Pig latin

Riccardo Torlone

Università Roma Tre

Credits: C. Olston et al. (Yahoo! Research) & James Jolly (Univ. of Wisconsin-Madison).

Motivation

- Big Data
  - Could be from multiple sources and in different formats
  - Data sets are typically huge
  - Don't need to alter the original data; just need to do reads
  - May be temporary; could discard the data set after analysis
- Data analysis goals
  - Quick: exploit parallel processing power of a distributed system
  - Easy
    - Be able to write a program or query without a huge learning curve
    - Have some common analysis tasks predefined
  - Flexible
    - Transform a dataset(s) into a workable structure without much overhead
    - Perform customized processing
  - Transparent: have a say in how the data processing is executed on the system

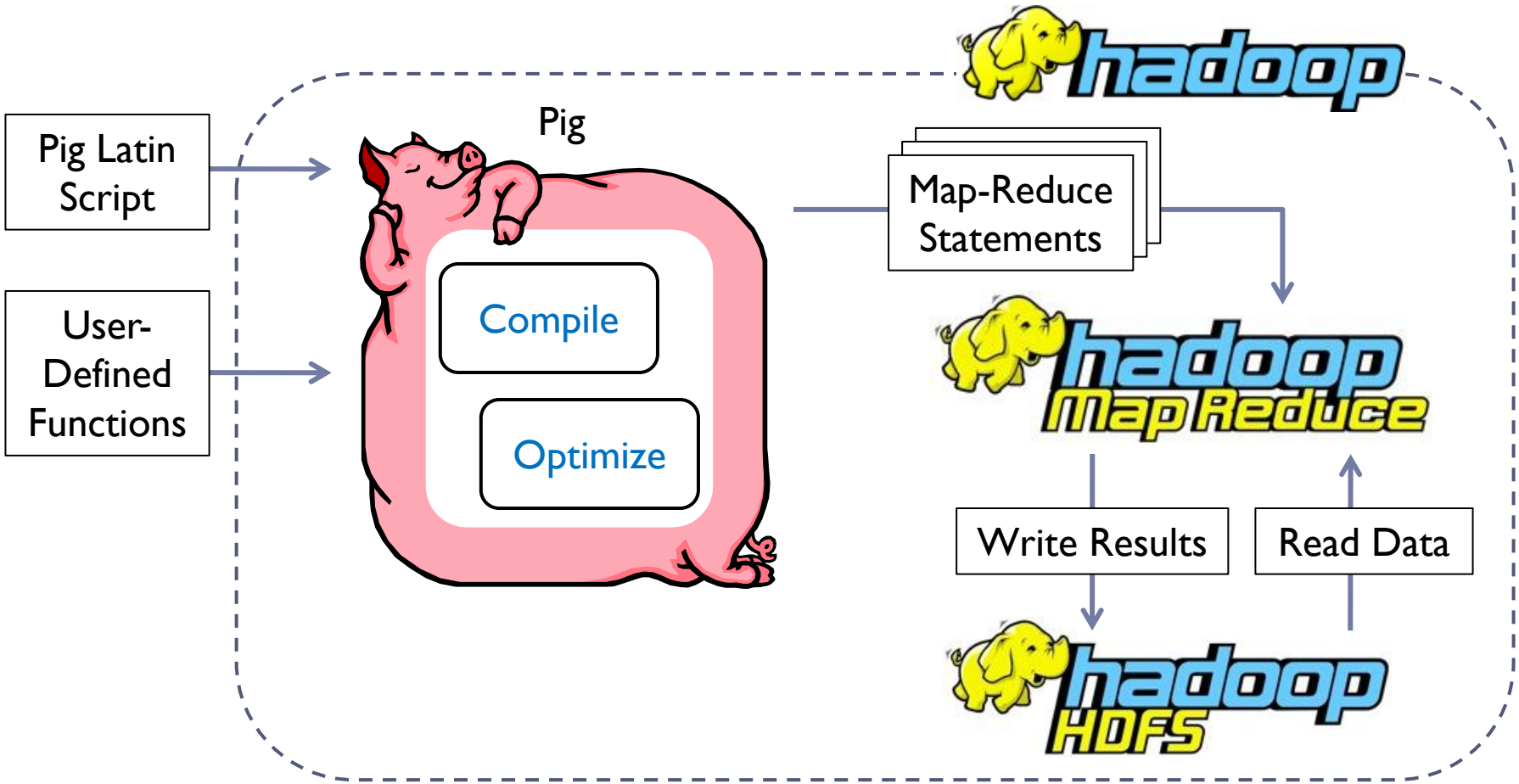
Solution?

- Relational Distributed Databases
  - Parallel database products expensive
  - Rigid schemas
  - Data has to be imported into system-managed tables
  - Processing requires SQL query construction and may not scale
- Map-Reduce
  - Relies on custom code for even common operations
  - Need to do workarounds for tasks that have different data flows other than the expected Map ➔Combine➔Reduce
- Take the best of both SQL and Map-Reduce; combine high-level declarative querying with low-level procedural programming
  - Pig Latin!

What is Pig Latin?

- Set-oriented data transformation language
  - primitives filter, combine, split, and order data
  - users describe transformations in steps
  - each set transformation is stateless
- Flexible data model
  - nested bags of tuples
  - semi-structured datatypes
- Extensible
  - supports user-defined functions
- Executable in Hadoop
  - A compiler converts Pig Latin scripts to MapReduce dataflows

Big picture


hadoop
Pig
Pig Latin Script
User-Defined Functions
Compile
Optimize
Map-Reduce Statements
hadoop MapReduce
Write Results
Read Data
hadoop HDFS

How is it used in practice?

- Useful for computations across large, distributed datasets
  - abstracts away details of execution framework
  - users can change order of steps to improve performance
- Used in tandem with Hadoop and HDFS
  - transformations converted to MapReduce dataflows
  - HDFS tracks where data is stored
  - operations scheduled nearby their data

A practical example

- Given two datasets:
  - list of words and their frequency of appearance on webpages
  - list of users and webpages they visit
- Let's find words users might be interested in lately

Dataset: words and their frequency of appearance…

website	word	frequency	date
news.bbc.co.uk	obama	0.010	20151005
abcnews.go.com	scheme	0.025	20151010
abcnews.go.com	bombing	0.021	20151006
www.foxnews.com	clinton	0.001	20151006
www.cnn.com	trump	0.031	20151017
www.cnn.com	obama	0.001	20151002
www.reuters.com	clinton	0.012	20150921
abcnews.go.com	congress	0.002	20150927
www.reuters.com	clinton	0.012	20150921
www.foxnews.com	clinton	0.001	20151006
www.latimes.com	abortion	0.001	20151015
www.latimes.com	attack	0.010	20151015
www.reuters.com	obama	0.005	20150917
www.foxnews.com	economy	0.038	20151006

Dataset: webpages users visit…

website	user
www.reuters.com	bill
news.bbc.co.uk	mike
www.cnn.com	mike
www.foxnews.com	bill
www.reuters.com	drew
www.latimes.com	james
abcnews.go.com	james

Loading word frequency data...

```
freqs = LOAD '/home/jolly/TestData/NewsWords.txt' USING PigStorage('\t')
        AS (website_indexed, word, freq, date);
```

(news.bbc.co.uk, obama, 0.010, 20151005)
(abcnews.go.com, scheme, 0.025, 20151010)
(abcnews.go.com, bombing, 0.021, 20151006)
(www.foxnews.com, clinton, 0.001, 20151006)
(www.cnn.com, trump, 0.031, 20151017)
(www.cnn.com, obama, 0.001, 20151002)
(www.reuters.com, clinton, 0.012, 20150921)
(abcnews.go.com, congress, 0.002, 20150927)
(www.reuters.com, clinton, 0.012, 20150921)
(www.foxnews.com, clinton, 0.001, 20151006)
(www.latimes.com, abortion, 0.001, 20151015)
(www.latimes.com, attack, 0.010, 20151015)
(www.reuters.com, obama, 0.005, 20150917)
(www.foxnews.com, economy, 0.038, 20151006)

Hmm, we have some repeats…

(news.bbc.co.uk, obama, 0.010, 20151005)

(abcnews.go.com, scheme, 0.025, 20151010)

(abcnews.go.com, bombing, 0.021, 20151006)

(www.foxnews.com, clinton, 0.001, 20151006)

(www.cnn.com, trump, 0.031, 20151017)

(www.cnn.com, obama, 0.001, 20151002)

(www.reuters.com, clinton, 0.012, 20150921)

(abcnews.go.com, congress, 0.002, 20150927)

(www.reuters.com, clinton, 0.012, 20150921)

(www.foxnews.com, clinton, 0.001, 20151006)

(www.latimes.com, abortion, 0.001, 20151015)

(www.latimes.com, attack, 0.010, 20151015)

(www.reuters.com, obama, 0.005, 20150917)

(www.foxnews.com, economy, 0.038, 20151006)

No more duplicate data!

```
distinct_freqs = DISTINCT freqs;
```

(www.cnn.com, obama, 0.001, 20151002)
(www.cnn.com, trump, 0.031, 20151017)
(abcnews.go.com, scheme, 0.025, 20151010)
(abcnews.go.com, bombing, 0.021, 20151006)
(abcnews.go.com, congress, 0.002, 20150927)
(news.bbc.co.uk, obama, 0.010, 20151005)
(www.foxnews.com, clinton, 0.001, 20151006)
(www.foxnews.com, economy, 0.038, 20151006)
(www.latimes.com, attack, 0.010, 20151015)
(www.latimes.com, abortion, 0.001, 20151015)
(www.reuters.com, clinton, 0.012, 20150921)
(www.reuters.com, obama, 0.005, 20150917)

Hmm, these tuples are old…

(www.cnn.com, obama, 0.001, 20151002)

(www.cnn.com, trump, 0.031, 20151017)

(abcnews.go.com, scheme, 0.025, 20151010)

(abcnews.go.com, bombing, 0.021, 20151006)

(abcnews.go.com, congress, 0.002, 20150927)

(news.bbc.co.uk, obama, 0.010, 20151005)

(www.foxnews.com, clinton, 0.001, 20151006)

(www.foxnews.com, economy, 0.038, 20151006)

(www.latimes.com, attack, 0.010, 20151015)

(www.latimes.com, abortion, 0.001, 20151015)

(www.reuters.com, clinton, 0.012, 20150921)

(www.reuters.com, obama, 0.005, 20150917)

… and these tuples are not very significant

(www.cnn.com, obama, 0.001, 20151002)

(www.cnn.com, trump, 0.031, 20151017)

(abcnews.go.com, scheme, 0.025, 20151010)

(abcnews.go.com, bombing, 0.021, 20151006)

(abcnews.go.com, congress, 0.002, 20150927)

(news.bbc.co.uk, obama, 0.010, 20151005)

(www.foxnews.com, clinton, 0.001, 20151006)

(www.foxnews.com, economy, 0.038, 20151006)

(www.latimes.com, attack, 0.010, 20151015)

(www.latimes.com, abortion, 0.001, 20151015)

(www.reuters.com, clinton, 0.012, 20150921)

(www.reuters.com, obama, 0.005, 20150917)

Let's filter them out.

```
important_freqs = FILTER distinct_freqs
                  BY date > 20151001 AND freq > 0.002;
```

(www.cnn.com, trump, 0.031, 20151017)

(abcnews.go.com, scheme, 0.025, 20151010)

(abcnews.go.com, bombing, 0.021, 20151006)

(news.bbc.co.uk, obama, 0.010, 20151005)

(www.foxnews.com, economy, 0.038, 20151006)

(www.latimes.com, attack, 0.010, 20151015)

Hmm, we don’t need these anymore…

(www.cnn.com, trump, 0.031, 20151017)

(abcnews.go.com, scheme, 0.025, 20151010)

(abcnews.go.com, bombing, 0.021, 20151006)

(news.bbc.co.uk, obama, 0.010, 20151005)

(www.foxnews.com, economy, 0.038, 20151006)

(www.latimes.com, attack, 0.010, 20151015)

Let's project them out.

```
websites_to_words = FOREACH important_freqs
                    GENERATE website_indexed, word;
```

(www.cnn.com, trump)

(abcnews.go.com, scheme)

(abcnews.go.com, bombing)

(news.bbc.co.uk, obama)

(www.foxnews.com, economy)

(www.latimes.com, attack)

Now we are ready to join our lists.

Websites to Users

(news.bbc.co.uk, mike)
(www.cnn.com, mike)
(www.foxnews.com, bill)
(www.reuters.com, drew)
(www.latimes.com, james)
(abcnews.go.com, james)

Websites to Words

(www.cnn.com, trump)
(abcnews.go.com, scheme)
(abcnews.go.com, bombing)
(news.bbc.co.uk, obama)
(www.foxnews.com, economy)
(www.latimes.com, attack)

Joining on website: finding words interesting to users…

```
users_to_words_equijoin = JOIN websites_to_users BY website_visited,
                                    websites_to_words BY website_indexed;
users_to_words = FOREACH users_to_words_equijoin
                    GENERATE user, word;
```

(mike, trump)

(james, scheme)

(james, bombing)

(mike, obama)

(bill, economy)

(james, attack)

Let’s group our results.

```
interests = GROUP users_to_words BY user;
```

(bill, {(bill, economy)})

(mike, {(mike, trump), (mike, obama)})

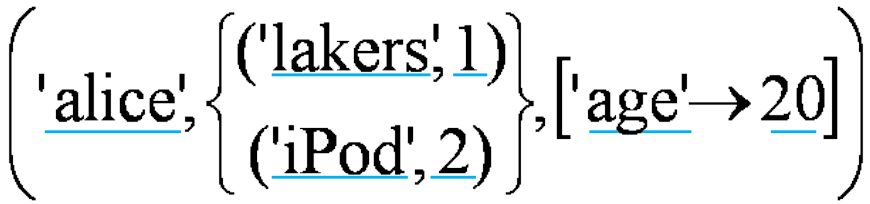
(james, {(james, scheme), (james, bombing), (james, attack)})

The whole PIG Latin program

```
freqs = LOAD '/home/jolly/TestData/NewsWords.txt' USING PigStorage(',')
        AS (website_indexed, word, freq, date);
distinct_freqs = DISTINCT freqs;
important_freqs = FILTER distinct_freqs
                  BY date > 20151001 AND freq > 0.002;
websites_to_words = FOREACH important_freqs
                    GENERATE website_indexed, word;
users_to_words_equijoin = JOIN websites_to_users BY website_visited,
                              websites_to_words BY website_indexed;
users_to_words = FOREACH users_to_words_equijoin
                 GENERATE user, word;
interests = GROUP users_to_words BY user;
```

Data model

- Atom – simple atomic value (ie: number or string)
- Tuple – sequence of fields; each field any type
- Bag – collection of tuples
  - Duplicates possible
  - Tuples in a bag can have different field lengths and field types
- Map – collection of key-value pairs
  - Key is an atom; value can be any type



Data model

- Use of data structures
  - Increased flexibility in data representation
- Fully nested
  - More natural for procedural programmers (target user) than normalization
  - Facilitates ease of writing user-defined functions
- No schema required

Data model

- User-Defined Functions (UDFs)
  - Can be used in many Pig Latin statements
  - Useful for custom processing tasks
  - Can use non-atomic values for input and output
  - Can be written in Java, Python, and JavaScript

Speaking Pig Latin

- LOAD
  - Input is assumed to be a bag (sequence of tuples)
  - Can specify a serializer with "USING"
  - Can provide a schema with "AS"

```
newBag = LOAD 'filename'
            <USING functionName()>
            <AS (fieldName1, fieldName2,...)>;
```

Speaking Pig Latin

- FOREACH
  - Apply some processing to each tuple in a bag
  - Each field can be:
    - A fieldname of the bag
    - A constant
    - A simple expression (ie: f1+f2)
    - A predefined function (ie: SUM,AVG, COUNT, FLATTEN)
    - A UDF (ie: tax(gross, percentage))

```
newBag = FOREACH bagName
         GENERATE field1, field2, ...;
```

Speaking Pig Latin

- FILTER
  - Select a subset of the tuples in a bag

    ```
    newBag = FILTER bagName BY expression;
    ```

  - Expression uses simple comparison operators (==, !=, <, >, …) and logical connectors (AND, NOT, OR)

    ```
    some_apples = FILTER apples BY colour != 'red';
    ```

  - Can use UDFs

    ```
    some_apples = FILTER apples BY NOT isRed(colour);
    ```

Speaking Pig Latin

- GROUP
  - groups together tuples that have the same group key

    ```
    newBag = GROUP bagName BY expression;
    ```

  - Usually the expression is a field

    ```
    stat1 = GROUP students BY age;
    ```

  - Expression can use operators

    ```
    stat2 = GROUP employees BY salary + bonus;
    ```

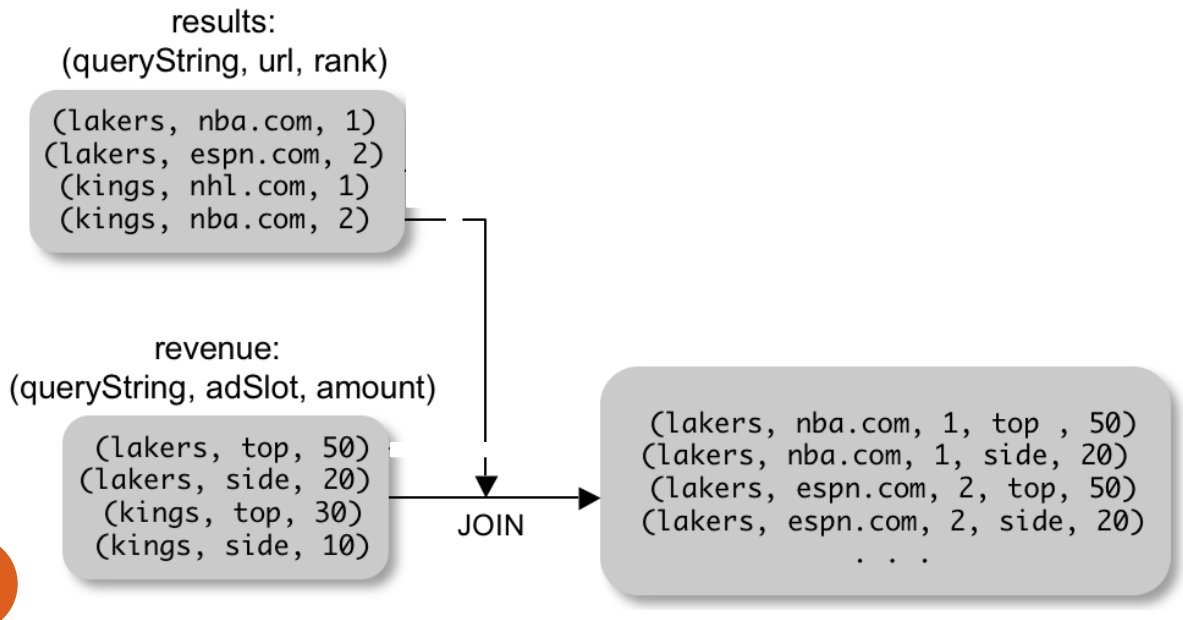
  - Can use UDFs

    ```
    stat3 = GROUP employees BY netsal(salary, taxes);
    ```

Speaking Pig Latin

- JOIN
  - join two datasets by a common attribute

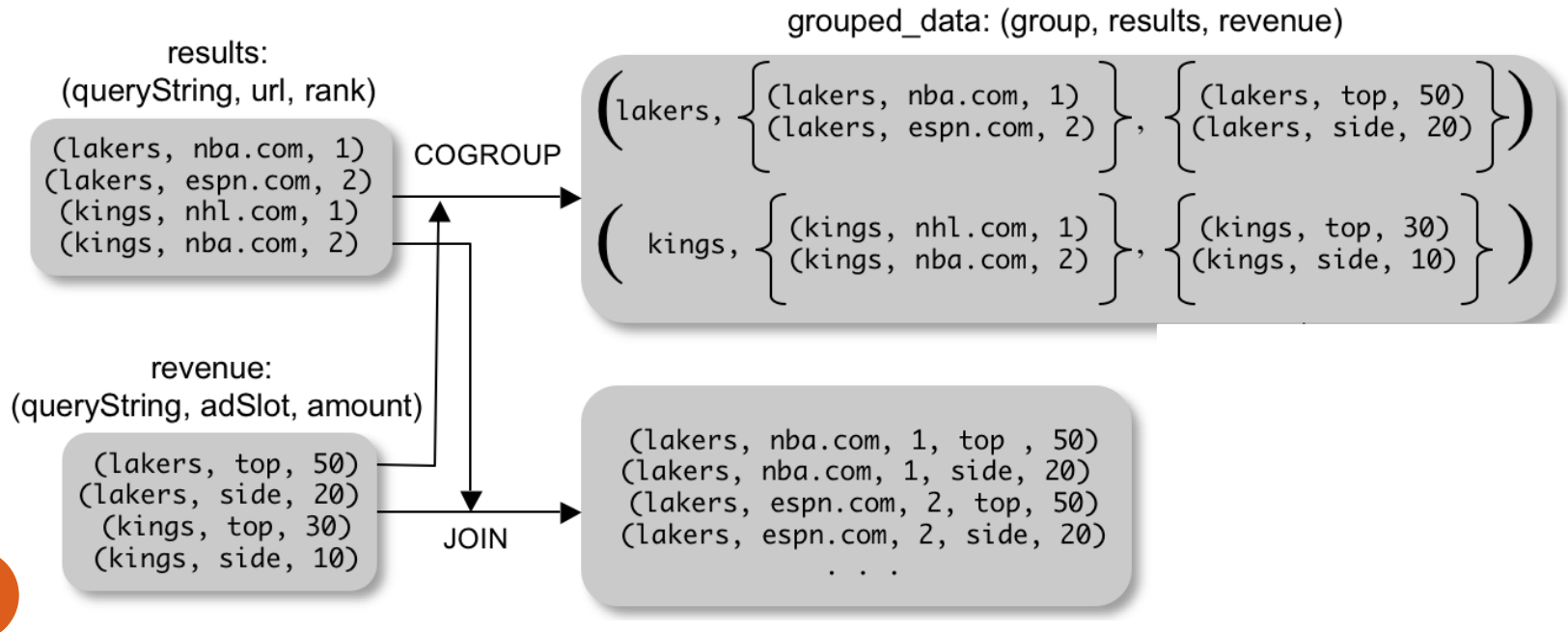
```
joined_data = JOIN results BY queryString,
                   revenue BY queryString;
```



Speaking Pig Latin

- COGROUP
  - Group two datasets together by a common attribute
  - Groups data into nested bags

```
grouped_data = COGROUP results BY queryString,
                       revenue BY queryString;
```



Speaking Pig Latin

- STORE (& DUMP)
  - Output data to a file (or screen)

    STORE bagName INTO ‘filename’ <USING deserializer()>;

- Other Commands (incomplete)
  - UNION – return the union of two or more bags
  - CROSS – take the cross product of two or more bags
  - ORDER – order tuples by a specified field(s)
  - DISTINCT – eliminate duplicate tuples in a bag
  - LIMIT – Limit results to a subset

How does it work?

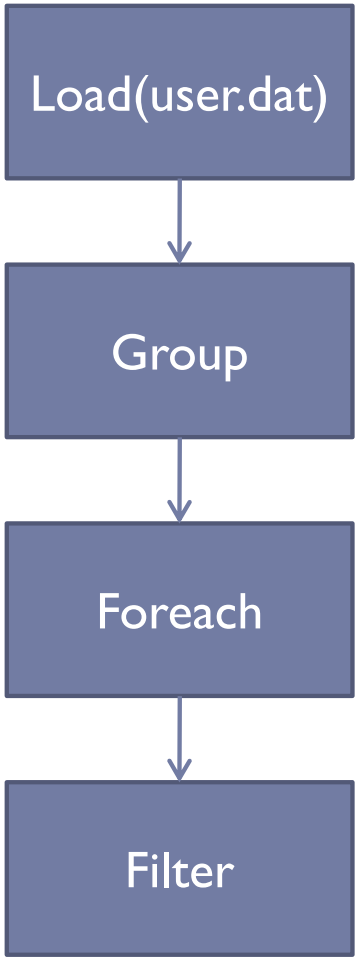
- Many programs can be executed using a single MapReduce job;
- Logic factored into MapReduce jobs:
  - mapper processes run on machines with input tuples;
  - input tuples processed using MAP function
    - producing intermediate tuples
  - intermediate tuples grouped together
    - transferred to reducer nodes
  - reducer processes intermediate tuples with REDUCE function

Compilation

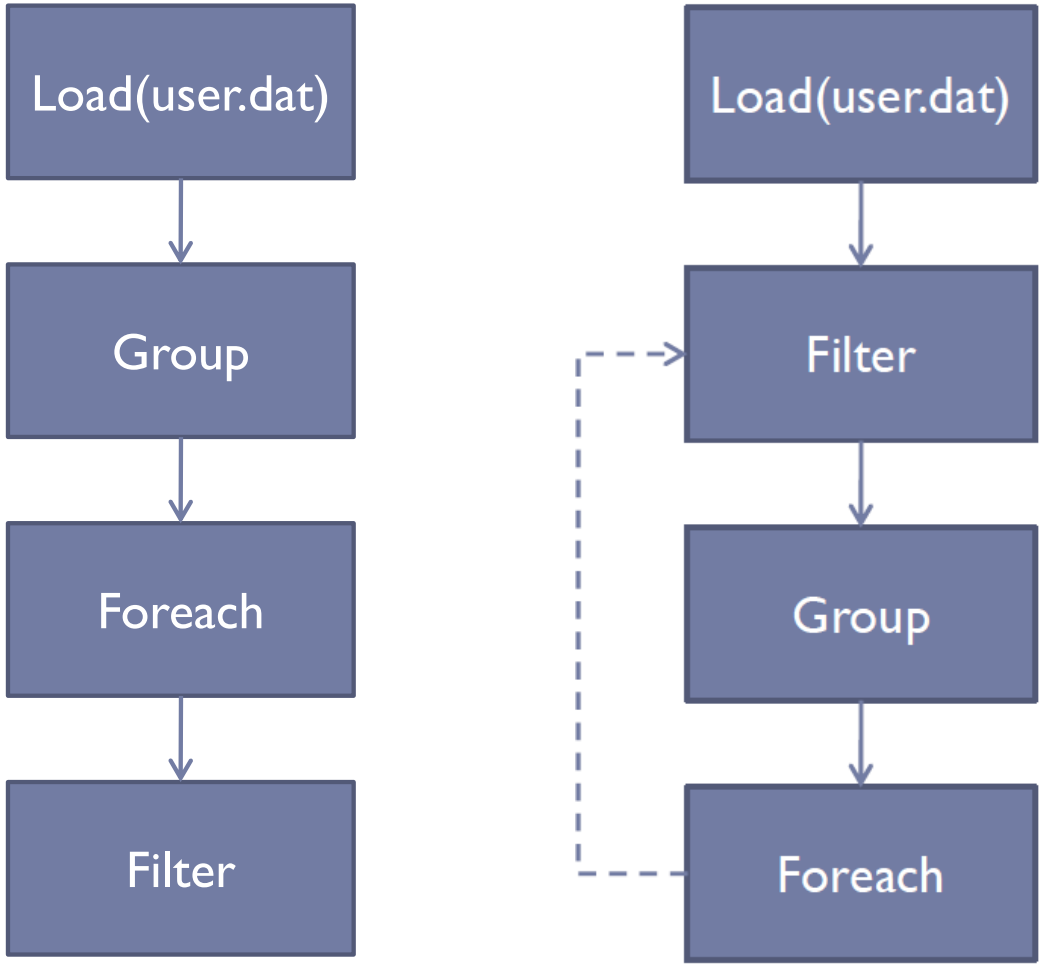
- Pig system does two tasks:
  - Builds a Logical Plan from a Pig Latin script
    - Supports execution platform independence
    - No processing of data performed at this stage
  - Compiles the Logical Plan to a Physical Plan and Executes
    - Convert the Logical Plan into a series of Map-Reduce statements to be executed by Hadoop Map-Reduce

Building a Logical Plan: Example

```
A = LOAD ‘user.dat’ AS (name, age, city);

B = GROUP A BY city;

C = FOREACH B GENERATE stat AS city, COUNT(A);

D = FILTER C BY city IS ‘kitchener’ OR city IS ‘waterloo’;

STORE D INTO ‘local_user_count.dat’;
```

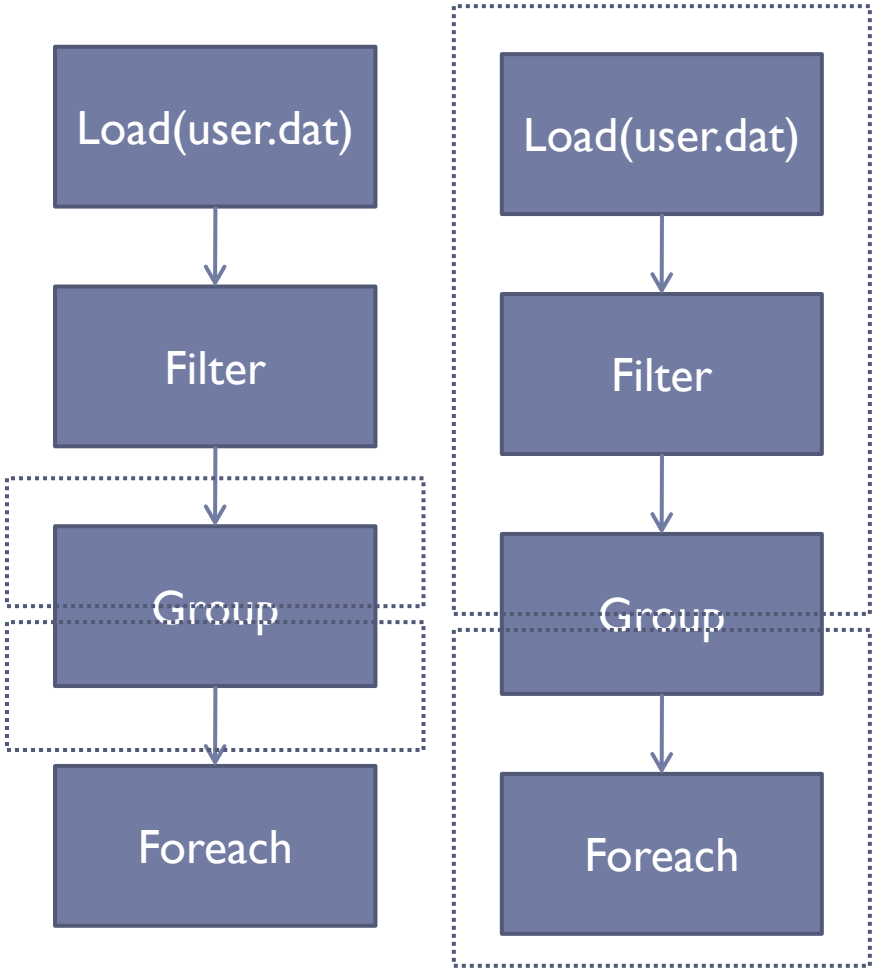


Optimization

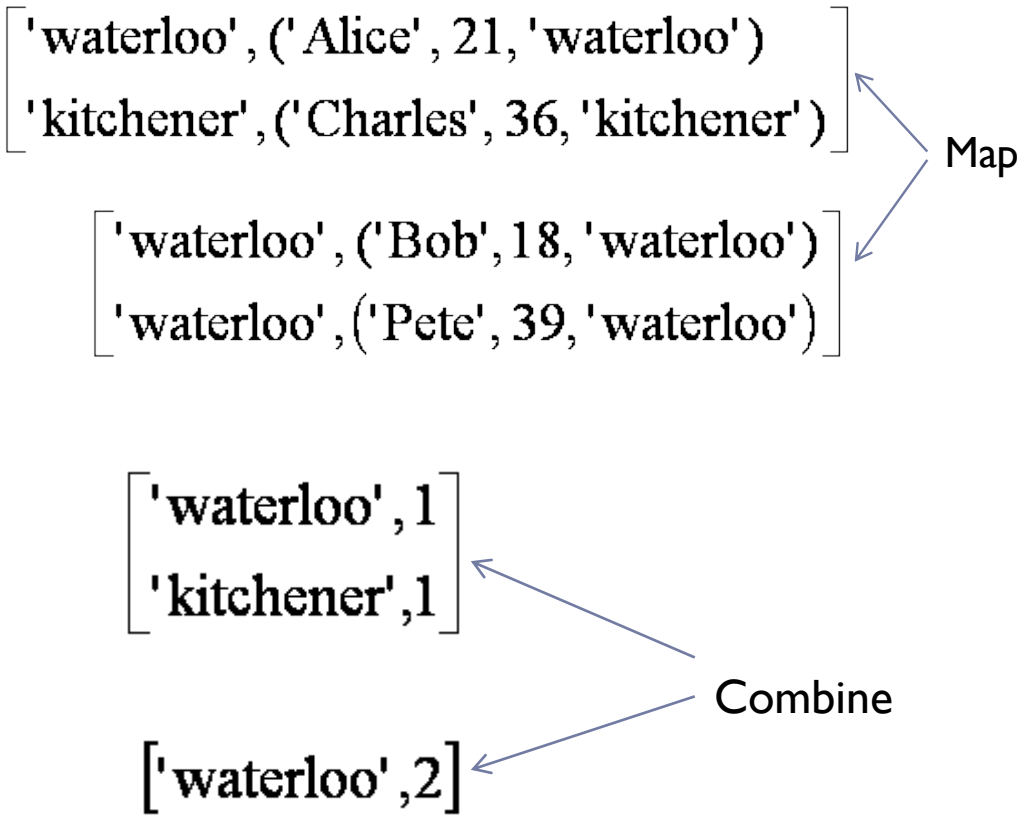


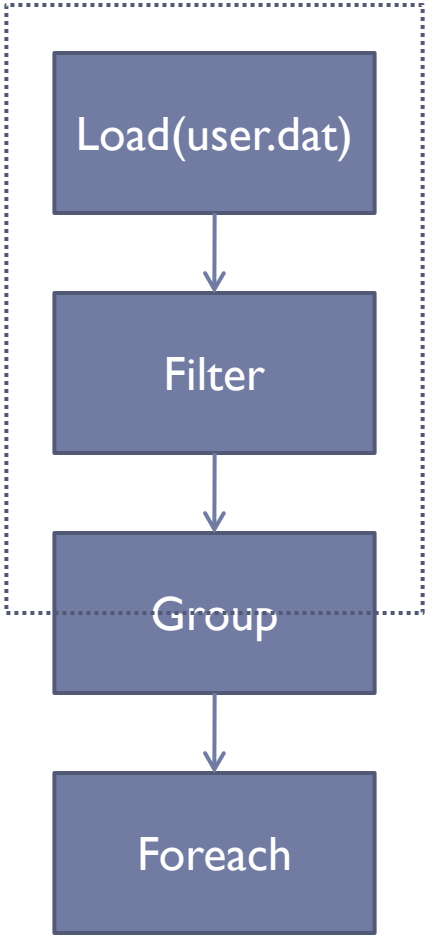
Building a Physical Plan

- Step 1: Create a map-reduce job for each GROUP
- Step 2: Push other commands into the map and reduce functions where possible
- May be the case certain commands require their own map-reduce job (ie: ORDER needs two map- reduce jobs)

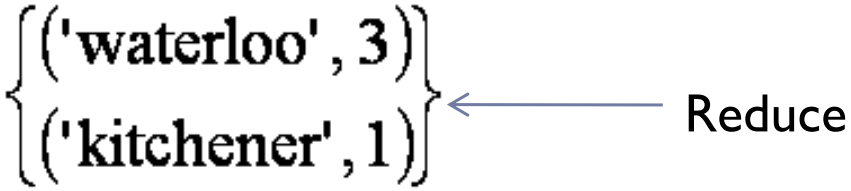


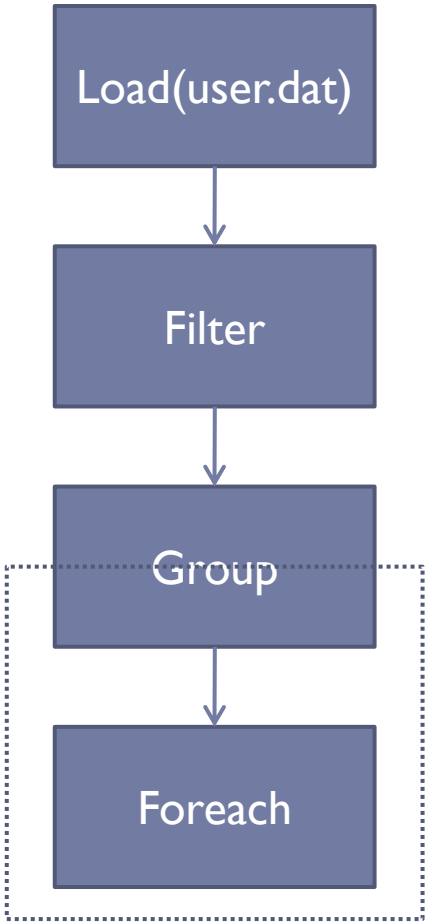
Run-time


'waterloo', ('Alice', 21, 'waterloo')
'kitchener', ('Charles', 36, 'kitchener')
'waterloo', ('Bob', 18, 'waterloo')
'waterloo', ('Pete', 39, 'waterloo')
Map
'waterloo', 1
'kitchener', 1
['waterloo', 2]
Combine

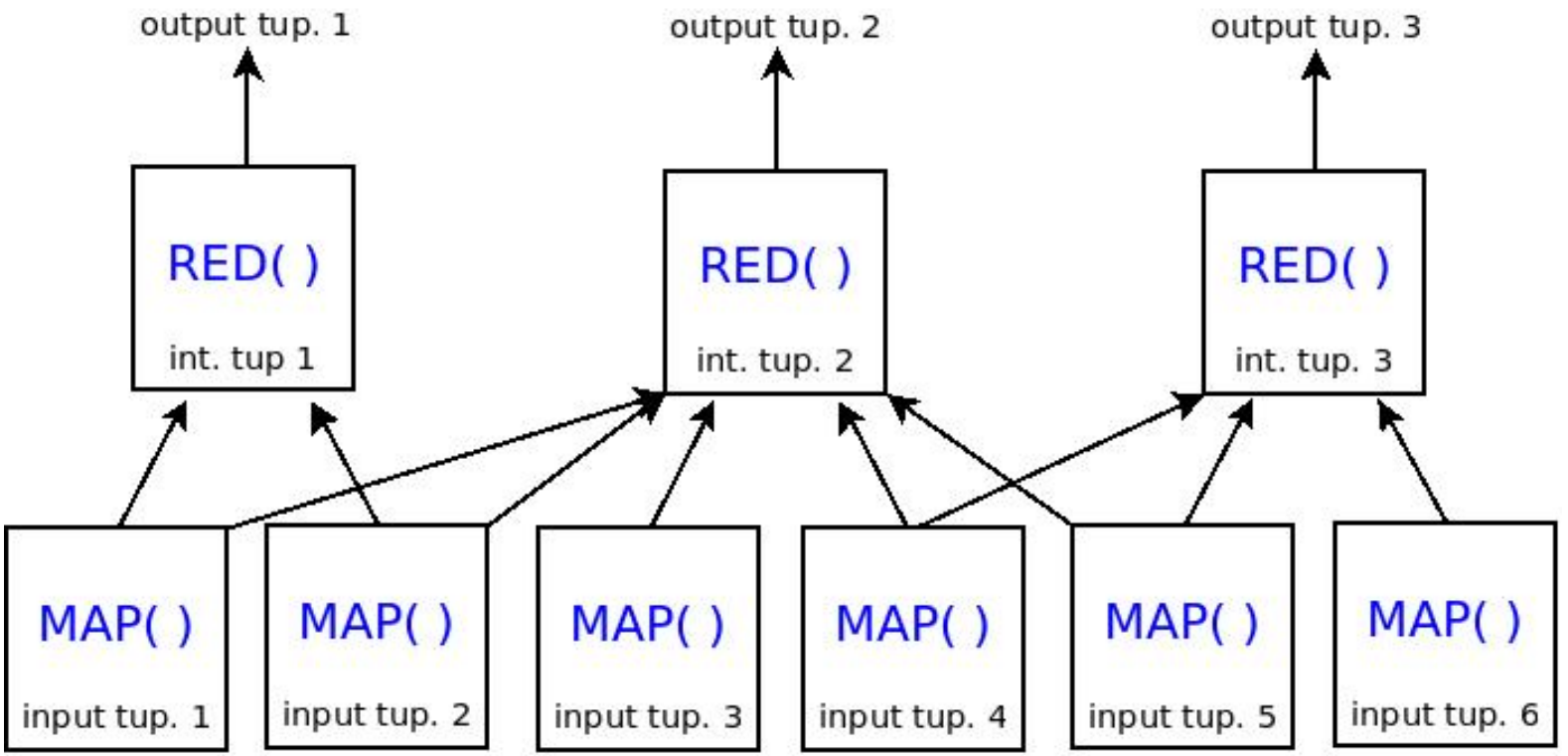

Load(user.dat)
Filter
Group
Foreach

Run-time


('waterloo', 3)
('kitchener', 1)
Reduce


Load(user.dat)
Filter
Group
Foreach

Example message traffic…



Why Pig Latin? Why not Java?

- We could just supply MAP( ) and REDUCE( ) to Java API…
- Because it is easier!
- Pig Latin allows you to:
  - describe long tasks
    - in a friendly scripting language
  - use many built-in datatypes
    - support for semi-structured data
  - use many built-in functions
    - filters, projections, joins, unions, splits, etc.
    - tends to make user-defined functions simpler

Why Pig Latin? Why not SQL?

- Source data might be not in a relational database
- Pig Latin:
  - is imperative
    - lets users manually tune query execution plan
  - doesn't need a schema
    - can easily read, write, and represent semi-structured data

Pros and cons

- Pros:
  - a high-level platform for creating MapReduce programs
  - abstracts the programming from Java MapReduce
  - an SQL-like syntax
  - a procedural style
  - extensible with Java UDF
- Cons:
  - it is not always desirable to add an abstract layer
  - performances!!

Performance evaluation

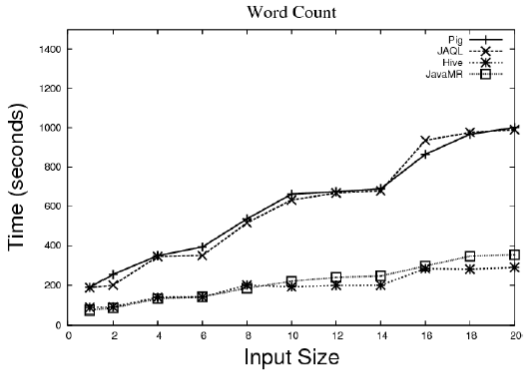


Fig. 6. Word Count Scale Up - Uniform Distribution

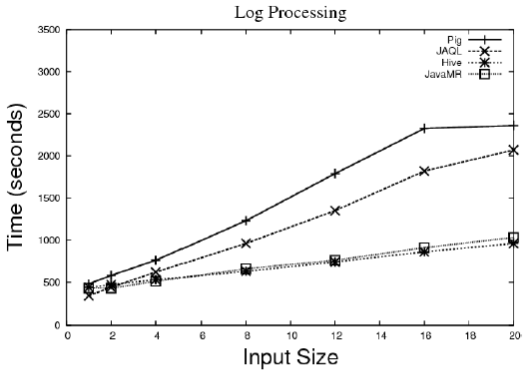


Fig. 7. Web Log Processing Scale Up - Uniform Distribution

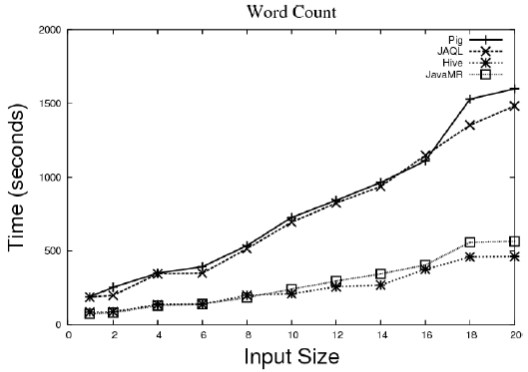


Fig. 8. Word Count Scale Up - Skewed Distribution

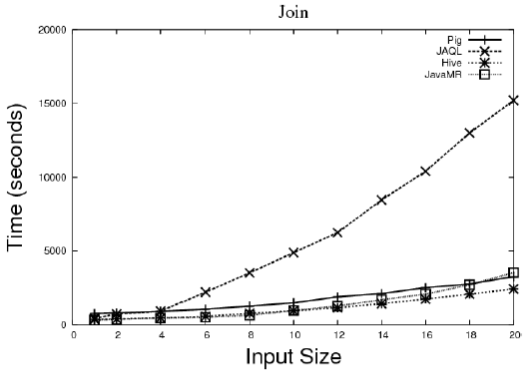
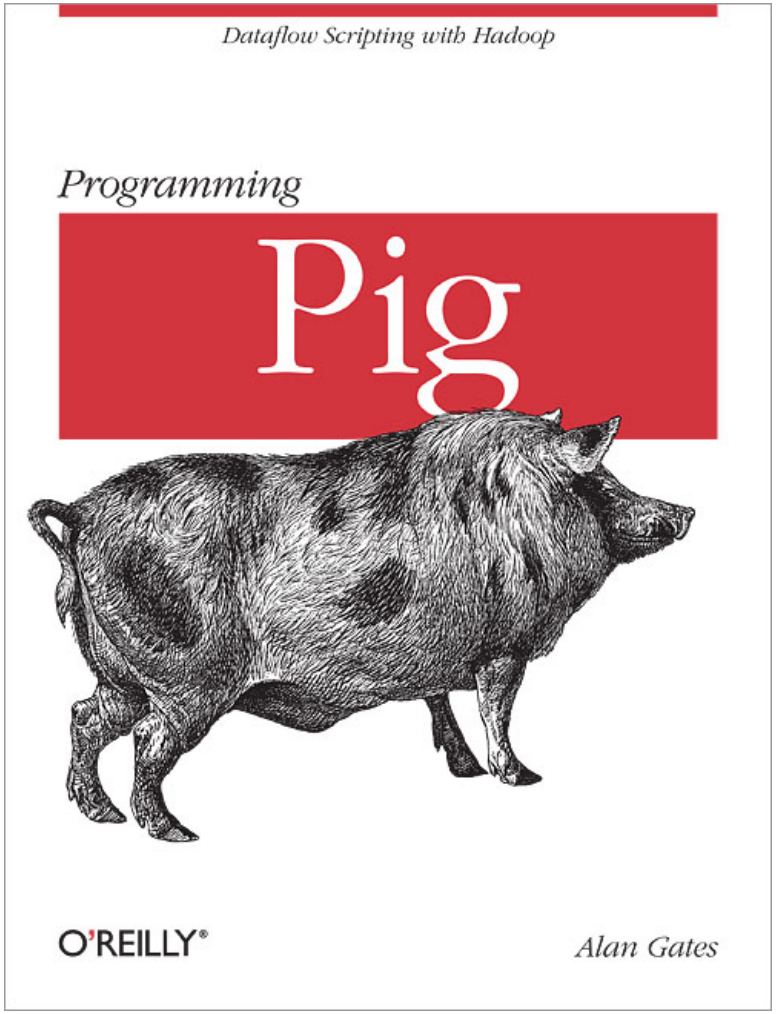


Fig. 9. Dataset Join Scale Up - Skewed Distribution

References


Dataflow Scripting with Hadoop
Programming
Pig
O'REILLY®
Alan Gates