Large Scale Data Processing
Pig

Dr. Wenceslao PALMA
wenceslao.palma@ucv.cl

Pig

- Pig raises the level of abstraction for processing large datasets. Pig is made up of two pieces:
  - The language used to express data flows called *Pig Latin*.
  - An execution environment to run Pig Latin program which transforms a data flow into a series of MapReduce jobs.

Pig

- Pig raises the level of abstraction for processing large datasets. Pig is made up of two pieces:
  - The language used to express data flows called *Pig Latin*.
  - An execution environment to run Pig Latin program which transforms a data flow into a series of MapReduce jobs.

Pig allows to focus on the data rather than the nature of the execution.

Pig

- Pig raises the level of abstraction for processing large datasets. Pig is made up of two pieces:
  - The language used to express data flows called *Pig Latin*.
  - An execution environment to run Pig Latin program which transforms a data flow into a series of MapReduce jobs.

Pig allows to focus on the data rather than the nature of the execution.

"The [Hofmann PLSA E/M] algorithm was implemented in pig in 30-35 lines of pig-latin statements. **Took a lot less compared to what it took in implementing the algorithm in Map-Reduce Java**. Exactly that's the reason I wanted to try it out in Pig. It took 3-4 days for me to write it, starting from learning pig."
– Prasenjit Mukherjee, Mahout project [a]

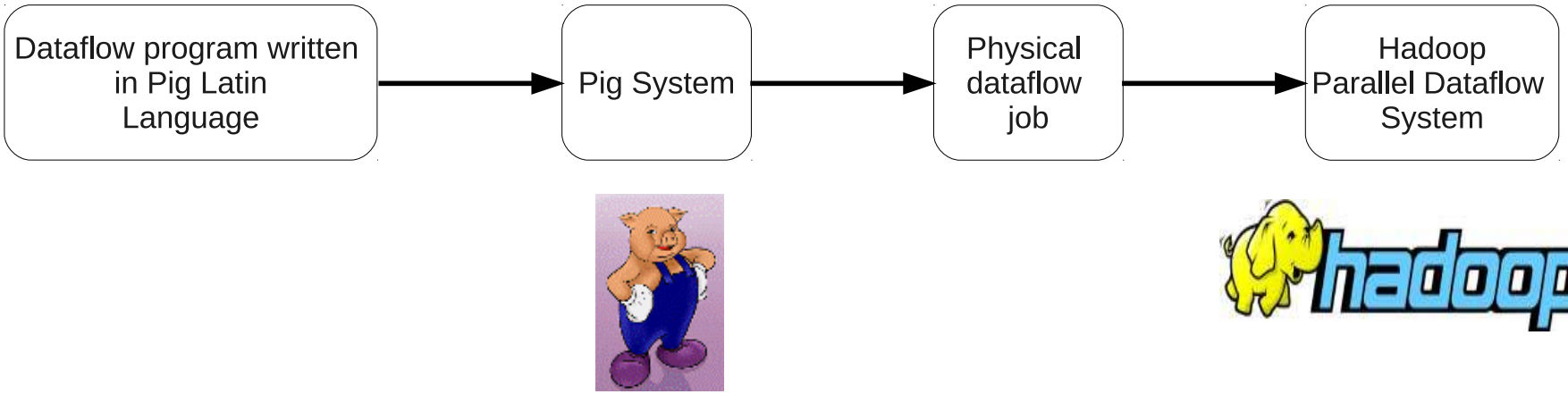
[a] Olston C. et al. Programmming and Debugging Large-Scale Data Processing Workflows

Pig

- Pig doesn't perform as well as programs written in MapReduce (see paper *A Comparison of Join Algorithms for Log Processing in MapReduce*). However, the gap is narrowing with each release.
- Writting queries in Pig will save your time :)
- Pig has two execution modes:
  - Local mode. This mode is suitable only for small datasets and when trying out Pig. ($pig -x local)
  - MapReduce mode. Pig translates queries into MapReduce jobs and runs them in a cluster. In this case, we must verify that the Pig version is compatible with the version of Hadoop we are using. This is documented on the releases page.

PigPen is an Eclipse plugin that provides a Pig script text editor and a button for running the script on an Hadoop cluster. It includes an operator graph window which shows a script in graph form for visualizing the data flow.

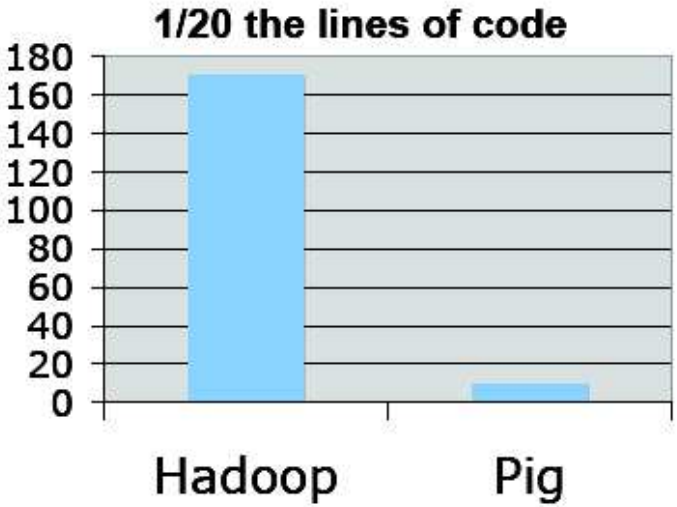
Pig

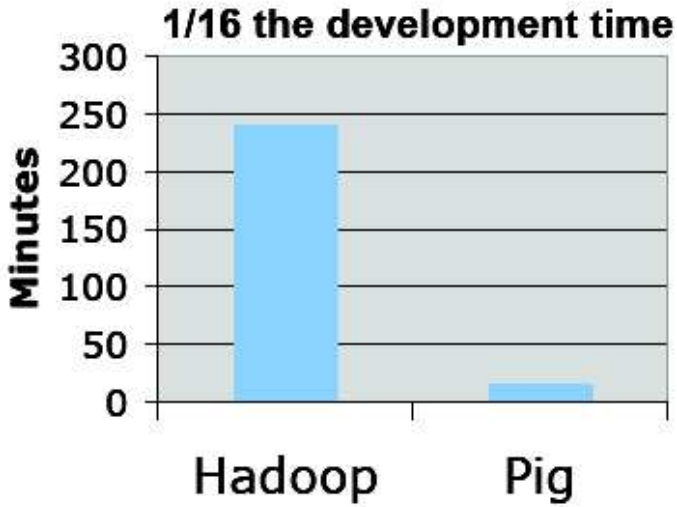


Pig Latin provides:

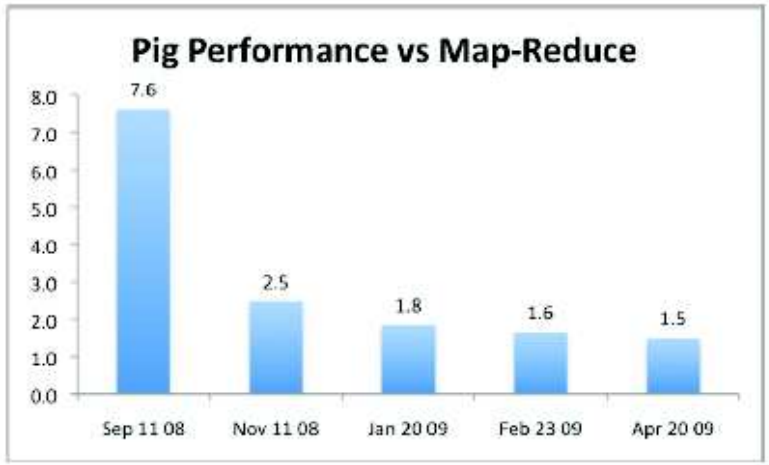
- more transparent program structure
- easier program development and maintenance
- automatic optimization opportunities

Comparison





performance
1.5x Hadoop



[1] from Pig talk at SIGMOD08

Example: find the top 10 most visited pages in each category

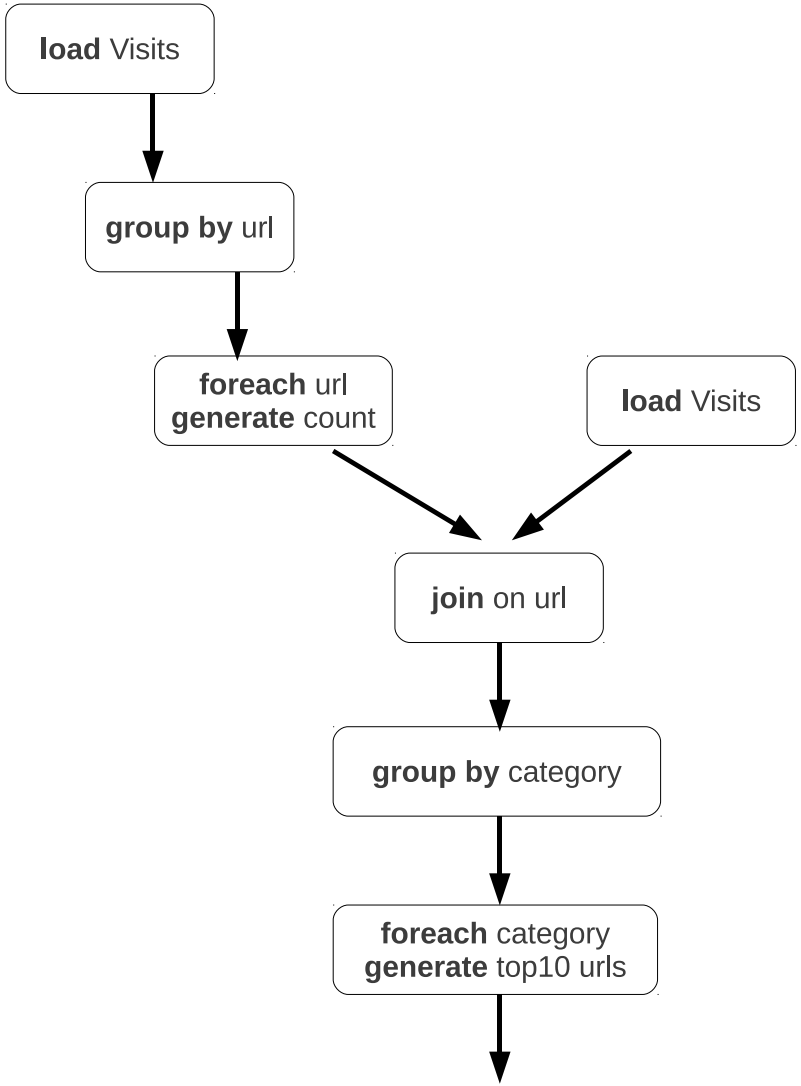
Visits

User	Url	Time
Amy	cnn.com	8:00
Amy	bbc.com	10:00
Amy	flickr.com	10:05
Fred	cnn.com	12:00

Url Info

Url	Category	PageRank
cnn.com	News	0.9
bbc.com	News	0.8
flickr.com	Photos	0.7
espn.com	Sports	0.9

A dataflow diagram for the previous query



Pig Latin Language

The dataflow in Pig Latin:

```
visits = load '/data/visits' as (user,url,time);
gVisits = group visits by url;
visitsCounts = foreach gVisits generate url,count(visits);

urlInfo = load '/data/urlInfo/' as (url,category,pRank);
visitCounts = join visitsCounts by url, urlInfo by url;

gCategories = group visitCounts by category;
topUrls = foreach gCategories generate top(visitCounts,10);

store topUrls into '/data/topUrls';
```

Pig Latin Language

- Operates directly over files. Ex.: '/data/visits', '/data/urlInfo/', '/data/topUrls'
- Schemas can be assigned dynamically but they are optional. Ex.: **as** (user,url,time), **as** (url,category,pRank)
- User defined functions (UDFs) can be used in every construct: **load**,**store**,**group**,**filter**,**foreach**. Ex.: top(visitCounts,10);

UDFs are written in Java, and filter functions are all subclasses of FilterFunc, which itself is a subclass of EvalFunc.

```
public abstract class EvalFunc<T> {
     public abstract T exec(Tuple input) throws IOException;
}
```

To use the new function:

- It must be compiled and packaged it in a JAR file.
- We tell Pig about the JAR file with the REGISTER operator

Pig Latin Language

Command	Description
LOAD	Read data from file system.
STORE	Write data to file system.
FOREACH...GENERATE	Apply an expression to each record and output one or more records.
FILTER	Apply a predicate and remove records that do not return true.
GROUP/COGROUP	Collect records with the same key from one or more inputs.
JOIN	Join two or more inputs based on a key.
CROSS	Cross product two or more inputs.
UNION	Merge two or more datasets.
SPLIT	Split data into two or more sets, based on filter conditions.
ORDER	Sort records based on a key.
DISTINCT	Remove duplicate tuples.
STREAM	Send all records through a user provided binary.
DUMP	Write ouput to stdout
LIMIT	Limit the number of records

Pig vs SQL-DBMS

Pig:

- Data flow programming language. A Pig Latin program is a step-by-step set of operations on an input relation, in which each step is a single transformation.
- More relaxed about the data being processed: schemas can be defined at runtime but it's optional.
- All reads are bulk, streaming writes (like MapReduce).

Programming in Pig Latin is like working at the level of an RDBMS query planner, which figure out how to turn a declarative statement into a system of steps.

SQL:

- Declarative programming language.
- Data is stored in tables with tightly predefined schemas.
- Low-latency queries (random read/writes), transactions and indexes.

Counting Triangles: The Pig solution (www.vertica.com)

```
set default_parallel N;
set mapreduce.job.maps M;
EDGES         = load 'input/graph.txt' using PigStorage(' ') as (source:long, dest:long);
CANON_EDGES_1 = filter EDGES by source < dest;
CANON_EDGES_2 = filter EDGES by source < dest;
TRIAD_JOIN    = join CANON_EDGES_1 by dest, CANON_EDGES_2 by source;
OPEN_EDGES    = foreach TRIAD_JOIN generate CANON_EDGES_1::source, CANON_EDGES_2::dest;
TRIANGLE_JOIN = join CANON_EDGES_1 by (source,dest), OPEN_EDGES by (CANON_EDGES_1::source,CANON_EDGES_2::dest);
TRIANGLES     = foreach TRIANGLE_JOIN generate 1 as a:int;
CONST_GROUP   = group TRIANGLES ALL parallel 1;
FINAL_COUNT   = foreach CONST_GROUP generate COUNT(TRIANGLES);
dump FINAL_COUNT;
```

References

- *Hadoop: The Definitive Guide*. Tom White. O'Reilly 2010.
- *Pig: Web-scale data processing*. Christopher Olston. Pig talk at SIGMOD 2008.