# Machine-Learning Summer School - 2015

## Big Data Programming

David Franke – Vast.com

http://www.cs.utexas.edu/~dfranke/

# Goals for Today

- Issues to address when you have big data
- Understand two popular "big data" tools/systems
  - How they work, how they are used
  - Vocabulary of these ecosystems
- Some direction on how you can get started
- Identify some ML work using big data

# Learning from Data

- What can we do when the data gets big?
  - Too big for the CPU memory of any single machine
  - Larger than the disk storage of a single machine

- Recent data point:
  - Facebook has ~800 petabyte data cluster (Hadoop)
  - 1 petabyte = $10^{15}$ bytes

- Big data is spread across a network of machines

# Learning from Big Data

- Need to bring distributed storage and distributed processing to bear to handle big data

- Issues:
  - Distributing computation across many machines
  - Maximizing performance
    - Minimize I/O to disk, minimize transfers across the network
  - Combining the results of distributed computation
  - Recovering from failures

# Managing Big Data

- We'll look at two popular tools/systems

- One well established – Hadoop
- One up and coming – Spark

- Basic concepts of each
- How they address the aforementioned issues

# Managing Big Data

- When writing a program with these tools …
  - You don't know the size of the data
  - You don't know the extent of the parallelism
- Both try to collocate the computation with the data
  - Parallelize the I/O
  - Make the I/O local (versus across network)
- Data is often unstructured (vs. relational model)

# Big Data vs. Relational

- RDBMS normalization
  - Goal is to remove redundancy and retain/insure integrity
- Big data apps want reads to be local
  - Send the code to the data, as it much smaller (Jim Gray)
  - Normalization makes read non-local
- Processing examines one input record at a time
  - Minimal state in programs – it's in the data

# Big Data Tools

- This all sounds great. What are the issues?
  - Coordinating the distributed computation
  - Handling partial failures
  - Combining the results of distributed computation

- Tools offer a programming model that abstracts
  - Disk read and write
  - Parallelization (computation and I/O)
  - Combining data (keys and values)

# Hadoop

- Open source project, supported by Yahoo
- Based on work done inside Google
  - MapReduce, GFS
  - Paper: http://research.google.com/archive/mapreduce.html

- Hadoop implements the MapReduce programming paradigm
  - Provides a template for programs
- HDFS – Hadoop Distributed File System

# Unstructured Data

- Tom White, in *Hadoop: The Definitive Guide*

- "*MapReduce works well on unstructured or semistructured data because it is designed to interpret the data at processing time. In other words, the input keys and values for MapReduce are not intrinsic properties of the data, but they are chosen by the persona analyzing the data*."

# MapReduce

- What characterizes a problem suitable for MR?
  - Most or all of the data is processed
    - But viewed in small increments – one input line/record
    - For the most part, map and reduce tasks are stateless
  - Write once, read multiple times
    - Data Warehouse has this intended usage (write once)
  - Unstructured data vs. structured/normalized

- Data pipelines (chains of map-reduce tasks) are common

# MapReduce

Table 1-1, Hadoop – The Definitive Guide

| | Traditional RDBMS | MapReduce |
|---|---|---|
| Data Size | Gigabytes | Petabytes |
| Access | Interactive and batch | Batch |
| Updates | Read and write many times | Write once, read many |
| Structure | Static schema | Dynamic schema |
| Integrity | High | Low |
| Scaling | Nonlinear | Linear |

# Structure of MapReduce

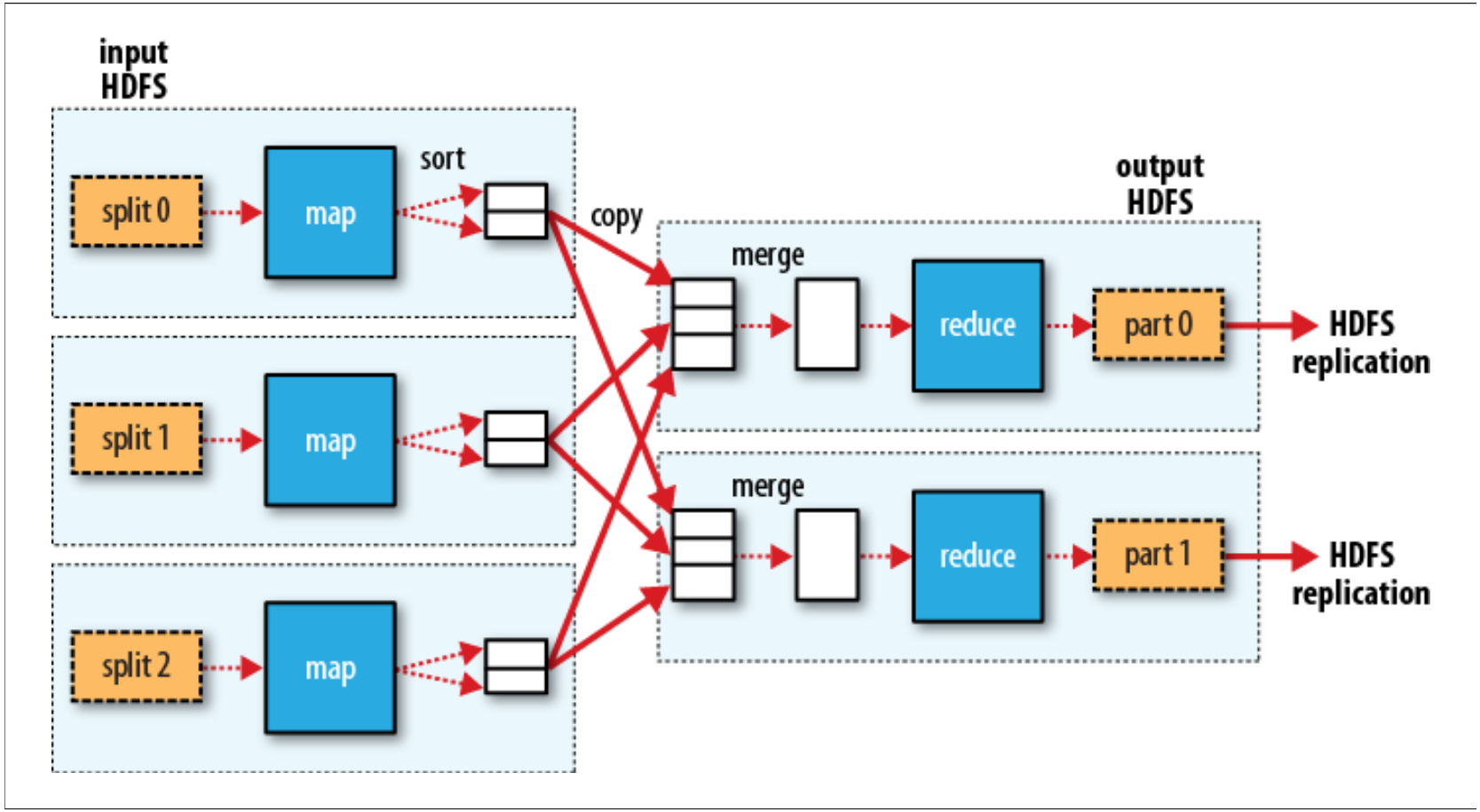
- HDFS stores a data file into multiple parts (partitions)
  - Each partition can be located on a different machine

- A map-reduce program defines
  - Map task
  - Reduce task

- A map task reads and processes one or more partitions
- A reduce task writes one partition of a file

# MapReduce in Hadoop

Figure 2.4, Hadoop - The Definitive Guide



# Map Function

- Map input is a stream of key/value pairs
  - Web logs: Server name (key), log entry (value)
  - Sensor reading: sensor ID (key), sensed values (value)
  - Record number (key), record (value)

- Map function processes each input pair in turn

- For each input pair, the map function can (but isn't required to) emit one or more key/value pairs

# Reduce Function

- Reduce input is a stream of key/value-list pairs
  - These are the key value pairs emitted by the map function
- Reduce function processes each input pair in turn
  - All key/value pairs with the same key sent to same reducer
- For each input pair, the reduce function can (but isn't required) to emit a key/value pair

# WordCount Example

- Multiple text files of arbitrary size, or
- An arbitrary number of documents

- Count the occurrences of all the words in the input

- Output:
  - `word1, count`
  - `word2, count`
  - …

# WordCount Example - Map

- Map task input is a stream of key/value pairs
  - Document ID (key), document (value)

- Map task processes each input pair in turn
  - Extract each word from the document
  - Emits a key/value pair for each word: *<the-word, 1>*

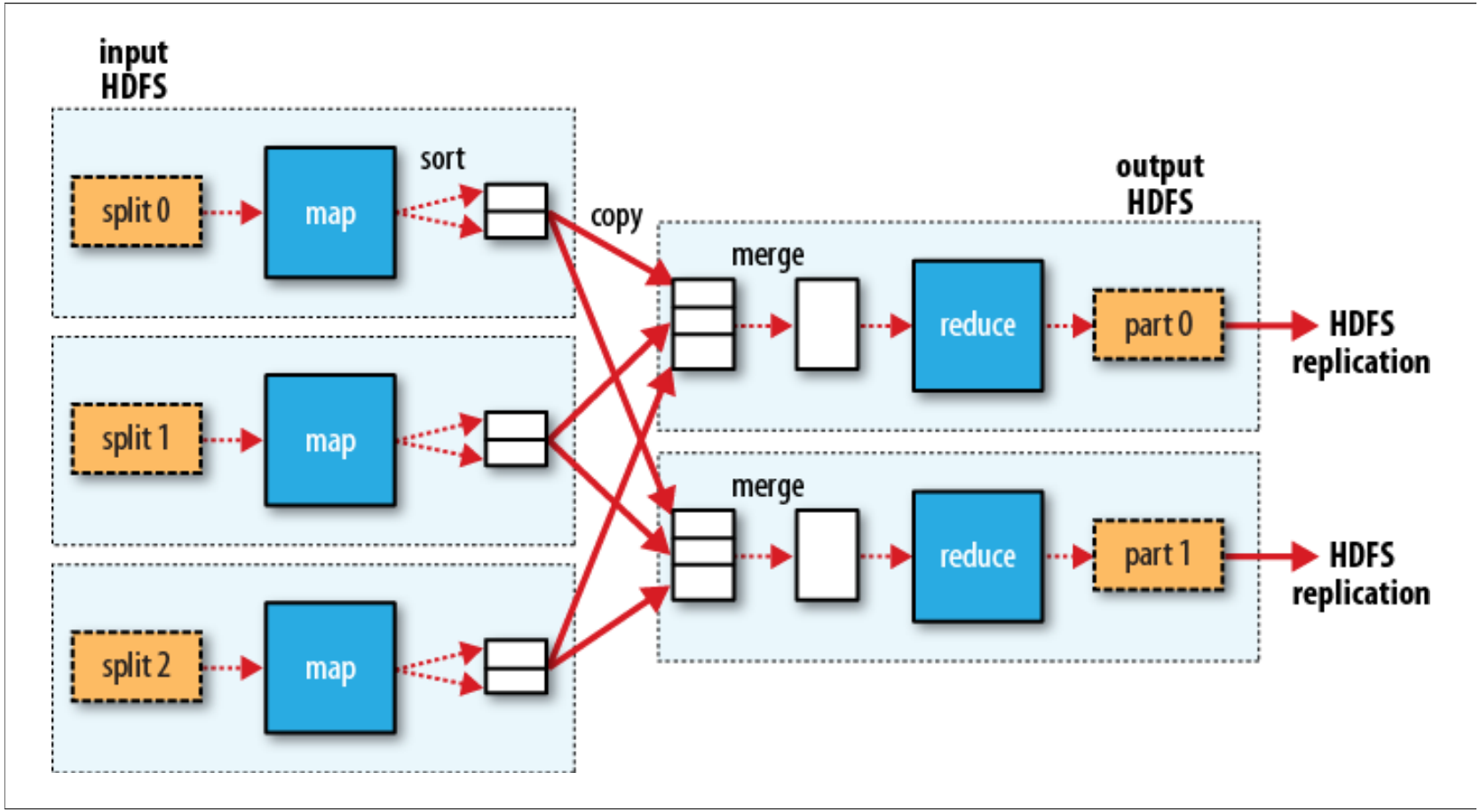
- Each map task emits multiple key/value pairs
  - For all words in the documents received as input

# WordCount Example - Reduce

- Reduce task input is a stream of key/value-list pairs
  - These are the key value pairs emitted by map tasks
  - Key is a text string (the word), value is a list of counts
- Reduce task processes each input pair in turn
  - Sums the values in the value-list
- For each input, the reduce task emits a key/value pair
  - Key is a text string (the word), value is the total count

# MapReduce in Hadoop

Figure 2.4, Hadoop - The Definitive Guide



# WordCount Example Code

- Hadoop map-reduce code example

# MapReduce Design Patterns

- Summarization
- Filtering
- Data Organization
  - Partitioning/binning, sorting, shuffle
- Joins
  - Merging data sets
- Meta-patterns
  - Optimizing map-reduce chains (data pipelines)

# Issues with MapReduce

- One “template”: map, then reduce
- HDFS is its own file system

- In a data pipeline, each map-reduce step
  - Reads all input data from disk
  - Writes all output data to disk
  - Even if output is just an intermediate result

- Addresses failure handling with replicated data
  - Can help performance though

# Apache Spark

- Open source project out of AMPLab at UC Berkeley

- A Spark program defines:
  - Transformations and actions on data sets
  - Data flow, or lineage graph among data sets, induced by the transformations

- Data sets in Spark are called RDDs
  - Resilient Distributed Datasets

# Spark Features

- Provide domain specific libraries
  - Example: map-reduce library
  - Promotes functional programming model
- Access to multiple data (file) systems
  - Local, HDFS, Cassandra, S3, database tables, …
- Lazy evaluation, and caching for performance
  - Reduce or eliminate disk I/O
- Support multi-stage and iterative apps

# Spark RDDs

- Resilient Distributed Dataset
  - One RDD has one or more partition
  - Partitions are distributed across machines
  - Rebuilt from base data on failure (versus replication)
  - Lazy evaluation – created on demand

- RDD types offer various functions
  - map, reduce
  - groupBy, reduceByKey
  - joins (inner, leftOuterJoin, rightOuterJoin)
  - filter, sample

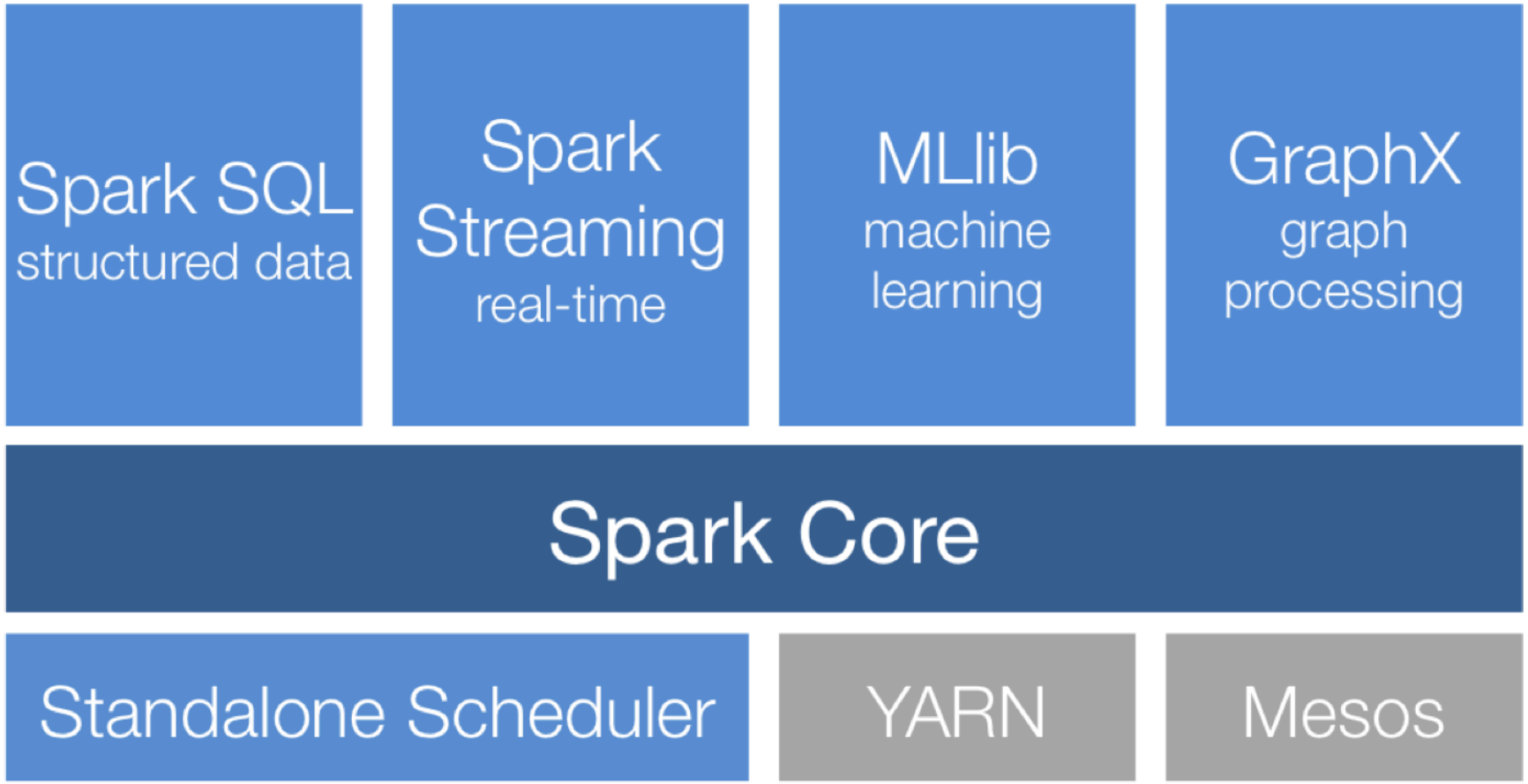
# Spark

- Provides a higher level of abstraction for coding
  - Multi-stage map-reduce pipeline in Hadoop …
  - Can be composed functions in Spark

- RDD support and libraries
  - Spark SQL – RDD representing relational table
  - Streaming data – D-Stream, Twitter stream
  - Graph data – GraphX
  - …

# Spark Stack

Learning Spark, Figure 1-1.



# Spark Programs

- A Spark program defines:
  - RDDs
    - Input from external sources
    - Produced by a transformation
  - Transformations
    - Produce a new RDD from the input RDD
  - Actions
    - Compute something from the input RDD
      - Return non-RDD objects (e.g., number)
    - Write an RDD to external storage

# Spark Transformations

- Transformations take functions as arguments
  - This function defines the details of the transform
  - Function is applied to each element of the RDD

- Filter
  - `rdd2 = rdd1.filter(fn)`
  - `fn` is a boolean function applied to elements of `rdd1`
    - Evaluates to true, element is in `rdd2`
- Union
  - `rdd3 = rdd1.union(rdd2)`
    - `rdd3` has all elements of `rdd1 and rdd2`

# Spark Transformations

- Map
  - `rdd2 = rdd1.map(fn)`
  - `fn` is applied to each element of `rdd1`
- Flat map
  - `rdd2 = rdd1.flatMap(fn)`
    - `fn` returns a list for each element of `rdd1`
    - `rdd2` is the concatenation of these lists
- ....

# Spark Actions

- Transformations do not initiate computation (lazy eval)
- Actions on RDDs initiate computation

- Counting
  - `rdd.count()`
- Extract some elements
  - `rdd1.take(10)`
- Get all elements
  - `rdd.collect()`
  - Careful, as this assembles the entire RDD. Filter first.

# Spark Actions

- Each action on an RDD causes it to be recomputed
- To use an RDD for more than one action
  - We tell Spark to persist the RDD
- Multiple levels of persistence
  - Memory only
  - Memory and disk
  - Disk only
- Can result in big performance wins over map-reduce

# Spark Code Example

- WordCount again

# Spark – Other Stuff

- Transformations allow the program to control the amount of parallelism
  - Specify the partitions in the RDD
  - By default, partitions in output RDD same as input RDD

- The Spark PairRDD (key/value)
  - Provides map-reduce functionality

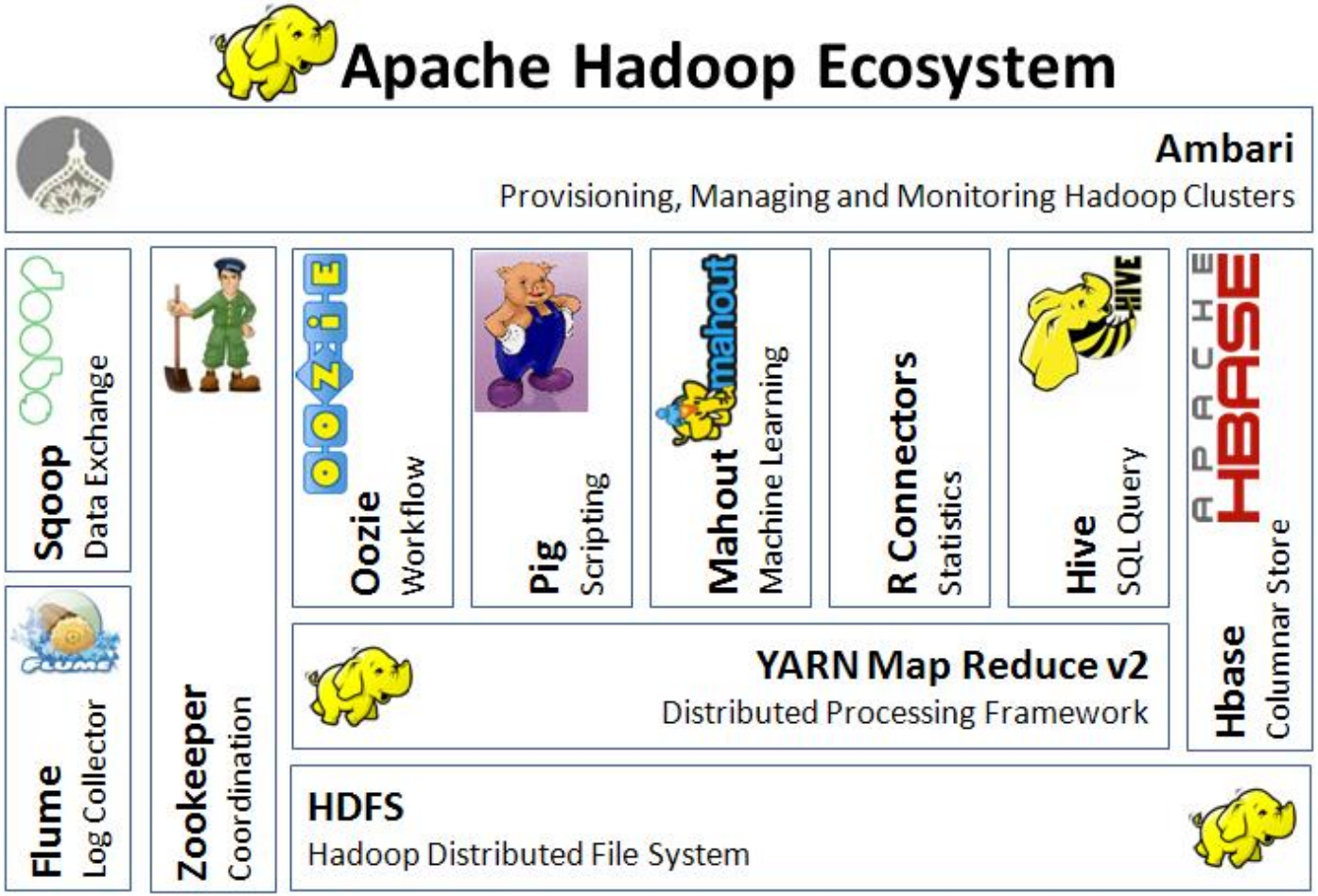
# Machine Learning on Big Data

- Mahout
  - Algorithms implemented using map-reduce
  - As of last year (April 2014), moving to Spark
  - See: http://mahout.apache.org/

- Spark library for machine learning: MLlib
  - http://spark.apache.org/docs/1.1.1/mllib-guide.html
-
- $H_2O$: Another Spark based machine learning tool
  - http://oxdata.com
  - Mention of possible integration with Mahout

# Hadoop Ecosystem

thebigdatablog.weebly.com



# Resources for Hadoop

- *Hadoop: The Definitive Guide, 3rd Edition*, by Tom White
  - O'Reilly Media
  - Print ISBN: 978-1-4493-1152-0 | ISBN 10: 1-4493-1152-0
  - Ebook ISBN: 978-1-4493-1151-3 | ISBN 10: 1-4493-1151-2
- *MapReduce Design Patterns*, by Donald Miner and Adam Shook
  - O'Reilly Media
  - Print ISBN: 978-1-4493-2717-0 | ISBN 10: 1-4493-2717-6
  - Ebook ISBN: 978-1-4493-4197-8 | ISBN 10: 1-4493-4197-7

- http://hadoop.apache.org/
- Several vendors provide Hadoop distributions
- Amazon Web Services – ElasticMapReduce

# Resources for Spark

- *Learning Spark*, (early release) by Holden Karau, Andy Konwinsky, Patrick Wendell, Matei Zaharia
  - O'Reilly Media
  - Print ISBN: 978-1-4493-5862-4 | ISBN 10: 1-4493-5862-4
  - Ebook ISBN: 978-1-4493-5860-0 | ISBN 10: 1-4493-5860-8

- http://spark.apache.org/
  - Can download a version that runs on your local machine

- Cloud services
  - Spark on AWS
  - DataBricks offers a cloud service
  - Others will join the party