Pig Latin: A Not-So-Foreign Language for Data Processing

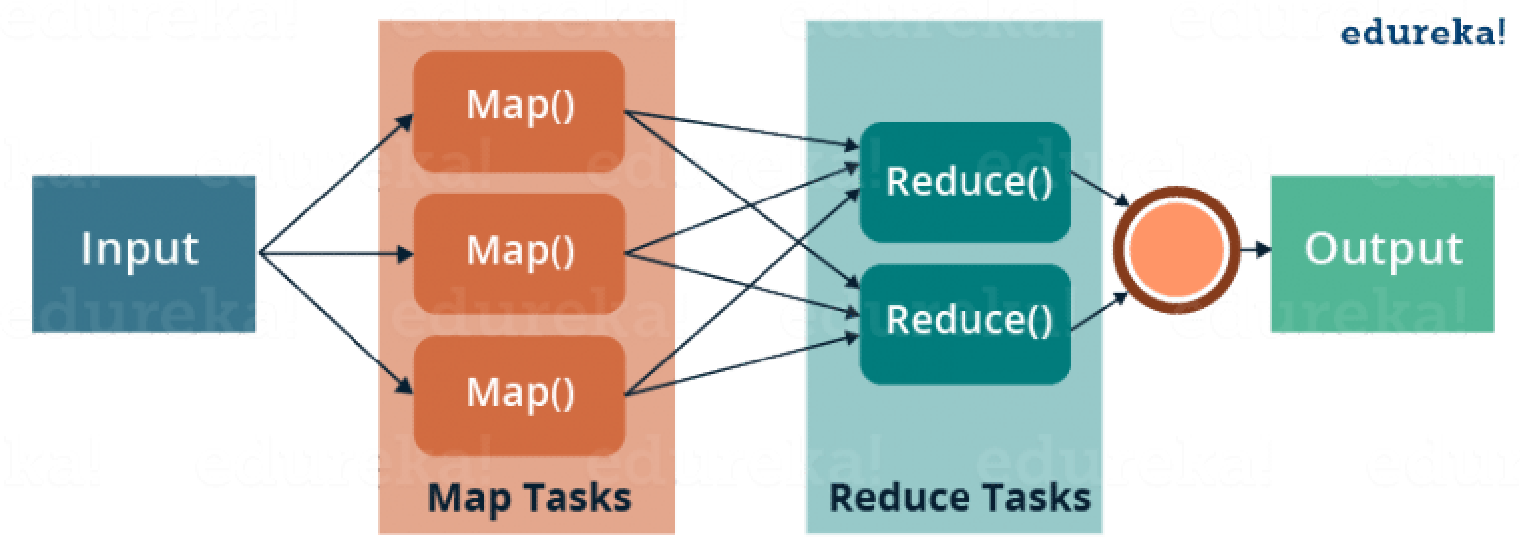
Ravi Kumar, Christopher Olston, Benjamin Reed, Utkarsh Srivastava, and Ansrew Tomkins

Published by the Association for Computing Machinery

Taylor Doering, Charlotte Hettrich, Amanda Lee, Joanita Nandaula

Declarative SQL-style Language

Developers find unnatural



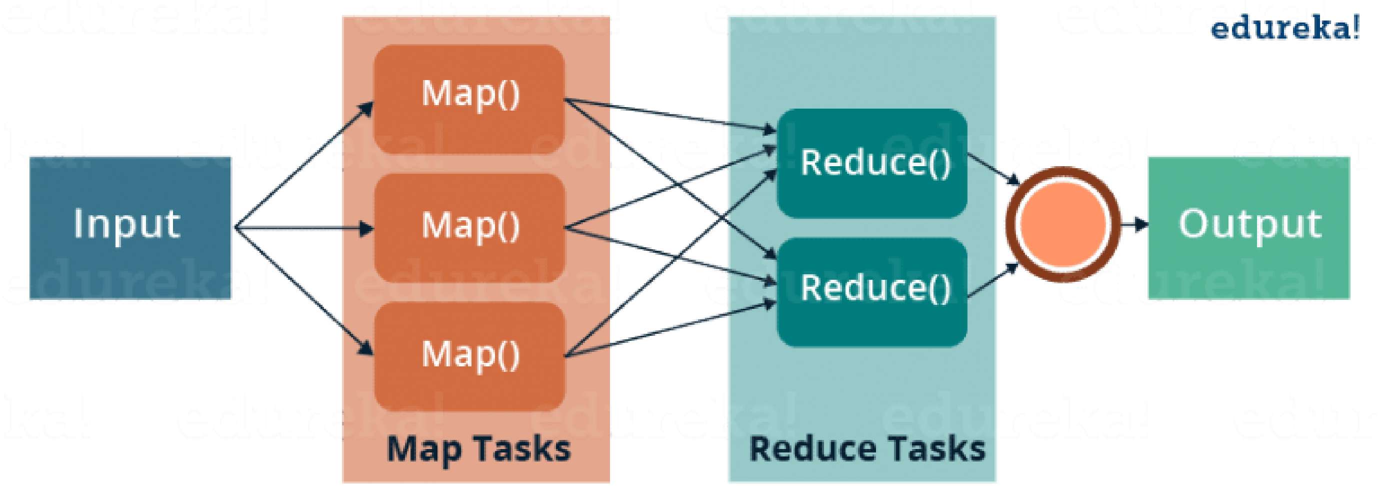
Procedural Map-Reduce Model

Low-level and rigid

Pig-Latin

Declarative SQL-style Language

Developers find unnatural



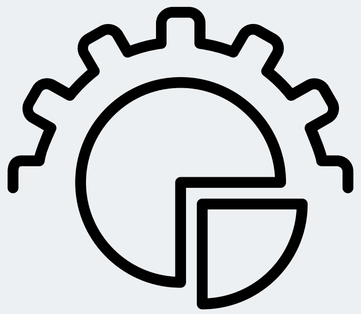
Procedural Map-Reduce Model

Low-level and rigid

Why it Matters

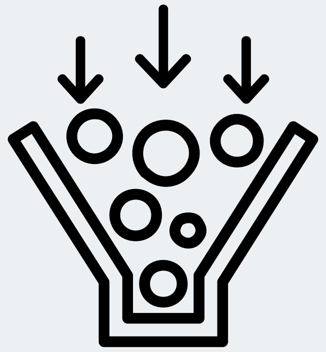
1

Impact Across Sectors



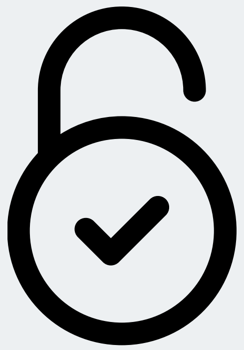
2

Overcoming the Bottleneck



3

Broadening Access



Example

Suppose we have a table:
urls: (url, category, page rank)

Simple SQL query that finds, for each sufficiently large category, the average pagerank of high-pagerank urls in that category:

```
SELECT category, AVG(pagerank)
FROM urls WHERE pagerank > 0.2
GROUP BY category HAVING COUNT(*) > 10^6
```

- SQL is **concise and familiar** for structured queries
- **Limited** in handling more **complex, unstructured data**
- Does not explicitly show the data transformation **steps**

```
SELECT category, AVG(pagerank)
FROM urls WHERE pagerank > 0.2
GROUP BY category HAVING COUNT(*) > 10^6
```

With Pig-Latin

```
good_urls = FILTER urls BY pagerank > 0.2;
groups = GROUP good_urls BY category;
big_groups = FILTER groups BY COUNT(good_urls)>10^6;
output = FOREACH big_groups GENERATE
            category, AVG(good_urls.pagerank);
```

- Pig Latin **simplifies** complex data transformations
- **Sequence of steps** shows clear data flow
- **Balances** high-level abstraction and control over data processing

Related Work

Dryad

- Microsoft's solution for complex, parallel workflows
- More adaptable than MapReduce, but still requires simplification through DryadLINQ

Dynamo

- Amazon's key-value store designed for internal use
- Excels in scalability and distributed storage, focusing on transactional data

Pig Latin

- Handles multi-stage data operations
- Balances ease-of-use with analytical power
- Innovative in handling complex data and debugging

Dynamo: Amazon’s highly available key-value store
Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks

Overview

Bridging the gap between SQL ease and MapReduce power

Proposed Solution:

Pig Latin: A high-level data processing language

Main Claims:

1. Simplification of Data Processing Workflows
2. Enhanced Flexibility and User Control
3. Support for Complex, Multi-Stage Data Operations
4. User-Defined Functions for Custom Processing

Technical Features

Dataflow Language

SQL	PIg Latin	Map-Reduce
Set of declarative constraints	Sequence of (single transformation) steps	
Order of operation based on engine	No fixed order of operation	Reordering not possible

- Easy to track variables and overall analysis process
- Optimization through reordering

Technical Features

Quickstart

- Pig's functionality to parse a file into tuples
  - Eliminates time-consuming data imports

Interoperability

- Output of a program formatted according to the user's choosing
- Operates over data in external files and does not take control over data
  - Facilitates interoperability with other applications

Technical Features

Nested Data Model

Several benefits compared to flat tables:

- Closer to how we think
- Data is stored on disk in a nested format
- Allows for algebraic language and richer user-defined functions
    - **GROUP** construct example

```
grouped_data = GROUP dataset1 BY field1;
query_data = FOREACH grouped_data GENERATE field1,
SUM(field2.amount);
```

Technical Features

UDFs: User-defined functions

- Customization of common functions i.e. grouping, filtering, joining, etc.
- Accepts non-atomic values as input and output

SQL	PIg Latin
• SELECT: only scalar functions • FROM: only set-valued functions • Aggregation functions only with GROUP BY	One type fits all

UDFs Example

Continuing with our Example 1, suppose we want to find the **top 10 highest ranking urls for each category**:

`top10()` UDF that accepts a set of urls and outputs a set with the top 10 urls by page rank, one group at a time

```
groups = GROUP urls BY category;
output = FOREACH groups GENERATE category, top10(urls);
```

Technical Features

Parallelism

- Handles large volumes of data by distributing the workload across multiple nodes in the cluster
- Only for a small set of primitives

Debugging

- Generates an example data table to illustrate the output of each step

Data Model of Pig Latin

The data model consists of four types:

- Atom: A simple atomic value such as a string
- Tuple: A sequence of fields each of which can be any of the data types
- Bag: A collection of tuples with possible duplicates
- Map: A collection of data types where each item has an associated key

$$t = \left(\text{'alice'}, \left\{ \begin{array}{c} (\text{'lakers'}, 1) \\ (\text{'iPod'}, 2) \end{array} \right\}, \left[\text{'age'} \rightarrow 20 \right] \right)$$

Let fields of tuple t be called f1, f2, f3

Expression Type	Example	Value for t
Constant	'bob'	Independent of t
Field by position	$0	'alice'
Field by name	f3	['age' → 20]
Projection	f2.$0	{ ('lakers') ('iPod') }
Map Lookup	f3#'age'	20
Function Evaluation	SUM(f2.$1)	1 + 2 = 3
Conditional Expression	f3#'age'>18? 'adult':'minor'	'adult'
Flattening	FLATTEN(f2)	'lakers', 1 'iPod', 2

Table 1: Expressions in Pig Latin.

Specifying Data Input: LOAD

The “LOAD” command is used.
First, specify what
the input data files are, and how the file contents are to be deserialized, i.e., converted into Pig’s data model.

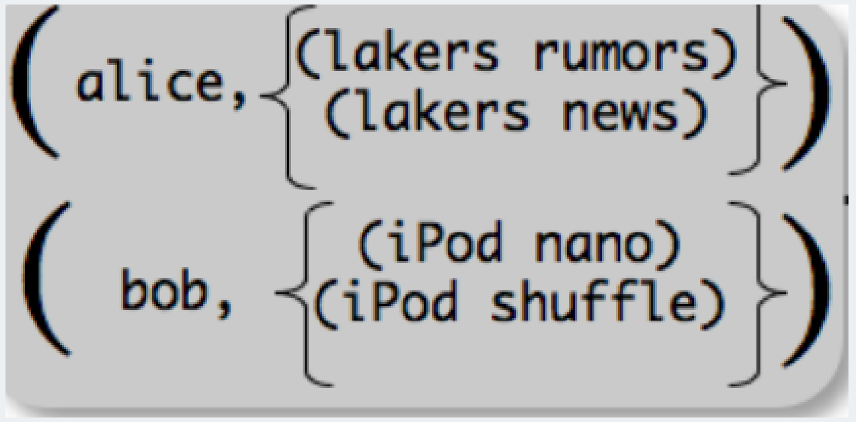
Here, the input files are deserialized using a custom myLoad() deserializer.

```
queries = LOAD
‘query_log.txt’USING
myLoad() AS (userId,
queryString,timestamp;
```

Per Tuple Processing: FOREACH

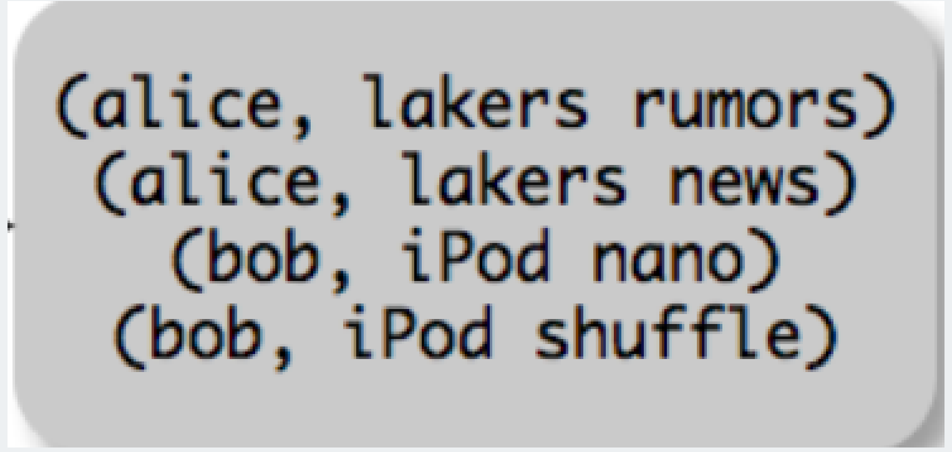
- The FOREACH command processes individual tuples.
- The GENERATE clause can be followed by a list of expressions in various forms. In this example, we are FLATTENING the data

WITHOUT FLATTENING:




```
expanded_queries = FOREACH
queries GENERATE userId,
FLATTEN(expandQuery(queryStr
ing));
```

WITH FLATTENING:



Discarding Unwanted Data: FILTER

- The FILTER command is used to retain a subset of data of interest and discard the rest
- This command filters out bot traffic from the bag of queries.
- Filtering conditions in Pig Latin can involve a combination of expressions, comparison operators (==, eq, !=, neq), and logical connectors (AND, OR, NOT).
- User-Defined Functions (UDFs) can be used in filtering.

```
Use of FILTER command:
real_queries = FILTER
queries BY userId neq ‘bot’;

real_queries = FILTER
queries BY NOT
isBot(userId);
```

The UDF isBot(userId) is used to filter out bot traffic.

Getting Related Data Together: COGROUP

- COGROUP command allows for the aggregation of tuples from different datasets that share common attributes.
- Each tuple in the 'grouped_data' output contains a group identifier (the 'queryString'), followed by bags containing tuples from each input dataset belonging to that group.

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

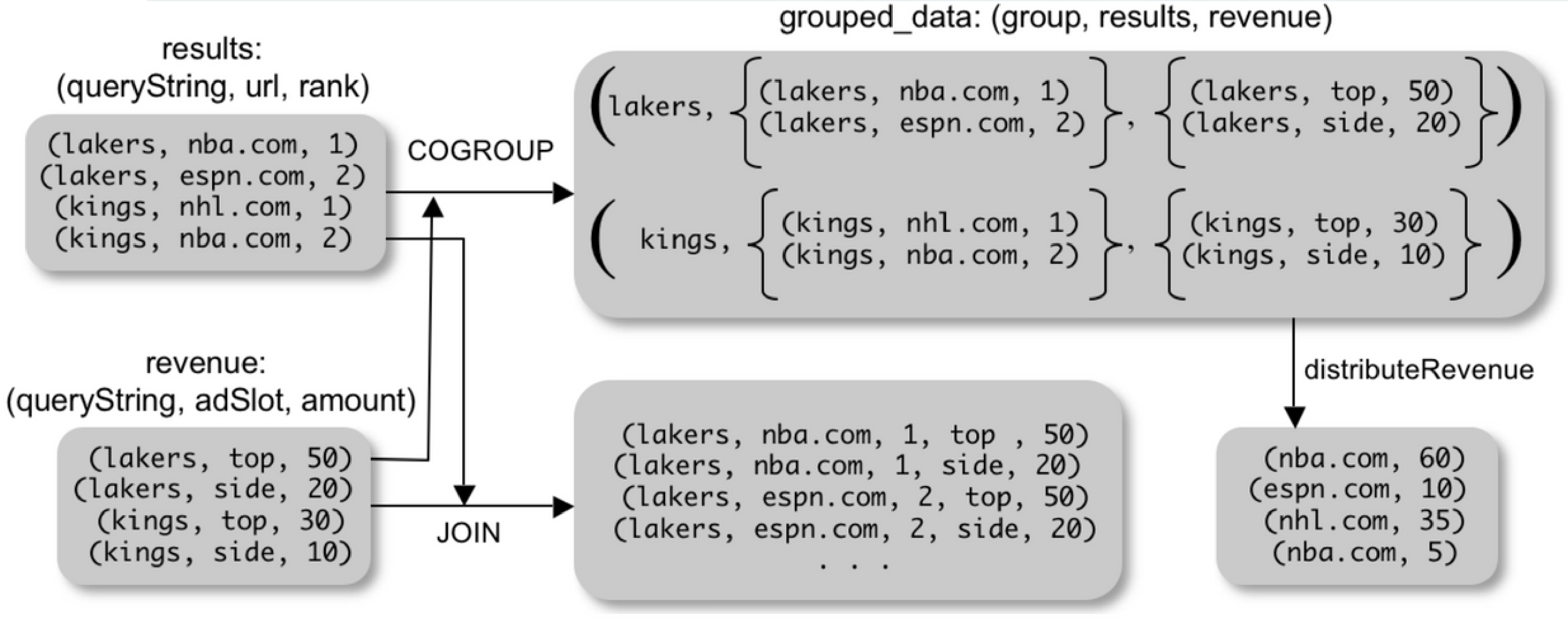


Figure 2: COGROUP **versus** JOIN.

Nested Operations

- When we have nested bags within tuples, Pig Latin allows some commands to be nested within a FOREACH command.
- Currently, only FILTER, ORDER, and DISTINCT are allowed to be nested within FOREACH.

```
query_revenues = FOREACH
grouped_revenue{ top_slot =
FILTER revenue BY adSlot eq
‘top’; GENERATE queryString,
SUM(top_slot.amount),
SUM(revenue.amount); }
```

Map-Reduce in Pig Latin

- Map-reduce program in Pig Latin is straightforward with the GROUP and FOREACH statements.
- Map function operates on one input tuple at a time and outputs a bag of key-value pairs.

```
map_result = FOREACH input GENERATE
FLATTEN(map(*));

key_groups = GROUP map_result BY $0;

output = FOREACH key_groups GENERATE
reduce(*);
```

OTHER COMMANDS:

- UNION: Returns the union of two or more bags
- CROSS: Returns the cross product of two or more bags
- ORDER: Order a bag by the specified field(s)
- DISTINCT: Eliminates duplicate tuples in a bag.

Implementation

- Pig Latin is **fully implemented** by the system, **Pig**.
- Current implementation **uses *Hadoop***, an open-source implementation of map-reduce as the execution platform.
- Pig Latin programs are **compiled into map-reduce jobs**, and **executed using Hadoop**.

```
c = COGROUP a BY ..., b BY ...
```

- Pig verifies that the bags **a** and **b** have already been defined.
- Pig builds a **logical plan** for every bag the user defines.
- The logical plan for **c** consists of a cogroup command having the logical plans for **a** and **b** as inputs.
- Processing is only triggered when the user invokes a **STORE** command on a bag.

Map-Reduce Plan Compilation

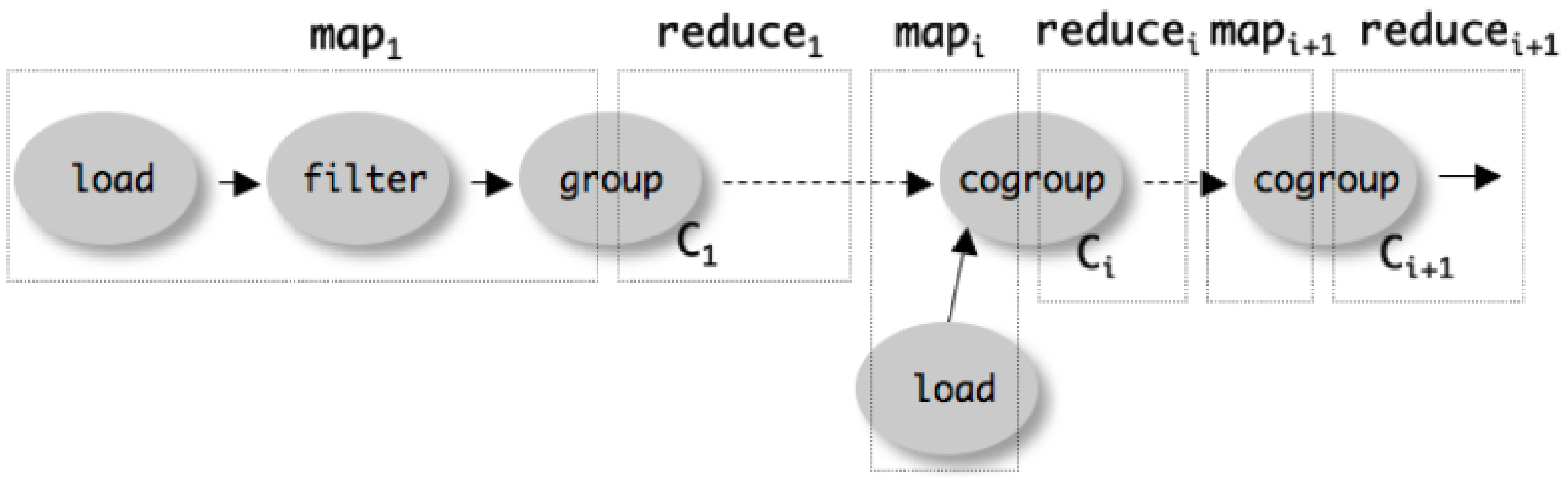


Figure 3: Map-reduce compilation of Pig Latin.

Their compiler begins by converting each **(CO)GROUP command** into the logical plan into a distinct map-reduce job with its **own map and reduce functions**.

Debugging Environment

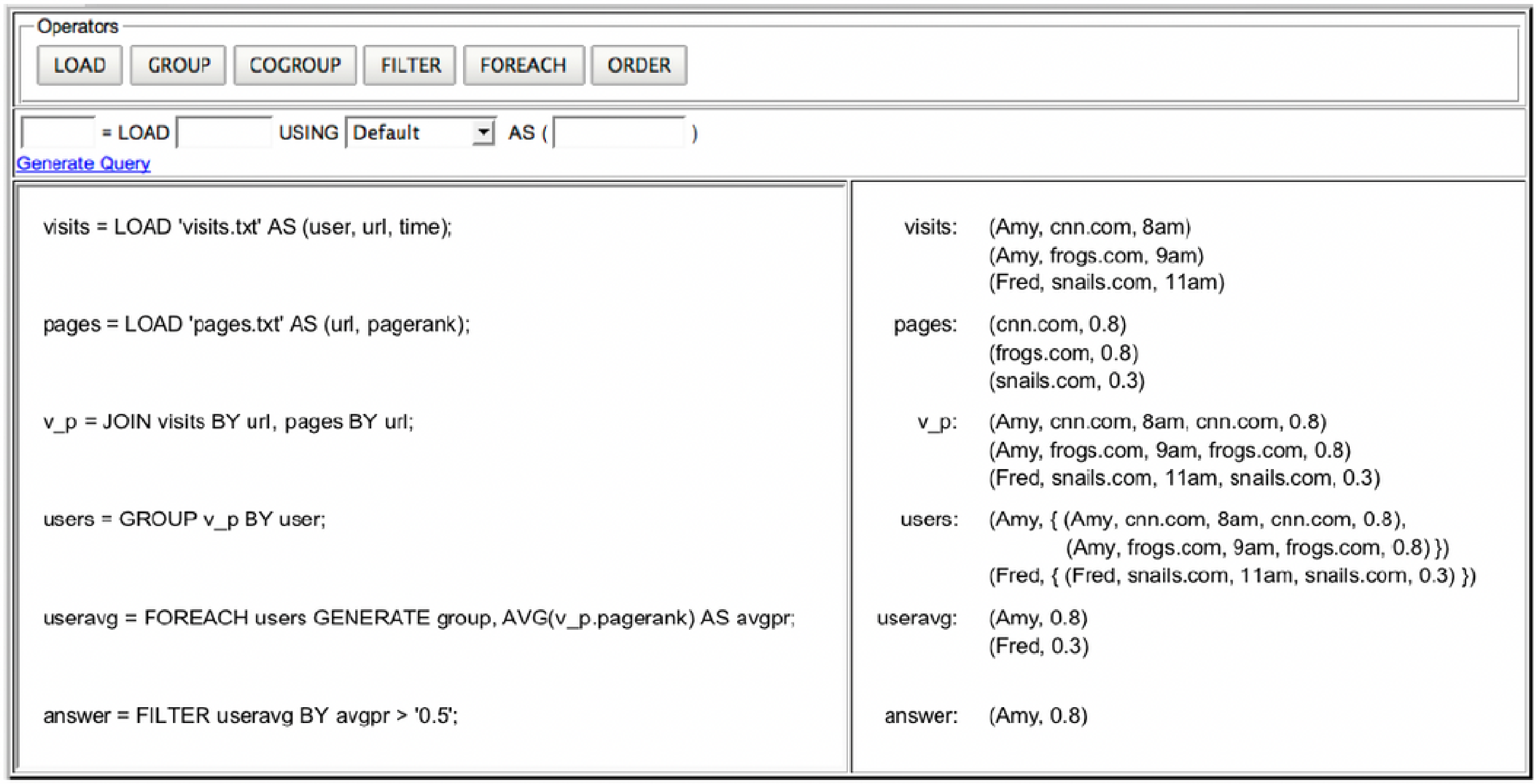


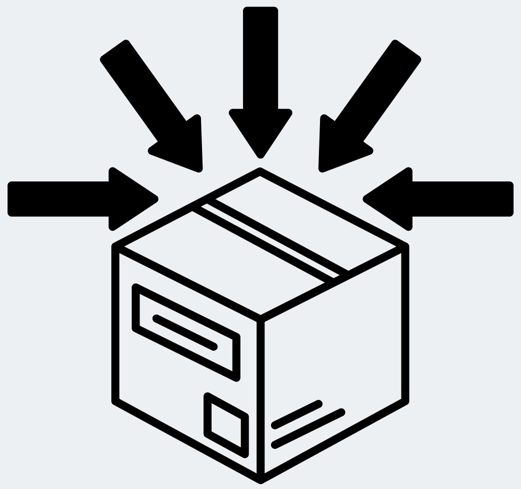
Figure 4: Pig Pen screenshot; displayed program finds users who tend to visit high-pagerank pages.

- The **left-hand panel** is where the user enters Pig Latin commands.
- The **right-hand panel** is populated automatically, and shows the effect of the user's program on the sandbox data set.
- The **sandbox data set** also helps users understand the schema at each step.

Three primary objectives in selecting a sandbox data set: **realism**, **conciseness**, and **completeness**.

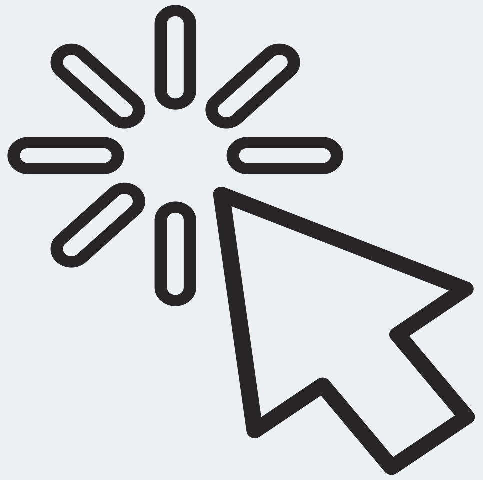
Usage Scenarios

Rollup Aggregates



Temporal Analysis

Session Analysis

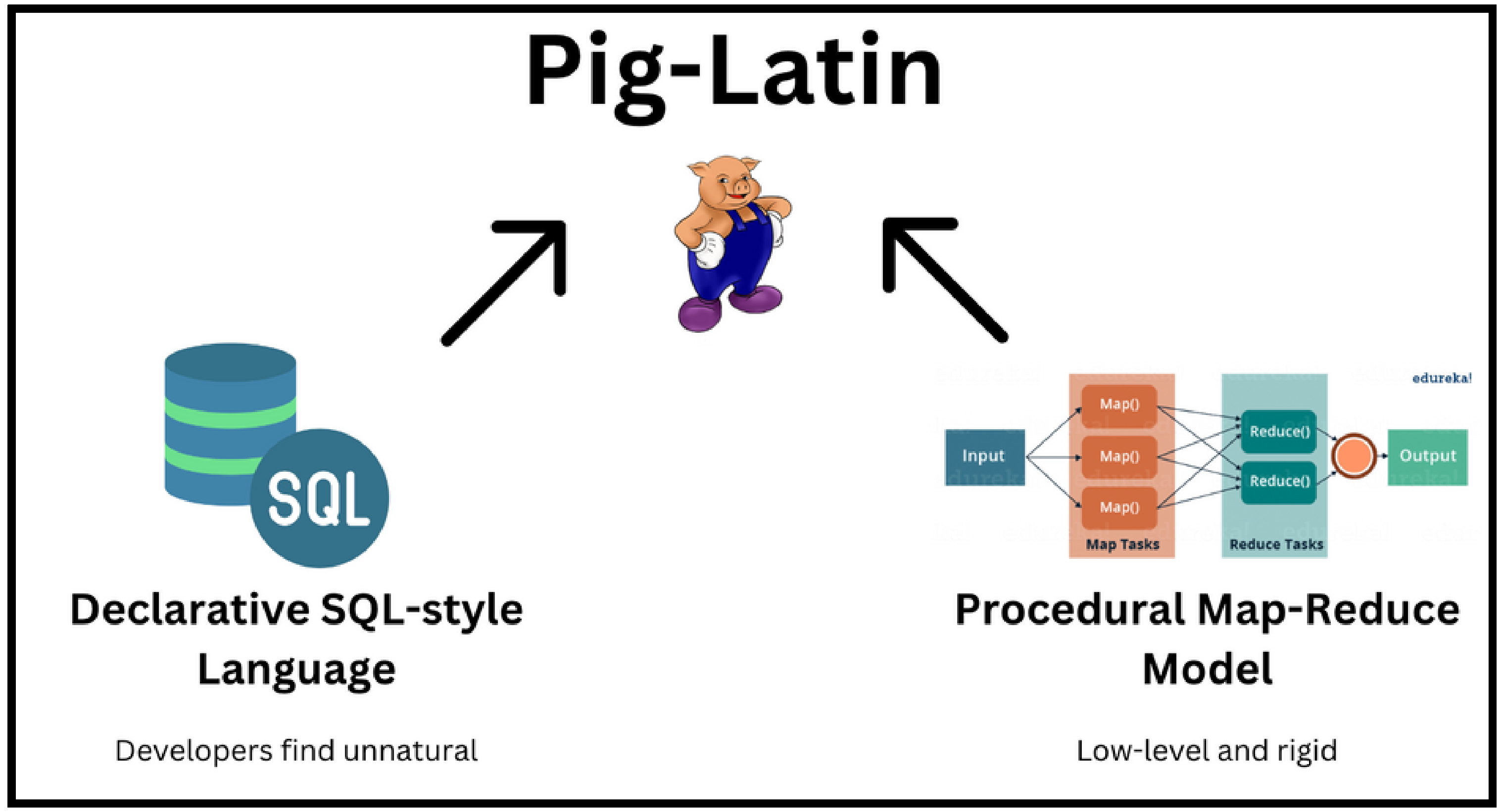


The primary reason to **use Pig rather than a database/OLAP system** for these rollup analyses, is that the search logs are too big and continuous to be curated and loaded into databases.

The Main Hypothesis

Pig Latin fits in a sweet spot between the **declarative SQL-style language**, and the **procedural map-reduce model**, and improves user productivity for **ad-hoc analysis of large-scale datasets**.

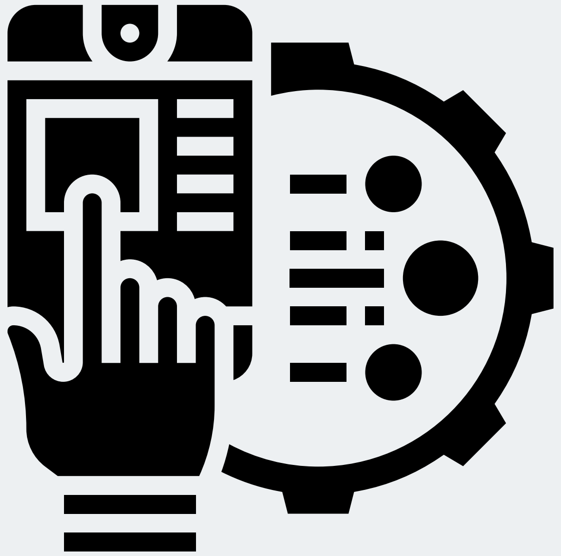
Goes back to Slide 3



Future Work

“Safe” Optimizer

User Interfaces



External Functions

Unified Environment

Summary

- We have described a new data processing environment being deployed at Yahoo! called **Pig**, and its associated language **Pig Latin**.
- We also described a debugging environment, **Pig Pen**.
- Pig has an active and growing user base inside **Yahoo!**, and with their recent open-source release they are beginning to attract users in the **broader community**.

Study Questions

1. What are the **advantages** of using Apache Pig for data processing compared to writing MapReduce jobs directly?
2. Explain the concept of **flattening** in Pig Latin and provide an example.

Reference

G. DeCandia et al. Dynamo: Amazon’s highly available key-value store. In Proc. SOSP, 2007.

M. Isard et al. Dryad: Distributed data-parallel programs from sequential building blocks. In European Conference on Computer Systems (EuroSys), pages 59–72, Lisbon, Portugal, March 21-23 2007.

Olston, C., Reed, B., Srivastava, U., Kumar, R. & Tomkins, A. (2008). Pig latin: a not-so-foreign language for data processing.. In J. T.-L. Wang (ed.), SIGMOD Conference (p./pp. 1099-1110), : ACM. ISBN: 978-1-60558-102-6