L08: Just enough Scala for Spark

ANLY 502: Massive Data Fundamentals

Simson Garfinkel & Marck Vaisman

March 20, 2017

Agenda

Administrivia

- A04 not graded yet
- A05 will be posted mid-week (hopefully)

Virtual Machine tools (useful for learning and playing with new tools)

- Vagrant
- Docker

Brief overview Functional programming concepts

Just Enough Scala for Spark

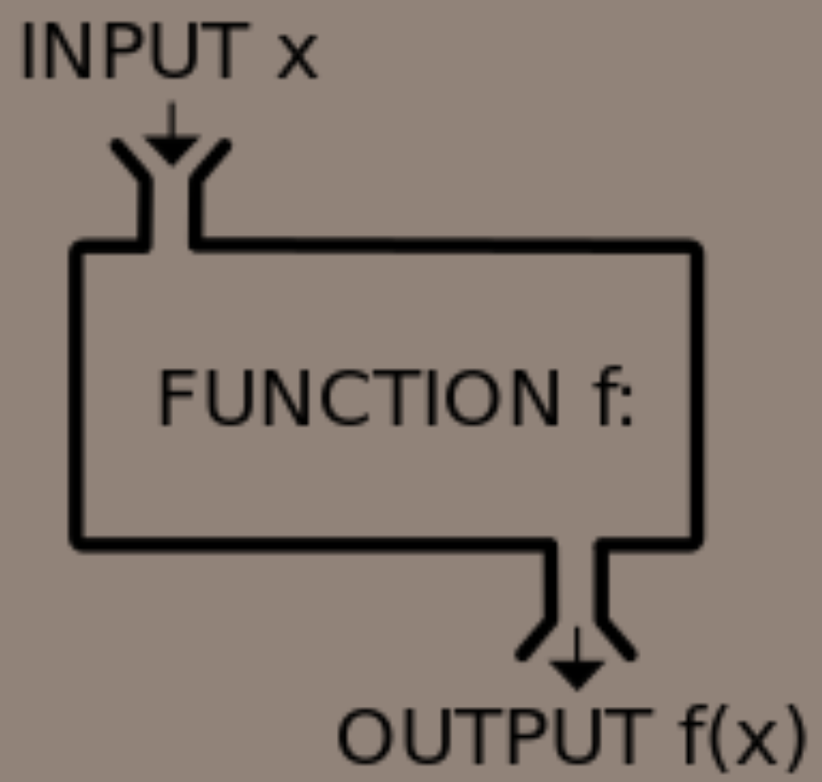
- Basic Scala data types
- Methods and Functions
- RDD Operations

Lab - play with the Dean Wampler's Spark Notebook and look at non-trivial Scala code

- Walk through steps involved in creating an inverted index

Time Permitting - Brief overview of PIG

VAGRANT



https://codesachin.files.wordpress.com/2016/04/220px-function_machine2-svg.png

Functional programming

Technical functional code characteristics:

- immutable data
- first class functions
- tail call

Functional code is characterized by:

- the absence of side effects
- not relying on data outside the current function
- not changing the data outside of the current function

https://maryrosecook.com/blog/post/a-practical-introduction-to-functional-programming

No side effects…

This is an unfunctional function:

```
a = 0
def increment():
    global a
    a += 1
```

This is a functional function:

```
def increment(a):
    return a + 1
```

Don't iterate over lists, use map and reduce

```
name_lengths = map(len, ["Mary", "Isla", "Sam"])

print name_lengths
# => [4, 4, 3]
```

This is a map that squares every number in the passed collection:

```
squares = map(lambda x: x * x, [0, 1, 2, 3, 4])

print squares
# => [0, 1, 4, 9, 16]
```

Rewrite as map

```
import random

names = ['Mary', 'Isla', 'Sam']
code_names = ['Mr. Pink', 'Mr. Orange', 'Mr. Blonde']

for i in range(len(names)):
    names[i] = random.choice(code_names)

print names
# => ['Mr. Blonde', 'Mr. Blonde', 'Mr. Blonde']
```

```
import random

names = ['Mary', 'Isla', 'Sam']

secret_names = map(lambda x: random.choice(['Mr. Pink',
                                            'Mr. Orange',
                                            'Mr. Blonde']),
                   names)
```

Rewrite as reduce

This code counts how often the word `'Sam'` appears in a list of strings:

```
sentences = ['Mary read a story to Sam and Isla.',
             'Isla cuddled Sam.',
             'Sam chortled.']

sam_count = 0
for sentence in sentences:
    sam_count += sentence.count('Sam')

print sam_count
# => 3
```

This is the same code written as a reduce:

```
sentences = ['Mary read a story to Sam and Isla.',
             'Isla cuddled Sam.',
             'Sam chortled.']

sam_count = reduce(lambda a, x: a + x.count('Sam'),
                   sentences,
                   0)
```

Write declaratively, not imperatively

Imperative

```
from random import random

time = 5
car_positions = [1, 1, 1]

while time:
    # decrease time
    time -= 1

    print ''
    for i in range(len(car_positions)):
        # move car
        if random() > 0.3:
            car_positions[i] += 1

    # draw car
    print '-' * car_positions[i]
```

```
-
--
--

--
--
---

---
--
---

----
---
----

----
----
-----
```

```
from random import random

def move_cars():
    for i, _ in enumerate(car_positions):
        if random() > 0.3:
            car_positions[i] += 1

def draw_car(car_position):
    print '-' * car_position

def run_step_of_race():
    global time
    time -= 1
    move_cars()

def draw():
    print ''
    for car_position in car_positions:
        draw_car(car_position)

time = 5
car_positions = [1, 1, 1]

while time:
    run_step_of_race()
    draw()
```

Remove state

This is a functional version of the car race code:

```
from random import random

def move_cars(car_positions):
    return map(lambda x: x + 1 if random() > 0.3 else x,
               car_positions)

def output_car(car_position):
    return '-' * car_position

def run_step_of_race(state):
    return {'time': state['time'] - 1,
            'car_positions': move_cars(state['car_positions'])}

def draw(state):
    print ''
    print '\n'.join(map(output_car, state['car_positions']))

def race(state):
    draw(state)
    if state['time']:
        race(run_step_of_race(state))

race({'time': 5,
      'car_positions': [1, 1, 1]})
```

The code is still split into functions, but the functions are functional. There are three signs of this. First, there are no longer any shared variables. `time` and `car_positions` get passed straight into `race()`. Second, functions take parameters. Third, no variables are instantiated inside functions. All data changes are done with return values. `race()` recurses[3] with the result of `run_step_of_race()`. Each time a step generates a new state, it is passed immediately into the next step.


Scala

Why Scala?

Non-Technical

Spark is written in Scala

- Most new features will be available trough the Scala API first before Python or R

There is industry demand

Technical

Statically typed

Mixed paradigm - object oriented programming

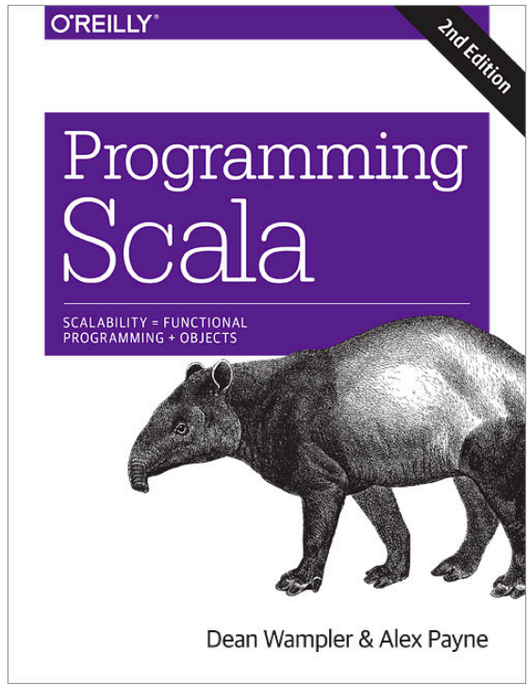
Mixed paradigm - functional programming

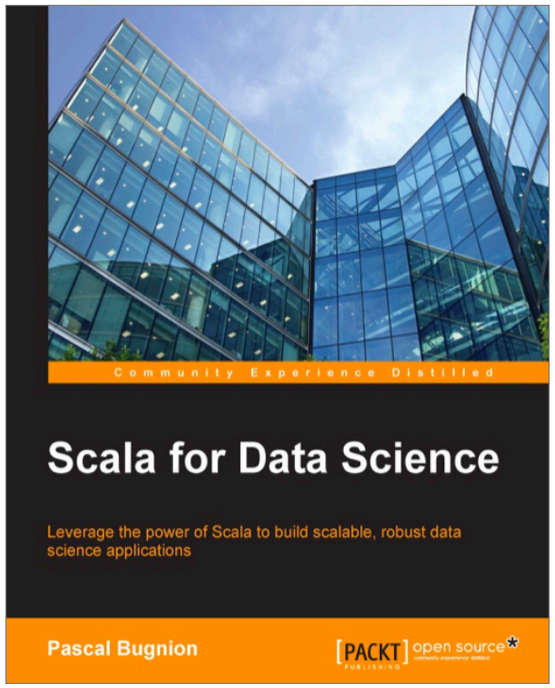
Sophisticated type system

Succinct, elegant, flexible syntax

Scalable

References used for this lecture





https://www.youtube.com/watch?v=LBoSgiLV_NQ

https://maryrosecook.com/blog/post/a-practical-introduction-to-functional-programming

Basics of OOP (like Python)

Everything in Scala is an object

- Objects have states and behaviors. An object is an instance of a class
  - —*Objects have types, the type determines what we can do with the object.*
  - —*Example: numbers, strings, files, digital images*
- A Class can be defined as a template/blueprint that describes the behavior/state of an object.
- A Method is a behavior. A Class can contain many methods.

Scala Data Types

Scala has all the same data types as Java, with the same memory footprint and precision. Following is the table giving details about all the data types available in Scala:

Data Type	Description
Byte	8 bit signed value. Range from -128 to 127
Short	16 bit signed value. Range -32768 to 32767
Int	32 bit signed value. Range -2147483648 to 2147483647
Long	64 bit signed value. -9223372036854775808 to 9223372036854775807
Float	32 bit IEEE 754 single-precision float
Double	64 bit IEEE 754 double-precision float
Char	16 bit unsigned Unicode character. Range from U+0000 to U+FFFF
String	A sequence of Chars
Boolean	Either the literal true or the literal false
Unit	Corresponds to no value
Null	null or empty reference
Nothing	The subtype of every other type; includes no values
Any	The supertype of any type; any object is of type *Any*
AnyRef	The supertype of any reference type

All the data types listed above are objects. There are no primitive types like in Java. This means that you can call methods on an Int, Long, etc.

Variables

val - immutable variable

```
scala> val array: Array[String] = new Array(5)
array: Array[String] = Array(null, null, null, null, null)

scala> array = new Array(2)
<console>:8: error: reassignment to val
       array = new Array(2)

scala> array(0) = "Hello"

scala> array
res1: Array[String] = Array(Hello, null, null, null, null)
```

```
val m = 17
m = 18       // error!
```

var - mutable variable (but can only be reassigned with another value of same type)

```
scala> var stockPrice: Double = 100.0
stockPrice: Double = 100.0

scala> stockPrice = 200.0
stockPrice: Double = 200.0
```

```
var m : Int = 17
m = 18           // ok
m = "seventeen" // error!
m = 18.0         // error!
```

Define a "Person" class with immutable and mutable parts

```
// src/main/scala/progscala2/typelessdomore/person.sc
scala> class Person(val name: String, var age: Int)
defined class Person

scala> val p = new Person("Dean Wampler", 29)
p: Person = Person@165a128d

scala> p.name
res0: String = Dean Wampler       // Show the value of firstName.

scala> p.age
res2: Int = 29                    // Show the value of age.

scala> p.name = "Buck Trends"
<console>:9: error: reassignment to val    // Disallowed!
       p.name = "Buck Trends"
              ^

scala> p.age = 30
p.age: Int = 30                   // Okay!
```

The `var` and `val` keywords only specify whether the reference can be changed to refer to a different object (`var`) or not (`val`). They don't specify whether or not the object they reference is mutable.

Ranges

```
scala> 1 to 10                  // Int range inclusive, interval of 1, (1 to 10)
res0: scala.collection.immutable.Range.Inclusive =
  Range(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

scala> 1 until 10               // Int range exclusive, interval of 1, (1 to 9)
res1: scala.collection.immutable.Range = Range(1, 2, 3, 4, 5, 6, 7, 8, 9)

scala> 1 to 10 by 3             // Int range inclusive, every third.
res2: scala.collection.immutable.Range = Range(1, 4, 7, 10)

scala> 10 to 1 by -3            // Int range inclusive, every third, counting down.
res2: scala.collection.immutable.Range = Range(10, 7, 4, 1)

scala> 1L to 10L by 3           // Long
res3: scala.collection.immutable.NumericRange[Long] = NumericRange(1, 4, 7, 10)

scala> 1.1f to 10.3f by 3.1f    // Float with an interval != 1
res4: scala.collection.immutable.NumericRange[Float] =
  NumericRange(1.1, 4.2, 7.2999997)

scala> 1.1f to 10.3f by 0.5f    // Float with an interval < 1
res5: scala.collection.immutable.NumericRange[Float] =
  NumericRange(1.1, 1.6, 2.1, 2.6, 3.1, 3.6, 4.1, 4.6, 5.1, 5.6, 6.1, 6.6,
    7.1, 7.6, 8.1, 8.6, 9.1, 9.6, 10.1)

scala> 1.1 to 10.3 by 3.1       // Double
res6: scala.collection.immutable.NumericRange[Double] =
  NumericRange(1.1, 4.2, 7.300000000000001)

scala> 'a' to 'g' by 3          // Char
res7: scala.collection.immutable.NumericRange[Char] = NumericRange(a, d, g)

scala> BigInt(1) to BigInt(10) by 3
res8: scala.collection.immutable.NumericRange[BigInt] =
  NumericRange(1, 4, 7, 10)

scala> BigDecimal(1.1) to BigDecimal(10.3) by 3.1
res9: scala.collection.immutable.NumericRange.Inclusive[scala.math.BigDecimal]
  = NumericRange(1.1, 4.2, 7.3)
```

Statically typed languages can be very verbose. Consider this typical declaration in Java, before Java 7:

```
import java.util.HashMap;
...
HashMap<Integer, String> intToStringMap = new HashMap<Integer, String>();
```

We have to specify the type parameters <Integer, String> twice. Scala uses the term *type annotations* for explicit type declarations like HashMap<Integer, String>.

Java 7 introduced the *diamond operator* to infer the generic types on the righthand side, reducing the verbosity a bit:

```
HashMap<Integer, String> intToStringMap = new HashMap<>();
```

We've already seen some examples of Scala's support for type inference. The compiler can discern quite a bit of type information from the context, without explicit type notations. Here's the same declaration rewritten in Scala, with inferred type information:

```
val intToStringMap: HashMap[Integer, String] = new HashMap
```

If we specify HashMap[Integer, String] on the righthand side of the equals sign, it's

even more concise:

```
val intToStringMap2 = new HashMap[Integer, String]
```

Some functional programming languages, like Haskell, can infer almost all types, be− cause they can perform global type inference. Scala can't do this, in part because Scala has to support *subtype polymorphism* (inheritance), which makes type inference much harder.

Here is a summary of the rules for when explicit t

Literal Values

Integer - type Int

```
0
035
21
0xFFFFFFFF
0777L
```

Floating Point - type Float

```
0.0
1e30f
3.14159f
1.0e100
.1
```

Character - single character, single quote

```
'a'
'\u0041'
'\n'
'\t'
```

String - sequence of characters, double quote

```
"Hello,\nWorld!"
"This string contains a \" character."
```

Multi Line Strings

```
"""the present string
spans three
Lines."""
```

Tuples

Multiple assignments

Tuples combine a fixed number of items together so that they can be passed around as a whole. Unlike an array or list, a tuple can hold objects of different types but they are immutable.

The following is an example of a tuple holding an integer, a string, and the console.

```
val t = (1, "hello", Console)
```

Which is syntactic sugar (shortcut) for the following:

```
val t = new Tuple3(1, "hello", Console)
```

Accessing tuple elements

```
val t = (4,3,2,1)
```

To access elements of a tuple t, you can use method t._1 to access the first element, t._2 to access the second, and so on. For example, the following expression computes the sum of all elements of t.

```
val sum = t._1 + t._2 + t._3 + t._4
```

What is an RDD

An immutable, partitioned collection of elements that can be operated on in parallel. This class contains the basic operations available on all RDDs,
such as map, filter, and persist. In addition, PairRDDFunctions contains operations available only on RDDs of key-value pairs, such as groupByKey and join;
DoubleRDDFunctions contains operations available only on RDDs of Doubles; and SequenceFileRDDFunctions contains operations available on RDDs that can be saved as SequenceFiles. All operations are automatically available on any RDD of the right type (e.g. RDD[(Int, Int)] through implicit.
Internally, each RDD is characterized by five main properties:

- A list of partitions - A function for computing each split - A list of dependencies on other RDDs - Optionally, a Partitioner for key-value RDDs (e.g. to say that the RDD is hash-partitioned) - Optionally, a list of preferred locations to compute each split on (e.g. block locations for an HDFS file)

All of the scheduling and execution in Spark is done based on these methods, allowing each RDD to implement its own way of computing itself. Indeed, users can implement custom RDDs (e.g. for reading data from a new storage system) by overriding these functions. Please refer to the Spark paper for more details on RDD internals.

RDDs

RDDs support many of the operations supported by native Scala collections

RDDs are immutable

- Cannot change an RDD once created. All operations create new RDDs or other Scala objects

RDDs are lazy

- Unlike native Scala collections, RDD operations are only evaluated when needed. (In the REPL any operation on a collection prints the values of the new collection to screen.

Transformations on RDDs: create new RDD from current one. Lazy evaluation

Actions on RDDs: force the evaluation of an RDD and normally return a Scala object rather than an RDD. Actions are evaluated immediately.

List of Transformations and Actions

Some Scala Examples

https://gist.github.com/mmakowski/379028

https://gist.github.com/mmakowski/379031

https://github.com/dcsobral/ConwayLife

Inverted Index Example (from Spark Notebook)

```
6  val iiFirstPass1 = sc.wholeTextFiles(shakespeare.toString).
7      flatMap { location_contents_tuple2 =>   // two-element tuple: (filename, contents)
8          val words = location_contents_tuple2._2.split("""\W+""")
9          val fileName = location_contents_tuple2._1.split(File.separator).last
10         words.map(word => ((word, fileName), 1))  // two-element tuple with two-elem. tuple key!
11     }.
12     // The previous and next expressions are effectively this query:
13     // "SELECT word, COUNT(*) FROM ... GROUP BY word"
14     reduceByKey((count1, count2) => count1 + count2).
15     map { word_file_count_tup3 =>   // Setup the word as the key, not the (word,filename)
16         (word_file_count_tup3._1._1, (word_file_count_tup3._1._2, word_file_count_tup3._2))
17         // output: (word, (filename, count))
18     }.
19     groupByKey.  // group by words
20     sortByKey(ascending = true).
21     mapValues { iterable =>
22         val vect = iterable.toVector.sortBy { file_count_tup2 =>
23             (-file_count_tup2._2, file_count_tup2._1)  // sort by count descending, then file
24         }
25         vect.mkString(",")   // make a comma-separated string of the (file,count) pairs
26     }
```

Coming Up

A05 - Due Friday 3/31

Q07 and Q08

Project proposals due 3/22

L09 - Scalable Machine Learning

Apache Pig

Apache Pig

Started at Yahoo! Research

- Easier approach for MapReduce
- Procedural language
- PigLatin scripts *interpreted and run as* MapReduce jobs.

Pig Advantages:

- Easier to program than MapReduce.
- Declarative statements directly describe data transformations.
- Optimizer makes efficient decisions.
- Debugging operators:
  - *DESCRIBE, EXPLAIN, ILLUSTRATE*
- Can run “locally” or on Hadoop.

Pig Disadvantages:

- Simple statements may generate many MapReduce jobs.
- Can be hard to debug.
- Keywords are case insensitive
  - *LOAD, USING, AS, GROUP, BY, ...*
- Functions, relations, fields are case sensitive:
  - *PigStorage, COUNT,*

Pig reference materials in Readings/L05 Databases

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

Christopher Olston* Yahoo! Research
Benjamin Reed† Yahoo! Research
Utkarsh Srivastava‡ Yahoo! Research
Ravi Kumar§ Yahoo! Research
Andrew Tomkins¶ Yahoo! Research

ABSTRACT

There is a growing need for ad-hoc analysis of extremely large data sets, especially at internet companies where innovation critically depends on being able to analyze terabytes of data collected every day. Parallel database products, e.g., Teradata, offer a solution, but are usually prohibitively expensive at this scale. Besides, many of the people who analyze this data are entrenched procedural programmers, who find the declarative, SQL style to be unnatural. The success of the more procedural *map-reduce* programming model, and its associated scalable implementations on commodity hardware, is evidence of the above. However, the map-reduce paradigm is too low-level and rigid, and leads to a great deal of custom user code that is hard to maintain, and reuse.

We describe a new language called *Pig Latin* that we have designed to fit in a sweet spot between the declarative style of SQL, and the low-level, procedural style of map-reduce. The accompanying system, Pig, is fully implemented, and compiles Pig Latin into physical plans that are executed over *Hadoop*, an open-source, map-reduce implementation. We give a few examples of how engineers at Yahoo! are using Pig to dramatically reduce the time required for the development and execution of their data analysis tasks, compared to using Hadoop directly. We also report on a novel debugging environment that comes integrated with Pig, that can lead to even higher productivity gains. Pig is an open-source, Apache-incubator project, and available for general use.

Categories and Subject Descriptors:
H.2.3 Database Management: Languages

General Terms: Languages.

*olston@yahoo-inc.com
†breed@yahoo-inc.com
‡utkarsh@yahoo-inc.com
§ravikuma@yahoo-inc.com
¶atomkins@yahoo-inc.com

SIGMOD'08, June 9-12, 2008, Vancouver, BC, Canada.

1. INTRODUCTION

At a growing number of organizations, innovation revolves around the collection and analysis of enormous data sets such as web crawls, search logs, and click streams. Internet companies such as Amazon, Google, Microsoft, and Yahoo! are prime examples. Analysis of this data constitutes the innermost loop of the product improvement cycle. For example, the engineers who develop search engine ranking algorithms spend much of their time analyzing search logs looking for exploitable trends.

The sheer size of these data sets dictates that it be stored and processed on highly parallel systems, such as shared-nothing clusters. Parallel database products, e.g., Teradata, Oracle RAC, Netezza, offer a solution by providing a simple SQL query interface and hiding the complexity of the physical cluster. These products however, can be prohibitively expensive at web scale. Besides, they wrench programmers away from their preferred method of analyzing data, namely writing imperative scripts or code, toward writing declarative queries in SQL, which they often find unnatural, and overly restrictive.

As evidence of the above, programmers have been flocking to the more procedural *map-reduce* [4] programming model. A map-reduce program essentially performs a group-by-aggregation in parallel over a cluster of machines. The programmer provides a map function that dictates how the grouping is performed, and a reduce function that performs the aggregation. What is appealing to programmers about this model is that there are only two high-level declarative primitives (map and reduce) to enable parallel processing, but the rest of the code, i.e., the map and reduce functions, can be written in any programming language of choice, and without worrying about parallelism.

Unfortunately, the map-reduce model has its own set of limitations. Its one-input, two-stage data flow is extremely rigid. To perform tasks having a different data flow, e.g., joins or n stages, inelegant workarounds have to be devised. Also, custom code has to be written for even the most common operations, e.g., projection and filtering. These factors lead to code that is difficult to reuse and maintain, and in which the semantics of the analysis task are obscured. Moreover, the opaque nature of the map and reduce functions impedes the ability of the system to perform optimizations.

We have developed a new language called *Pig Latin* that combines the best of both worlds: high-level declarative querying in the spirit of SQL, and low-level, procedural programming à la map-reduce.

Olston 2008 Pig

Building a High-Level Dataflow System on top of Map-Reduce: The Pig Experience

Alan F. Gates, Olga Natkovich, Shubham Chopra, Pradeep Kamath, Shravan M. Narayanamurthy, Christopher Olston, Benjamin Reed, Santhosh Srinivasan, Utkarsh Srivastava
Yahoo!, Inc.*

ABSTRACT

Increasingly, organizations capture, transform and analyze enormous data sets. Prominent examples include internet companies and e-science. The *Map-Reduce* scalable dataflow paradigm has become popular for these applications. Its simple, explicit dataflow programming model is favored by some over the traditional high-level declarative approach: SQL. On the other hand, the extreme simplicity of Map-Reduce leads to much low-level hacking to deal with the many-step, branching dataflows that arise in practice. Moreover, users must repeatedly code standard operations such as *join* by hand. These practices waste time, introduce bugs, harm readability, and impede optimizations.

Pig is a high-level dataflow system that aims at a sweet spot between SQL and Map-Reduce. Pig offers SQL-style high-level data manipulation constructs, which can be assembled in an explicit dataflow and interleaved with custom Map- and Reduce-style functions or executables. Pig programs are compiled into sequences of Map-Reduce jobs, and executed in the *Hadoop* Map-Reduce environment. Both Pig and Hadoop are open-source projects administered by the Apache Software Foundation.

This paper describes the challenges we faced in developing Pig, and reports performance comparisons between Pig execution and raw Map-Reduce execution.

1. INTRODUCTION

Organizations increasingly rely on ultra-large-scale data processing in their day-to-day operations. For example, modern internet companies routinely process petabytes of web content and usage logs to populate search indexes and perform ad-hoc mining tasks for research purposes. The data includes unstructured elements (e.g., web page text; images) as well as structured elements (e.g., web page click records; extracted entity-relationship models). The processing combines generic relational-style operations (e.g., filter; join; count) with specialized domain-specific operations (e.g., part-of-speech tagging; face detection). A similar situation arises in e-science, national intelligence, and other domains.

The popular *Map-Reduce* [8] scalable data processing framework, and its open-source realization *Hadoop* [1], cater to these workloads and offer a simple dataflow programming model that appeals to many users. However, in practice, the extreme simplicity of the Map-Reduce programming model leads to several problems. First, it does not directly support complex N-step dataflows, which often arise in practice. Map-Reduce also lacks explicit support for combined processing of multiple data sets (e.g., joins and other data matching operations), a crucial aspect of knowledge discovery. Lastly, frequently-needed data manipulation primitives like filtering, aggregation and top-k thresholding must be coded by hand.

Consequently, users end up stitching together Map-Reduce dataflows by hand, hacking multi-input flows, and repeatedly implementing standard operations inside black-box functions. These practices slow down data analysis, introduce mistakes, make data processing programs difficult to read, and impede automated optimization.

Our *Pig* system [4] offers composable high-level data manipulation constructs in the spirit of SQL, while at the same time retaining the properties of Map-Reduce systems that make them attractive for certain users, data types, and workloads. In particular, as with Map-Reduce, Pig programs encode explicit dataflow graphs, as opposed to implicit dataflow as in SQL. As one user from Adobe put it:

> "Pig seems to give the necessary parallel programming constructs (FOREACH, FLATTEN, COGROUP .. etc) and also give sufficient control back to the programmer (which a purely declarative approach like [SQL on top of Map-Reduce][1] doesn't)."

Pig dataflows can interleave built-in relational-style operations like filter and join, with user-provided executables (scripts or pre-compiled binaries) that perform custom processing. Schemas for the relational-style operations can be supplied at the last minute, which is convenient when working with temporary data for which system-managed metadata is more of a burden than a benefit. For data used

*Author email addresses: {gates, olgan, shubhamc, pradeepk, shravanm, olston, breed, sms, utkarsh}@yahoo-inc.com.

VLDB '09, August 24-28, 2009, Lyon, France

[1] Reference to specific software project removed.

Gates 2009 The Pig Experience

Pig Latin Reference Manual 2

by

Table of contents

PigLatin Reference Manu

Famous example:
Pig program to find top 5 websites for Twitter users age 18-25

```
Users    = load 'users' as (name, age);
Filtered = filter Users by age >= 18 and age <=
 25;
Pages    = load 'pages' as (user, url);
Joined   = join Filtered by name, Pages by
 user;
Grouped  = group Joined by url;
Summed   = foreach Grouped generate group,
           count(Joined) as clicks;
Sorted   = order Summed by clicks desc;
Top5     = limit Sorted 5;
store Top5 into 'top5sites';
```

Equivalent MapReduce program (in Java)

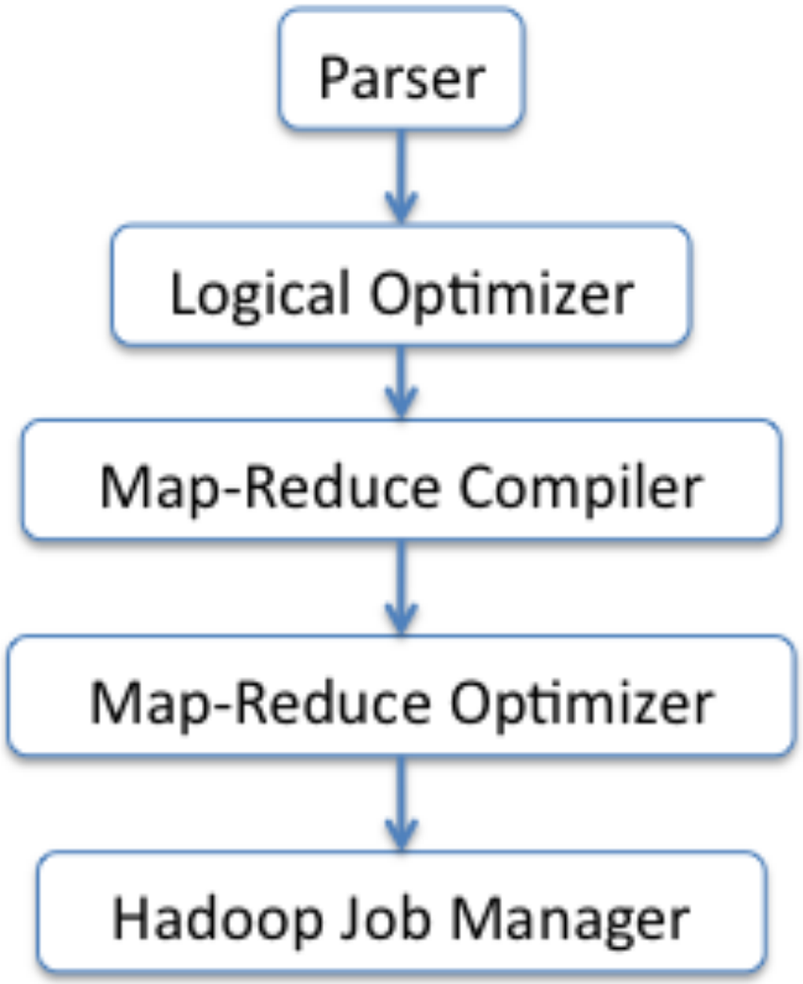
```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.KeyValueTextInputFormat;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.RecordReader;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.SequenceFileInputFormat;
import org.apache.hadoop.mapred.SequenceFileOutputFormat;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.jobcontrol.Job;
import org.apache.hadoop.mapred.jobcontrol.JobControl;
import org.apache.hadoop.mapred.lib.IdentityMapper;

public class MRExample {
    public static class LoadPages extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String key = line.substring(0, firstComma);
            String value = line.substring(firstComma + 1);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("1" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class LoadAndFilterUsers extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String value = line.substring(firstComma + 1);
            int age = Integer.parseInt(value);
            if (age < 18 || age > 25) return;
            String key = line.substring(0, firstComma);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("2" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class Join extends MapReduceBase
        implements Reducer<Text, Text, Text, Text> {

        public void reduce(Text key,
                Iterator<Text> iter,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // For each value, figure out which file it's from and
store it
            // accordingly.
            List<String> first = new ArrayList<String>();
            List<String> second = new ArrayList<String>();

            while (iter.hasNext()) {
                Text t = iter.next();
                String value = t.toString();
                if (value.charAt(0) == '1')
first.add(value.substring(1));
                else second.add(value.substring(1));

                reporter.setStatus("OK");
            }

            // Do the cross product and collect the values
            for (String s1 : first) {
                for (String s2 : second) {
                    String outval = key + "," + s1 + "," + s2;
                    oc.collect(null, new Text(outval));
                    reporter.setStatus("OK");
                }
            }
        }
    }
    public static class LoadJoined extends MapReduceBase
        implements Mapper<Text, Text, Text, LongWritable> {

        public void map(
                Text k,
                Text val,
                OutputCollector<Text, LongWritable> oc,
                Reporter reporter) throws IOException {
            // Find the url
            String line = val.toString();
            int firstComma = line.indexOf(',');
            int secondComma = line.indexOf(',', firstComma);
            String key = line.substring(firstComma, secondComma);
            // drop the rest of the record, I don't need it anymore,
            // just pass a 1 for the combiner/reducer to sum instead.
            Text outKey = new Text(key);
            oc.collect(outKey, new LongWritable(1L));
        }
    }
    public static class ReduceUrls extends MapReduceBase
        implements Reducer<Text, LongWritable, WritableComparable,
Writable> {

        public void reduce(
                Text key,
                Iterator<LongWritable> iter,
                OutputCollector<WritableComparable, Writable> oc,
                Reporter reporter) throws IOException {
            // Add up all the values we see

            long sum = 0;
            while (iter.hasNext()) {
                sum += iter.next().get();
                reporter.setStatus("OK");
            }

            oc.collect(key, new LongWritable(sum));
        }
    }
    public static class LoadClicks extends MapReduceBase
        implements Mapper<WritableComparable, Writable, LongWritable,
Text> {

        public void map(
                WritableComparable key,
                Writable val,
                OutputCollector<LongWritable, Text> oc,
                Reporter reporter) throws IOException {
            oc.collect((LongWritable)val, (Text)key);
        }
    }
    public static class LimitClicks extends MapReduceBase
        implements Reducer<LongWritable, Text, LongWritable, Text> {

        int count = 0;
        public void reduce(
            LongWritable key,
            Iterator<Text> iter,
            OutputCollector<LongWritable, Text> oc,
            Reporter reporter) throws IOException {

            // Only output the first 100 records
            while (count < 100 && iter.hasNext()) {
                oc.collect(key, iter.next());
                count++;
            }
        }
    }
    public static void main(String[] args) throws IOException {
        JobConf lp = new JobConf(MRExample.class);
        lp.setJobName("Load Pages");
        lp.setInputFormat(TextInputFormat.class);
        lp.setOutputKeyClass(Text.class);
        lp.setOutputValueClass(Text.class);
        lp.setMapperClass(LoadPages.class);
        FileInputFormat.addInputPath(lp, new
Path("/user/gates/pages"));
        FileOutputFormat.setOutputPath(lp,
            new Path("/user/gates/tmp/indexed_pages"));
        lp.setNumReduceTasks(0);
        Job loadPages = new Job(lp);

        JobConf lfu = new JobConf(MRExample.class);
        lfu.setJobName("Load and Filter Users");
        lfu.setInputFormat(TextInputFormat.class);
        lfu.setOutputKeyClass(Text.class);
        lfu.setOutputValueClass(Text.class);
        lfu.setMapperClass(LoadAndFilterUsers.class);
        FileInputFormat.addInputPath(lfu, new
Path("/user/gates/users"));
        FileOutputFormat.setOutputPath(lfu,
            new Path("/user/gates/tmp/filtered_users"));
        lfu.setNumReduceTasks(0);
        Job loadUsers = new Job(lfu);

        JobConf join = new JobConf(MRExample.class);
        join.setJobName("Join Users and Pages");
        join.setInputFormat(KeyValueTextInputFormat.class);
        join.setOutputKeyClass(Text.class);
        join.setOutputValueClass(Text.class);
        join.setMapperClass(IdentityMapper.class);
        join.setReducerClass(Join.class);
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/indexed_pages"));
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/filtered_users"));
        FileOutputFormat.setOutputPath(join, new
Path("/user/gates/tmp/joined"));
        join.setNumReduceTasks(50);
        Job joinJob = new Job(join);
        joinJob.addDependingJob(loadPages);
        joinJob.addDependingJob(loadUsers);

        JobConf group = new JobConf(MRExample.class);
        group.setJobName("Group URLs");
        group.setInputFormat(KeyValueTextInputFormat.class);
        group.setOutputKeyClass(Text.class);
        group.setOutputValueClass(LongWritable.class);
        group.setOutputFormat(SequenceFileOutputFormat.class);
        group.setMapperClass(LoadJoined.class);
        group.setCombinerClass(ReduceUrls.class);
        group.setReducerClass(ReduceUrls.class);
        FileInputFormat.addInputPath(group, new
Path("/user/gates/tmp/joined"));
        FileOutputFormat.setOutputPath(group, new
Path("/user/gates/tmp/grouped"));
        group.setNumReduceTasks(50);
        Job groupJob = new Job(group);
        groupJob.addDependingJob(joinJob);

        JobConf top100 = new JobConf(MRExample.class);
        top100.setJobName("Top 100 sites");
        top100.setInputFormat(SequenceFileInputFormat.class);
        top100.setOutputKeyClass(LongWritable.class);
        top100.setOutputValueClass(Text.class);
        top100.setOutputFormat(SequenceFileOutputFormat.class);
        top100.setMapperClass(LoadClicks.class);
        top100.setCombinerClass(LimitClicks.class);
        top100.setReducerClass(LimitClicks.class);
        FileInputFormat.addInputPath(top100, new
Path("/user/gates/tmp/grouped"));
        FileOutputFormat.setOutputPath(top100, new
Path("/user/gates/top100sitesforusers18to25"));
        top100.setNumReduceTasks(1);
        Job limit = new Job(top100);
        limit.addDependingJob(groupJob);

        JobControl jc = new JobControl("Find top 100 sites for users
18 to 25");
        jc.addJob(loadPages);
        jc.addJob(loadUsers);
        jc.addJob(joinJob);
        jc.addJob(groupJob);
        jc.addJob(limit);
        jc.run();
    }
}
```

Pig takes your program and compiles it into a Hadoop job.

Building a High-Level Dataflow System on top of Map-Reduce: The Pig Experience

Alan F. Gates, Olga Natkovich, Shubham Chopra, Pradeep Kamath, Shravan M. Narayanamurthy, Christopher Olston, Benjamin Reed, Santhosh Srinivasan, Utkarsh Srivastava

Yahoo!, Inc.*




```
urls = LOAD ‘dataset’ AS (url, category, pagerank);
groups = GROUP urls BY category;
bigGroups = FILTER groups BY COUNT(urls)>1000000;
result = FOREACH bigGroups GENERATE
             group, top10(urls);
STORE result INTO ‘myOutput’;
```

Pig builds a "data flow" model from your program.

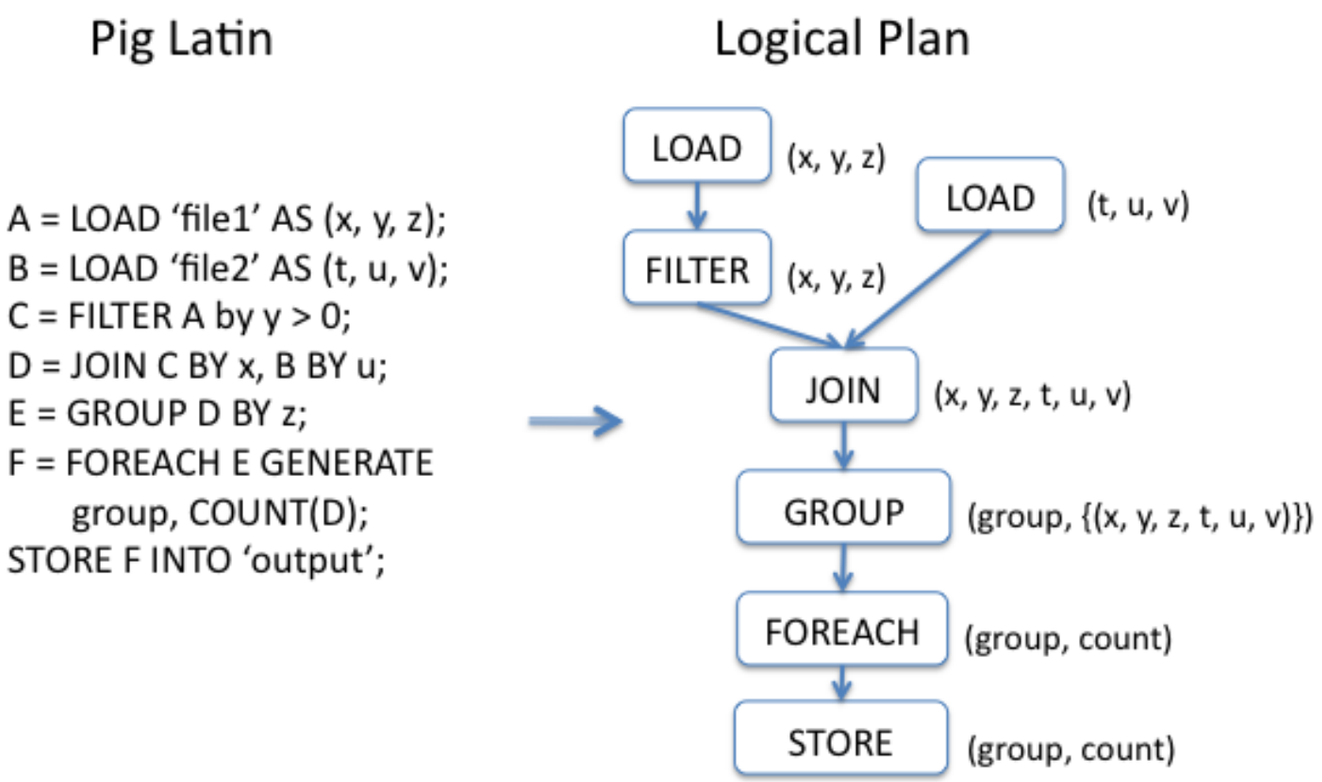
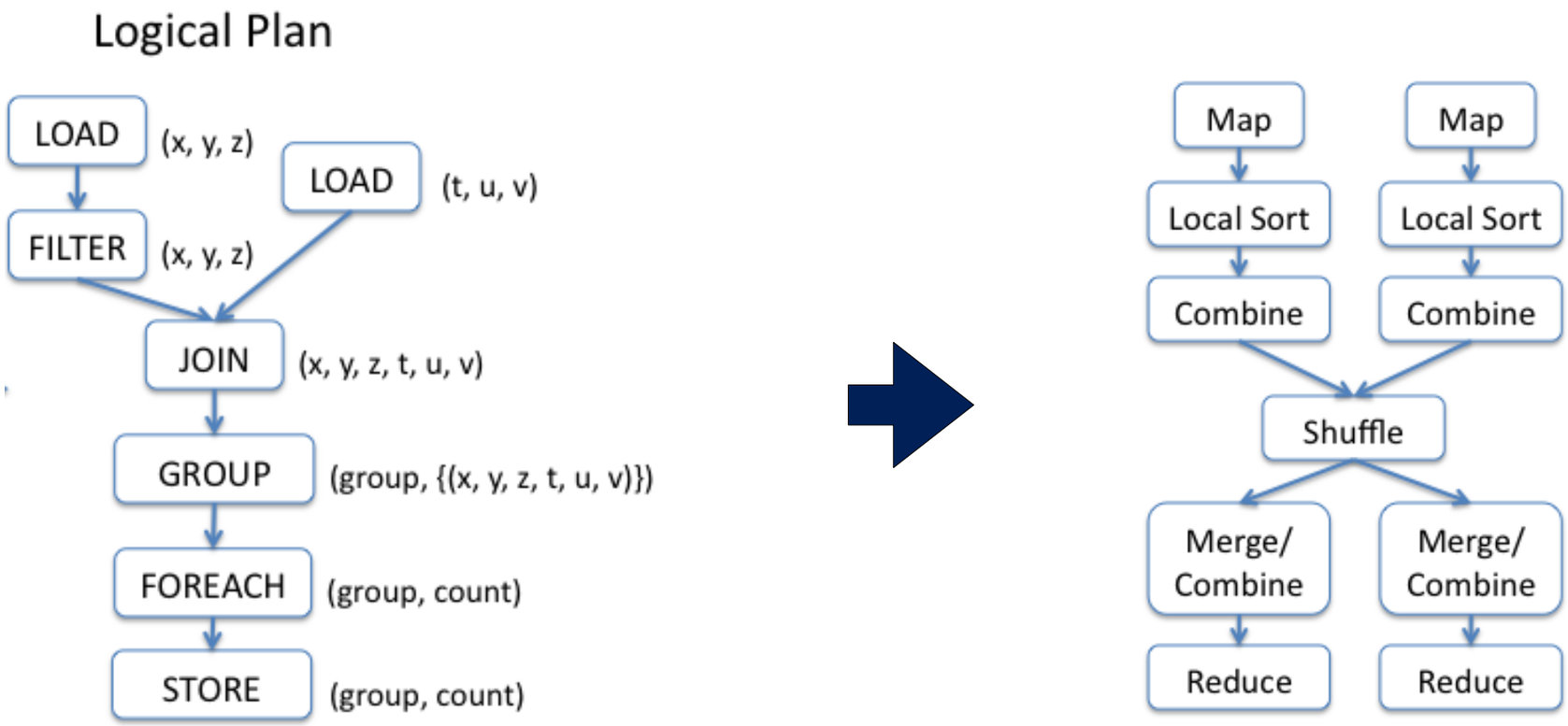


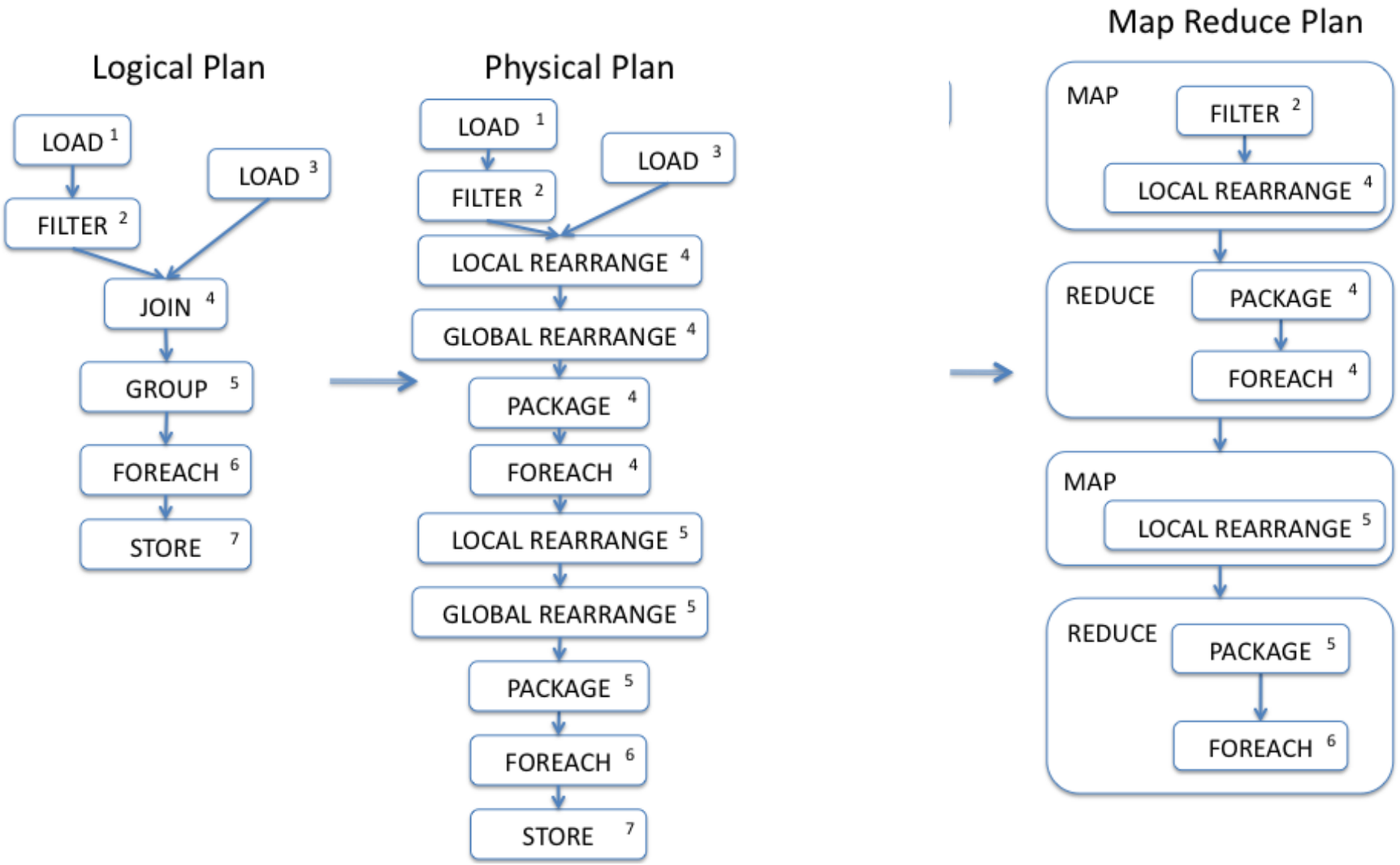
Figure 2: Pig Latin to logical plan translation.

The data flow is translated into a series of MapReduce steps.



Pig Latin to logical plan translation.

Which are translated to an efficient Map Reduce Plan.



Pig Latin Program — Basic Program Design

Basic Pig Latin program:

- LOAD data from a file system (HDFS or S3)
- Transform the data.
- STORE to file system or DUMP to output.

- A = LOAD *filename* [USING function] [AS schema];

e.g.:

- A = LOAD 'file';
- A = LOAD *filename* USING BinStorage();
- A = LOAD *filename* USING PigStorage(*field_delimiter)*;
- A = LOAD *filename* USING PigStorage() AS (*field_desc);*

Pig Data Loading Functions:

Pig Latin Program — Basic Program Design

Basic Pig Latin program:

- LOAD data from a file system (HDFS or S3)
- Transform the data.
- STORE to file system or DUMP to output.

- FILTER

```
B = FILTER A BY $1 == 1;
B = FILTER A BY date ==
“1980-01-01”;
B = FILTER A BY $1 > 50;
```

- ORDER BY

```
C = ORDER B BY $0;
C = ORDER B BY date;
```

- LIMIT

```
D = LIMIT B 30;
```

- JOIN

```
D = JOIN C BY $1, B BY $1;
D = JOIN C BY ipaddress, D BY
ipaddress;
```

Pig transformation examples:

Pig Latin Program — Basic Program Design

Basic Pig Latin program:

- LOAD data from a file system (HDFS or S3)
- Transform the data.
- STORE to file system or DUMP to output.

- STORE

```
STORE A INTO 'outputfile';
STORE A INTO 'outputfile.gz';


-- Store UTF-8:
STORE A INTO 'output' USING PigDump();

-- Store in Binary
STORE A INTO 'output' USING
BinStorage();

--- Store with delimiters:
STORE A INTO 'output' USING
PigStorage('*');
```

Pig Storage examples:

Pig can run locally or on MapReduce

Which version am I running?

```
$ pig -help
```

Pig modes of operation:

Warning: EMR has problems with pig -x local

	Local Mode	MapReduce Mode
Interactive	`$ pig -x local`	`$ pig -x mapreduce`
Batch	`$ pig -x local` *`filename.pig`*	`$ pig -x mapreduce` *`filename.pig`*

Pig Latin statements work with relations.
A = LOAD 'foo.txt' A is a *relation*.

A relation is a "bag."

- A **bag** is a collection of **tuples**.
- A **tuple** is an ordered set of **fields**
- A **field** is a piece of **data**.

Pig Data Types:

- Scalar types: **int, long, double, chararray**
- map — An “associative array” (like a python dictionary)

```
chararray : anytype
```

—*e.g.*

```
“first” : “George”
“last”  : “Washington”
“born”  : 1732
```

- tuple

```
(v0, v1, v2, ...)
```

- bag — a collection of tuples

```
((a, b, c),
 (d, e, f),
 ...
)
```

Example (from reference guide)

```
A = LOAD 'student' USING
PigStorage()
          AS (name:chararray,
age:int, gpa:float);
DUMP A;
(John,18,4.0F)
(Mary,19,3.8F)
(Bill,20,3.9F)
(Joe,18,3.8F)
```

	First Field	SecondField	Third Field
Data Type	chararray	int	float
Positional notation	$0	$1	$2
Possible name	name	age	gpa
Field value	John	18	4.0

It's best to use names!

Pig Latin FOREACH ... GENERATE

FOREACH ... GENERATE creates new relations from old ones.

Example (from reference guide):

```
A = LOAD 'student' USING
PigStorage()
AS (name:chararray, age:int,
gpa:float);
DUMP A;
(John,18,4.0F)
(Mary,19,3.8F)
(Bill,20,3.9F)
(Joe,18,3.8F)


X = FOREACH A GENERATE name,$2;
DUMP X;
(John,4.0F)
(Mary,3.8F)
(Bill,3.9F)
(Joe,3.8F)
```

Simple data types:

Simple Data Types	Description	Example
Scalars		
int	Signed 32-bit integer	10
long	Signed 64-bit integer	Data: 10L or 10l Display: 10L
float	32-bit floating point	Data: 10.5F or 10.5f or 10.5e2f or 10.5E2F Display: 10.5F or 1050.0F
double	64-bit floating point	Data: 10.5 or 10.5e2 or 10.5E2 Display: 10.5 or 1050.0
Arrays		
chararray	Character array (string) in Unicode UTF-8 format	hello world
bytearray	Byte array (blob)	

Pig is a complete data flow programming language

Functions:

- +, -, *, /, %,

NULL:

- Operations can return NULL;
  NULL is ignored by AVG(), MIN(),
  MAX(), SUM(), COUNT()

Conditions:

- ==, !=, >, <, >=, <=

Conditionals:

- NO IF STATEMENT!
- *conditional* ? *if-true* : *if-false*

Pig is a complete data flow programming language

Functions:

- +, -, *, /, %,

NULL:

- Operations can return NULL; NULL is ignored by AVG(), MIN(), MAX(), SUM(), COUNT()

Conditions:

- ==, !=, >, <, >=, <=

Conditionals:

- NO IF STATEMENT!
- *conditional* ? *if-true* : *if-false*

Example from Pig Latin Reference Manual:

```
A = LOAD 'data' AS (f1:int,
f2:int, :bag{T:tuple(t1:int,t2:int)});
DUMP A;
(10,1,{(2,3),(4,6)})
(10,3,{(2,3),(4,6)})
(10,6,{(2,3),(4,6),(5,7)})

X = FOREACH A GENERATE f1, f2, f1%f2;
DUMP X;
(10,1,0)
(10,3,1)
(10,6,4)

X = FOREACH A GENERATE f2, (f2==1?1:COUNT(B));
DUMP X;
(1,1L)
(3,2L)
(6,3L)
```

Word Count with Pig

```
lines     = LOAD 's3://gu-anly502/ps02/tobe.txt' as (line:chararray);
words     = FOREACH lines generate flatten(TOKENIZE(line)) as word;
grouped   = GROUP words by word;
wordcount = FOREACH grouped GENERATE group, COUNT(words);
dump wordcount;
```

LOAD — Loads the data

FOREACH — TOKENIZEs each line. Creates a "words" alias where each tuple is a "word"

GROUP — combines words that have the same word

FOREACH — counts the number of words in each group.

DUMP — sends to standard output.

Note:

- Put spaces around the equals sign (=) !
- Most Pig words are case-sensitive. (Exception: built-in statements like LOAD, FOREACH, GROUP and GENERATE).

grunt> — the Pig command line

```
grunt> help
Commands:
<pig latin statement>; - See the PigLatin manual for details: http://hadoop.apache.org/pig
File system commands:
    fs <fs arguments> - Equivalent to Hadoop dfs command: http://hadoop.apache.org/common/docs/current/hdfs_shell.html
Diagnostic commands:
    describe <alias>[::<alias] - Show the schema for the alias. Inner aliases can be described as A::B.
    explain [-script <pigscript>] [-out <path>] [-brief] [-dot|-xml] [-param <param_name>=<param_value>]
        [-param_file <file_name>] [<alias>] - Show the execution plan to compute the alias or for entire script.
        -script - Explain the entire script.
        -out - Store the output into directory rather than print to stdout.
        -brief - Don't expand nested plans (presenting a smaller graph for overview).
        -dot - Generate the output in .dot format. Default is text format.
        -xml - Generate the output in .xml format. Default is text format.
        -param <param_name - See parameter substitution for details.
        -param_file <file_name> - See parameter substitution for details.
        alias - Alias to explain.
    dump <alias> - Compute the alias and writes the results to stdout.
Utility Commands:
    exec [-param <param_name>=param_value] [-param_file <file_name>] <script> -
        Execute the script with access to grunt environment including aliases.
        -param <param_name - See parameter substitution for details.
        -param_file <file_name> - See parameter substitution for details.
        script - Script to be executed.
    run [-param <param_name>=param_value] [-param_file <file_name>] <script> -
        Execute the script with access to grunt environment.
        -param <param_name - See parameter substitution for details.
        -param_file <file_name> - See parameter substitution for details.
        script - Script to be executed.
    sh  <shell command> - Invoke a shell command.
    kill <job_id> - Kill the hadoop job specified by the hadoop job id.
    set <key> <value> - Provide execution parameters to Pig. Keys and values are case sensitive.
        The following keys are supported:
        default_parallel - Script-level reduce parallelism. Basic input size heuristics used by default.
        debug - Set debug on or off. Default is off.
        job.name - Single-quoted name for jobs. Default is PigLatin:<script name>
        job.priority - Priority for jobs. Values: very_low, low, normal, high, very_high. Default is normal
        stream.skippath - String that contains the path. This is used by streaming.
        any hadoop property.
    help - Display this message.
    history [-n] - Display the list statements in cache.
        -n Hide line numbers.
    quit - Quit the grunt shell.
grunt>
```

Always ask for "help"

Always read the documentation

Grunt supports many Unix commands: ls, cat,

```
grunt> ls s3://gu-anly502/
16/02/15 15:48:52 INFO s3n.S3NativeFileSystem: listStatus s3://gu-anly502/
with recursive false
s3://gu-anly502/bootstrap.sh<r 1>   936
s3://gu-anly502/gutenberg   <dir>
s3://gu-anly502/ps02        <dir>
s3://gu-anly502/ps03        <dir>
s3://gu-anly502/ps04        <dir>
grunt>

grunt> ls s3://gu-anly502/ps02/
16/02/15 15:49:01 INFO s3n.S3NativeFileSystem: listStatus s3://gu-anly502/
ps02 with recursive false
s3://gu-anly502/ps02/hamlet.txt<r 1>        1644
s3://gu-anly502/ps02/tobe.txt<r 1>  43
grunt>

grunt> cat s3://gu-anly502/ps02/tobe.txt
16/02/15 15:49:05 INFO s3n.S3NativeFileSystem: Opening 's3://gu-anly502/
ps02/tobe.txt' for reading
To be, or not to be- that is the question:
grunt>
```

To minimize Pig output — lower the warning level

Pig uses log4j to log. Make a copy of the existing log4j.properties file and edit it:

```
$ cp /etc/pig/conf.dist/log4j.properties log4j_WARN
```

—*set these lines:*

```
# ***** Set root logger level to DEBUG and its only appender to A.
log4j.rootLogger=ERROR, A
log4j.logger.org.apache.pig=warn,A
log4j.logger.org.apache.hadoop=warn,A
```

When you run pig, type:

```
$ pig -4 log4j_WARN
```

Hadoop Word Count in Pig

```
$ pig -4 log4j_WARN
grunt> lines = load 's3://gu-anly502/ps02/tobe.txt' as (line:chararray);
...
grunt> dump lines;
...
(To be, or not to be- )
(that is the question:)
grunt>
...
grunt> words = FOREACH lines generate flatten(TOKENIZE(line)) as word;
grunt> grouped = GROUP words by word;
grunt> wordcount = FOREACH grouped GENERATE group, COUNT(words);
grunt> dump wordcount;
68560 [JobControl] WARN  org.apache.hadoop.mapreduce.JobResourceUploader  - No job jar file
set.  User classes may not be found. See Job or Job#setJar(String).
68560 [JobControl] WARN  org.apache.hadoop.mapreduce.JobResourceUploader  - No job jar file
set.  User classes may not be found. See Job or Job#setJar(String).
68934 [DataStreamer for file /tmp/hadoop-yarn/staging/hadoop/.staging/job_1455488005182_0020/
job.xml block BP-1229375385-172.31.42.104-1455487984302:blk_1073742532_7091] INFO
amazon.emr.metrics.MetricsSaver  - 1 aggregated HDFSWriteDelay 113 raw values into 1
aggregated values, total 1
(To,1)
(be,1)
(is,1)
(or,1)
(to,1)
(be-,1)
(not,1)
(the,1)
(that,1)
(question:,1)
grunt>
```

Sorting the output…

```
grunt> dump wordcount;
68560 [JobControl] WARN  org.apache.hadoop.mapreduce.JobResourceUploader  - No job jar file
set.  User classes may not be found. See Job or Job#setJar(String).
68560 [JobControl] WARN  org.apache.hadoop.mapreduce.JobResourceUploader  - No job jar file
set.  User classes may not be found. See Job or Job#setJar(String).
68934 [DataStreamer for file /tmp/hadoop-yarn/staging/hadoop/.staging/
job_1455488005182_0020/job.xml block
BP-1229375385-172.31.42.104-1455487984302:blk_1073742532_7091] INFO
amazon.emr.metrics.MetricsSaver  - 1 aggregated HDFSWriteDelay 113 raw values into 1
aggregated values, total 1
(To,1)
(be,1)
(is,1)
(or,1)
(to,1)
(be-,1)
(not,1)
(the,1)
(that,1)
(question:,1)

grunt> sorted_wordcount = ORDER wordcount by $0;
grunt> dump sorted_wordcount;
(To,1)
(be,1)
(be-,1)
(is,1)
(not,1)
(or,1)
(question:,1)
(that,1)
(the,1)
(to,1)
```

Working with a larger data set — use LIMIT to limit output.

```
grunt> hamlet = LOAD 's3://gu-anly502/ps02/hamlet.txt' AS (line:chararray);
grunt> words = foreach hamlet generate flatten(TOKENIZE(line)) as word;
grunt> grouped = GROUP words by word;
grunt> wordcount = FOREACH grouped GENERATE group, COUNT(words);
grunt> sorted_words = ORDER wordcount BY $1 DESC;
grunt> sorted_words20 = limit sorted_words 20;
grunt> dump sorted_words20;
(of,14)
(the,14)
(to,9)
(and,7)
(The,6)
(a,5)
(To,5)
(And,5)
(that,4)
(we,4)
(bear,3)
(That,3)
(us,3)
(in,3)
(make,2)
(end,2)
(makes,2)
(all,2)
(For,2)
(have,2)
grunt>
```

Pig Latin scripts can be put in files and run from the command line (like mrjob).

```
$ cat top20.pig
hamlet = LOAD 's3://gu-anly502/ps02/hamlet.txt' AS (line:chararray);
words = foreach hamlet generate flatten(TOKENIZE(line)) as word;
grouped = GROUP words by word;
wordcount = FOREACH grouped GENERATE group, COUNT(words);
sorted_words = ORDER wordcount BY $1 DESC;
sorted_words20 = limit sorted_words 20;
dump sorted_words20;
quit;
$ pig top20.pig -stop-on-failure
...
(of,14)
(the,14)
(to,9)
(and,7)
(The,6)
(a,5)
(To,5)
(And,5)
(that,4)
(we,4)
(bear,3)
(That,3)
(us,3)
(in,3)
(make,2)
(end,2)
(makes,2)
(all,2)
(For,2)
(have,2)
$
```

-stop-on-failure is recommended

Pig Status — don't just ignore it.
Use *store lines into 'outputfile';* to write output to a file.

```
4064342 [main] INFO  org.apache.pig.tools.pigstats.mapreduce.SimplePigStats  - Script Statistics:

HadoopVersion PigVersion UserId     StartedAt  FinishedAt Features
2.7.1-amzn-0  0.14.0-amzn-0        hadoop     2016-02-15 17:10:13   2016-02-15 17:10:34
UNKNOWN

Success!

Job Stats (time in seconds):
JobId         Maps       Reduces    MaxMapTime MinMapTime AvgMapTime MedianMapTime
MaxReduceTime MinReduceTime         AvgReduceTime          MedianReducetime      Alias
Feature       Outputs
job_1455488005182_0036   1          0          6          6          6          0
   0          0          0          lines      MAP_ONLY   hdfs://ip-172-31-42-104.ec2.internal:
8020/user/hadoop/outputfile,

Input(s):
Successfully read 2 records (356 bytes) from: "s3://gu-anly502/ps02/tobe.txt"

Output(s):
Successfully stored 2 records (44 bytes) in: "hdfs://ip-172-31-42-104.ec2.internal:8020/user/
hadoop/outputfile"

Counters:
Total records written : 2
Total bytes written : 44
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_1455488005182_0036


...
```

```
16/02/15 17:10:34 INFO mapreduce.SimplePigStats: Script Statistics:

HadoopVersion          PigVersionUserId    StartedAt FinishedAtFeatures
2.7.1-amzn-0 0.14.0-amzn-0       hadoop    2016-02-15 17:10:13 2016-02-15 17:10:34
UNKNOWN

Success!

Job Stats (time in seconds):
JobId        Maps      Reduces   MaxMapTimeMinMapTimeAvgMapTimeMedianMapTime
MaxReduceTime          MinReduceTime       AvgReduceTime       MedianReducetime    Alias
   Feature   Outputs
job_1455488005182_0036 1         0         6         6         6         6         0
   0         0         0         lines     MAP_ONLY  hdfs://
ip-172-31-42-104.ec2.internal:8020/user/hadoop/outputfile,

Input(s):
Successfully read 2 records (356 bytes) from: "s3://gu-anly502/ps02/tobe.txt"

Output(s):
Successfully stored 2 records (44 bytes) in: "hdfs://ip-172-31-42-104.ec2.internal:8020/
user/hadoop/outputfile"

Counters:
Total records written : 2
Total bytes written : 44
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_1455488005182_0036
```

```
16/02/15 17:10:34 INFO mapreduce.SimplePigStats: Script Statistics:

HadoopVersion          PigVersionUserId    StartedAt FinishedAtFeatures
2.7.1-amzn-0 0.14.0-amzn-0       hadoop    2016-02-15 17:10:13 2016-02-15 17:10:34
UNKNOWN

Success!

Job Stats (time in seconds):
JobId        Maps      Reduces   MaxMapTimeMinMapTimeAvgMapTimeMedianMapTime
MaxReduceTime          MinReduceTime       AvgReduceTime       MedianReducetime    Alias
   Feature   Outputs
job_1455488005182_0036 1         0         6         6         6         6         0
   0         0         0         lines     MAP_ONLY  hdfs://
ip-172-31-42-104.ec2.internal:8020/user/hadoop/outputfile,

Input(s):
Successfully read 2 records (356 bytes) from: "s3://gu-anly502/ps02/tobe.txt"

Output(s):
Successfully stored 2 records (44 bytes) in: "hdfs://ip-172-31-42-104.ec2.internal:8020/
user/hadoop/outputfile"

Counters:
Total records written : 2
Total bytes written : 44
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_1455488005182_0036
```

```
grunt> cat hdfs:///user/hadoop/output
cat hdfs:///user/hadoop/outputfile
To be, or not to be-
that is the question:
grunt>
```

Grunt built-in commands:

```
Was expecting one of:
    <EOF>
    "cat" ...
    "clear" ...
    "fs" ...
    "sh" ...
    "cd" ...
    "cp" ...
    "copyFromLocal" ...
    "copyToLocal" ...
    "dump" ...
    "\\d" ...
    "describe" ...
    "\\de" ...
    "aliases" ...
    "explain" ...
    "\\e" ...
    "help" ...
    "history" ...
    "kill" ...
    "ls" ...
    "mv" ...
    "mkdir" ...
    "pwd" ...
    "quit" ...
    "\\q" ...
    "register" ...
    "rm" ...
    "rmf" ...
    "set" ...
    "illustrate" ...
    "\\i" ...
    "run" ...
    "exec" ...
    "scriptDone" ...
    "" ...
    "" ...
    <EOL> ...
    ";" ...
```

Describe and Illustrate show the structure of relations.

```
grunt> describe lines
describe lines
16/02/15 17:14:10 INFO
Configuration.deprecation:
fs.default.name is deprecated.
Instead, use fs.defaultFS
lines: {line: chararray}

grunt> illustrate lines;
-------------------------------------
--
| lines     | line:chararray
|
-------------------------------------
--
|           | that is the question:
|
-------------------------------------
--
grunt>
```

Pig User Defined Functions (UDFs)

UDFs expand Pig's functionality.

- Parse input lines
- Perform complex operations.
- Example — a UDF could search the MaxMind IP address geolocation database

  —*provided that the database is on each node.*

Coding Options:

- Write in Java — import as registered jar files.
- Write in jython — (Python that generates jar files) — import as registered jar files.
- Write in python — Access with "pig streaming API" (similar to Hadoop streaming)

Pig can process any tab-delimited data.
How do you process data that aren't tab-delimited? (e.g. Apache log files)

Piggybank — a collection of algorithms for pig.

- CommonLogLoader —
  - https://pig.apache.org/docs/r0.14.0/api/org/apache/pig/piggybank/storage/apachelog/CommonLogLoader.html
- CombinedLogLoader:
  - https://pig.apache.org/docs/r0.14.0/api/org/apache/pig/piggybank/storage/apachelog/CombinedLogLoader.html

```
raw = LOAD 'combined_log' USING
org.apache.pig.piggybank.storage.apachelog.CombinedLogLoader AS (remoteAddr,
remoteLogname, user, time, method, uri, proto, status, bytes, referer, userAgent);
```

- Note: I was not able to get CombinedLogLoader to work with the ForensicsWiki logs!

I used REGEX_EXTRACT to extract the log file entries:

```
logs_base =
  FOREACH
   raw_logs
  GENERATE
   FLATTEN ( EXTRACT( line,
     '^(\\S+) (\\S+) (\\S+) \\[([\\w/]+):(\\d{2}:\\d{2}:\\d{2}) [+\\-]\\d{4}\\] "(\
\S+) (\\S+) \\S+" (\\S+) (\\S+) "([^"]*)" "([^"]*)"'
     ) ) AS (
     host: chararray, identity: chararray, user: chararray, date: chararray, time:
chararray, verb: chararray, url: chararray, request: chararray, status: int,
     size: chararray, referrer: chararray, agent: chararray
     );
```

Pig program to produce hits-by-day

```
DEFINE EXTRACT      org.apache.pig.piggybank.evaluation.string.EXTRACT();

raw_logs = load 's3://gu-anly502/ps03/forensicswiki.2012.txt' as (line:chararray);

logs_base =
  FOREACH raw_logs GENERATE FLATTEN (
     EXTRACT( line,
     '^(\\S+) (\\S+) (\\S+) \\[([\\w/]+):(\\d{2}:\\d{2}:\\d{2}) [+\\-]\\d{4}\\]
"(\\S+) (\\S+) \\S+" (\\S+) (\\S+) "([^"]*)" "([^"]*)"'
     ) ) AS (
     host: chararray, identity: chararray, user: chararray, date: chararray, time:
chararray, verb: chararray, url: chararray, request: chararray, status: int,
     size: chararray, referrer: chararray, agent: chararray
     );

by_date = GROUP logs_base BY (date);

date_counts = FOREACH by_date GENERATE
    group as date,       -- the key you grouped on
    COUNT(logs_base);    -- the number of log lines wiht this date

dump date_counts;
```

Pig output

```
$ pig parse_apache.pig
16/02/21 20:18:47 INFO pig.ExecTypeProvider: Trying ExecType : LOCAL
16/02/21 20:18:47 INFO pig.ExecTypeProvider: Trying ExecType : MAPREDUCE
16/02/21 20:18:47 INFO pig.ExecTypeProvider: Picked MAPREDUCE as the ExecType
45   [main] INFO  org.apache.pig.Main  - Apache Pig version 0.14.0-amzn-0 (r:
unknown) compiled Jan 14 2016, 02:55:53
16/02/21 20:18:47 INFO pig.Main: Apache Pig version 0.14.0-amzn-0 (r: unknown)
compiled Jan 14 2016, 02:55:53
...
16/02/21 20:23:09 INFO util.MapRedUtil: Total input paths to process : 5
(01/Jul/2012,35039)
(01/Sep/2012,33272)
(02/Jul/2012,46445)
(02/Sep/2012,36225)
(03/Jul/2012,43922)
(03/Sep/2012,40703)
(04/Jul/2012,38576)
...
(30/Jul/2012,45488)
(30/Sep/2012,37817)
(31/Jul/2012,48353)
263298 [main] INFO  org.apache.pig.Main  - Pig script completed in 4 minutes, 23
seconds and 386 milliseconds (263386 ms)
16/02/21 20:23:10 INFO pig.Main: Pig script completed in 4 minutes, 23 seconds and
386 milliseconds (263386 ms)

[20:23:11 last: 266s][~/ANLY502/L05]
$
```

266 seconds to process 4GB file!

Parse the date as a "datetime" and create a new relation with just the desired fields.

Old regular expression:

```
logs_base =
  FOREACH raw_logs GENERATE FLATTEN (
     EXTRACT( line,
     '^(\\S+) (\\S+) (\\S+) \\[([\\w/]+):(\\d{2}:\\d{2}:\\d{2}) [+\\-]\\d{4}\\]
"(\\S+) (\\S+) \\S+" (\\S+) (\\S+) "([^"]*)" "([^"]*)"'
     ) ) AS (
     host: chararray, identity: chararray, user: chararray, date: chararray, time:
chararray, verb: chararray, url: chararray, request: chararray, status: int,
     size: chararray, referrer: chararray, agent: chararray);
```

New:

```
logs_base =
  FOREACH
   raw_logs
  GENERATE
   FLATTEN ( EXTRACT( line,
     '^(\\S+) (\\S+) (\\S+) \\[([^\\]]+)\\] "(\\S+) (\\S+) \\S+" (\\S+) (\\S+)
"([^"]*)" "([^"]*)"'
     ) ) AS (
     host: chararray, identity: chararray, user: chararray, datetime_str:
chararray, verb: chararray, url: chararray, request: chararray, status: int,
     size: int, referrer: chararray, agent: chararray
     );
```

"schema"

```
logs = FOREACH logs_base GENERATE ToDate(datetime_str,'dd/MMM/yyyy:HH:mm:ss Z') AS
date, host, url, size;
```

"describe" and "explain"

```
logs = FOREACH logs_base GENERATE ToDate(datetime_str,'dd/MMM/yyyy:HH:mm:ss Z') AS
date, host, url, size;
```

Describe logs:

```
logs: {date: datetime,host: chararray,url: chararray,size: int}
```

Explain logs:

"describe" and "explain"

```
logs = FOREACH logs_base GENERATE ToDate(datetime_str,'dd/MMM/yyyy:HH:mm:ss Z') AS
date, host, url, size;
```

Describe logs:

```
logs: {date: datetime,host: chararray,url: chararray,size: int}
```

Explain logs:

```
#-----------------------------------------------
# New Logical Plan:
#-----------------------------------------------
logs: (Name: LOStore Schema: date#58:datetime,host#46:chararray,url#51:chararray,size#54:int)
|
|---logs: (Name: LOForEach Schema: date#58:datetime,host#46:chararray,url#51:chararray,size#54:int)
    |   |
    |   (Name: LOGenerate[false,false,false,false] Schema: date#58:datetime,host#46:chararray,url#51:chararray,size#5
Uids=[51, 54, 58, 46]
    |   |   |
    |   |   (Name: UserFunc(org.apache.pig.builtin.ToDate2ARGS) Type: datetime Uid: 58)
    |   |   |
    |   |   |---datetime_str:(Name: Project Type: chararray Uid: 49 Input: 0 Column: (*))
    |   |   |
    |   |   |---(Name: Constant Type: chararray Uid: 57)
    |   |   |
    |   |   host:(Name: Project Type: chararray Uid: 46 Input: 1 Column: (*))
    |   |   |
    |   |   url:(Name: Project Type: chararray Uid: 51 Input: 2 Column: (*))
    |   |   |
    |   |   size:(Name: Project Type: int Uid: 54 Input: 3 Column: (*))
    |   |
    |   |---(Name: LOInnerLoad[3] Schema: datetime_str#49:chararray)
    |   |
    |   |---(Name: LOInnerLoad[0] Schema: host#46:chararray)
    |   |
    |   |---(Name: LOInnerLoad[5] Schema: url#51:chararray)
    |   |
    |   |---(Name: LOInnerLoad[8] Schema: size#54:int)
    |
    |---logs_base: (Name: LOForEach Schema: host#74:chararray,identity#75:chararray,user#76:chararray,datetime_str#77
rray,status#81:int,size#82:int,referrer#83:chararray,agent#84:chararray)
        |   |
        |   (Name: LOGenerate[true] Schema: host#74:chararray,identity#75:chararray,user#76:chararray,datetime_str#77
rray,status#81:int,size#82:int,referrer#83:chararray,agent#84:chararray)
        |   |   |
        |   |   (Name: UserFunc(org.apache.pig.piggybank.evaluation.string.EXTRACT) Type: tuple Uid: 73)
        |   |   |
        |   |   |---(Name: Cast Type: chararray Uid: 17)
        |   |   |   |
        |   |   |   |---line:(Name: Project Type: bytearray Uid: 17 Input: 0 Column: (*))
        |   |   |
        |   |   |---(Name: Constant Type: chararray Uid: 72)
        |   |
        |   |---(Name: LOInnerLoad[0] Schema: line#17:bytearray)
        |
        |---raw_logs: (Name: LOLoad Schema: line#17:bytearray)RequiredFields:null
"
```

Explain "Logical Plan"

"describe" and "explain"

```
logs = FOREACH logs_base GENERATE ToDate(datetime_str,'dd/MMM/yyyy:HH:mm:ss Z') AS
date, host, url, size;
```

Describe logs:

```
logs: {date: datetime,host: chararray,url: chararray,size: int}
```

Explain logs:

"describe" and "explain"

```
logs = FOREACH logs_base GENERATE ToDate(datetime_str,'dd/MMM/yyyy:HH:mm:ss Z') AS
date, host, url, size;
```

Describe logs:

```
logs: {date: datetime,host: chararray,url: chararray,size: int}
```

Explain logs:

```
#-----------------------------------------------
# Physical Plan:
#-----------------------------------------------
logs: Store(fakefile:org.apache.pig.builtin.PigStorage) - scope-18
|
|---logs: New For Each(false,false,false,false)[bag] - scope-17
    |   |
    |   POUserFunc(org.apache.pig.builtin.ToDate2ARGS)[datetime] - scope-9
    |   |
    |   |---Project[chararray][3] - scope-7
    |   |
    |   |---Constant(dd/MMM/yyyy:HH:mm:ss Z) - scope-8
    |   |
    |   Project[chararray][0] - scope-11
    |   |
    |   Project[chararray][5] - scope-13
    |   |
    |   Project[int][8] - scope-15
    |
    |---logs_base: New For Each(true)[bag] - scope-6
        |   |
        |   POUserFunc(org.apache.pig.piggybank.evaluation.string.EXTRACT)[tuple] - scope-4
        |   |
        |   |---Cast[chararray] - scope-2
        |   |   |
        |   |   |---Project[bytearray][0] - scope-1
        |   |
        |   |---Constant(^(\S+) (\S+) (\S+) \[([^\]]+)\] "(\S+) (\S+) \S+" (\S+) (\S+) "([^"]*)" "([^"]*)") - scope-3
        |
        |---raw_logs: Load(s3://gu-anly502/ps03/forensicswiki.2012-01.unzipped/access.log.2012-01-01:org.apache.pig.builtin.PigStorage) - scope-0
```

Explain Physical Plan

"describe" and "explain"

```
logs = FOREACH logs_base GENERATE ToDate(datetime_str,'dd/MMM/yyyy:HH:mm:ss Z') AS
date, host, url, size;
```

Describe logs:

```
logs: {date: datetime,host: chararray,url: chararray,size: int}
```

Explain logs:

"describe" and "explain"

```
logs = FOREACH logs_base GENERATE ToDate(datetime_str,'dd/MMM/yyyy:HH:mm:ss Z') AS
date, host, url, size;
```

Describe logs:

```
logs: {date: datetime,host: chararray,url: chararray,size: int}
```

Explain logs:

```
#-----------------------------------------------
# Map Reduce Plan
#-----------------------------------------------
MapReduce node scope-19
Map Plan
logs: Store(fakefile:org.apache.pig.builtin.PigStorage) - scope-18
|
|---logs: New For Each(false,false,false,false)[bag] - scope-17
    |   |
    |   POUserFunc(org.apache.pig.builtin.ToDate2ARGS)[datetime] - scope-9
    |   |
    |   |---Project[chararray][3] - scope-7
    |   |
    |   |---Constant(dd/MMM/yyyy:HH:mm:ss Z) - scope-8
    |   |
    |   Project[chararray][0] - scope-11
    |   |
    |   Project[chararray][5] - scope-13
    |   |
    |   Project[int][8] - scope-15
    |
    |---logs_base: New For Each(true)[bag] - scope-6
        |   |
        |   POUserFunc(org.apache.pig.piggybank.evaluation.string.EXTRACT)[tuple] - scope-4
        |   |
        |   |---Cast[chararray] - scope-2
        |   |   |
        |   |   |---Project[bytearray][0] - scope-1
        |   |
        |   |---Constant(^(\S+) (\S+) (\S+) \[([^\]]+)\] "(\S+) (\S+) \S+" (\S+) (\S+) "([^"]*)" "([^"]*)") - scope-3
        |
        |---raw_logs: Load(s3://gu-anly502/ps03/forensicswiki.2012-01.unzipped/access.log.2012-01-01:org.apache.pig.builtin.PigStorage) - scope-0--------
```

Explain Map Reduce Plan

"describe" and "explain"

```
logs = FOREACH logs_base GENERATE ToDate(datetime_str,'dd/MMM/yyyy:HH:mm:ss Z') AS
date, host, url, size;
```

Describe logs:

```
logs: {date: datetime,host: chararray,url: chararray,size: int}
```

Explain logs:

Final demo: list of forensicswiki hits by date:

Program:

```
raw_logs = load 's3://gu-anly502/ps03/forensicswiki.2012.txt' as (line:chararray);
logs_base =
  FOREACH
   raw_logs
  GENERATE
   FLATTEN ( EXTRACT( line,
     '^(\\S+) (\\S+) (\\S+) \\[([^\\]]+)\\] "(\\S+) (\\S+) \\S+" (\\S+) (\\S+) "([^"]*)"
"([^"]*)"'
     ) ) AS (
     host: chararray, identity: chararray, user: chararray, datetime_str: chararray, verb:
chararray, url: chararray, request: chararray, status: int,
     size: int, referrer: chararray, agent: chararray
     );

by_date = GROUP logs BY (date);
date_counts = FOREACH by_date GENERATE
    group as date,      -- the key you grouped on
    COUNT(logs_base);   -- the number of log lines wiht this date
dump date_counts;
```

Output:

```
(,0)
(2012-01-01T00:00:00.000Z,29116)
(2012-01-02T00:00:00.000Z,38188)
...
(2012-12-31T00:00:00.000Z,36631)
(2013-01-01T00:00:00.000Z,1283)
329255 [main] INFO  org.apach pig.Main  - Pig script completed in 5 minutes, 29 seconds and 337
milliseconds (329337 ms)
16/02/22 00:43:57 INFO pig.Main: Pig script completed in 5 minutes, 29 seconds and 337
milliseconds (329337 ms)

[00:43:58 last: 331s][~/ANLY502/L05]
$
```

331 seconds! (4x faster that mrjob)

A little cleaner — (2012-12-31T00:00:00.000Z,36631) -> (2012-12-31,36631)

Add a second GENERATE:

```
logs  = FOREACH logs_base GENERATE ToDate(SUBSTRING(datetime_str,0,11),'dd/MMM/
yyyy') AS date, host, url, size;
logs2 = FOREACH logs      GENERATE SUBSTRING(ToString(date),0,10) AS date, host,
url, size;

by_date = GROUP logs2 BY (date);
date_counts = FOREACH by_date GENERATE
    group AS date,    -- the key you grouped on
    COUNT(logs2);      -- the number of log lines wiht this date

date_counts_sorted = ORDER date_counts BY date;
dump date_counts_sorted;
```

And run...

```
(2012-12-28,39090)
(2012-12-29,54360)
(2012-12-30,40828)
(2012-12-31,36631)
(2013-01-01,1283)
368896 [main] INFO  org.apache.pig.Main  - Pig script completed in 6 minutes, 8
seconds and 977 milliseconds (368977 ms)
16/02/22 01:21:35 INFO pig.Main: Pig script completed in 6 minutes, 8 seconds and
977 milliseconds (368977 ms)
[hadoop@ip-172-31-37-188 L05]$ %
```

368 seconds (up from 331)

MaxMind Join with the Forensicswiki Data

```
DEFINE EXTRACT       org.apache.pig.piggybank.evaluation.string.EXTRACT();

raw_logs = load 's3://gu-anly502/ps03/forensicswiki.2012.txt' as (line:chararray);

maxmind  = load 's3://gu-anly502/ps03/maxmind' as (ipaddr:chararray, country:chararray);


logs_base =
  FOREACH
   raw_logs
  GENERATE
   FLATTEN ( EXTRACT( line,
     '^(\\S+) (\\S+) (\\S+) \\[([^\\]]+)\\] "(\\S+) (\\S+) \\S+" (\\S+) (\\S+) "([^"]*)"
"([^"]*)"'
     ) ) AS (
     host: chararray, identity: chararray, user: chararray, datetime_str: chararray, verb:
chararray, url: chararray, request: chararray, status: int,
     size: int, referrer: chararray, agent: chararray
     );

geolocated_logs = JOIN logs_base BY host, maxmind BY ipaddr;
geolocated_50 = LIMIT geolocated_logs 50;
dump geolocated_50;
...
(180.76.5.67,-,-,01/Jan/2012:13:02:39 -0800,GET,/wiki/Special:WhatLinksHere/User_talk:Marc_Yu,
200,3799,-,Mozilla/5.0 (compatible; Baiduspider/2.0; +http://www.baidu.com/search/
spider.html),180.76.5.67,China)
(180.76.5.89,-,-,01/Jan/2012:02:27:53 -0800,GET,/wiki/Special:RecentChangesLinked/Libvshadow,
200,4391,-,Mozilla/5.0 (compatible; Baiduspider/2.0; +http://www.baidu.com/search/
spider.html),180.76.5.89,China)
(180.76.5.89,-,-,01/Jan/2012:21:47:55 -0800,GET,/images/7/79/?C=S;O=D,200,553,-,Mozilla/5.0
(compatible; Baiduspider/2.0; +http://www.baidu.com/search/spider.html),180.76.5.89,China)
```