pig || web-scale data processing

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)

- Data analysis is "inner loop" at Yahoo! et al.

- Big data necessitates parallel processing

Examples

1. Detect faces
   - You have a function detectFaces()
   - You want to run it over *n* images
   - n is big
2. Study web usage
   - You have a web crawl and click log
   - Find sessions that end with the “best” page

Existing Work

- Parallel architectures
  - cluster computing
  - multi-core processors
- Data-parallel software
  - parallel DBMS
  - Map-Reduce, Dryad
- Data-parallel languages
  - SQL
  - NESL

Pig Project

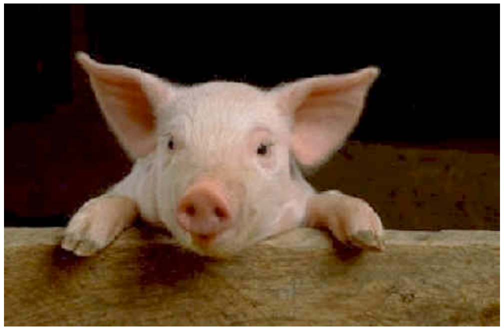
- Data-parallel language (“Pig Latin”)
  - Relational data manipulation primitives
  - Imperative programming style
  - Plug in code to customize processing
- Various crazy ideas
  - Multi-program optimization
  - Adaptive data placement
  - Automatic example data generator

Pig Latin Language

[SIGMOD'08]

Example 1

Detect faces in many images.



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

Example 2

Find sessions that end with the "best" page.

Visits

user	url	time
Amy	www.cnn.com	8:00
Amy	www.crap.com	8:05
Amy	www.myblog.com	10:00
Amy	www.flickr.com	10:05
Fred	cnn.com/index.htm	12:00

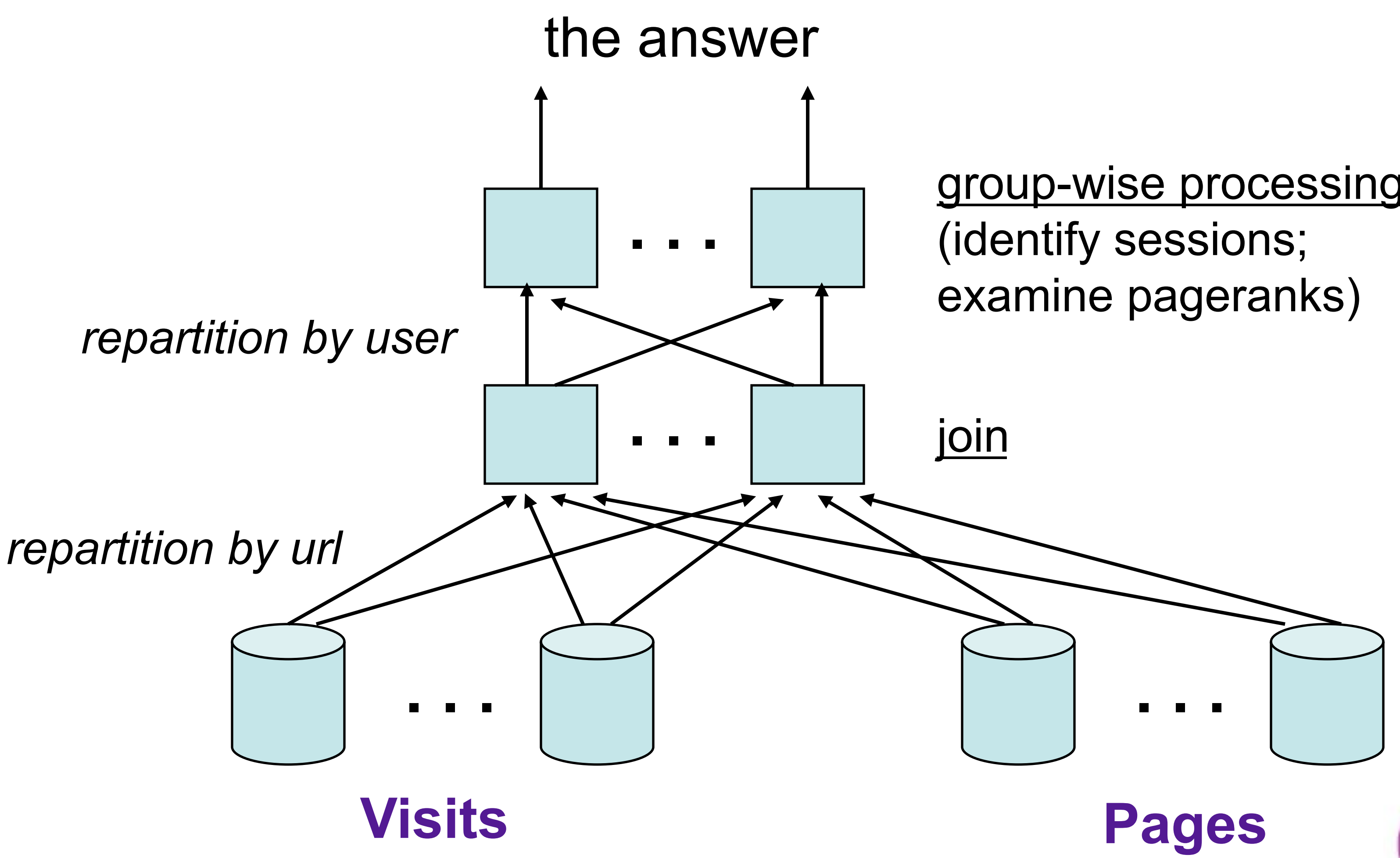
⋮

Pages

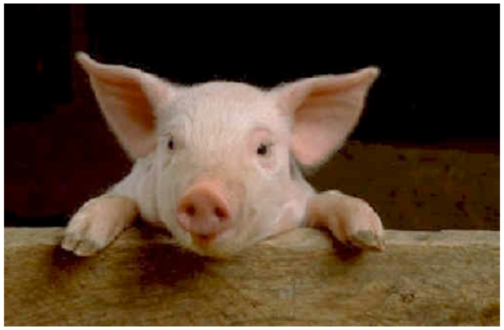
url	pagerank
www.cnn.com	0.9
www.flickr.com	0.9
www.myblog.com	0.7
www.crap.com	0.2

⋮

Efficient Evaluation Method



In Pig Latin



```
      Visits = load '/data/visits' as (user, url, time);
      Visits = foreach Visits generate user, Canonicalize(url), time;

       Pages = load '/data/pages' as (url, pagerank);

          VP = join Visits by url, Pages by url;
  UserVisits = group VP by user;
    Sessions = foreach UserVisits generate flatten(FindSessions(*));
HappyEndings = filter Sessions by BestIsLast(*);

       store HappyEndings into '/data/happy_endings';
```

Pig Latin, in general

- transformations on sets of records
- easy for users
  - high-level, extensible data processing primitives
- easy for the system
  - exposes opportunities for parallelism and reuse

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

Related Languages

- **SQL:** declarative all-in-one blocks
- **NESL:** lacks join, cogroup
- **Map-Reduce:** special case of Pig Latin

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

- **Sawzall:** rigid map-then-reduce structure

Pig Latin = Sweet Spot Between SQL & Map-Reduce

	SQL	Map-Reduce
Progra[mming style]	[Large blocks of declarative]	"[Plug together pipes]" (but ... -step
Built-in ... manipu...		
Execution model	Fancy; trust the query ...	Simple, transparent
Oppor... autom...		... map()

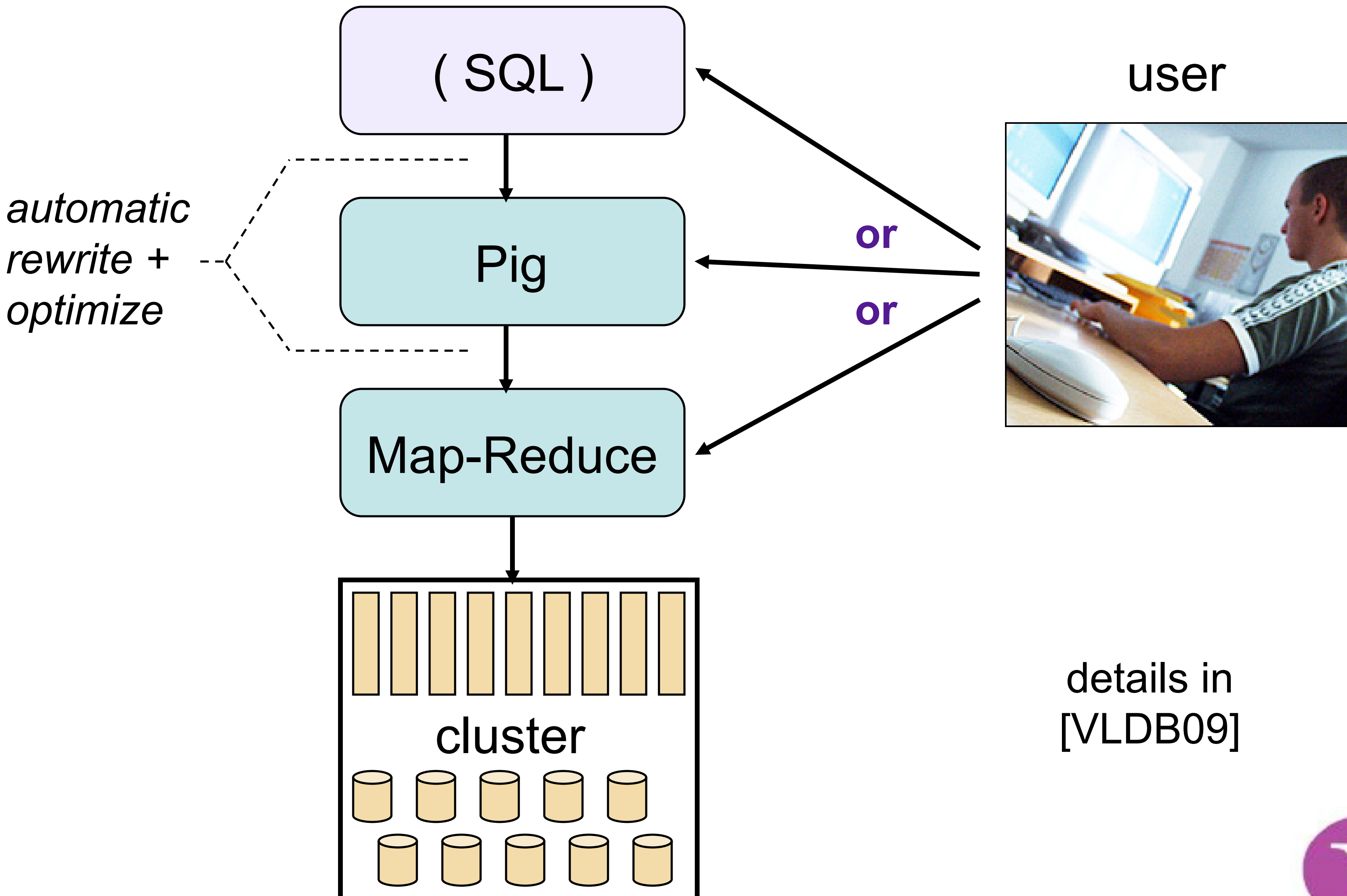
"I much prefer writing in Pig [Latin] versus SQL. The step-by-step method of creating a program in Pig [Latin] is much cleaner and simpler to use than the single block method of SQL. It is easier to keep track of what your variables are, and where you are in the process of analyzing your data."

-- Jasmine Novak, Engineer, Yahoo!

"PIG seems to give the necessary parallel programming construct (FOREACH, FLATTEN, COGROUP .. etc) and also give sufficient control back to the programmer (which purely declarative approach like [SQL on top of Map-Reduce] doesn't)."

-- Ricky Ho, Adobe Software

Map-Reduce as Backend



details in
[VLDB09]

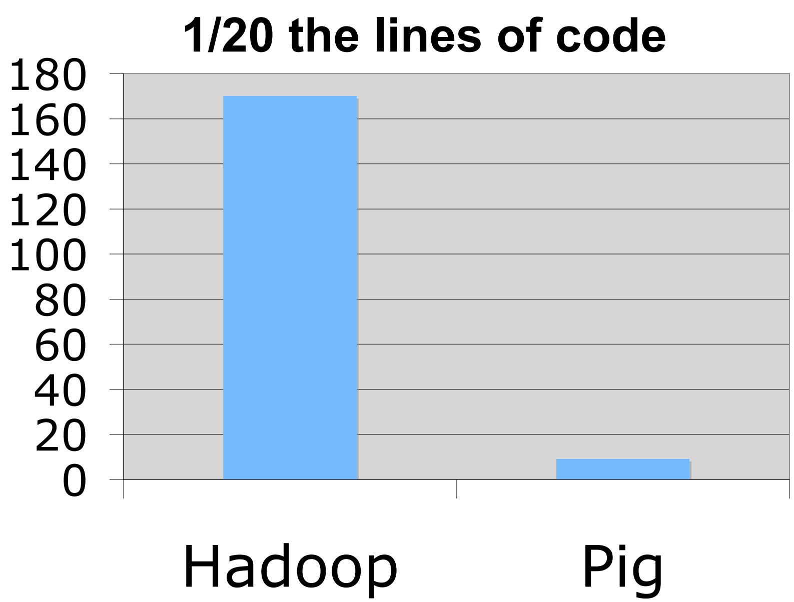
Pig Latin vs. Map-Reduce: Code Reduction

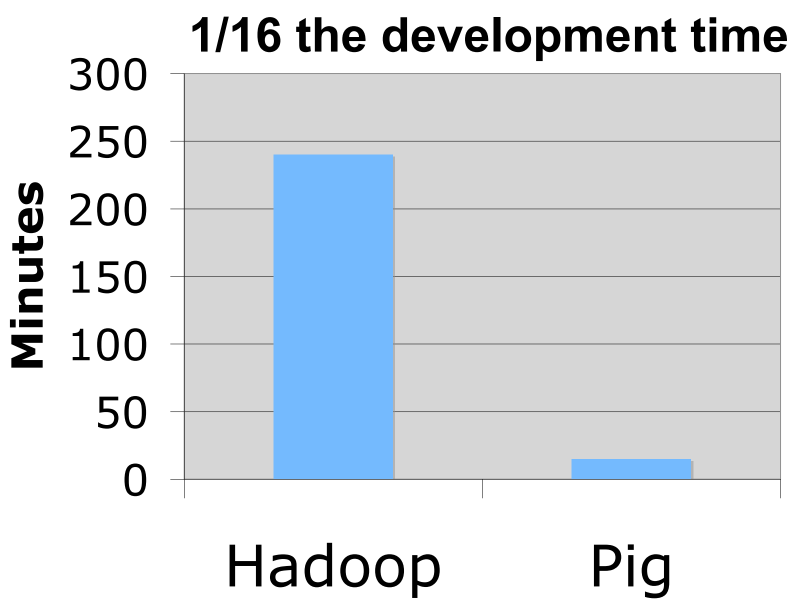
```
import java.io.IOException;                reporter.setStatus("OK");                lp.setOutputKeyClass(Text.class);
```

```
Users  = load 'users' as (name, age);
Fltrd  = filter Users by
         age >= 18 and age <= 25;
Views  = load 'views' as (user, url);
Jnd    = join Fltrd by name, Views by user;
Grpd   = group Jnd by url;
Smmd   = foreach Grpd generate group,
         COUNT(Jnd) as clicks;
Srtd   = order Smmd by clicks desc;
Top5   = limit Srtd 5;
store Top5 into 'top5sites';
```

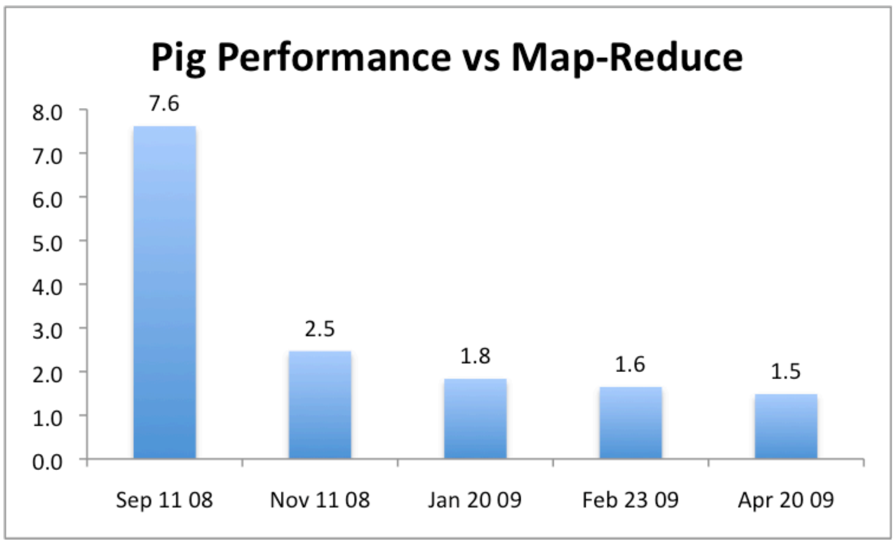
```
first.add(value.substring(1));                lp.setJobName("Load Pages");                }
            else second.add(value.substring(1));                lp.setInputFormat(TextInputFormat.class);                }
```

Comparison





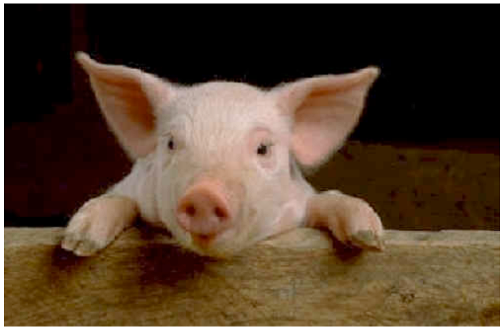
performance 1.5x Hadoop



Ways to Run Pig

- Interactive shell
- Script file
- Embed in host language (e.g., Java)
- *soon:* Graphical editor

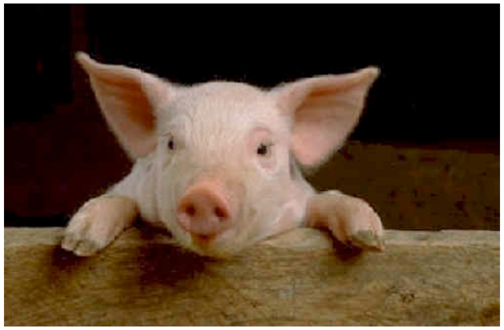
Status



- Open-source implementation
  - **http://hadoop.apache.org/pig**
  - Runs on Hadoop or local machine
  - Active project; many refinements in the works

- Wide adoption in Yahoo
  - 100s of users
  - 1000s of Pig jobs/day
  - 60% of ad-hoc Hadoop jobs are via Pig
  - 40% of production jobs via Pig

Status



- Gaining traction externally
  - log processing & aggregation
  - building text indexes
  - collaborative filtering, applied to image & video recommendation systems

"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

Crazy Ideas

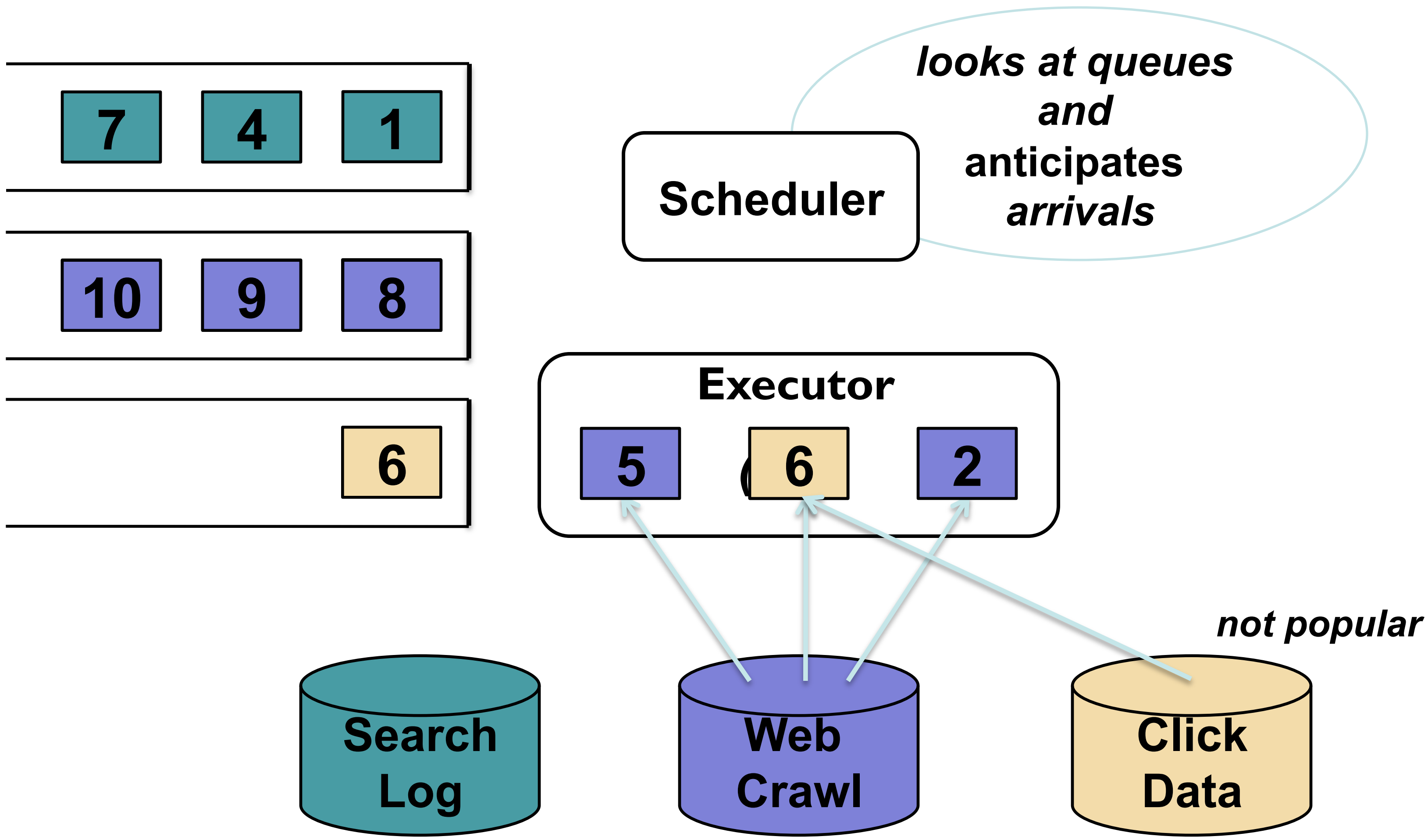
[USENIX'08]
[VLDB'08]
[SIGMOD'09]

Crazy Idea #1
Multi-Program Optimization

Motivation

- User programs repeatedly scan the same data files
  - web crawl
  - search log
- Goal:
  - Reduce redundant IOs, and hence improve overall system throughput
- Approach:
  - Introduce “shared scans” capability
  - Careful scheduling of jobs, to maximize benefit of shared scans

Scheduling Shared Scans



Crazy Idea #2

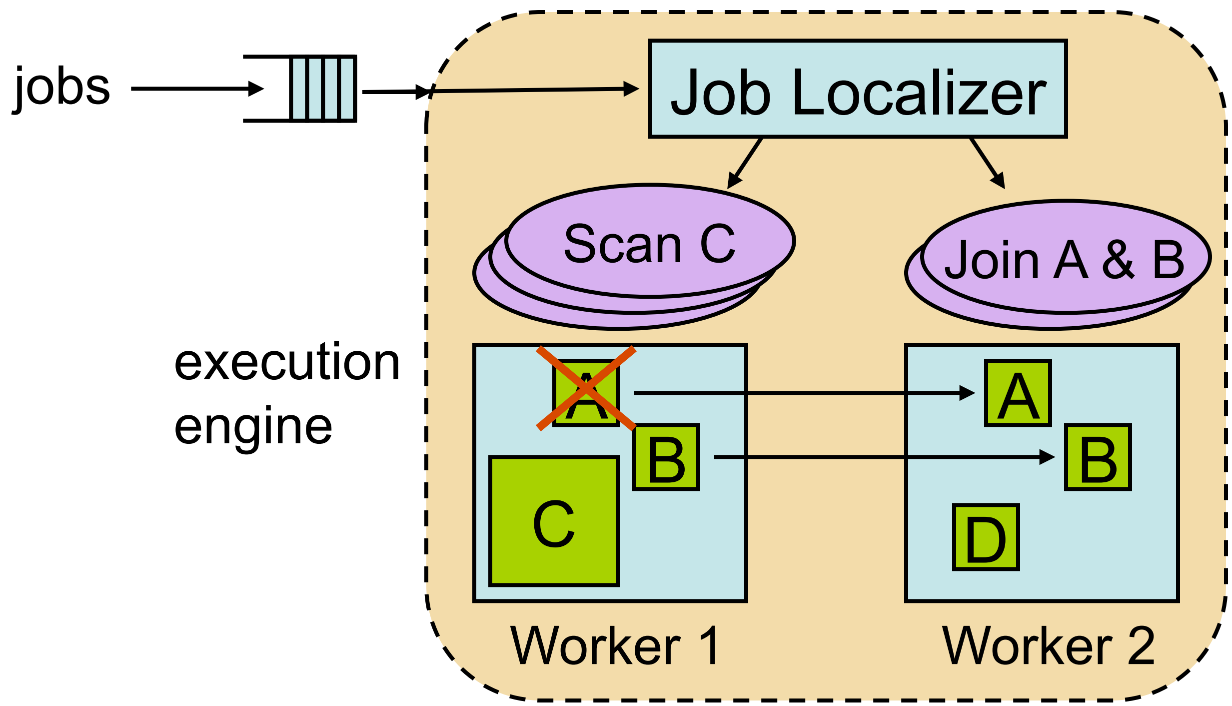
Adaptive Data Placement

Motivation

- Hadoop is good at localizing computation to data, for conventional map-reduce scenarios
- However, advanced query processing operations change this:
  - Pre-hashed join of A & B: co-locate A & B?
  - Frag-repl join of A & B: more replicas of B?

- **Our idea:**
  - Adaptive "pressure-based" mechanism to move data s.t. better locality arises

Adaptive Data Placement

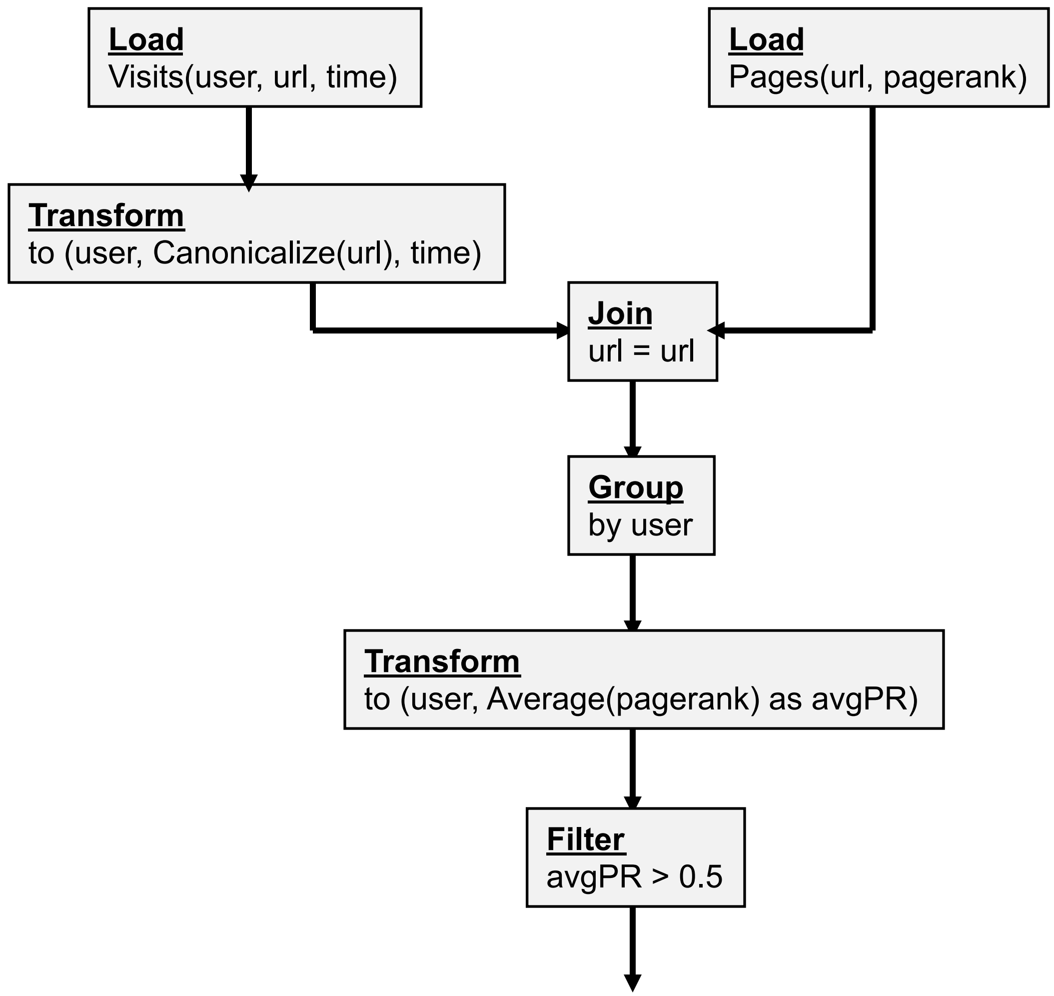


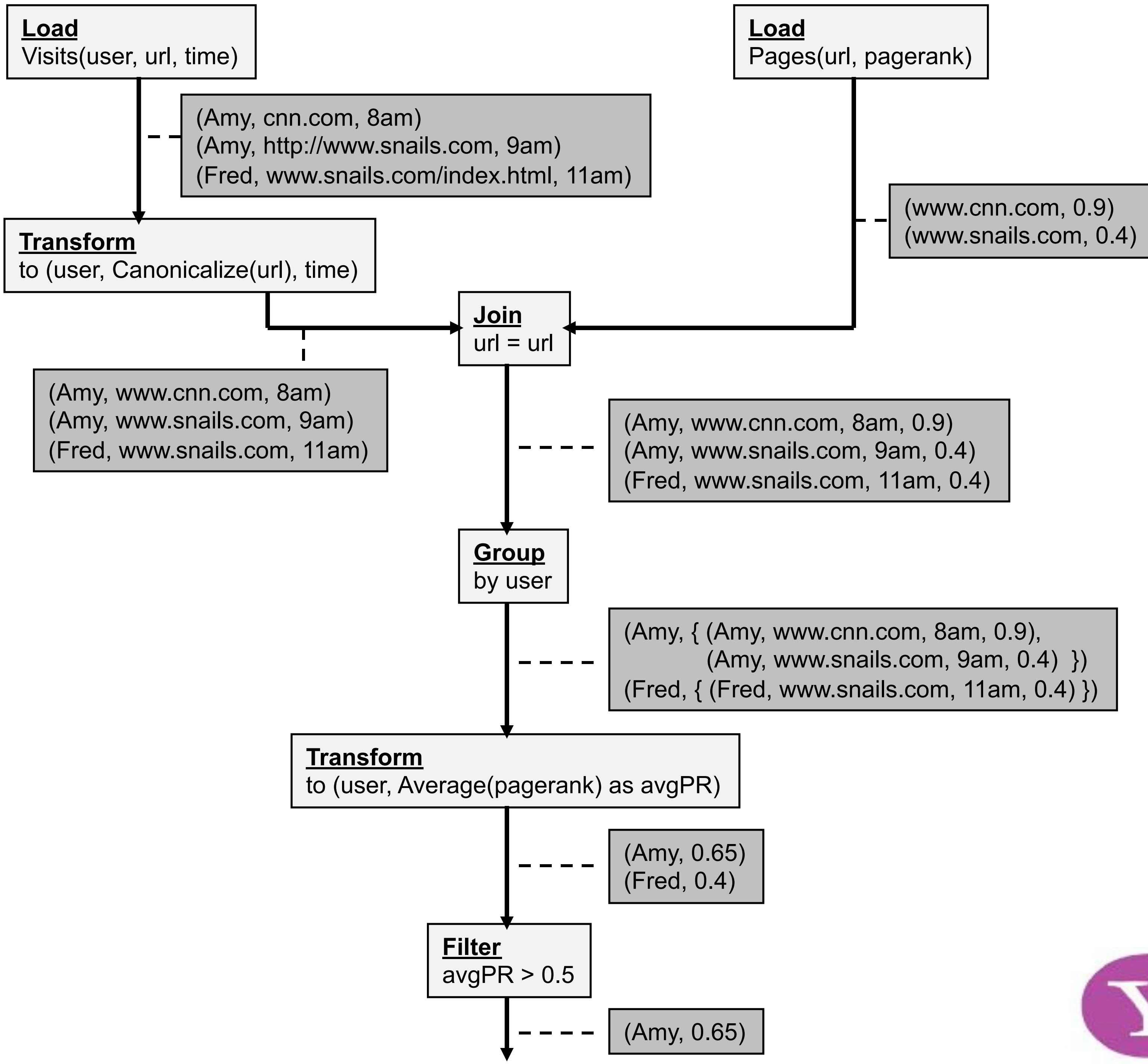
Crazy Idea #3

Automatic Example Generator

Example Program (abstract view)

Find users who tend to visit “good” pages.




Load
Visits(user, url, time)
(Amy, cnn.com, 8am)
(Amy, http://www.snails.com, 9am)
(Fred, www.snails.com/index.html, 11am)
Load
Pages(url, pagerank)
(www.cnn.com, 0.9)
(www.snails.com, 0.4)
Transform
to (user, Canonicalize(url), time)
(Amy, www.cnn.com, 8am)
(Amy, www.snails.com, 9am)
(Fred, www.snails.com, 11am)
Join
url = url
(Amy, www.cnn.com, 8am, 0.9)
(Amy, www.snails.com, 9am, 0.4)
(Fred, www.snails.com, 11am, 0.4)
Group
by user
(Amy, { (Amy, www.cnn.com, 8am, 0.9),
(Amy, www.snails.com, 9am, 0.4) })
(Fred, { (Fred, www.snails.com, 11am, 0.4) })
Transform
to (user, Average(pagerank) as avgPR)
(Amy, 0.65)
(Fred, 0.4)
Filter
avgPR > 0.5
(Amy, 0.65)

Automatic Example Data Generator

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

Talk Summary

- Data-parallel language (“Pig Latin”)
  - Sequence of data transformation steps
  - Users can plug in custom code
- Research nuggets
  - Joint scheduling of related programs, to amortize IOs
  - Adaptive data placement, to enhance locality
  - Automatic example data generator, to make user’s life easier

Credits

<u>Yahoo! Grid Team</u>

project leads:

Alan Gates

Olga Natkovich

<u>Yahoo! Research</u>

project leads:

Chris Olston

Utkarsh Srivastava

Ben Reed