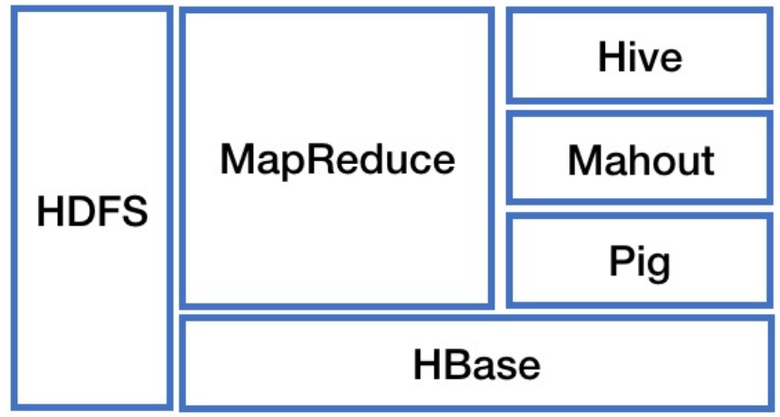
II CONGRESO
COLOMBIANO
DE ESTADISTICA

BIG DATA Y COMPUTACIÓN DE ALTO DESEMPEÑO: UNA INTRODUCCIÓN PARA EL ESTADÍSTICO

JUAN DAVID VELÁSQUEZ HENAO, MSc, PhD
Profesor Titular
Departamento de Ciencias de la Computación y la Decisión
Facultad de Minas
Universidad Nacional de Colombia, Sede Medellín

jdvelasq@unal.edu.co
@jdvelasquezh
https://github.com/jdvelasq
https://goo.gl/prkjAq
https://goo.gl/vXH8jy

Evolución del Big Data



Pig 0.1.0
Lenguaje de alto nivel para análisis y manipulación de datos

Chukwa 0.3.0
Sistema para recolección de datos para el monitoreo de grandes sistemas distribuidos.

Avro 0.9.0
Sistema de serialización de datos

Cassandra 0.3.0
Sistema de bases de datos NoSQL

Giraph 0.1
Procesamiento de grafos

Hive 0.10.0
Lenguaje similar a SQL para manejo de bodegas de datos en hadoop

Ambari 0.9.0
Herramienta para la gestión y monitoreo de clusters Hadoop

Zeppelin 0.5.0
Notebook para procesamiento interactivo

2007 2008 2009 2010 2012 2013 2014 2015

Hadoop 0.10.1
Procesamiento distribuido de grandes conjuntos de datos en clusters de computadores.

Mahout 0.1
Implementación de algoritmos de machine learning en Hadoop

Hbase 0.20.5
Sistema distribuido y escalable no relacional para almacenamiento de grandes volúmenes de datos (BigTable)

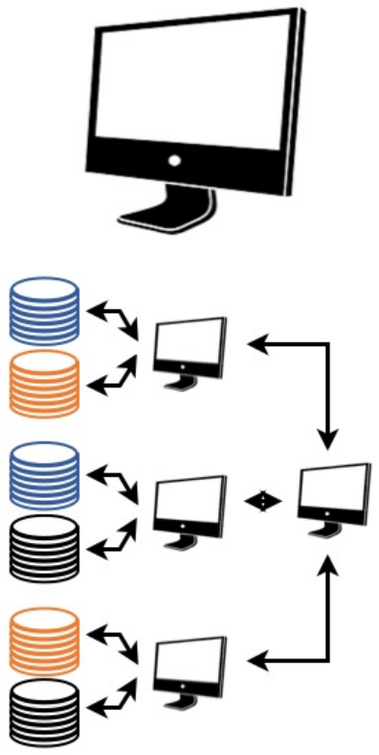
Tez 0.5.0
Procesamiento de grafos acíclicos complejos

Spark 0.8.0
Herramienta general para el procesamiento de datos a gran escala

Servicios Web (2002)

Computación local

Servidores + red + clientes



Cloud computing / utility computing

Servidores y almacenamiento en la nube + internet + clientes locales

Software as a Service (SaaS)

Software almacenado en máquinas suministradas por un tercero.

Aplicaciones accesadas vía un cliente o la Web.

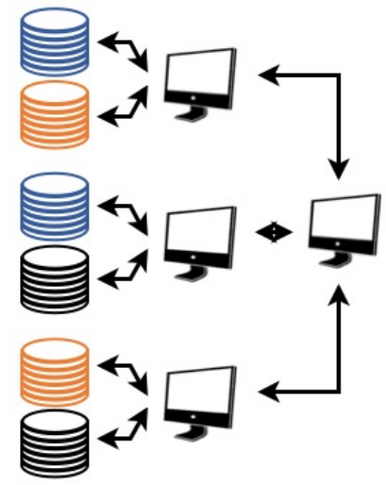
Orientado a aplicaciones de usuario final.

Platform as a Service (PaaS)

Orientado a desarrolladores.
Ambiente de desarrollo gestionado por un tercero.

Infrastructure as a Service (IaaS)

Bloques básicos para construcción de ambientes manejados por un tercero
Capacidad de procesamiento, almacenamiento, conectividad, seguridad, etc.



Apache Hadoop / MapReduce

Hadoop / MapReduce

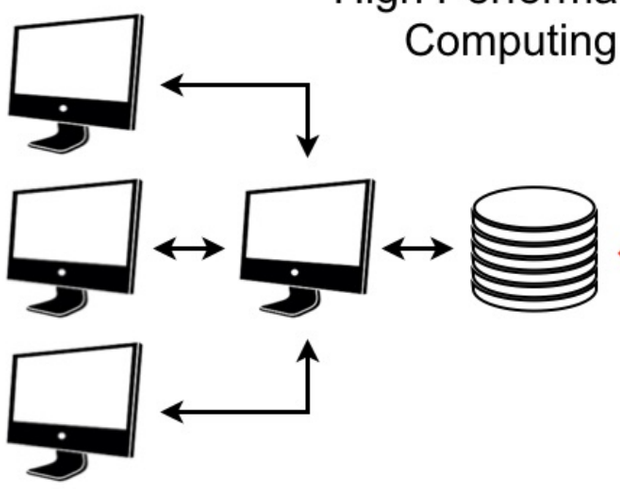
Aproximación Tradicional
(RDBMS y data warehouse)

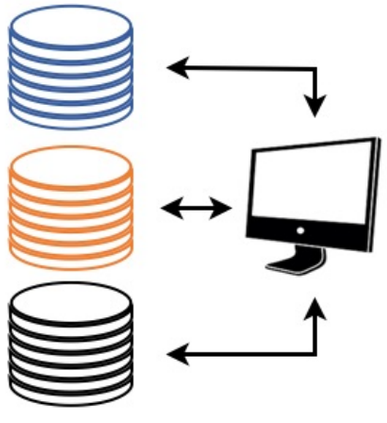
- Alto volumen de datos
- Lectura lenta
- Escritura + lenta
- Búsqueda ++ lenta

RDBMS
- Orientado a filas
- Estructura fija
- SQL
- Consistencia
- Integridad referencial
- Abstracción

Procesamiento distribuido

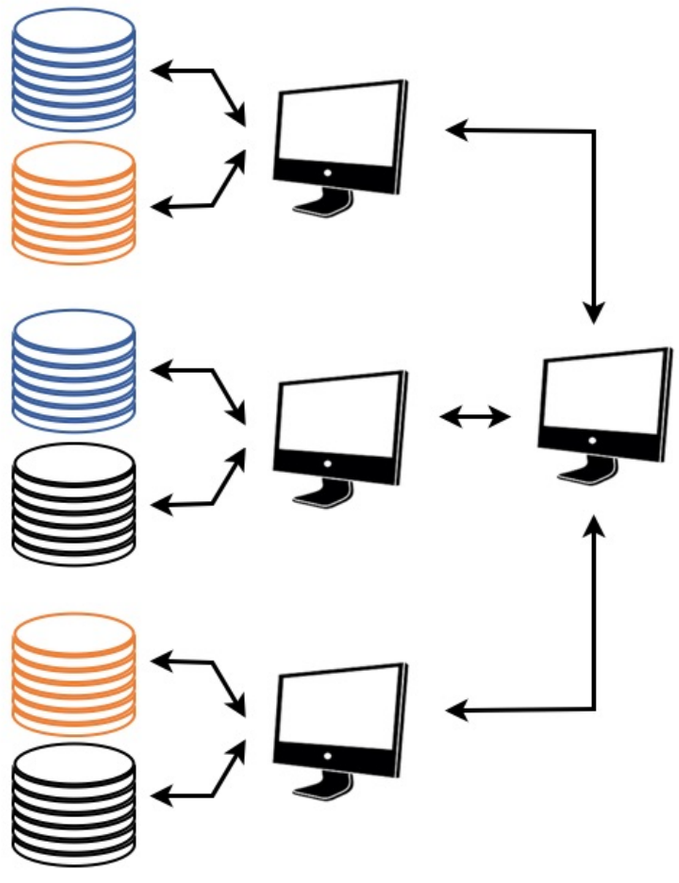
High Performance Computing





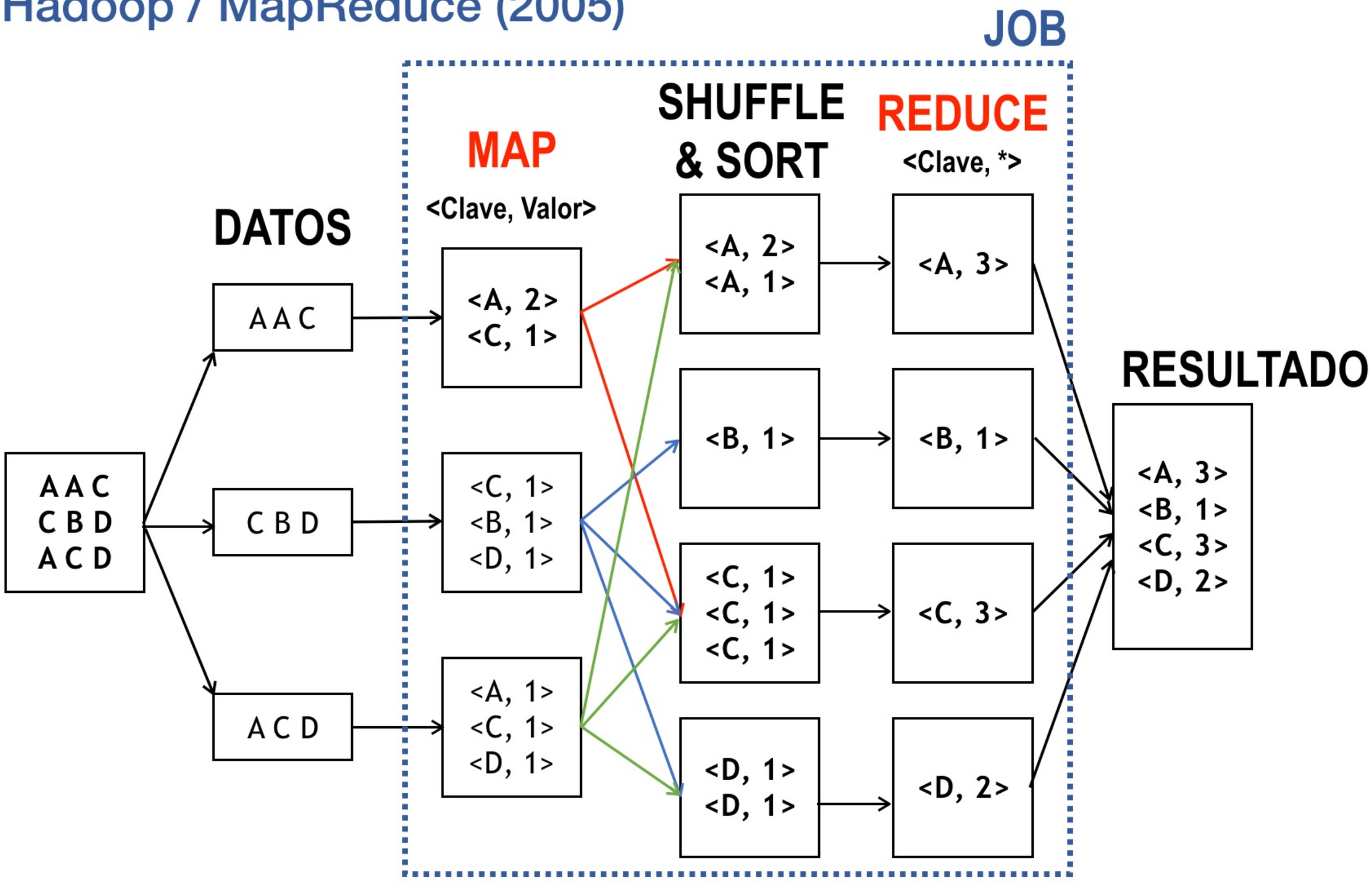
Almacenamiento distribuido

Hadoop Distributed Filesystem (HDFS)



Nodos de Datos + Nodos de Computo

Hadoop / MapReduce (2005)



Generalidades

Modos de ejecución

- Modo local (un solo hilo de ejecución en la máquina local, para aprendizaje y desarrollo). El sistema local de archivos funciona como el HDFS.
- Modo seudo-distribuido (múltiples hilos en la máquina local, para desarrollo). El HDFS es alcanzable como un servicio.
- Cluster (ambiente productivo)

Desarrollo de aplicaciones

- Programación directa en Java (Hadoop)
- Programación usando un lenguaje interpretado (Hadoop-streaming

Sistemas Operativos

- Microsoft Windows (difícil).
- Mac OS: instalación directa.
- Ubuntu Linux: instalación directa.

Conteo de palabras usando Hadoop-Streaming

Datos

In [93]:
```
%%writefile input/text0.txt
A B C A
A D D A
A K M C
```

```
Overwriting input/text0.txt
```

In [94]:
```
%%writefile input/text1.txt
B A C Y B
U O Y Y A
A B I T
```

```
Overwriting input/text1.txt
```

In [95]:
```
%%writefile input/text2.txt
A C D A
A K B
A N H I D A
```

```
Overwriting input/text2.txt
```

Conteo de palabras usando Hadoop-Streaming

mapper.R

In [96]:
```
%%writefile mapper.R
#! /usr/bin/env Rscript

input <- file('stdin', 'r')
while(TRUE) {
    row <- readLines(input, n=1)
    if( length(row) == 0 ){
        break
    }
    words <- strsplit(row, " ")[[1]]
    for(word in words){
        if(word != '')
            write(cat(word,'\t1',sep=''), "")
    }
}
```

```
Overwriting mapper.R
```

In [119]:
```
## El programa anterior se hace ejecutable
!chmod +x mapper.R

## Verificación
!cat ./input/text*.txt | ./mapper.R | head
```

```
A	1
B	1
C	1
A	1
A	1
D	1
D	1
A	1
A	1
K	1
```

Conteo de palabras usando Hadoop-Streaming

reducer.R

In [98]:
```
%%writefile reducer.R
#!/usr/bin/env Rscript

curkey <- NULL
total <- 0
input <- file('stdin', 'r')
while(TRUE) {
    row <- readLines(input, n=1)
    if( length(row) == 0 ){
        break
    }
    x <- strsplit(row, "\t")[[1]]
    key <- x[1]
    value <- strtoi(x[2])
    if(!is.null(curkey) && key == curkey){
        total <- total + value
    }
    else{
        if( !is.null(curkey) ) {
            write(cat(curkey,'\t', total), "")
        }
        curkey <- key
        total <- value
    }
}
write(cat(curkey,'\t', total), "")
```

```
Overwriting reducer.R
```

In [99]:
```
!chmod +x reducer.R
!cat ./input/text*.txt | ./mapper.R | sort | ./reducer.R
```

```
Warning message:
In readLines(input, n = 1) : incomplete final line found on 'stdin'
A	13
B	5
C	4
D	4
H	1
I	2
K	2
M	1
N	1
O	1
T	1
U	1
Y	3
```

Conteo de palabras usando Hadoop-Streaming

In [115]:
```
!hadoop fs -mkdir /user
!hadoop fs -mkdir /user/jdvelasq
!hadoop fs -mkdir /user/jdvelasq/input
!hadoop fs -copyFromLocal  input/* /user/jdvelasq/input
!hadoop fs -ls /user/jdvelasq/input/*
```

```
-rw-r--r--   1 jdvelasq supergroup         25 2018-11-08 23:36 /user/jdvelasq/input/text0.txt
-rw-r--r--   1 jdvelasq supergroup         29 2018-11-08 23:36 /user/jdvelasq/input/text1.txt
-rw-r--r--   1 jdvelasq supergroup         28 2018-11-08 23:36 /user/jdvelasq/input/text2.txt
```

In [116]:
```
!hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-*.jar \
-input input -output output  -mapper mapper.R -reducer reducer.R
```

In [117]:
```
!hadoop fs -ls /user/jdvelasq/output
```

```
Found 2 items
-rw-r--r--   1 jdvelasq supergroup          0 2018-11-08 23:37 /user/jdvelasq/output/_SUCCESS
-rw-r--r--   1 jdvelasq supergroup         79 2018-11-08 23:37 /user/jdvelasq/output/part-000
00
```

In [118]:
```
!hadoop fs -cat /user/jdvelasq/output/part-00000
```

```
A	13
B	5
C	4
```

Conteo de palabras usando Hadoop-Streaming

Hadoop Overview Datanodes Datanode Volume Failures Snapshot Startup Progress Utilities

Browse the file system
Logs

Browse Directory

/ Go!

Show 25 entries Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	jdvelasq	supergroup	0 B	Nov 09 07:37	0	0 B	Volumes
drwxr-xr-x	jdvelasq	supergroup	0 B	Nov 08 23:31	0	0 B	input
drwxr-xr-x	jdvelasq	supergroup	0 B	Nov 09 07:36	0	0 B	tmp
drwxr-xr-x	jdvelasq						

Showing 1 to 4 of 4 entries

Hadoop, 2018.

Hadoop Overview Datanodes Datanode Volume Failures Snapshot Startup Progress Utilities

Browse Directory

/user/jdvelasq/output Go!

Show 25 entries Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	jdvelasq	supergroup	0 B	Nov 08 23:37	1	128 MB	_SUCCESS
-rw-r--r--	jdvelasq	supergroup	79 B	Nov 08 23:37	1	128 MB	part-00000

Showing 1 to 2 of 2 entries Previous 1 Next

Hadoop, 2018.

Problemas típicos en los que puede usarse Map/Reduce

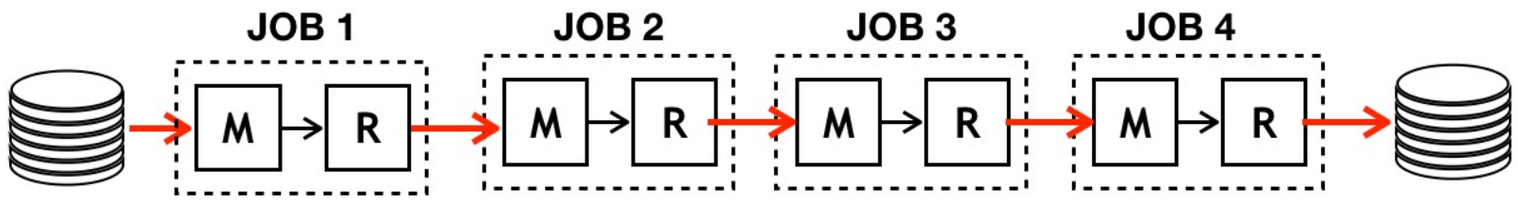
- Valores máximo o mínimo para una clave.
- Ordenamiento del archivo.
- Sumas y promedios por claves.
- Obtener los N registros más ...
- Asociar por claves:

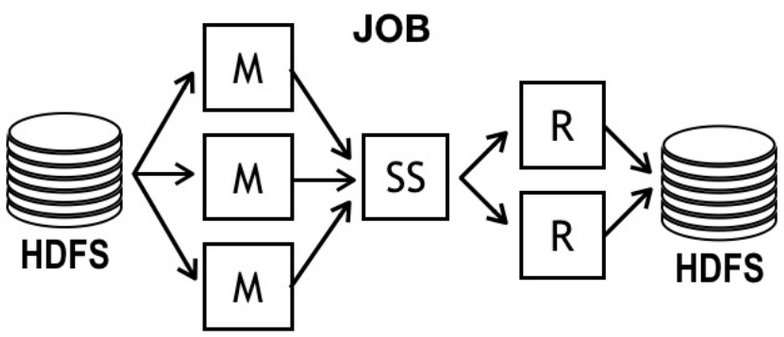
```
0   A, B                    A   0, 1
1   A, B        ======>     B   0, 1, 2
2   B, C                    C   2
```

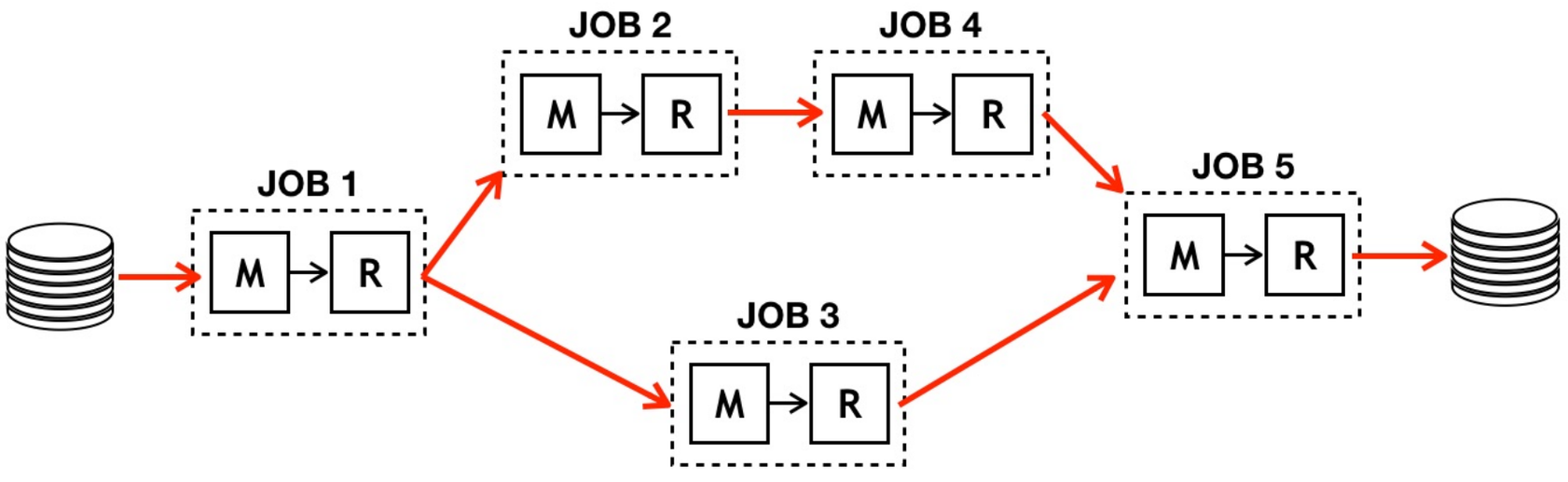
La mayor parte de problemas no pueden solucionarse usando un proceso simple Map/Reduce

Jobs

Hadoop / MapReduce (2005)

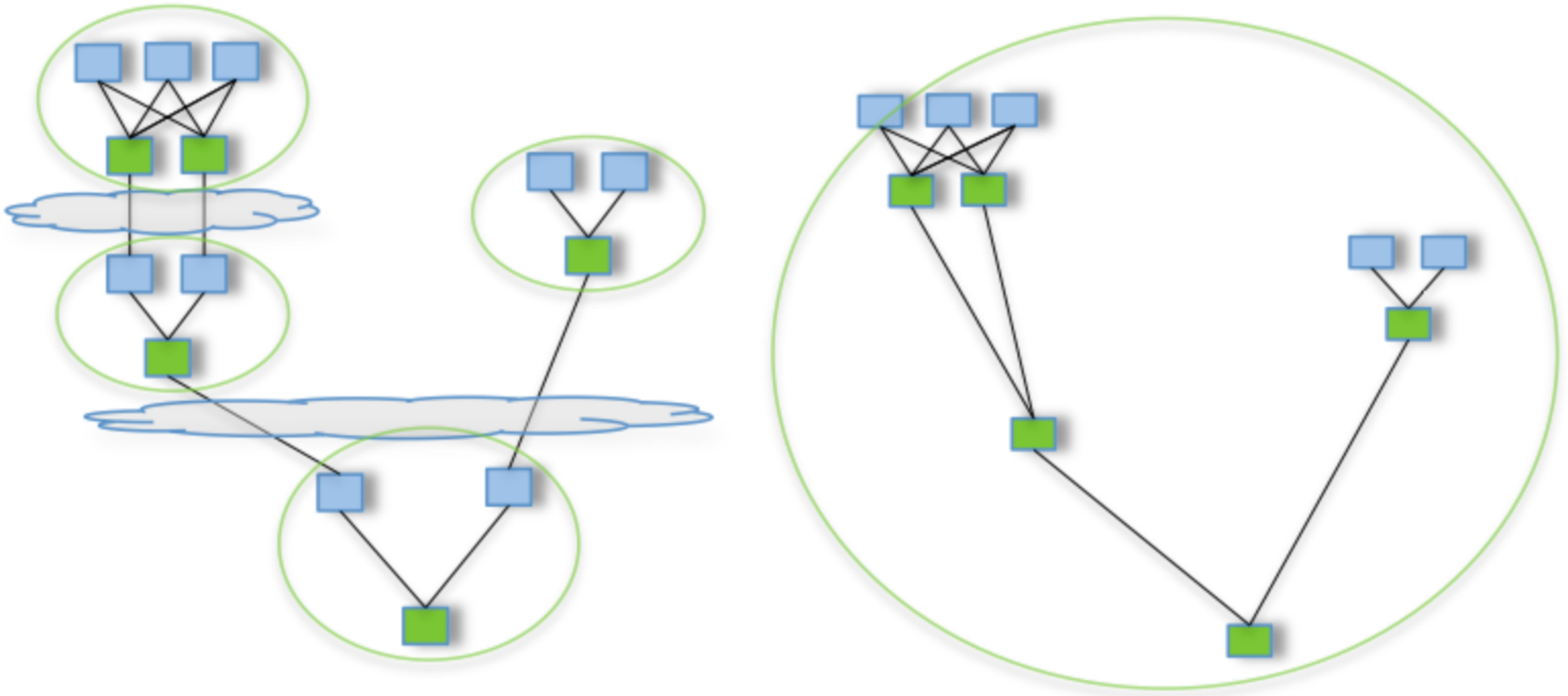






La coordinación entre jobs debe realizarse explícitamente en la consola de comandos

Apache Tez



Pig/Hive - MR

Pig/Hive - Tez

Apache Pig

Conteo de palabras usando Pig

In [1]:
```
%load_ext bigdata
%pig_init
```

In [5]:
```
%%pig
lines = LOAD 'input/text*.txt' AS (line:CHARARRAY);

-- genera una tabla llamada words con una palabra por registro
words = FOREACH lines GENERATE FLATTEN(TOKENIZE(line)) AS word;

-- agrupa los registros que tienen la misma palabra
grouped = GROUP words BY word;

-- genera una variable que cuenta las ocurrencias por cada grupo
wordcount = FOREACH grouped GENERATE group, COUNT(words);

-- selecciona las primeras 15 palabras
s = LIMIT wordcount 15;

-- imprime en pantalla las primeras 15 palabras
STORE s INTO 'output-pig';
```

In [6]:
```
!hadoop fs -ls output-pig/
```

```
Found 2 items
-rw-r--r--   1 jdvelasq supergroup          0 2018-11-09 07:22 output-pig/_SUCCESS
-rw-r--r--   1 jdvelasq supergroup         53 2018-11-09 07:22 output-pig/part-r-00000
```

In [7]:
```
!hadoop fs -cat /user/jdvelasq/output-pig/part-r-00000
```

```
A	13
B	5
C	4
```

Hadoop Overview Datanodes Datanode Volume Failures Snapshot Startup Progress Utilities

Conteo de palabras usando Pig

Browse Directory

/user/jdvelasq Go!

Show 25 entries Search:

	Replication	Block Size	Name
	0	0 B	input
	0	0 B	output
	0	0 B	output-pig

Showing 1 to

Previous 1 Next

Hadoop, 201

File information - part-r-00000

Download Head the file (first 32K) Tail the file (last 32K)

Block information -- Block 0

Block ID: 1073741847

Block Pool ID: BP-1224940242-192.168.1.58-1541734190285

Generation Stamp: 1023

Size: 53

Availability:

- 192.168.1.58

File contents

```
A   13
B   5
C   4
D   4
H   1
I   2
K   2
M   1
```

http://nbviewer.jupyter.org/github/jdvelasq/AGD-03-Pig/blob/master/02-basics.ipynb

http://nbviewer.jupyter.org/github/jdvelasq/AGD-03-Pig/blob/master/04-tipos-de-datos.ipynb

Apache Hive

Conteo de palabras

In [8]:
```
%hive_init
```

```
Hive initialized!
```

In [9]:
```
%%hive
DROP TABLE IF EXISTS docs;
DROP TABLE IF EXISTS word_counts;

CREATE TABLE docs (line STRING);

LOAD DATA LOCAL INPATH
    'input/text*.txt'
OVERWRITE INTO TABLE docs;

CREATE TABLE word_counts
AS
    SELECT word, count(1) AS count
    FROM
        (SELECT explode(split(line, '\\s')) AS word FROM docs) w
GROUP BY
    word
ORDER BY
    word;

SELECT * FROM word_counts LIMIT 10;
```

```
        7
A       13
B       5
C       4
D       4
H       1
I       2
K       2
M       1
N       1
```

Caraterísticas de Apache Hive

- El lenguaje tiene muchas similitudes con SQL y resulta fácil

- Los archivos se guardan directamente en disco duro como texto.

- A continuación se presentan los principales puntos de diferencia con otros sistemas de bases de datos.

- Obtener los N registros más

Hive

In [11]:
```
%%writefile tbl0.csv
1,D,4,2016-06-25,a:e:c:d,bb#10:dd#20:cc#40
2,C,4,2015-05-14,a:c,dd#40:bb#20:cc#10
3,D,6,2014-12-26,b:d,aa#10:bb#40
4,D,5,2016-06-25,a:c:e:b:d,bb#40:dd#20:aa#10:cc#30
5,C,7,2016-05-23,d:e:c,cc#10:aa#20
6,A,2,2018-06-14,a:d,aa#20
7,B,3,2014-12-22,a:e:d,cc#20
8,C,6,2015-08-20,d:a:c:e,cc#10:aa#20
9,B,3,2017-12-08,b:a:c:e,cc#40:dd#10:aa#30:bb#20
10,B,7,2015-07-28,c:d:e:a:b,aa#10:dd#40:cc#30
```

```
Writing tbl0.csv
```

In [12]:
```
%%hive
DROP TABLE IF EXISTS tbl0;
CREATE TABLE tbl0 (
    c1 INT,
    c2 STRING,
    c3 INT,
    c4 DATE,
    c5 ARRAY<CHAR(1)>,
    c6 MAP<STRING, INT>
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
COLLECTION ITEMS TERMINATED BY ':'
MAP KEYS TERMINATED BY '#'
LINES TERMINATED BY '\n';

LOAD DATA LOCAL INPATH 'tbl0.csv' INTO TABLE tbl0;

SELECT * FROM tbl0;
```

```
1       D       4       2016-06-25      ["a","e","c","d"]       {"bb":10,"dd":20,"cc":40}
2       C       4       2015-05-14      ["a","c"]       {"dd":40,"bb":20,"cc":10}
3       D       6       2014-12-26      ["b","d"]       {"aa":10,"bb":40}
4       D       5       2016-06-25      ["a","c","e","b","d"]
{"bb":40,"dd":20,"aa":10,"cc":30}
5       C       7       2016-05-23      ["d","e","c"]   {"cc":10,"aa":20}
6       A       2       2018-06-14      ["a","d"]       {"aa":20}
7       B       3       2014-12-22      ["a","e","d"]   {"cc":20}
8       C       6       2015-08-20      ["d","a","c","e"]       {"cc":10,"aa":20}
9       B       3       2017-12-08      ["b","a","c","e"]
{"cc":40,"dd":10,"aa":30,"bb":20}
10      B       7       2015-07-28      ["c","d","e","a","b"]   {"aa":10,"dd":40,"cc":30}
```

Apache Spark

Conteo de palabras

In [6]:
```
!hadoop fs -cat /user/jdvelasq/output-spark/part-00000
```

```
('B', 5)
('C', 4)
('D', 4)
('T', 1)
('N', 1)
```

In [1]:
```
import findspark
findspark.init()

from pyspark import SparkConf, SparkContext
from operator import add

APP_NAME = "My First Spark Application"

def tokenize(text):
    return text.split()

def main(sc):
    text = sc.textFile('input/text*.txt')
    words = text.flatMap(tokenize)
    wc = words.map(lambda x: (x,1))
    counts = wc.reduceByKey(add)
    counts.saveAsTextFile("output-spark")

if __name__ == "__main__":
    conf = SparkConf().setAppName(APP_NAME)
    conf = conf.setMaster("local[*]")
    sc = SparkContext(conf=conf)
    main(sc)
```

In [2]:
```
!hadoop fs -ls output-spark
```

```
Found 4 items
-rw-r--r--   1 jdvelasq supergroup          0 2018-11-09 08:39 output-spark/_SUCCESS
-rw-r--r--   1 jdvelasq supergroup         45 2018-11-09 08:39 output-spark/part-00000
-rw-r--r--   1 jdvelasq supergroup         37 2018-11-09 08:39 output-spark/part-00001
-rw-r--r--   1 jdvelasq supergroup         36 2018-11-09 08:39 output-spark/part-00002
```

Aplicación arbitraria de funciones

In [7]:
```
## en este ejemplo se pasa una función arbitraria a `map`
from operator import add
rdd = sc.textFile('input/text*.txt')
rdd = rdd.map(len)
print(rdd.collect())
rdd = rdd.reduce(add)
rdd
```

```
[7, 7, 7, 1, 9, 9, 7, 1, 7, 6, 11, 1]
```

Out[7]: 73

Spark QL

In [7]:
```
from pyspark.sql import Row

## crea el contexto
sc = spark.sparkContext

## lee el archivo como lineas de texto
rdd = sc.textFile("people.csv")

## parte por las comas
rdd = rdd.map(lambda row: row.split(","))

## cambia los tipos de los datos
rdd = rdd.map(lambda p: Row(id=p[0],
                            firstname=p[1],
                            surname=p[2],
                            birthdate=p[3],
                            color=p[4],
                            quantity=int(p[5])))

## crea el DataFrame
df = spark.createDataFrame(rdd)
df.createOrReplaceTempView("df")
df.show()
```

```
+----------+------+---------+---+--------+--------+
| birthdate| color|firstname| id|quantity| surname|
+----------+------+---------+---+--------+--------+
|1971-07-08| green|   Vivian|  1|
|1974-05-23| green|    Karen|  2|
|1973-04-22|orange|     Cody|  3|
|1975-01-29| black|     Roth|  4|
|1974-07-03|  blue|      Zoe|  5|
```

In [ ]:
```
query = """
SELECT DISTINCT
    _C5
FROM
    csv.`datos.csv`
ORDER BY
    _c5
"""

spark.sql(query).write.save('temp', format="csv")
```

In [13]:
```
## Filtrado
df.filter(df['color'] == 'blue').show()
```

```
+---+---------+-------+-------------------+-----+--------+
| id|firstname|surname|          birthdate|color|quantity|
+---+---------+-------+-------------------+-----+--------+
|  5|      Zoe| Conway|1974-07-03 00:00:00| blue|       2|
|  7| Driscoll|  Klein|1970-10-05 00:00:00| blue|       5|
| 15|     Hope|  Silva|1970-07-01 00:00:00| blue|       5|
+---+---------+-------+-------------------+-----+--------+
```

Machine Learning Library (MLlib) Guide

MLlib is Spark's machine learning (ML) library. Its goal is to make practical machine learning sca
it provides tools such as:

- ML Algorithms: common learning algorithms such as classification, regression, clustering, a
- Featurization: feature extraction, transformation, dimensionality reduction, and selection
- Pipelines: tools for constructing, evaluating, and tuning ML Pipelines
- Persistence: saving and load algorithms, models, and Pipelines
- Utilities: linear algebra, statistics, data handling, etc.

Announcement: DataFrame-based API is prir

The MLlib RDD-based API is now in maintenance mode.

As of Spark 2.0, the RDD-based APIs in the `spark.mllib` package have entered maintenance m
Learning API for Spark is now the DataFrame-based API in the `spark.ml` package.

What are the implications?

- MLlib will still support the RDD-based API in `spark.mllib` with bug fixes.
- MLlib will not add new features to the RDD-based API.
- In the Spark 2.x releases, MLlib will add features to the DataFrames-based API to reach fea API.
- After reaching feature parity (roughly estimated for Spark 2.3), the RDD-based API will be de
- The RDD-based API is expected to be removed in Spark 3.0.

Why is MLlib switching to the DataFrame-based API?

- DataFrames provide a more user-friendly API than RDDs. The many benefits of DataFrames SQL/DataFrame queries, Tungsten and Catalyst optimizations, and uniform APIs across lan
- The DataFrame-based API for MLlib provides a uniform API across ML algorithms and acro
- DataFrames facilitate practical ML Pipelines, particularly feature transformations. See the Pi

What is "Spark ML"?

- "Spark ML" is not an official name but occasionally used to refer to the MLlib DataFrame-b

Scala Java Python R

In SparkR we provide Spark SQL data source API for loading image data as DataFrame.

```
> df = read.df("data/mllib/images/origin/kittens", "image")
> head(select(df, df$image.origin, df$image.width, df$image.height))

1              file:///spark/data/mllib/images/origin/kittens/54893.jpg
2            file:///spark/data/mllib/images/origin/kittens/DP802813.jpg
3 file:///spark/data/mllib/images/origin/kittens/29.5.a_b_EGDP022204.jpg
4            file:///spark/data/mllib/images/origin/kittens/DP153539.jpg
  width height
1   300    311
2   199    313
3   300    200
4   300    296
```

Scala Java Python R

More details on parameters can be found in the R API documentation.

```
# Load training data
df <- read.df("data/mllib/sample_libsvm_data.txt", source = "libsvm")
training <- df
test <- df

# Fit an binomial logistic regression model with spark.logit
model <- spark.logit(training, label ~ features, maxIter = 10, regParam = 0.3, elasticNetParam = 0.8)

# Model summary
summary(model)

# Prediction
predictions <- predict(model, test)
head(predictions)
```

sparklyr: R interface for Apache Spark

- Connect to Spark from R. The sparklyr package provides a complete dplyr backend.
- Filter and aggregate Spark datasets then bring them into R for analysis and visualization.
- Use Spark's distributed machine learning library from R.
- Create extensions that call the full Spark API and provide interfaces to Spark packages.

Installation

You can install the **sparklyr** package from CRAN as follows:

```
install.packages("sparklyr")
```

You should also install a local version of Spark for development purposes:

```
library(sparklyr)
spark_install(version = "2.1.0")
```

Apache Zeppelin

Zeppelin Notebook Job Search anonymous

demo-python

Head default

```
print('hola mund0')
```

FINISHED

```
hola mund0
```

READY

Interpreters

Overview

Spark

JDBC

Python

Alluxio

Beam

BigQuery

Cassandra

Elasticsearch

Flink

Geode

Groovy

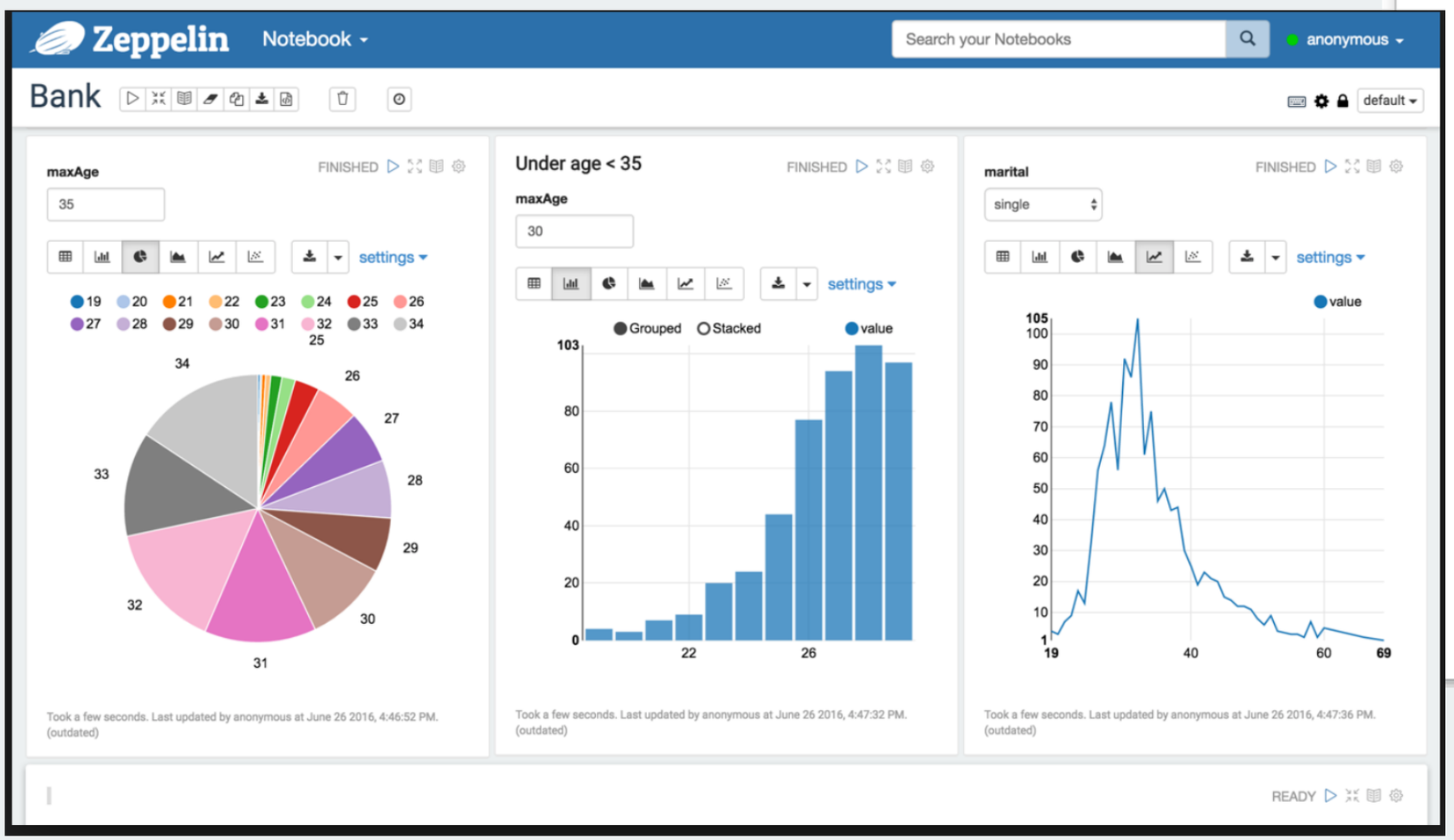
HBase

HDFS

Hive

Ignite

Kylin



Apache DRILL

```
SELECT * FROM dfs.`<path-to-installation>/apache-drill-<version>/sample-data/region.parquet`;
```

```
+--------------+--------------+-----------------------+
| R_REGIONKEY  |    R_NAME    |       R_COMMENT       |
+--------------+--------------+-----------------------+
| 0            | AFRICA       | lar deposits. blithe  |
| 1            | AMERICA      | hs use ironic, even   |
| 2            | ASIA         | ges. thinly even pin  |
| 3            | EUROPE       | ly final courts cajo  |
| 4            | MIDDLE EAST  | uickly special accou  |
+--------------+--------------+-----------------------+
5 rows selected (0.409 seconds)
```

II CONGRESO
COLOMBIANO
DE ESTADISTICA

BIG DATA Y COMPUTACIÓN DE ALTO DESEMPEÑO: UNA INTRODUCCIÓN PARA EL ESTADÍSTICO

JUAN DAVID VELÁSQUEZ HENAO, MSc, PhD
Profesor Titular
Departamento de Ciencias de la Computación y la Decisión
Facultad de Minas
Universidad Nacional de Colombia, Sede Medellín

jdvelasq@unal.edu.co
@jdvelasquezh
https://github.com/jdvelasq
https://goo.gl/prkjAq
https://goo.gl/vXH8jy