The Pig Experience

A. Gates et al., VLDB 2009
Why not Map-Reduce?

- Does not directly support complex N-Step dataflows
  - All operations have to be expressed using MR primitives
- Lacks explicit support for processing of structured data
  - JOINs
- Data Manipulation primitives are missing
  - Filtering, aggregation, top-k
Implications of Using MR

• Makes the coding cycle longer
• Hard to run ad-hoc data analyses
• Hard to read/debug MR programs
• Automatic optimization is hard
  – Too much custom-made code
Pig

- High-level data manipulation
- Modular
- Scalable (Pig Latin is translated into MR)
- Encodes explicit dataflow graphs
Pig vs SQL

• SQL
  – Purely declarative
  – Runs on a relational DB with pre-defined schema
  – Query optimization using indexes/compression

• Pig
  – Mixes declarative and imperative constructs
  – Runs on a non-normalized TSV files
  – Translates into MR: no query optimization
Pig Data Types

- A relation is a bag
- A bag is a collection of tuples
- A tuple is an ordered set of fields
- A field is a piece of data

```pig
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))
```
Relational Operators

• FILTER
  – Selects tuples from a relation based on some condition

  \[ X = \text{FILTER} \ A \ \text{BY} \ (f1 == 8) \ \text{OR} \ (\text{NOT} \ (f2+f3 > f1)); \]
Relational Operators

• **FOREACH**
  
  - Generates data transformations based on columns of data

DUMP B;
(2,4) (8,9) (1,3) (2,7) (2,9) (4,6) (4,9)

DUMP C;
(1,{{1,2,3}},{(1,3)})
(4,{{4,2,1},{4,3,3}},{(4,6),(4,9)})
(8,{{8,3,4},{8,4,3}},{(8,9)})

X = FOREACH C GENERATE group, B.b2;

DUMP X;
(1,{{3}}) (4,{{6),(9}}) (8,{{9}})
Relational Operators

• GROUP BY
  – Groups the data in one or multiple relations

DUMP A;

B = GROUP A BY url;

DUMP B;
Relational Operators

• **FLATTEN**
  – un-nests tuples as well as bags

• 
  \[(a, (b, c))\]
  – \( \text{GENERATE } 0, \text{flatten}(\$1) \rightarrow (a, b, c) \)

• 
  \[(a, \{(b,c),(d,e)\})\]
  – \( \text{GENERATE } 0, \text{flatten}(\$1) \rightarrow (a, b, c), (a, d, e) \)
Relational Operators

• JOIN

  – Performs inner, equijoin of two or more relations based on common field values.

  – Shorthand for (CO)GROUP followed by FLATTEN
UDFs

- Pig provides extensive support for user-defined functions (UDFs) as a way to specify custom processing.
- UDFs can be a part of any operator in Pig.

```pig
-- myscript.pig
REGISTER myudfs.jar;

A = LOAD 'student_data' AS (name: chararray, age: int, gpa: float);

B = FOREACH A GENERATE myudfs.UPPER(name);
```
Streaming

- Input: standard input/file
- Output: standard input/file
- Both input and output are treated as standard pig relations

\[
A = \text{LOAD 'data'}; \\
\text{DEFINE cmd `stream.pl –n 5`;} \\
B = \text{STREAM A THROUGH cmd;} \\
\]
Data Guarantees in Streaming

• Unordered data
  – No guarantee for the order in which the data is delivered to the streaming application.

• Grouped data
  – The data for the same grouped key is guaranteed to be provided to the streaming application contiguously.

• Grouped and ordered data
  – The data for the same grouped key is guaranteed to be provided to the streaming application contiguously.
  – The data within the group is guaranteed to be sorted by the provided secondary key.
**Pig Compilation and Execution**

**Parser**
Verify that the program is syntactically correct

Output a canonical (non-optimized) logical plan
Pig Compilation and Execution

Logical Optimizer
Optimize the canonical logical plan

Push Up Filters
Push the FILTER operators up the data flow graph

Push Down Explodes
Reduce the number of records that flow through the pipeline by moving FOREACH operators with a FLATTEN down the data flow graph.
Pig Latin

A = LOAD 'file1' AS (x, y, z);
B = LOAD 'file2' AS (t, u, v);
C = FILTER A by y > 0;
D = JOIN C BY x, B BY u;
E = GROUP D BY z;
F = FOREACH E GENERATE
    group, COUNT(D);
STORE F INTO 'output';

Logical Plan

LOAD (x, y, z)
FILTER (x, y, z)
JOIN (x, y, z, t, u, v)
GROUP (group, {(x, y, z, t, u, v)})
FOREACH (group, count)
STORE (group, count)
Pig Compilation and Execution

MR Translation

*Compile the optimized logical plan into a DAG of MR jobs*
Logical-MR Compilation

• Logical Plan $\rightarrow$ Physical Plan
  – Embeds each physical operator within MR stage

• Most operators have one-to-one mapping
  – FILTER, LOAD, STORE

• Others have more complex translations
  – GROUP, JOIN
• (CO)GROUP becomes a series of

1. Local rearrange (M)
   *Local tuple sort by group-by key*

2. Global rearrange (M)
   *All tuples with same group-by key are on the same machine*

3. Package (R)
   *Create a single-tuple package (id, {tuples}) per group-by key*
• JOIN is a
  – (CO)GROUP (M/R)
  – FLATTEN (R)
Pig Compilation and Execution

Running the jobs

*Topologically sort the DAG of MR jobs*
*Submit jobs to Hadoop in the sorted order*
*Monitor the execution status*
Flow Control

• Pig uses an *iterator model*
  – all algebra operators are implemented as *iterators*
    and support a simple open-next-close protocol
  – Simple API for UDFs
  – Some extensions to support synchronization
    between branches in data-flow graph
Branching

• Branching can be obtained through SPLIT/MULTIPLEX operators
  – Processing data in multiple ways without loading it multiple times

• Using too many SPLITs can harm combiner effectiveness
  – A smaller portion of data can be held in memory
  – Up to the user to reason about this tradeoff
Nesting

clicks = LOAD 'clicks'
    AS (userid, pageid, linkid, viewedat);
byuser = GROUP clicks BY userid;
result = FOREACH byuser {
    uniqPages = DISTINCT clicks.pageid;
    uniqLinks = DISTINCT clicks.linkid;
    GENERATE group, COUNT(uniqPages),
          COUNT(uniqLinks);
};

• # distinct pages and links visited by each user
• Outer FOREACH has a nested sub-graph with two DISTINCT/COUNT pipelines for pages and links
  – Pipelines are executed sequentially
Pig In Practice

• Excellent for large processing of (sloppily) structured data
  – Query logs
  – Web dumps
  – Social network analysis

• Flexible due to
  – Lazy type conversion
  – Optional schemas
  – Text file storage
Some Cookbook Tips

• **Project/Filter Early and Often**
  – Pig does not (yet) determine when a field is no longer needed
  – Carrying large amounts of data through the pipeline can cause slowdowns
Some Cookbook Tips

• **Take Advantage of Join Optimization**
  – Insures that the last table in the join is not brought into memory but stream through instead.
  – Reduces the amount of memory used which means you can avoid spilling the data
  – Make sure that the table with the largest number of tuples per key is the last table in your query.
Some Cookbook Tips

• **Use PARALLEL Keyword**
  – PARALLEL controls the number of reducers. The default out of the box is 1.
  – Heuristic: \(<\text{num machines}> \times \text{num reduce slots per machine}> \times 0.9\)
  – Can be used with GROUP, COGROUP, JOIN, DISTINCT, LIMIT, ORDER BY.
More at Pig Cookbook