large-scale data analysis

Christopher Olston and many others
Yahoo! Research

Motivation

- Projects increasingly revolve around analysis of big data sets
  - Extracting structured data, e.g. face detection
  - Understanding complex large-scale phenomena
    - social systems (e.g. user-generated web content)
    - economic systems (e.g. advertising markets)
    - computer systems (e.g. web search engines)
  - Understanding ⇒ innovation
  - Data analysis is "inner loop" at Yahoo! et al.
- Big data necessitates parallel processing
- Need architectures, software & languages for parallel data analysis

Examples

1. Detect faces
   - You have function detectFaces()
   - You want to run it over *n* images
   - n is big
2. Study web search ranking
   - You have a log of web search results
   - You want to identify search queries for which "PageRank" was not the dominant ranking feature

Existing Work

- Parallel architectures
  - cluster computing
  - multi-core processors
- Data-parallel software
  - parallel DBMS
  - Map-Reduce
- Parallelizable data languages
  - Sawzall
  - SQL (mostly)

Pig Project

- Data-parallel language ("Pig Latin")
  - Goals:
    - Simple & customizable data analysis primitives
    - Easy for the system to parallelize & optimize
- Data-parallel software ("Pig")
  - Focus on cross-program optimizations:
    - Combined execution of related programs
    - Reuse of derived data

Talk Outline

- Pig Latin language
  - Examples and language overview
  - Related work
  - Ongoing work: Pig Latin IDE
- Pig system
  - System overview
  - Cross-job optimization
  - Related work

Pig Latin Language

Example 1

- You have function detectFaces()
- You want to run it over *n* images
- n is big

```
I = load '/mydata/images' using ImageParser() as (id, image);
F = foreach I generate id, detectFaces(image);
store F into '/mydata/faces';
```

Example 2

Find queries for which the highest-PageRank page in the result set did not appear among the top 5 results.

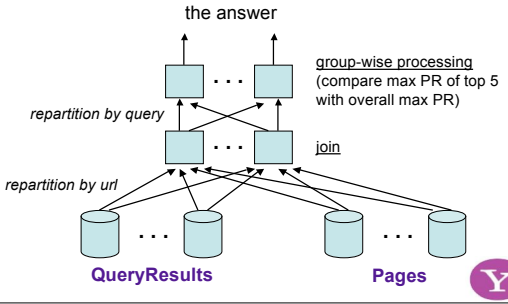
QueryResults

query	position	url
"news"	1	cnn.com
"news"	2	bbc.com
…		
"pig"	1	pigwheels.com
…		

Pages

url	pagerank
bbc.com	0.9
cnn.com	0.9
pigwheels.com	0.3
…	

Efficient Evaluation Method



In Pig Latin

```
QueryResults = load '/data/query_results';
Pages = load '/data/pages';

A = join QueryResults by url, Pages by url;

B = group A by query;

C = filter B by CheckTop5(*);

store C into '/data/interesting_queries';
```

Pig Latin, in general

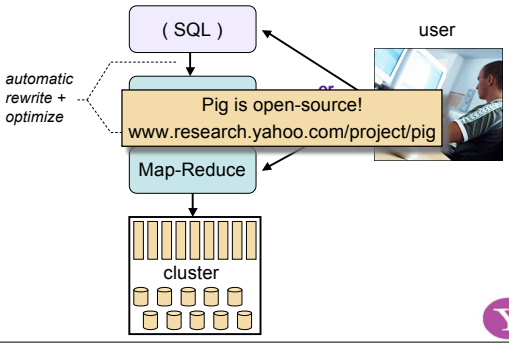
- transformations on sets of records
- easy for users
  - high-level abstraction for data analysis
- easy for the system
  - exposes opportunities for parallelism and reuse

operators:	*binary operators:*
• FILTER	• COGROUP
• FOREACH … GENERATE	• CROSS
• GROUP	• UNION

Related Work

- **SQL:** declarative all-in-one blocks
- **Map-Reduce:** special case of Pig Latin
- **Sawzall:** filter-aggregate ⇒ map-reduce

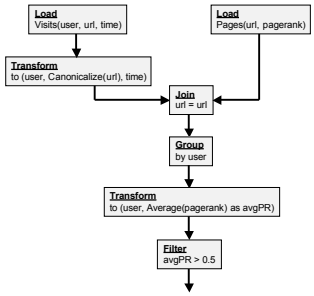
Current Deployment



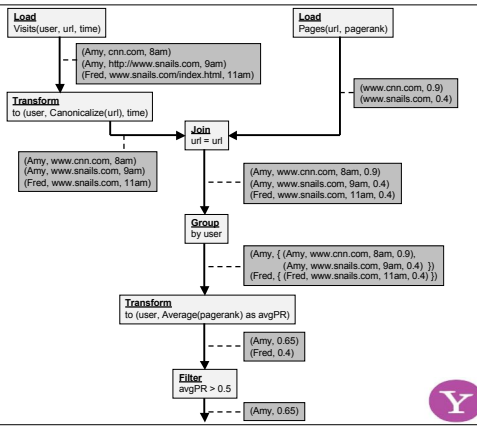
Ongoing Work: **Pig Pen**

- GUI for writing, debugging & sharing Pig Latin programs
  - Shared data & code repository
  - Syntax highlighting, error checking, etc.
  - "Boxes-and-arrows" dataflow graph
  - Automatically generated example data to assist in debugging

Example: Find Users Who Tend to Visit "Good" Pages



Correct answer?



Generating Example Data

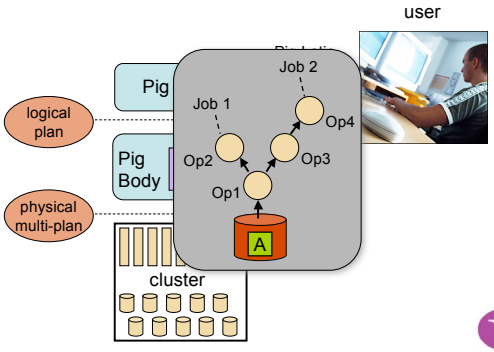
- Objectives:
  - Realism
  - Conciseness
  - Completeness
- Challenges:
  - Large original data
  - Selective operators (e.g., join, filter)
  - Noninvertible operators (e.g., UDFs)

NEXT: Pig System

- Pig Latin language
  - Examples and language overview
  - Related work
  - Ongoing work: Pig Latin IDE

➡ Pig system
  - System overview
  - Cross-job optimization
  - Related work

Pig System



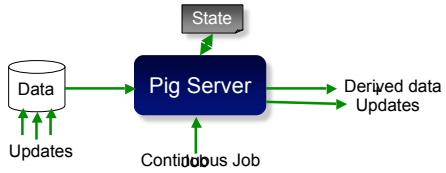
Key Issue: Work Sharing

- Repeated work across jobs
  - Scan common tables
    - web crawl
    - search log
  - Do common transformations
    - eliminate spam pages
    - group pages by host
    - join web crawl with search log
  - Redo derived computations
    - most viewed page in past 3 days
    - number of hyperlinks incident to each page
- Goal: Minimize redundant work by system

Techniques

- execute similar jobs together
- cache moves & transformations
- update derived data incrementally

Updating Derived Data: **iPig**



Example Job:

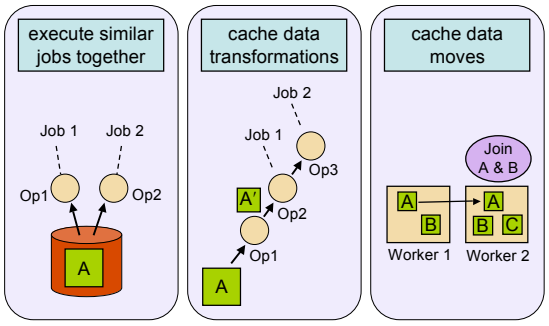
- For every web site, tell me the most visited page in the last 5 hours
- Do this every hour

```
views      = LOAD 'page_views' CONTINUOUSLY;
site_views = GROUP views BY site WINDOW BY time(5hr, 1hr);
top_page   = FOREACH site_views GENERATE group, mostFrequent(views.url);
```

iPig Challenges

- API for user-defined functions (UDFs)
  - Stateful cursors over input data
  - Persist custom UDF state across invocations
  - Okay for UDFs to ignore "incrementalness"
- Parallel batch processing *(vs. serial continuous)*
  - Transactional state updates
  - State migration
- Multiple windows over same data feed
  - Avoid redundant state

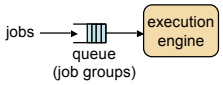
Sharing Work Across Independent Jobs



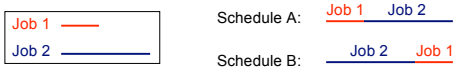
Work-Sharing Challenges

- System architecture
  - Built-in support for work-sharing modalities
- Algorithms to govern work-sharing
  - "Model-light" approach

Executing Similar Jobs Together

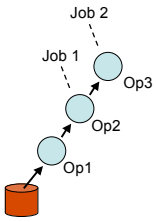


Optimal queue ordering policy?



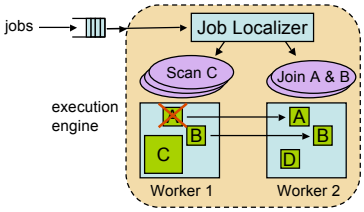
- New "sharable" jobs arrive with frequency λ_1, λ_2
- Which schedule is best:
  - If $\lambda_1 >> \lambda_2$
  - If $\lambda_1 = \lambda_2$

Caching Data Transformations



- Options:
  - Cache Op2 output
  - Cache Op3 output
  - Cache both
- Considerations:
  - Space
  - Utility
  - Cost to generate
- ☹ Difficult to estimate a priori
- ☺ Can materialize "fragments", and learn

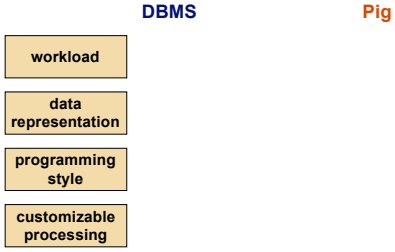
Caching Data Moves



Related Work

- FS & DB data placement techniques
  - model-based, random or round-robin
  - some incorporate fault-tolerance considerations
- DB work on selecting materialized views
  - model-based
  - some combine with query optimization
- Prior data-parallel systems, e.g., Map-Reduce, Dryad, parallel DBMSs
  - no work sharing

Is Pig a DBMS?



Summary

- Pig Latin
  - high-level language for data parallelism
    - sequence of data transformation steps
    - users can plug in custom code
- Pig Pen
  - automated data sandbox, for debugging
- Pig System
  - mechanisms & algorithms to share work

Credits

Shubham Chopra
Alan Gates
Dan Kifer
Ravi Kumar
Antonio Magnaghi
Shravan Narayanamurthy
Olga Natkovich

Chris Olston
Ben Reed
Adam Silberstein
Utkarsh Srivastava
Andrew Tomkins
Erik Vee
Rob Weltman

interns: Parag Agarwal, Tyson Condie, Sandeep Pandey, Ying Xu

Additional slides …

Pig Latin vs. SQL

SQL	declarative (**what**, not **how**); bundle many aspects into one statement
Pig Latin	sequence of simple transformations

Pig Latin vs. Map-Reduce

- Map-reduce welds together 3 primitives:
  process records → create groups → process groups

```
a = FOREACH input GENERATE flatten(Map(*));
b = GROUP a BY $0;
c = FOREACH b GENERATE Reduce(*);
```

- In Pig, these primitives are:
  - explicit
  - independent
  - fully composable
- Pig adds primitives for:
  - filtering tables
  - projecting tables
  - combining 2 or more tables

more natural programming model

optimization opportunities

Pig Latin vs. Sawzall

- Sawzall translates filter-group-aggregate expressions into map-reduce
  - Rigid two-part structure
  - Single-input processing