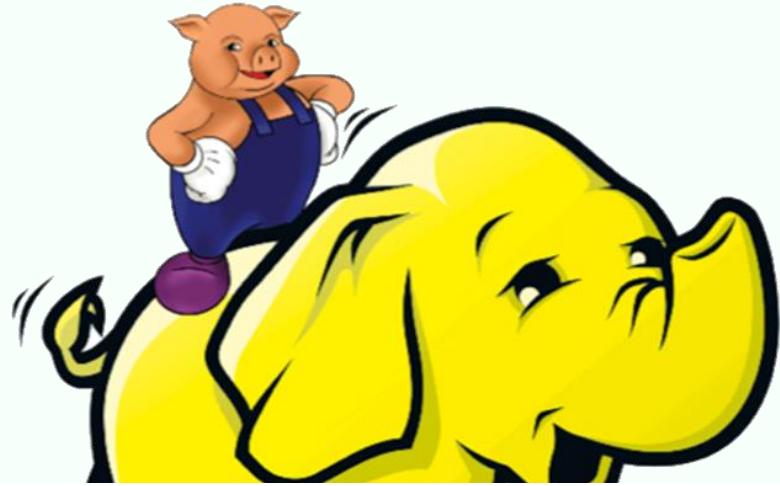
Apache Pig

- 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**



Apache Pig

- Create MapReduce programs to **run on Hadoop**
- Use a high-level “scripting” language called **Pig Latin**



- Can embed **Functions**: (or Python, Java, etc.)
- Based on F

pig Lat·in

/'pig_lətn/

noun

a made-up language formed from English by transferring the initial consonant or consonant cluster of each word to the end of the word and adding a vocalic syllable (usually 'pig', latn: so *chicken soup* would be translated to *ickenchay oupsay*). Pig Latin is typically spoken playfully, as if to convey secrecy.



Translations, word origin, and more definitions



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;
```

Map

```
-- count the entries in each group
```

```
word_count = FOREACH word_groups GENERATE COUNT(filtered_words) AS count, group AS word;
```

Reduce

```
-- order the records by count
```

```
ordered_word_count = ORDER word_count BY count DESC;
```

Map + Reduce

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

Any ideas which lines correspond to map and which to reduce?

APACHE PIG: AN 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
customer415	Nescafe	2014-03-31T08:48:35Z	\$2.000
customer415	12 Huevos	2014-03-31T08:48:35Z	\$2.200
...			

Find the number of items sold per hour of the day



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 proces 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
...

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	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	Gillette_Mach3	2014-03-31T08:48:03Z	\$8.250
...

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	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
customer413	Gillette_Mach3	2014-03-31T08:48:03Z	\$8.250
...

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
...

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;
grunt> hrItem = 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] <i>count</i>	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]	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:**

More readable!

- Reference using name

- `premium = FILTER raw BY org.udf.MinPrice1000(price);`

- ... or position

- `premium = FILTER raw BY org.udf.MinPrice1000($3);`

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
...

APACHE PIG: TYPES

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;           ← hour an integer  
C = FOREACH A GENERATE hour + 4f % 24;          ← 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;
```

A:

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

A:

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	2	5	8
	1	4	7

B:

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: UNNESTING (FLATTEN)



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';
```

Pig Complex Types: Flatten Tuples

```
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 flatten(t1), flatten(t2);
```

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

Pig Complex Types: Flatten Tuples

```
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 flatten(t1), flatten(t2);
```

```
DUMP X;
```

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

X:

t1a	t1b	t1c	t2a	t2b	t2c
3	8	9	4	5	6
1	4	7	3	7	5
2	5	8	9	5	8

Pig Complex Types: Flatten Tuples

```
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))
```

```
Y = FOREACH A GENERATE t1, flatten(t2);
```

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

Pig Complex Types: Flatten Tuples

```
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))
```

```
Y = FOREACH A GENERATE t1, flatten(t2);
```

```
DUMP Y;
```

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

Y:

t1			t2a	t2b	t2c
t1a	t1b	t1c			
3	8	9	4	5	6
1	4	7	3	7	5
2	5	8	9	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)})
```

```
C = FOREACH B GENERATE flatten(A);
```

B:

group (c1)	A		
	c1	c2	c3
3	3	8	9
2	2	3	6
1	2	5	8
	1	4	7

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)})
```

```
C = FOREACH B GENERATE flatten(A);
```

```
DUMP C;
```

```
(3,8,9)
```

```
(2,3,6)
```

```
(2,5,8)
```

```
(1,4,7)
```

C:

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

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)})
```

```
D = FOREACH B GENERATE group, flatten(A);
```

B:

group (c1)	A		
	c1	c2	c3
3	3	8	9
2	2	3	6
1	2	5	8
	1	4	7

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)})
```

```
D = FOREACH B GENERATE group, flatten(A);
```

```
DUMP D;
```

```
(3,3,8,9)
```

```
(2,2,3,6)
```

```
(2,2,5,8)
```

```
(1,1,4,7)
```

D:

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

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)  
(El_Mercurio,08,142)  
(Nescafe,09,153)
```

```
cat data2;  
(2000,Nescafe)  
(8250, Gillette_Mach3)  
(500, El_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:

```
(El_Mercurio,08,142, 500, El_Mercurio)  
(Nescafe,08,120, 2000,Nescafe)  
(Nescafe,09,153, 2000,Nescafe)
```

X:

prod	hour	count	price	name
Nescafe	08	120	2000	Nescafe
Nescafe	09	153	2000	Nescafe
El_Mercurio	08	142	500	El_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)
```

which prodName?

```
Y = FOREACH X GENERATE prodName
```

```
Y = FOREACH X GENERATE A::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:

--

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
...

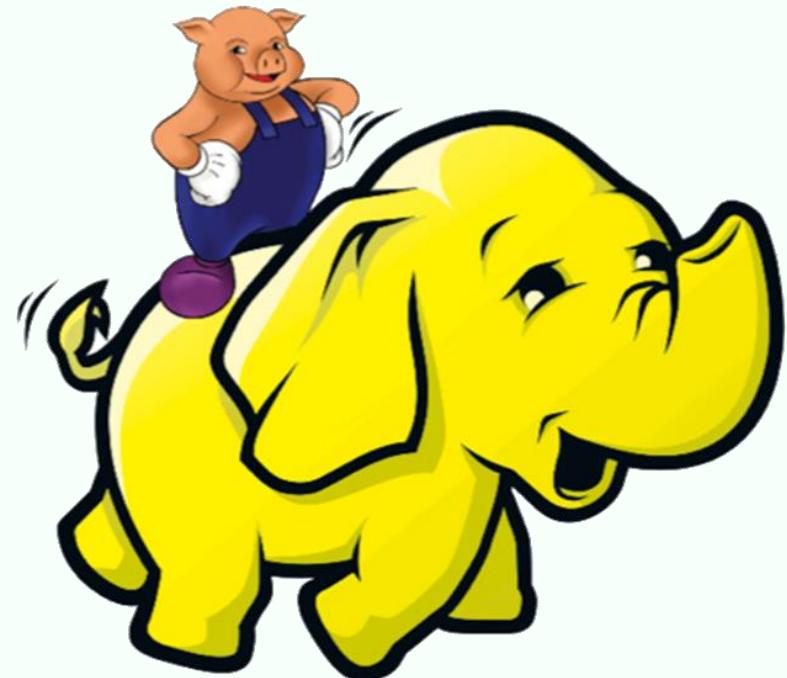
cust	item	time	price
customer412	1L_Leche	2014-03-31T08:47:57Z	900
...

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

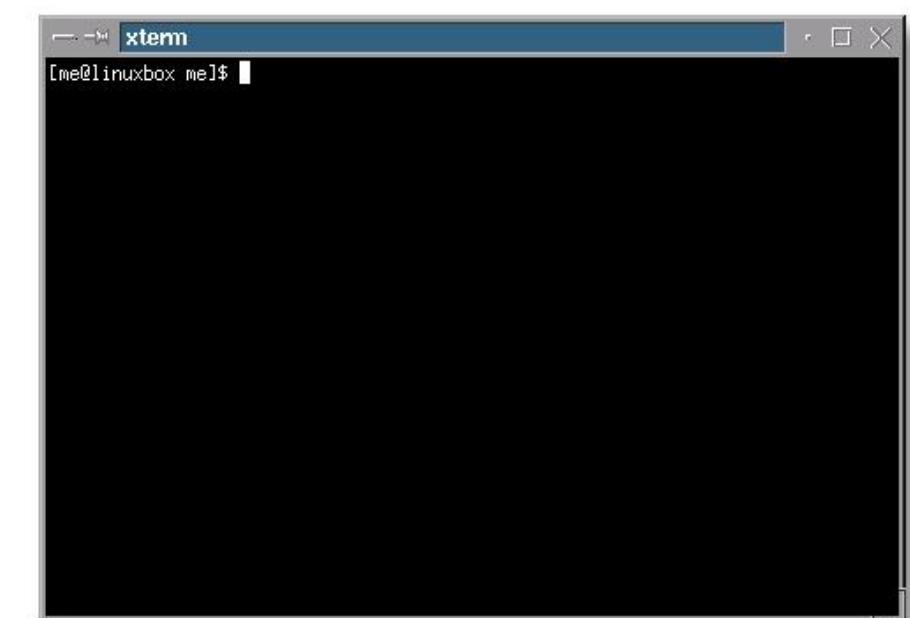
Pig translated to MapReduce in Hadoop

- Pig is only an interface/scripting language for MapReduce



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
```

wordcount.pig

```
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';
```

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();
        }
    }
}
```



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



**IN-CLASS EXERCISE:
COUNT CO-STARS WITH PIG**

RECAP ...

Apache Pig (Latin)

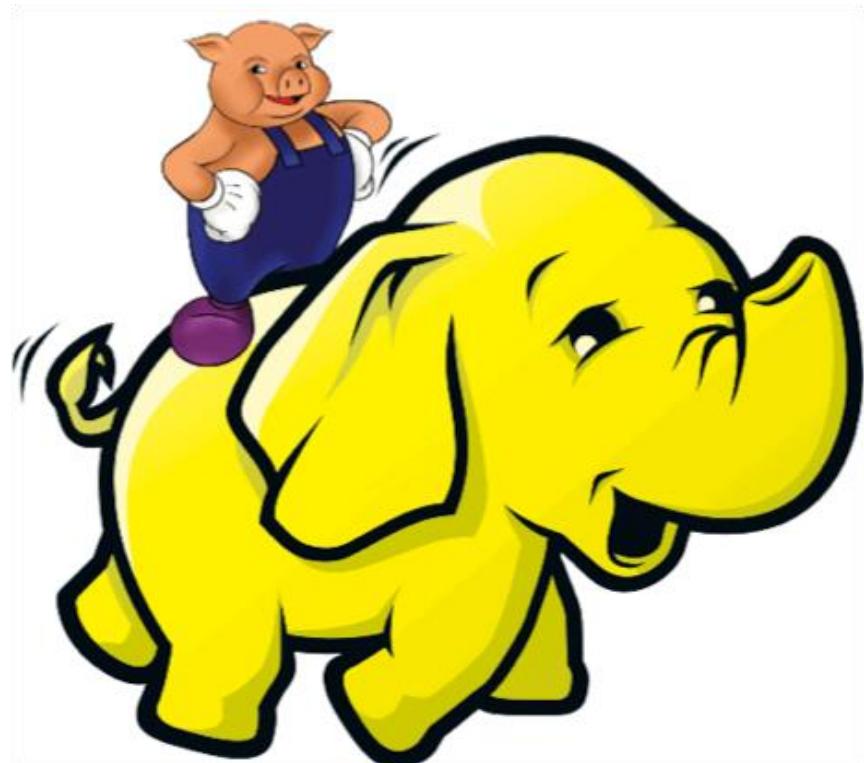
- Allows for scripting MapReduce jobs:
- Procedural, but makes use of relational algebra
- Three ways to run:
 1. Interactive command line
 2. Batch script
 3. Call from Java

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



</MAPREDUCE>
</HADOOP>

Apache Hadoop ... Internals (if interested)

Apache Hadoop (MapReduce) Internals - Diagrams

Fork me on GitHub

This project contains several diagrams describing Apache Hadoop internals (2.3.0 or later). Even if these diagrams are NOT specified in any formal or unambiguous language (e.g., UML), they should be reasonably understandable (here some [diagram notation conventions](#)) and useful for any person who want to grasp the main ideas behind Hadoop. Unfortunately, not all the internal details are covered by these diagrams. You are free to help :)



<http://ercoppa.github.io/HadoopInternals/>



Questions?