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

- Introduction
- Features and language description
- Implementation
- Debugging environment
- Optimizations
- Conclusion
Agenda

• **Introduction**
  • Features and language description
  • Implementation
  • Debugging environment
  • Optimizations
  • Conclusion
Agenda

- Introduction
  - Features and language description
  - Implementation
  - Debugging environment
  - Optimizations
  - Conclusion
  - need of this technology
  - map-reduce
  - parallel DB System
  - Pig Latin
Need of technology to work with big data sets

• Number and size of data increase exponentially
  – Development of Computer science
  – Development of Internet

• Internet companies need to process huge data sets
  – Part of the product improvement cycle
  – Analysis of terabytes of data every days
  – EX: ranking algorithms, books comparisons
Map reduce : Definition

• a framework used to do **distributed and parallel computing on large data sets**
• based on two simple primitives
  – **Map** : do an operation over all elements of a list. EX : increase all grades by 1
  – **Reduce** : take a list, do a combination and return a single element.
• looks like functional programming
Map reduce : Analysis

- low level

+ good performance and parallelization

+ language of your choice

- very rigid (one input, 2 functions)

- need custom code for all operations

- hard to maintain and to reuse

- hard to optimize
Parallel DB products : Definition

- solution proposed by the classical database companies (EX : Oracle)
- **SQL interface**
- derivate from classical DB systems
Parallel DB products: Analysis

+/-. high level
+ high level SQL query interface
+ hide complexity of parallel clusters
- not directly designed for Internet use
- performance
- SQL not a natural way of thinking
Pig Latin: the trade-off (1/2)

- proposed trade-off by Yahoo!
- tries to take the best of the both alternatives cited above.
  + high level declaratives (like in SQL)
  - Pig Latin programs are sequences of steps (like script, like map-reduce)
  + flexible and nested data model
  + user-defined functions
Pig Latin : the trade-off (2/2)

+ easier to optimize than map-reduce
  • compiled into map-reduce job (Hadoop), but this infrastructure can be replaced by another one

+ evolution is so guaranteed
Agenda

• Introduction
• **Features and language description**
• Implementation
• Debugging environment
• Optimizations
• Conclusion
Agenda

• Introduction

• Features and language description
  - data model
  - user-defined functions
  - commands (EX: LOAD, FILTER)
  - nested operations
  - parallelism

• Implementation

• Debugging environment

• Optimizations
Pig Latin : Data Model

• rich, **flexible and fully nested** data model.

• consisting of 4 different types :
  – Atom
  – Tuple
  – Bag
  – Map
Data Model : Atom

- a simple and **atomic value**
- like a number or a string

- **EX** : ‘eth’ ‘cake’ 12 20
Data Model : Tuple

- a **sequence of fields** (of any types)
- can be **nested** (tuples of tuples)
- fields of a tuple can be accessed by name ($f_1$) or by position ($0$)

**EX** : ('Alice', 'Bob')

('Alice', 1, ('Bob', 2))
Data Model: Bag

- a collection of tuples
- possible duplicates
- flexible: no need of same type and number of fields for all tuples

EX: \[\{(\text{Alice}, \text{Bob})\}, \{(\text{Alice}, 1, (\text{Ron}, 2))\}\]
Data Model : Map

• a collection of data items
• each item has an associated key
• this key can be used to look up the item

• EX : 
  \[
  \begin{cases}
  
  
  \text{‘like’} \rightarrow \{\text{(‘Alice’)}
  
  \text{\{(‘apple’, ‘orange’)}
  
  \text{‘age’} \rightarrow 20
  
  \end{cases}
  
  \]
Data Model : Map

• a collection of data items
• each item has an associated key
• this **key can be used to look up the item**

**EX :**

- 'like' $\rightarrow$ \{('Alice')
  \{('apple', 'orange')\} \}
- 'age' $\rightarrow$ 20
Input management (1/2)

• can use **input file of any form**
  – thanks to the flexible data model
  – no need of transactional guarantees
  – data schema are optional

• input file is assumed to be a **sequence of fields** (= a bag)
the input file is defined with LOAD

LOAD command only defines what and how to read. But the effective reading is only do when it’s necessary (lazy)

EX: queries = LOAD 'query_log.txt'
    USING myLoadfunction()
    AS (Id, queryString, timestamp);
Input management (2/2)

- The input file is defined with `LOAD` command only defines what and how to read. But the effective reading is only done when it's necessary (lazy).

- **EX:**
  ```java
  queries = LOAD 'query_log.txt'
  USING myLoadfunction()
  AS (Id, queryString, timestamp);
  ```

- **Names of fields (optional)**
- **Custom deserializer (optional)**
- **Input File**
Output management (1/2)

• can generate output file of any form
  – thanks to the flexible data model
  – with the help of user-defined functions
  – to fill requirements of another application
Output management (2/2)

- the output file is created with \texttt{STORE}

- \texttt{EX : STORE query INTO 'myoutput' USING myStorfunction()}
Output management (2/2)

- the output file is created with $STORE$

- EX: $STORE$ query INTO 'myoutput' USING myStorfunction()
User-defined functions

- input and return types are very flexible
- need to be written in Java
  - but use of Pearl or Python is planned

EX: `spam = FILTER urls BY isSpam(url);`
User-defined functions

• Input and Return types are very flexible
• Need to be written in Java
  – but use of Pearl or Python is planned

• EX: \( \text{spam} = \text{FILTER urls BY isSpam(url)}; \)

User defined functions
Input = url (‘www.inf.ethz.ch’) Return = boolean
FOREACH

- apply some processing to every tuple of a data set
- totally parallelizable

EX: \( \text{exqueries} = \text{FOREACH } \text{queries} \quad \text{GENERATE } \text{id}, \quad \text{exQuery}(\text{queryString}) \)
FOREACH

- apply some processing to every tuple of a data set
- Totally parallelizable
- EX: 
  \[
  \text{exqueries} = \text{FOREACH queries GENERATE } \text{id, } \text{exQuery(queryString)}
  \]

User defined functions
FLATTEN

- used to **eliminate nesting**

- **EX**: `FLATTEN (query)`
FILTER

- used to **discard a useless part** of a set
- we can use user-defined function to do the selection

EX: \( \text{spam} = \text{FILTER urls BY isSpam(url)}; \)
COGROUP (1/2)

• used to assemble tuples from one or more data sets
• we need to define a way to do the linkage (a common column)
• JOIN is done by a COGROUP followed by taking a cross product of the tuples in the nested bag
• EX: \texttt{COGROUP a \texttt{BY} id, b \texttt{BY} id;}

The first field of the result is the common element, and the other are bags, one for each input being merged.
Others commands

• **GROUP**: special case of COGROUP with only one bag involved
• **UNION**: to do the union of two or more bags
• **CROSS**: cross product of two bags
• **ORDER**: to order a bag by a field
• **DISTINCT**: to eliminate duplicate
Nested operations

- FOREACH can be used to apply commands into nested bag
- available commands are:
  - FILTER
  - ORDER
  - DISTINCT
- very useful to work with results of COGROUP (nested bags)
All the primitives available in Pig Latin were designed in a way to allow fully parallelization.

It’s why these operations can be converting to map-reduce sequence of operations.

It’s why JOIN for example is not available.
Agenda

- Introduction
- Features and language description
- **Implementation**
- Debugging environment
- Optimizations
- Conclusion
Implementation (1/3)

- Pig Latin is implemented by a system called Pig
- Pig is conceived to allow different systems to be used as execution platform.
- current version uses Hadoop as platform
- Hadoop is an open-source implementation of map-reduce
- therefore actually Pig Latin programs are compiled into map-reduce jobs!
• Pig Latin makes a logical plan for every bag defined by the user
  – a new bags, defined by a command, is build by a combination of input bags (and the command’s action of course).

• Pig Latin works in a lazy way, and the processing is done only when the user use the STORE command
  – at this point, logical plan is compiled into a physical plan and it’s executed
• when it’s needed (by a STORE) logical plans are compiled into map-reduce jobs
  - we can have some overhead due to the lack of flexibility of map-reduce
  - perhaps Pig Latin could be better with another platform?
Agenda

- Introduction
- Features and language description
- Implementation
- **Debugging environment**
- Optimizations
- Conclusion
• the system PIG contain Pig Pen a debugging environment
• Pig Pen uses a sample data set to see what a transformation does in each step
  – the creation of this “Sandbox” data set is done very carefully
  – this set should be a part of the real data set but need to be as small as possible
  – this set need to act like the real set and need to show the same behaviour in every step
  – that’s assured by a special procedure used by Pig Pen to generate this sample.
Debugging Environment (2/2)

<table>
<thead>
<tr>
<th>$a = \text{LOAD} \ 'a.txt' \ \text{AS} \ (id, name);$</th>
<th>$a: (1, Bob) \ (2, Alice) \ (3, Ron)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b = \text{FILTER} \ a \ \text{BY} \ \text{Zurich}(name);$</td>
<td>$b: (1, Bob) \ (2, Alice)$</td>
</tr>
</tbody>
</table>
Debugging Environment (2/2)

\[ a = \text{LOAD} \ 'a.txt' \ \text{AS} \ (id, name); \]

\[ b = \text{FILTER} \ a \ \text{BY} \ Zurich(name); \]

\[ a: (1, Bob) \\
   (2, Alice) \\
   (3, Ron) \]

\[ b: (1, Bob) \\
   (2, Alice) \]

Sample Data Set 1

Sample Data Set 2
Agenda

- Introduction
- Features and language description
- Implementation
- Debugging environment
- Optimizations
- Conclusion
Agenda

- Introduction
- Features and language
- Implementation
- Debugging environment
- Optimizations
  - Single-program optimizations
  - Cross-program optimizations
- Conclusion
Single-program optimizations (1/2)

• optimizations that can be done inside a Pig Latin program
• logical optimization: change the order of operations in a Program
  – With very huge data sets, it’s crucial!
  – a little error can implies a lot of useless computations.
• some of well-know database oriented optimization can be used to in Pig Latin
  – early filtering
    • EX: Use FILTER as fast as possible
  – operations rewriting
    • EX: Replace a JOIN by a ASYMMETRIC JOIN
  – early projection
    • EX: discard useless fields at the beginning
Cross-program optimizations (1/2)

• the idea is to avoid redundant computations by two or more different programs.

• we have two different ways to implement this idea:
  – concurrent work sharing
  – won-concurrent work sharing
Concurrent work sharing

• a running program check if another program would not make the same step that him

• if it’s the case, the both program synchronise together to do the computation only once
Non-concurrent work sharing

• the idea here is **to cache the more used data** or the result of very common computation

• EX: cache the result of the “remove BOT” filtering in the calculation of the numbers of visits of the company's website
Agenda

- Introduction
- Features and language description
- Implementation
- Debugging environment
- Optimizations
- Conclusion
Review of Pig Latin (1/2)

- Pig Latin tries to take the best part of map-reduce word and classical database world
  - useful and fully parallelizable primitives
  - flexible nested data model
  - independence of platform
  - good debugging environment
  - high-level
Pig Latin is useful to process ad-hoc, data-centric workloads on very huge data sets.

One example of application:

- **sessions analysis** (a session = sequences of clicks and page views by a user)
- The nested model provides a helpful abstraction to manage different sessions.
- Using FOREACH, to do some calculations inside nested structures, allows us to reduce the code complexity.
Pig Latin can not avoid all limitations coming from map-reduce

solution : Use of another implementation ?

• developments planned by Yahoo:
  – the use of another language for UDF
  – a better user interface
  – a safe optimizer (that will perform some basics optimizations alone)
  – the embedding of Pig Latin into other established language like Python
Agenda

- Introduction
- Features and language description
- Implementation
- Debugging environment
- Optimizations
- Conclusion
??? Questions ???

- Introduction
- Features and language description
- Implementation
- Debugging environment
- Optimizations
- Conclusion