uc3m | Universidad Carlos III de Madrid

OPENCOURSEWARE
APRENDIZAJE AUTOMÁTICO PARA EL ANÁLISIS DE DATOS
GRADO EN ESTADÍSTICA Y EMPRESA
Ricardo Aler

APRENDIZAJE EN BIG DATA
- MAPREDUCE -

QUE ES BIG DATA

- Tres características principales:
  - Volumen (terabytes, petabytes)
    - 1 petabyte = 1,000 TB, = 1,000,000 Gb
  - Velocidad (tiempo real, streaming, …)
    - Ej: detección de fraude, áudio, …
  - Variedad: mezcla de datos: estructurados, no-estructurados, texto, sensores, audio, video, click streams, ficheros de log, …

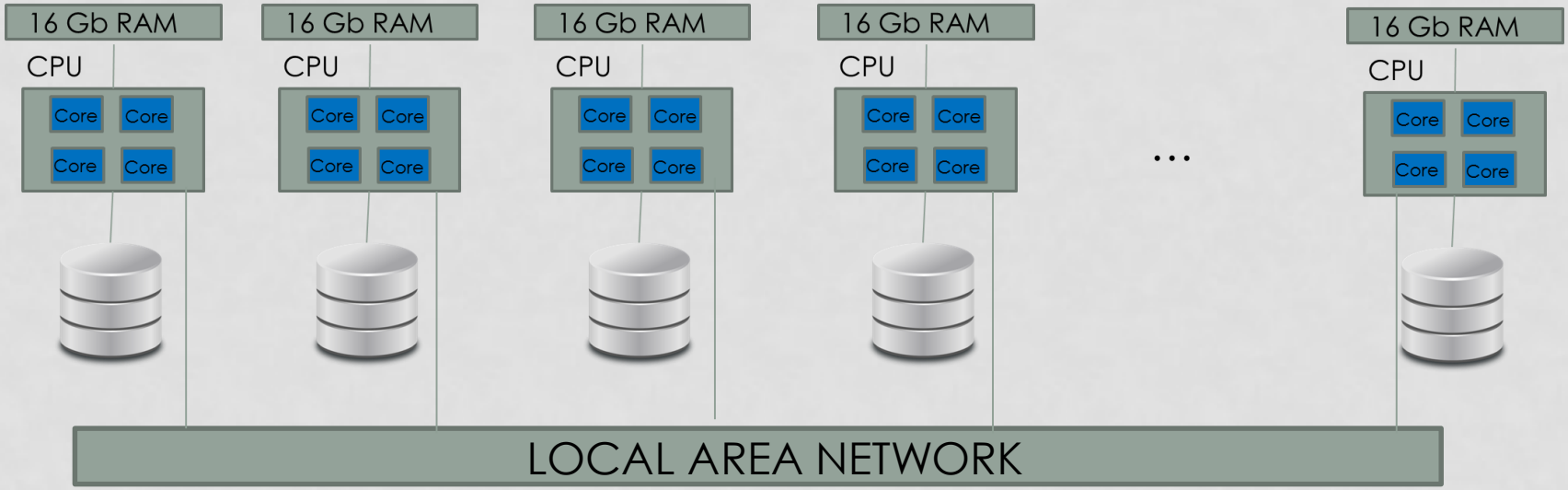
MOTIVACIÓN

- "*comodity* hardware" o "granjas de ordenadores": cientos o miles de PCs organizados en racks y con discos duros locales.

Source: https://sankartech.files.wordpress.com/2013/08/ibm_supercomputer.jpg

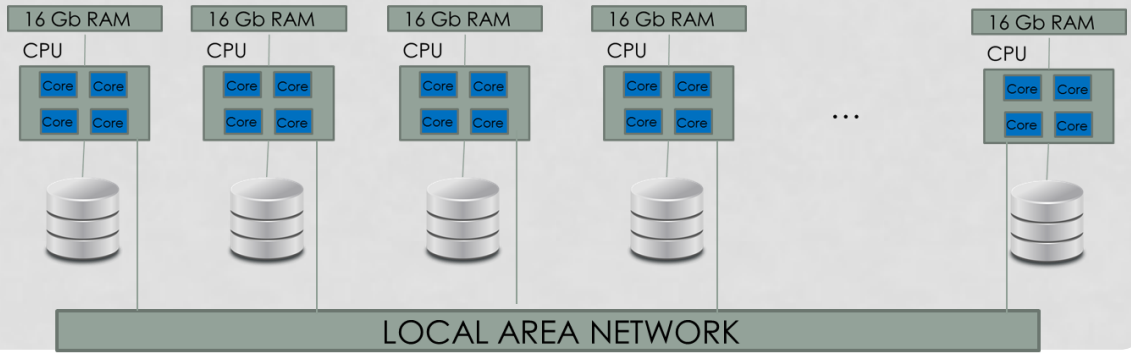
DATA PARALLELISM

- La misma tarea ejecutándose a la vez en datos distintos.



BIG DATA

- Hay que usarlo cuando:
  - El conjunto de datos no cabe en un solo ordenador
  - O se tarda demasiado en procesarlo en un solo ordenador
- Se usa "commodity hardware" (clusters de pc's normales)
- Modelos de programación: Mapreduce (Yahoo), Apache Spark (Databricks), Dryad (Microsoft), Vowpal Wabbit (Microsoft)



MAP REDUCE

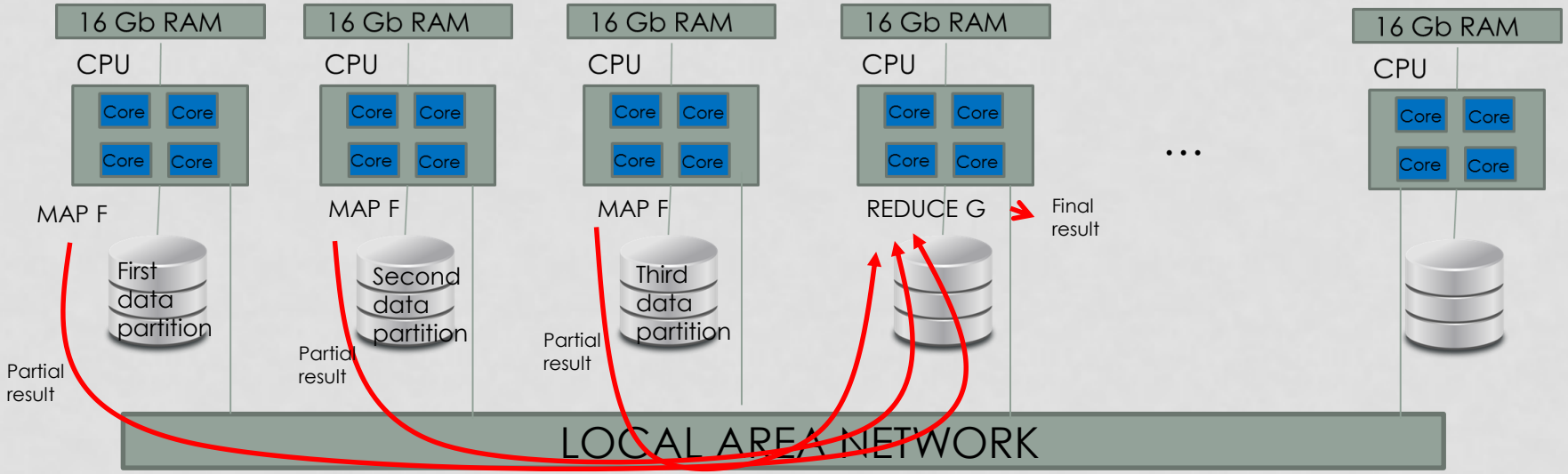
- Modelo de programación para data parallelism / computación distribuida
- Basado en dos operaciones:
  - **Map**: se ejecuta en paralelo en ordenadores distintos
  - **Reduce**: combina los resultados producidos por los maps

- El objetivo de este modelo es que las tareas de computación más pesadas **(map**), ocurran localmente donde están los datos
  - Es decir que se use la red de área local lo menos posible
  - Los resultados producidos por **Map** son más pequeños y se pueden combinar (**reducir**) en otros ordenadores.

- Origen: Google 2004 (indices de páginas, etc. Varios petabytes diariamente)
- Se usa en Facebook, LinkedIn, Tuenti, ebay, Yahoo, …
- Amazon AWS, Microsoft Azure, Google, … proporcionan Map-Reduce (pagando)

MAP REDUCE DATA PARALLELISM

- **Map** realizan la parte más pesada de la computación, allí donde los datos residen
- **Map** genera resultados parciales (de pequeño tamaño) que viajan por la red y son combinados en el **reducer** generando un resultado final
- **Nota:** la salida de cada **map** tiene el formato (key, value) o (clave, valor). En la red puede haber varios **reducers**, y cada uno procesa los resultados relativos a la misma clave (los resultados se agrupan por clave)

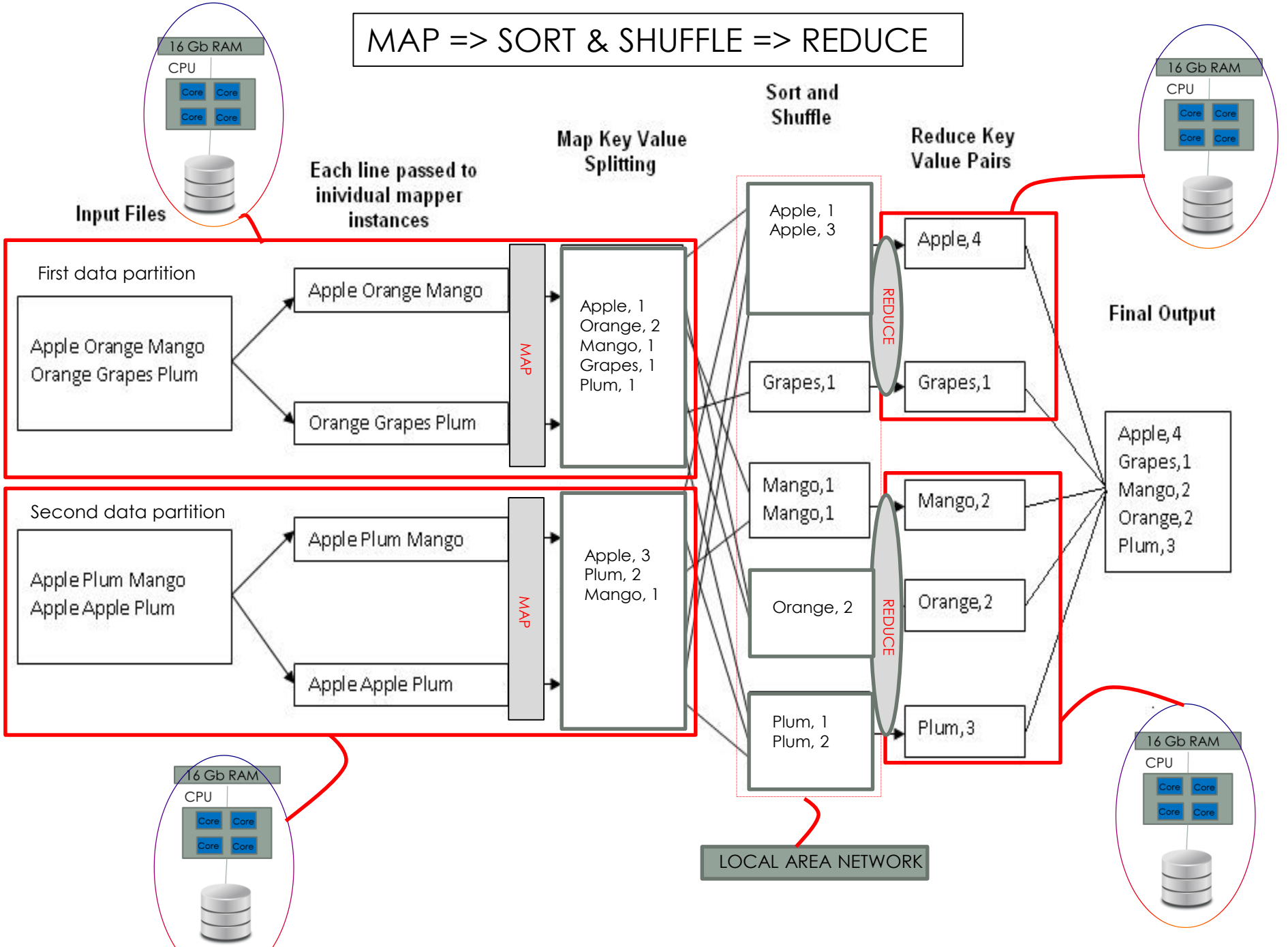


CONTANDO PALABRAS EN MAPREDUCE

- Supongamos que tenemos un gran conjunto de datos en forma de texto (ej: una biblioteca de libros)
- Nuestro objetivo es contar cuántas veces aparece cada palabra en el conjunto de datos:

1. El conjunto de datos se divide en diferentes particiones (tantas particiones como discos duros)
2. La función **map** cuenta las palabras en un texto
    1. Nota: cada CPU / equipo puede ser capaz de ejecutar varias map en paralelo (multi-core)
3. Sort & shuffle: los resultados parciales de los **maps** están agrupados por clave y se envían a los **reduce** en otros ordenadores según dicha clave. Esto lo hace mapreduce automáticamente
4. La función **reduce** suma los conteos parciales de las palabras

MAP => SORT & SHUFFLE => REDUCE



Source: http://1.bp.blogspot.com/-UvgLSDv7Rb4/Tbpn3veAOTI/AAAAAAAAAVk/kdaMzLa50BE/s1600/WordCountFlow.JPG

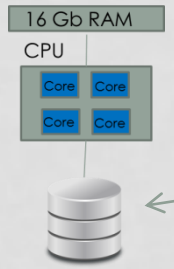
KNN IN MAPREDUCE (K=1)

Anchalia, P. P., & Roy, K. The k-Nearest Neighbor Algorithm Using MapReduce Paradigm.

Dataset distribuido en dos ordenadores (dos partiticiones)

Computer 1

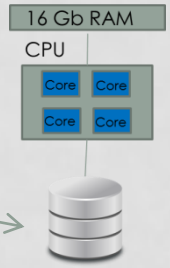
X	Y	Class
0.5	3	Y
1	2	N
-2	0,5	N



p

La instancia a ser clasificada se envía a los dos ordenadores

p = (p_x=1.3, p_y=2.1)



p

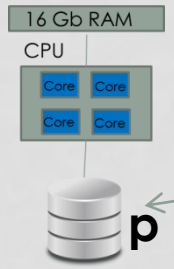
Computer 2

X	Y	Class
0.9	1	N
2	2	Y
-2	1	N

Dataset distribuido en dos ordenadores (dos partiticiones)

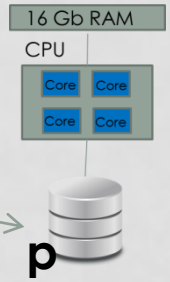
Computer 1

X	Y	Class
0.5	3	Y
1	2	N
-2	0,5	N



La instancia a ser clasificada se envía a los dos ordenadores

p = (p_x=1.3, p_y=2.1)



Computer 2

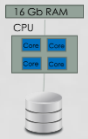
X	Y	Class
0.9	1	N
2	2	Y
-2	1	N

- La operación más costosa es el computo de las distancias **d($\mathbf{v_i}$, p)**, donde $\mathbf{v_i}$ es cada una de las instancias en el dataset.
- El cálculo de distancias es lo que **map** hace

```
Map(key = class, value = v=(x,y)):
  return(key = class, value (distance((x,y), (p_x, p_y)))
```

Computer 1

X	Y	Class
0.5	3	Y
1	2	N
-2	0,5	N



p

$p = (p_x=1.3, p_y=2.1)$
¿Qué clase?

```
Map(key = class, value = v=(x,y)):
  return(key = class, value = (distance((x,y), (px, py)))
```

Computer 2

X	Y	Class
0.9	1	N
2	2	Y
-2	1	N

p

map

(Y , 1.2)
(N , 0.32)
(N , 3.67)

combinación

(Y , 1.2)
(N , 0.32)

Resultados parciales que se envían por la red

map

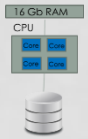
(Y , 1.17)
(N , 0.71)
(N , 3.48)

combinación

(Y , 1.17)
(N , 0.71)

Computer 1

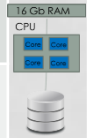
X	Y	Class
0.5	3	Y
1	2	N
-2	0,5	N



$p = (p_x=1.3, p_y=2.1)$
¿Qué clase?

Computer 2

X	Y	Class
0.9	1	N
2	2	Y
-2	1	N



map

combiner

(Y , 1.2)
(N , 0.32)

map

combiner

(Y , 1.17)
(N , 0.71)

Sort and shuffle (esta agrupación ocurre en la red)

(Y , 1.2)
(Y , 1.17)

(N , 0.71)
(N , 0.32)

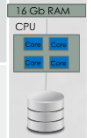
Computer 1

X	Y	Class
0.5	3	Y
1	2	N
-2	0,5	N

$p = (p_x=1.3, p_y=2.1)$
¿Qué clase?

Computer 2

X	Y	Class
0.9	1	N
2	2	Y
-2	1	N



map

combiner

(Y , 1.2)
(N , 0.32)

map

combiner

(Y , 1.17)
(N , 0.71)

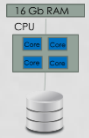
Sort and shuffle

(Y , 1.2)
(Y , 1.17)

(N , 0.71)
(N , 0.32)

Reduce(k=class, v=(d_1, d_2, …, d_n):
return (class, minimum(v))

Computer 3



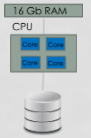
reduce

(Y, 1.17)

reduce

(N, 0.32)

Computer 4



min? = 0.32, N

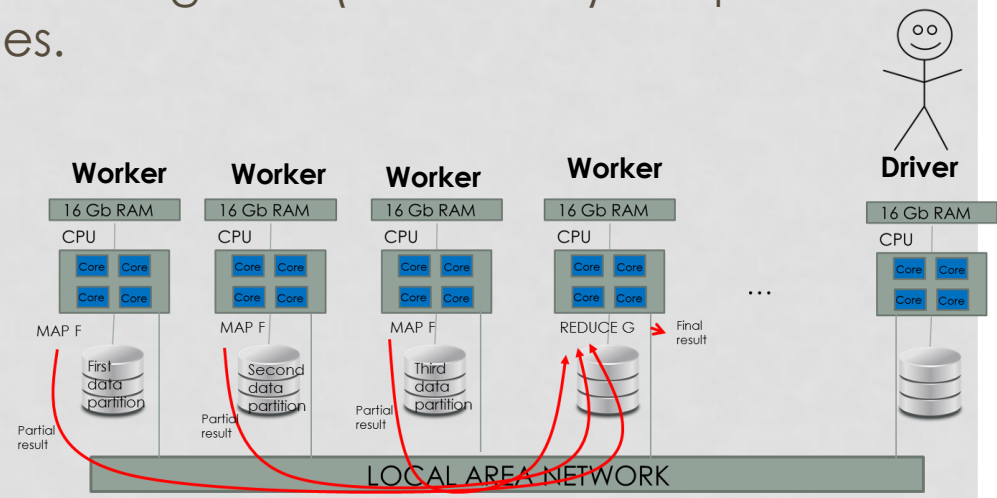
SPARK EN R

MAPREDUCE / HADOOP LIMITATIONS

- Para cada iteración map / reduce, MapReduce tiene que guardar los resultados en disco (replicados, para recuperación de fallos)
- El precio de la memoria RAM ha bajado. Es más rápido guardar los resultados en memoria (en la medida que se pueda)
- Spark usa algunas ideas de MapReduce, pero está orientado a usar más la memoria RAM

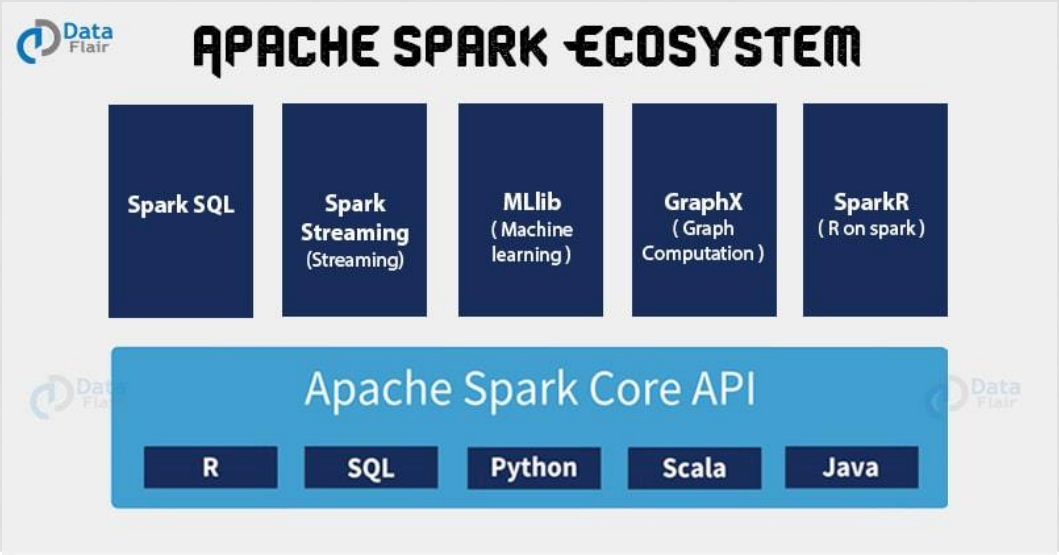
CONCEPTOS BÁSICOS

- **Nodo:** un ordenador / servidor
  - **Worker node:** ejecuta tareas Spark (en MapReduce, esas tareas serían Map o Reduce, en Spark es más variado). Dentro de cada worker node hay un programa que se ejecuta en él, llamado **Executor** (puede haber varios). Es el encargado de ejecutar las tareas.
  - **Master Node / cluster manager:** ordenador que coordina a los workers.
  - **Driver Node:** es el ordenador donde corre el programa del usuario (o **driver**). Accede a Spark a través de **SparkContext** o **SparkSession**.
- Usa dataframes distribuidos en dataframes locales en cada una de las particiones (típicamente una partición es un ordenador)
- Las transformaciones sobre el dataframe global (distribuido) se aplican a cada uno de los dataframes locales.



ECOSYSTEMA SPARK

- El lenguaje nativo de Spark es Scala, pero se puede programar en Python (Pyspark) y en R (SparkR)
- Scala es más rápido, pero SparkR permite usar R y sus librerías
- Es necesario decir que el interfaz de R con Spark es el más limitado, de momento



Source: https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/sites/2/2017/07/apache-spark-ecosystem-components.jpg

MANERAS DE TRABAJAR EN SPARK DESDE R

- Librería **sparkR**:
  - Similar a Scala o Pyspark, aunque menos desarrollado que estos
  - http://spark.apache.org/docs/latest/sparkr.html
- Librería **sparklyr**
  - Desarrollada por Rstudio
  - Más sencilla de manejar, puesto que usa dplyr, pero con dataframes distribuidos **spark**
  - https://spark.rstudio.com/
- **dplyr:** librería R para manejar data.frames, pero también spark data.frames.