

# UNIDAD 2: MODELO DE MAPEO Y REDUCCIÓN

## EXTENSIONES

---

Blanca Vázquez

Febrero 2020

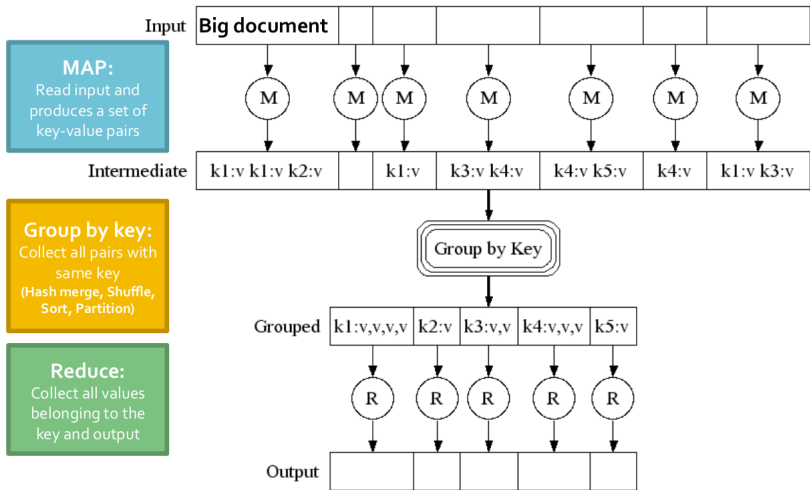
Mining of Massive Datasets

Jure Leskovec, Anand Rajaraman, Jeff Ullman

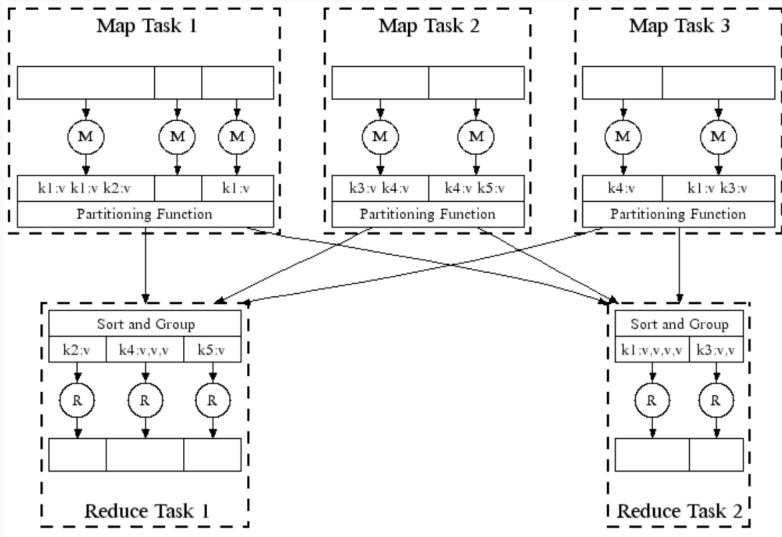
Stanford University

<http://www.mmds.org>

# MODELO MAPREDUCE



# MODELO MAPREDUCE EN PARALELO



El ambiente del modelo MapReduce se encarga de:

- **Particionar** los datos de entrada
- **Planificar** la ejecución de los programas en todo el clúster
- **Agrupar** por llaves (en la práctica esto es un cuello de botella)
- **Manejo de fallas**
- Administrar la **comunicación** entre máquinas

Se encarga de la coordinación de las actividades en el clúster de cómputo.

- **Estatus de las tareas / nodos:** esperando, en progreso, completada
- **Planifica** las tareas de mapeo y reducción
- Cuando una tarea de mapeo está completa, el maestro **envía la localización y tamaño de los archivos intermedios (R)** a los nodos de reducción.
- **Para detectar fallas**, periódicamente el maestro hace 'ping' a los nodos

- **Las entradas y salidas** son almacenadas en el sistema de archivo distribuido (DFS).
  - Las tareas de mapeo son planificadas en nodos cercanos a la localización física de los datos
- **Los resultados intermedios** son almacenados en sistemas locales de archivos (nodos de mapeo y reducción): evitar tráfico en la red y sobrecarga de datos
- Las salidas son frecuentemente la entrada hacia otra tarea de mapeo.

- **Cuando falla el nodo de mapeo**
  - Las tareas de mapeo completadas o en progreso se vuelven a programar para su ejecución
  - Los nodos de reducción son notificados indicando que la tarea de mapeo fue reprogramada.
- **Cuando falla el nodo de reducción**
  - Únicamente las tareas en progreso son reprogramadas hacia otro nodo.
- **Cuando falla el nodo de maestro**
  - Todas las tareas de mapeo y reducción son abortadas / canceladas.
  - El cliente es notificado

## ¿CUÁNTAS TAREAS DE MAPEO Y REDUCCIÓN DEBEMOS PLANIFICAR?

**Objetivo:** identificar  $M$  (número de tareas de mapeo) y  $R$  (número de tareas de reducción) que deben ser planificadas



# ¿CUÁNTAS TAREAS DE MAPEO Y REDUCCIÓN DEBEMOS PLANIFICAR?

**Objetivo:** identificar  $M$  (número de tareas de mapeo) y  $R$  (número de tareas de reducción) que deben ser planificadas

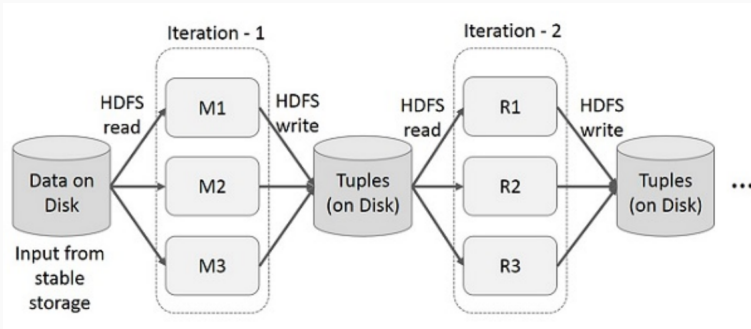
- El número de tareas  $M$  debe ser más grande que el número de nodos en el clúster.
- Mejora el balanceo de cargas y acelera la recuperación en caso de fallas.

# ¿CUÁNTAS TAREAS DE MAPEO Y REDUCCIÓN DEBEMOS PLANIFICAR?

**Objetivo:** identificar  $M$  (número de tareas de mapeo) y  $R$  (número de tareas de reducción) que deben ser planificadas

- El número de tareas  $M$  debe ser más grande que el número de nodos en el clúster.
- Mejora el balanceo de cargas y acelera la recuperación en caso de fallas.
- Usualmente  $R$  es más pequeño que  $M$ .
- El archivo de salida se reparte en los  $R$  nodos.

# PROBLEMAS CON MAPREDUCE



Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, <http://www.mmms.org>

**Problema:** MapReduce incurre en gastos considerables debido a la replicación de datos, E/S de disco, serialización.

## MapReduce tiene dos limitaciones principales

- **Dificultad para programar directamente en MapReduce:** muchos problemas no son fácilmente descritos en MapReduce.
- **Cuellos de botella:** para preservar la persistencia de los datos, el tiempo que se toma en dividir y almacenar los datos es considerable.

En resumen: MapReduce no está totalmente planeado para aplicaciones muy grandes

- Muchas veces es necesario encadenar múltiples tareas de mapeo y reducción



Apache Spark extiende el modelo de programación MapReduce

- Permite el procesamiento de datos distribuidos basado en clúster
- Se basa en un sistema de operaciones (transformaciones - acciones) realizadas sobre colecciones de datos distribuidos (RDD)
- Sistema de cómputo expresivo, no limitado a MapReduce.
- Actualmente, es el sistema más popular de flujo de datos

## Extensiones al modelo MapReduce

- **Más rápido**
  - Evita guardar resultados intermedios a disco
  - Activa la caché de datos para consultas repetitivas (ejemplo, para aprendizaje máquina)
- **Spark presenta su ejecución por grafos dirigidos acíclicos (DAGs)**
- **Presenta funciones extras** (más allá de M-R)
- **Compatible con Hadoop**

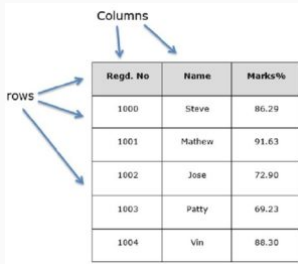


- **Es código abierto** (Apache Foundation)
- Soporta **Java, Scala y Python**
- Principal contribución: **Conjunto de datos distribuidos y resilientes (RDD)**
- **Integra APIs de alto nivel**
  - En las versiones más recientes de Spark incluye dataframes y datasets
  - Ofrece APIs para agregar datos, lo cual permite soportar SQL.

# ¿QUÉ ES UN DATASET?

# ¿QUÉ ES UN DATASET?

- Es un conjunto de datos
- Puede contener cualquier tipo de información



The diagram shows a table with three columns and five rows. The columns are labeled 'Regd. No', 'Name', and 'Marks%'. The rows are labeled 'rows' on the left. Arrows point from the labels to the corresponding parts of the table.

Regd. No	Name	Marks%
1000	Steve	86.29
1001	Mathew	91.63
1002	Jose	72.90
1003	Patty	69.23
1004	Vin	88.30

Imagen tomada de IMF BUSINESS SCHOOL

## Son datasets distribuidos tolerantes a fallos capaces de operar en paralelo

- Son un tipo de estructura principal de Spark
- Los RDDs están particionados sobre los nodos del clúster
- No son inmutables: cuando transformamos un RDD, realmente estamos creando uno nuevo (grafos DAG)
- Distribuye los datos en modo lectura
- Tolerante a fallos
- Los RDDs pueden ser creados desde Hadoop

## ¿CÓMO SE CREA UN RDD?

Existen dos formas comunes para crear un RDD:

- A través del objeto ***SparkContext***
- A partir de conjuntos de datos externos

El método *SparkContext.parallelize* nos permite crear un RDD a partir de una lista o tupla:

```
lista = ['en', 'un', 'lugar', 'de', 'la', 'mancha']  
listardd = sc.parallelize(lista, 4)
```

- `sc` es una instancia de la clase `SparkContext`
- la lista se pasa como argumento a `sc` y se paralelizará automáticamente por Spark
- El programador puede decidir en cuántas partes debe paralelizarse un RDD (ej., 4)

A partir de una fuente de almacenamiento utilizando la función *textFile* del SparkContext:

```
texto = sc.textFile("loremipsum.txt")
```

- Como argumento se pasa un archivo (texto, binario) almacenado en disco
- El método `textFile` cargaría el archivo como un RDD.

- Los RDDs no son valiosos solamente por los datos que contienen, sino por las operaciones que podemos realizar sobre ellos.
- Spark proporcionar un conjunto de acciones para procesar y extraer información: `collect()`, `reduce()`, `count()`, `first`, `foreach()`



`collect()` retorna todos los elementos de un RDD como una lista de Python.

```
rdd = sc.parallelize([4, 1, 2, 6, 1, 5, 3, 3, 2, 4])
lista = rdd.collect()
print ("El tercer elemento de la lista es%d"% lista[2])
>> El tercer elemento de la lista es 2
```

```
print(lista)
```

Importante: si el RDD es muy grande, podemos tener problemas al volcar toda la colección en memoria.

`count()` retorna el número elementos del RDD.

```
rdd = sc.parallelize([4, 1, 2, 6, 1, 5, 3, 3, 2, 4])  
print("El RDD contiene%d elementos"% rdd.count())
```

```
>> El RDD contiene 10 elementos
```

`countByValue()` retorna un diccionario con el número de apariciones de cada elemento en un RDD.

```
rdd = sc.parallelize([4, 1, 2, 6, 1, 5, 3, 3, 2, 4])  
rdd.countByValue()
```

```
>> 1: 2, 2: 2, 3: 2, 4: 2, 5: 1, 6: 1
```

`reduce(func)` agrega los elementos de un RDD según la función que se le pase como parámetro. La función debe cumplir con las siguientes propiedades para que pueda ser calculada en paralelo.

- **Conmutativa**  $(A + B) = B + A$

Asegurando que el resultado será independiente del orden de los elementos en el RDD.

- **Asociativa**  $(A + B) + C = A + (B + C)$

Asegurando que cualquiera de los dos elementos asociados en la agregación a la vez no afecta el resultado final.

## EJEMPLO: REDUCE()

Crea un RDD que multiplique por 2 sus valores y sumar los resultados:

```
rdd = sc.parallelize([1, 1, 1, 1, 2, 2, 2, 3, 3, 4])  
rdd2 = rdd.map(lambda x: x*2)  
tSum = rdd2.reduce(lambda x,y: x+y)  
print (tSum)
```

```
>> 40
```

## EJEMPLO: REDUCE()

Crea un diccionario con elementos (x,1) y suma las apariciones por elemento

```
rdd_text = sc.parallelize(['red', 'red', 'blue',  
'green', 'green','yellow'])  
rdd_aux = rdd_text.map(lambda x: (x,1))  
rdd_result = rdd_aux.reduceByKey(lambda x,y: x+y)  
print (rdd_result.collect())  
  
>> [('blue', 1), ('green', 2), ('yellow', 1), ('red',  
2)]
```

## ACCIÓN: FOREACH()

`foreach()` ejecuta la función que se le pasa por parámetro sobre cada elemento del RDD.

```
rdd = sc.parallelize([4, 1, 2, 6, 1, 5, 3, 3, 2, 4])
def impar(x):
    if x% 2 == 1:
        print ("%d es impar"% x)
rdd.foreach(impar)
>> 1 es impar
1 es impar
5 es impar
3 es impar
3 es impar
```

`saveAsTextFile(directorio)` guarda el RDD como un conjunto de archivos de texto dentro de directorio.



`collectAsMap()` retorna los elementos de un RDD  
clave/valor como un diccionario de python.

```
sc.parallelize ([('a','b'),('c','d')]).collectAsMap()  
>>  {'a': 'b', 'c': 'd'}
```

Función	Valor que retorna
<i>first()</i>	Devuelve el primer valor del RDD
<i>mean()</i>	Devuelve el valor medio
<i>variance()</i>	Devuelve la varianza
<i>stdev()</i>	Devuelve la desviación estándar
<i>take(n)</i>	Devuelve una lista con los n elementos del RDD

No siempre se podrá ejecutar las acciones directamente sobre un RDD, debido a las siguientes razones

- El RDD podría no estar en un formato adecuado
- El RDD podría tener más datos de los necesarios a analizar (*calcular la media de edades*)
- El RDD podría no contener todos los datos necesarios

Antes de realizar acciones sobre los RDD, **primero deben realizarse transformaciones sobre los RDDs**

- Para garantizar que cada RDD contenga los datos unificados, filtrados y formateados para evitar errores
- **Al aplicar una transformación sobre un RDD original, regresará un nuevo RDD**
- Las transformaciones no modifican el RDD original
- Spark evalúa las transformaciones de manera 'perezosa' (*lazy evaluation*)

Las transformaciones construyen RDDs a través de operaciones como: `map()`, `filter()`, `sample()` y `union()`.

`map(func)` retorna un nuevo RDD, resultado de pasar cada uno de los elementos de un RDD original como parámetro de la función

```
rdd = sc.parallelize([4, 0, 2, 6, 1, 5, 3, 9, 7, 8])  
t1 = rdd.map(lambda x: x*2)  
t1.collect()  
>> [8, 0, 4, 12, 2, 10, 6, 18, 14, 16]
```

`filter(func)` retorna un nuevo RDD que contiene los elementos que cumplen la función

```
num = sc.parallelize ([1,2,3,4,5,6,100,2000,4000])  
menor50 = num.filter(lambda x : x < 50)  
menor50.collect()
```

```
>> [1, 2, 3, 4, 5, 6]
```

`distinct()` retorna un nuevo RDD que contiene una sola copia de los diferentes elementos del RDD

```
num=sc.parallelize([1,2,3,4,4,3,2,5])  
num.distinct().collect()
```

```
>> [4, 1, 5, 2, 3]
```



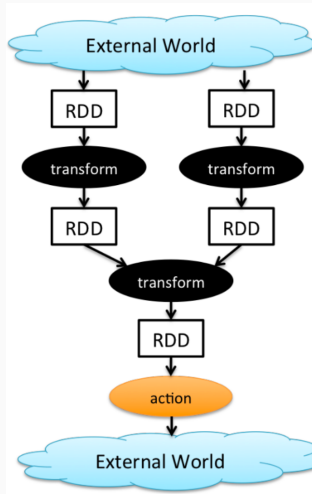
`union()` retorna un nuevo RDD que contiene la unión de los elementos de un RDD y del que se le pasa como argumento

```
city1=sc.parallelize(['Barcelona','Madrid','Paris'])  
city2=sc.parallelize(['Madrid','Londres','Roma'])  
city1.union(city2).collect()
```

```
>> ['Barcelona', 'Madrid', 'Paris', 'Madrid',  
    'Londres', 'Roma']
```

Función	Valor que retorna
<i>intersection()</i>	Devuelve la intersección de 2 RDDs
<i>keys()</i>	Devuelve únicamente las llaves del RDD
<i>sortBy(func)</i>	Ordena un RDD según un criterio

# GRAFO ACÍCLICO DIRIGIDO (DAG)



J. Leskovec, A.Rajaraman, J. Ullman: Mining of Massive Datasets, <http://www.mmds.org>

- Cada tarea de Spark crea un DAG para se ejecuten en un clúster
- Los DAG pueden tener cualquier número de estados (MapReduce, tiene 2 estado predefinidos)
- **Caché de datos**
- Los DAG permiten programar hilos complejos de ejecución en paralelo.

# COMPONENTES DEL ECOSISTEMA DE HADOOP

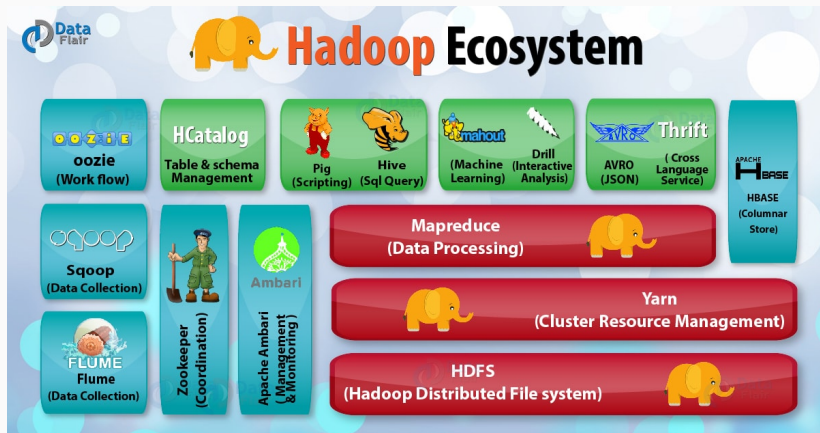
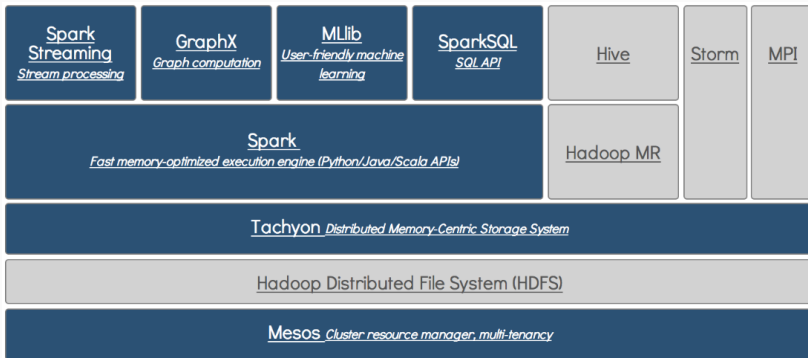


Imagen tomada de <https://bit.ly/2JgZwjx>

# MÓDULOS DE APACHE SPARK



J. Leskovec, A.Rajaraman, J. Ullman: Mining of Massive Datasets, <http://www.mmids.org>

- Conceptos básicos de Spark

[http://reader.digitalbooks.pro/content/preview/books/41061/book/0EBPS/Text/capitulo\\_3.html](http://reader.digitalbooks.pro/content/preview/books/41061/book/0EBPS/Text/capitulo_3.html)