poloclub.github.io/#cse6242

CSE6242/CX4242: Data & Visual Analytics

Scaling Up Pig

Duen Horng (Polo) Chau

Associate Professor, College of Computing Associate Director, MS Analytics Georgia Tech

Mahdi Roozbahani

Lecturer, Computational Science & Engineering, Georgia Tech Founder of Filio, a visual asset management platform

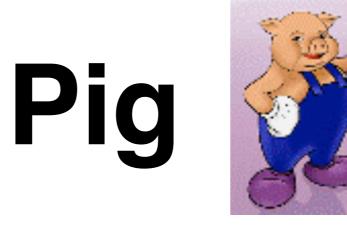
Partly based on materials by Guy Lebanon, Jeffrey Heer, John Stasko, Christos Faloutsos



http://pig.apache.org

High-level language

- instead of writing low-level map and reduce functions
- Easier to program, understand and maintain
- Created at Yahoo!
- Produces sequences of Map-Reduce programs
- (Lets you do "joins" much more easily)



http://pig.apache.org

Your data analysis task becomes a data flow sequence (i.e., data transformations)

Input data flow output

You specify **data flow** in **Pig Latin** (Pig's language). Then, Pig turns the data flow into a sequence of MapReduce jobs automatically!

Pig: 1st Benefit

Write only a few lines of Pig Latin

Typically, MapReduce development cycle is long

- Write mappers and reducers
- Compile code
- Submit jobs
- •

Pig: 2nd Benefit

Pig can perform a **sample run** on representative subset of your input data automatically!

Helps debug your code in smaller scale (much faster!), before applying on full data

What Pig is good for?

Batch processing

- Since it's built on top of MapReduce
- Not for random query/read/write

May be **slower** than MapReduce programs coded from scratch

 You trade ease of use + coding time for some execution speed

How to run Pig

Pig is a client-side application (run on your computer)

Nothing to install on Hadoop cluster

How to run Pig: 2 modes

Local Mode

- Run on your computer (e.g., laptop)
- Great for trying out Pig on small datasets

MapReduce Mode

- Pig translates your commands into MapReduce jobs
- Remember you can have a single-machine cluster set up on your computer

Difference between PIG local and mapreduce mode: <u>http://stackoverflow.com/questions/</u> 11669394/difference-between-pig-local-and-mapreduce-mode

Pig program: 3 ways to write

Script

Grunt (interactive shell)

Great for debugging

Embedded (into Java program)

- Use PigServer class (like JDBC for SQL)
- Use PigRunner to access Grunt

Grunt (interactive shell)

Provides code completion

Press **Tab** key to complete Pig Latin keywords and functions

Let's see an example Pig program run with Grunt

• Find highest temperature by year

records = LOAD 'input/ ncdc/ micro-tab/ sample.txt'
AS (year:chararray, temperature:int, quality:int);

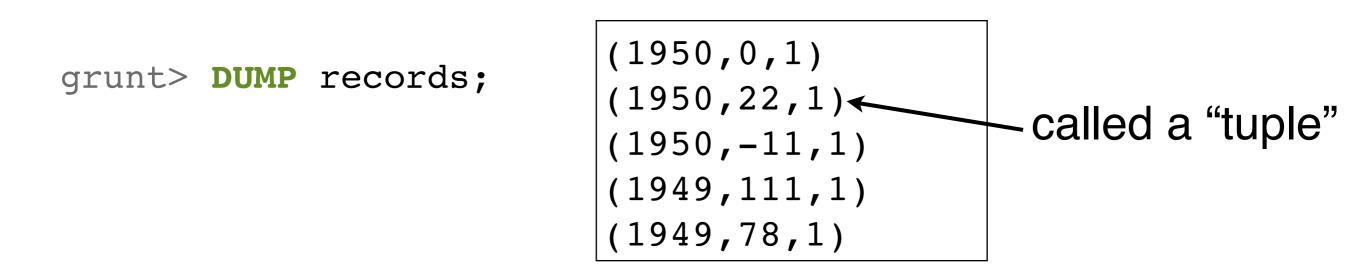
```
filtered_records =
  FILTER records BY temperature != 9999
  AND (quality = = 0 OR quality = = 1 OR
     quality = = 4 OR quality = = 5 OR
     quality = = 9);
```

grouped_records = GROUP filtered_records BY year;

```
max_temp = FOREACH grouped_records GENERATE
group, MAX(filtered_records.temperature);
```

DUMP max_temp;

```
grunt>
records = LOAD 'input/ncdc/micro-tab/sample.txt'
AS (year:chararray, temperature:int, quality:int);
```



grunt> DESCRIBE records;

records: {year: chararray, temperature: int, quality: int}

```
grunt>
filtered_records =
  FILTER records BY temperature != 9999
  AND (quality == 0 OR quality == 1 OR
      quality == 4 OR quality == 5 OR
      quality == 9);
```

grunt> DUMP filtered_records;

In this example, no tuple is filtered out

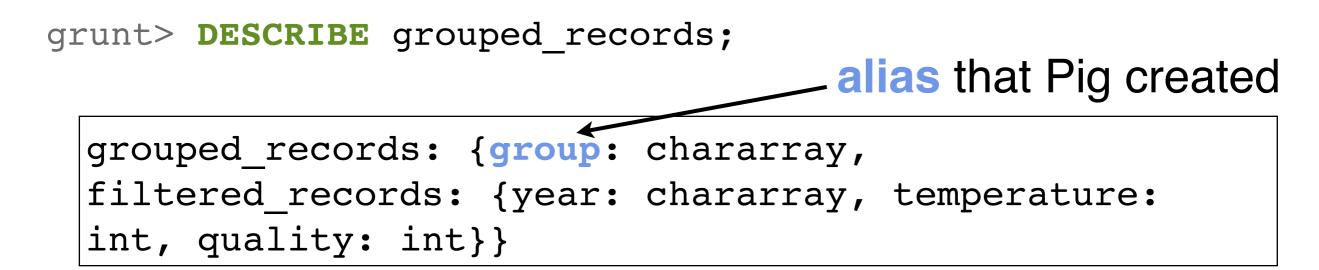
grunt> grouped_records = GROUP filtered_records BY year;

grunt> DUMP grouped_records;

(1949, {(1949, 111, 1), (1949, 78, 1)})
(1950, {(1950, 0, 1), (1950, 22, 1), (1950, -11, 1)})

called a "bag"

= unordered collection of tuples



(1949,{(1949,111,1), (1949,78,1)})
(1950,{(1950,0,1),(1950,22,1),(1950,-11,1)})

grouped_records: {group: chararray, filtered_records: {year: chararray, temperature: int, quality: int}}

grunt> max_temp = FOREACH grouped_records GENERATE
group, MAX(filtered_records.temperature);

grunt> DUMP max_temp;

(1949,111) (1950,22)

Run Pig program on a subset of your data

You saw an example run on a tiny dataset

How to do that for a larger dataset?

Use the ILLUSTRATE command to generate sample dataset

Run Pig program on a subset of your data

grunt> ILLUSTRATE max_temp;

records	year:chararray	temperature:int	quality:int	
 	1949 1949 1949	78 111 9999	1 1 1	
filtered_rec	ords year:chararra	y temperature:int	quality:int	
	1949 1949	78 111	1 1	
grouped_records group:chararray filtered_records:bag{:tuple(year:chararray, temperature:int,quality:int)}				
	1949	{(1949, 78, 1), (19	949, 111, 1)}	
max_temp 	group:chararray 1949	:int 111		

How does Pig compare to SQL?

SQL: "fixed" schema

PIG: loosely defined schema, as in

records = LOAD 'input/ncdc/micro-tab/sample.txt'
AS (year:chararray, temperature:int, quality:int);

How does Pig compare to SQL?

SQL: supports fast, random access

(e.g., <10ms, but of course depends on hardware, data size, and query complexity too)

PIG: batch processing

Pig vs SQL

1. Pig Latin is **procedural**, where SQL is **declarative**.

- 2. Pig Latin allows pipeline **developers to decide where to checkpoint data** in the pipeline.
- 3. Pig Latin allows the developer to select specific operator implementations directly **rather than relying on the optimizer**.
- 4. Pig Latin supports **splits** in the pipeline.
- 5. Pig Latin allows developers to **insert their own code** almost anywhere in the data pipeline.

Much more to learn about Pig

Relational Operators, Diagnostic Operators (e.g., describe, explain, illustrate), utility commands (cat, cd, kill, exec), etc.

Table 11-1. Pig Latin relational operators

Category	Operator	Description	
Loading and storing	LOAD	Loads data from the filesystem or other storage into a relation	
	STORE	Saves a relation to the filesystem or other storage	
	DUMP	Prints a relation to the console	
Filtering	FILTER	Removes unwanted rows from a relation	
	DISTINCT	Removes duplicate rows from a relation	
	FOREACH GENERATE	Adds or removes fields from a relation	
	MAPREDUCE	Runs a MapReduce job using a relation as input	
	STREAM	Transforms a relation using an external program	
	SAMPLE	Selects a random sample of a relation	
Grouping and joining	JOIN	Joins two or more relations	
	COGROUP	Groups the data in two or more relations	
	GROUP	Groups the data in a single relation	
	CROSS	Creates the cross-product of two or more relations	
Sorting	ORDER	Sorts a relation by one or more fields	
	LIMIT	Limits the size of a relation to a maximum number of tuples	
Combining and splitting	UNION	Combines two or more relations into one	
	SPLIT	Splits a relation into two or more relations	