Conceptos y Aplicaciones de Big Data

MapReduce

- Combiners
- Jobs
- Pasaje de información

Prof. Waldo Hasperué
whasperue@lidi.info.unlp.edu.ar

Temario

- Función combiner
- Problemas de varios jobs
- Pasaje de información entre mappers y reducers

Repaso – Etapa map

- Trabaja sobre un split de los datos.
- Es fácilmente paralelizable, se ejecutan tantos mappers como splits existan en el cluster.
- Por cada par clave-valor procesado, puede "escribir" como salida de 0 a M pares resultado.

<k1, v1> → list(<k2,v2>)

Repaso – Etapa map

Java

```
void map(LongWritable key1, Text value1 …{
      while(haya valores en value1){
            context.write(key2, value2);
      }
}
```

Python

```
for line in sys.stdin:
      (key1, value1) = line.split('\t', 1)
      while(haya valores en value1):
            print '%s\t%s' % (key2, value2)
```

Repaso – Etapa map

Java

```
void map(LongWritable key1, Text value1 …{
      while(haya valores en value1){
            cont  t write(key2, value2);
      }
}
```

Se invoca a la función tantas veces como datos se tengan que procesar.
No hay forma de saber cuando es la última llamada a esta función.

Python

```
for line in sys.stdin:
      (key1, value1) = line.split('\t', 1)
      while(haya valores en value1):
            print '%s\t%s' % (key2, value2)
```

Repaso – Etapa map

Java

```
void map(LongWritable key1, Text value1 …{
      while(haya valores en value1){
            context.write(key2, value2);
      }
}
```

En sys.stdin están todos los datos a procesar.
No hay forma de saber cuantos elementos posee esa colección.

Python

```
for line in sys.stdin:
      (key1, value1) = line.split('\t', 1)
      while(haya valores en value1):
            print '%s\t%s' % (key2, value2)
```

Repaso – Etapa map

Java

```
void map(LongWritable key1, Text value1 …{
      while(haya valores en value1){
            context.write(key2, value2);
      }
}
```

Los datos se procesan "como vienen". No existe ningún orden ni por clave, ni por valor

Python

```
for line in sys.stdin:
      (key1, value1) = line.split('\t', 1)
      while(haya valores en value1):
            print '%s\t%s' % (key2, value2)
```

Repaso – Etapa map

Java

```
void map(LongWritable key1, Text value1 …{
      while(haya valores en value1)
            context.wri  ey2, va  );
      }
}
```

En cada llamada se pasa la clave y el valor por parámetro

Python

```
for line in sys.stdin:
      (key1, value1) = line.split('\t', 1)
      while(haya valores en value1):
            print '%s\t%s' % (key2, value2)
```

Repaso – Etapa map

Java

```
void map(LongWritable key1, Text value1 …{
      while(haya valores en value1){
            context.write(key2, value2);
      }
}
```

La clave y el valor hay que "parsearlo"

Python

```
for line in sys.stdin:
      (key1, value1) = line.split('\t', 1)
      while(haya valores en value1):
            print '%s\t%s' % (key2, value2)
```

Repaso – Etapa map

Java

```
void map(LongWritable key1, Text value1 …{
      while(haya valores en value1){
            context.write(key2, value2);
      }
}
```

Dependiendo de lo que haya en **value1**, podemos escribir de 0 a M tuplas resultado

Python

```
for line in sy    din:
      (key1, va    e1) = line.split('\t', 1)
      while(haya valores en value1):
            print '%s\t%s' % (key2, value2)
```

Repaso – Etapa reduce

- Trabaja sobre todas la tuplas que tienen una misma clave.
- Se paraleliza por clave, se ejecutan tantos reducers como claves se generaron en la etapa map.
- Por cada par clave-lista(valor) procesado, puede "escribir" como salida de 0 a M pares resultado.

<k2, list(v2)> → list(<k3, v3>)

Repaso – Etapa reduce

Java

```
void reduce(Text key2, Iterable<IntWritable> values…
{
        for(Object value2 : values) {
                Procesar value2
        }
        context.write(key3, value3);
}
```

Python

```
current_key = None
for line in sys.stdin:
        (key2, value2) = line.split('\t', 1)
        if current_key == key2:
                Procesar value2
        else:
                print '%s\t%s' % (key3, value3)
```

Repaso – Etapa reduce

Java

```
void reduce(Text key2, Iterable<IntWritable> values…
{
        for(O     t value2 : values) {
                  value2

        }
        (
}
```

Se invoca a la función tantas veces como claves se tengan que procesar.
No hay forma de saber cuando es la última llamada a esta función.
La única garantía es que las claves aparecen ordenadas

Python

```
current_
for line in sys.stdin:
        (key2, value2) = line.split('\t', 1)
        if current_key == key2:
                Procesar value2
        else:
                print '%s\t%s' % (key3, value3)
```

Repaso – Etapa reduce

Java

```
void reduce(Text key2, Iterable<IntWritable> values…
{
        f
        }
}
```

En sys.stdin están todas las tuplas a procesar.
No hay forma de saber cuantos elementos tiene la colección.
La única garantía es que las claves están ordenadas

Python

```
current_key = N
for line in sys.stdin:
        (key2, value2) = line.split('\t', 1)
        if current_key == key2:
                Procesar value2
        else:
                print '%s\t%s' % (key3, value3)
```

Repaso – Etapa reduce

Java

```
void reduce(Text key2, Iterable<IntWritable> values…
{
        for(Object value2 : values) {
                Procesar value2
        }
        cont   .write(key3, value3);
}
```

Python

```
curr
for
        else:
                print '%s\t%s' % (key3, value3)
```

Por cada clave se recibe la lista de valores asociados.
Esta lista solo puede ser recorrida una única vez.
No se puede acceder a los elementos por índice.
No se puede adelantar ni rebobinar.

Repaso – Etapa reduce

Java

```
void reduce(Text key2, Iterable<IntWritable> values…
{
        f
        }
        c
}
```

Hay una única lista que se procesa secuencialmente de principio a fin.
El control de "cambio de clave" debe hacerse manualmente.

Python

```
current_key = None
for line in sys.stdin:
        (key2, value2) = line.split('\t', 1)
        if current_key == key2:
                Procesar value2
        else:
                print '%s\t%s' % (key3, value3)
```

Repaso – Etapa reduce

Java

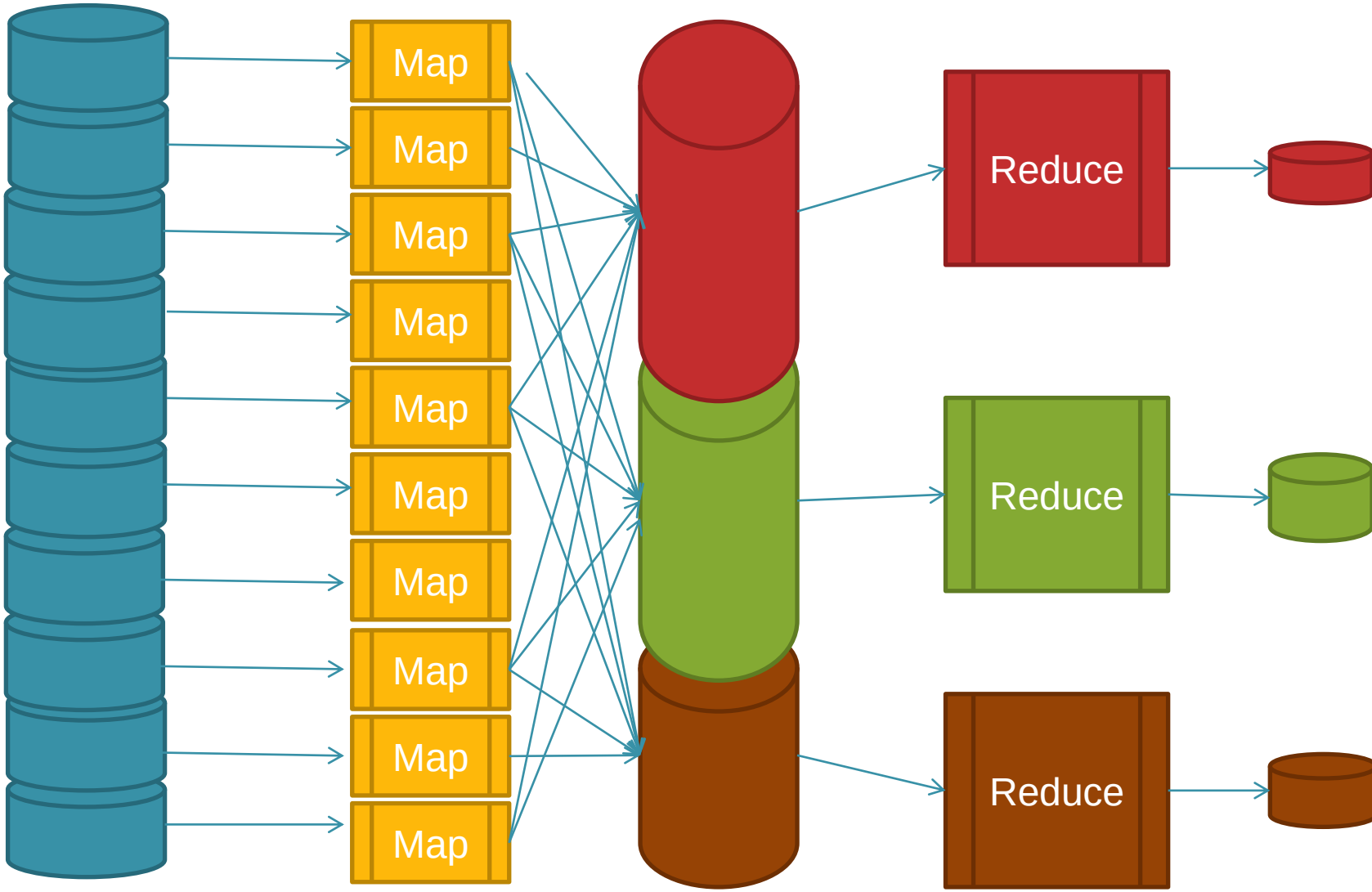
```
void reduce(Text key2, Iterable<IntWritable> values…
{
        for(Object value2 : values) {
                Procesar value2
        }
        context.write(key3, value3);
}
```

Python

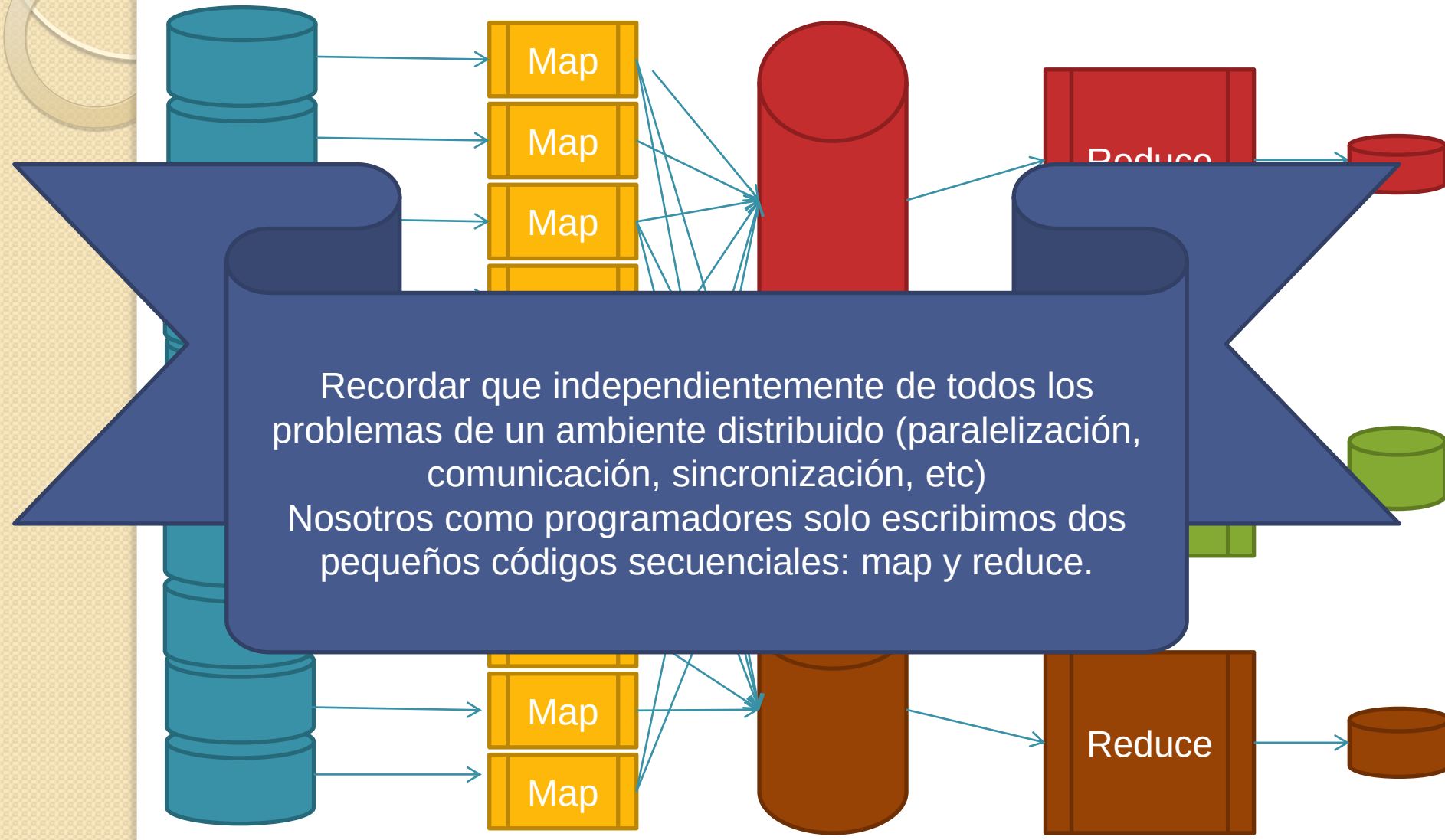
```
current_key = None
for line in sys.stdin:
        (key2, value2) = line                ('\t', 1)
        if current_key == key
                Procesar value
        else:
                print '%s\t%s' % (key3, value3)
```

Al igual que en la etapa map se pueden escribir todos los pares clave-valor que se necesiten.

Repaso – Map y Reduce



Repaso – Map y Reduce



Ejemplo

- Un hipermercado tiene almacenada todas las compras del último bimestre en todas sus sucursales.

  <id_sucursal, id_compra, monto>

- Se desea saber cuales son las *T* compras de mayor dinero.
  - ¿Qué hace el map?
  - ¿Qué hace el reduce?
  - ¿Con cuanta información trabaja el reduce?

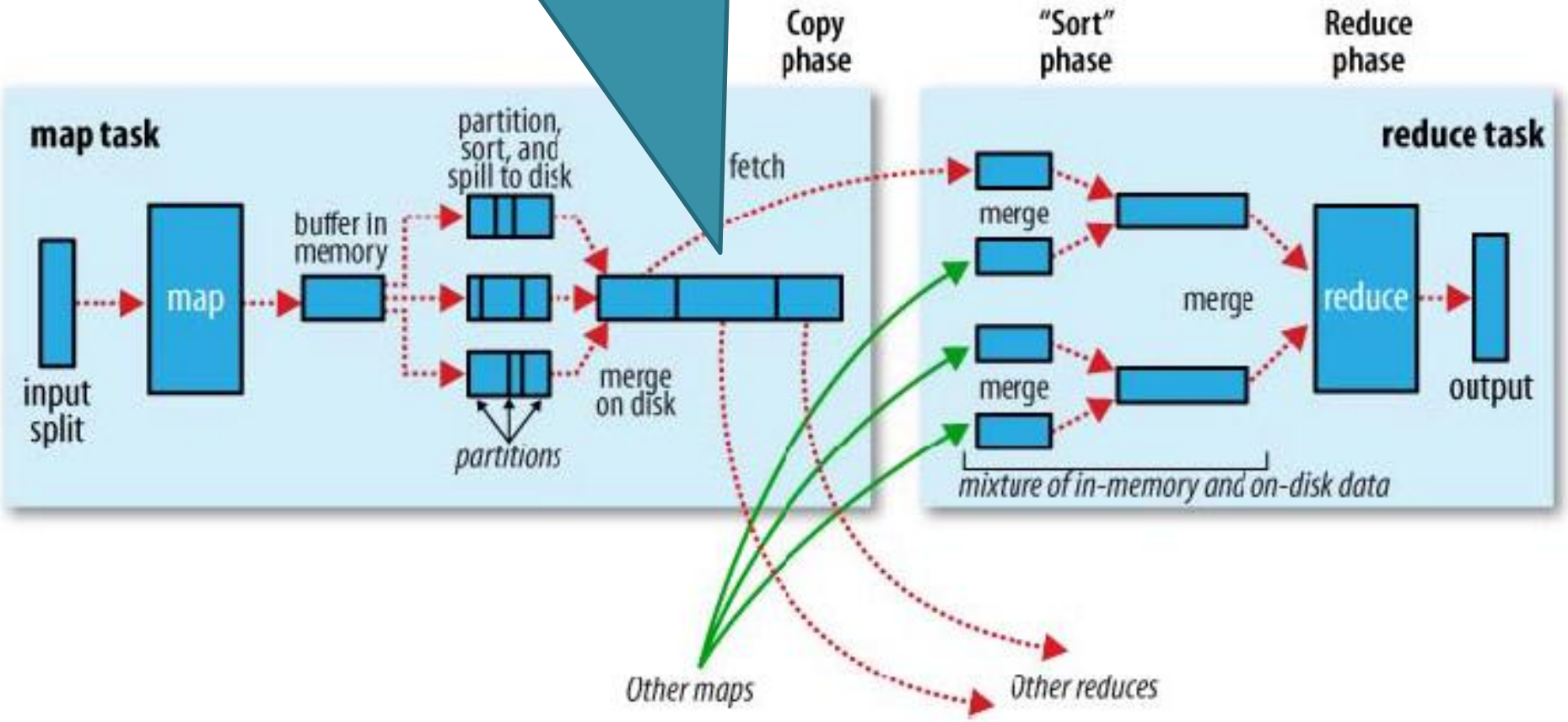
Problema

- Una solución "rebuscada" sería que cada mapper devuelva su top *T* de valores…
- …pero si queremos el top T por sucursal… ¿tiene sentido complicar aún más la función map?
- La solución más "elegante" y respetar la filosofía del paradigma es usar una función combiner.

Función combiner

- La gran desventaja que tiene el proceso de ejecución de un Job MapReduce es la enorme cantidad de datos que se deben mover en el cluster al finalizar la etapa map.
- En la mayoría de los problemas, los reducers trabajan con todos (o gran parte de ellos) los datos del dataset.
- La función combiner es una función que permite minimizar la cantidad de datos a "mover" por el cluster.

Lo que se busca es que el volumen de datos a guardar en disco sea lo más chico posible.



Función combiner

- Esta función es utilizada para "disminuir" la carga de datos en la tarea reduce.
- La función combiner se ejecuta inmediatamente finalizada la tarea map.
- Se ejecuta en el mismo nodo donde se ejecutó el map.

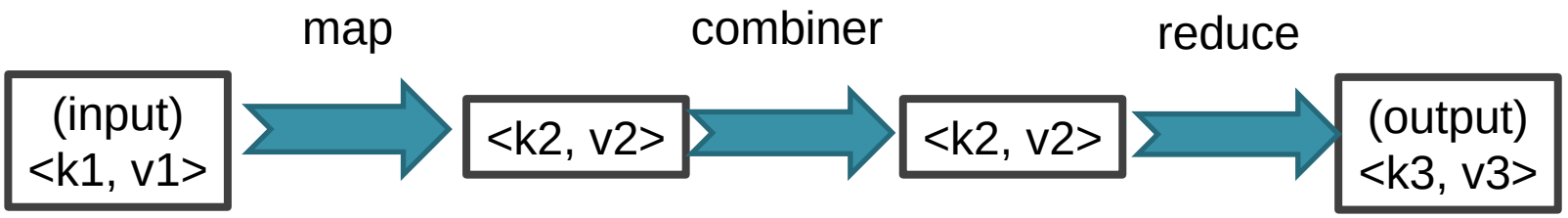
Función combiner

- Hadoop no garantiza cuantas veces es invocada esta función, ya que internamente es solo una "tarea de optimización".
- Solo trabaja con la salida producida por la tarea map (antes de escribir los datos a disco).
- Su objetivo es "disminuir" la carga de datos de los reducers.

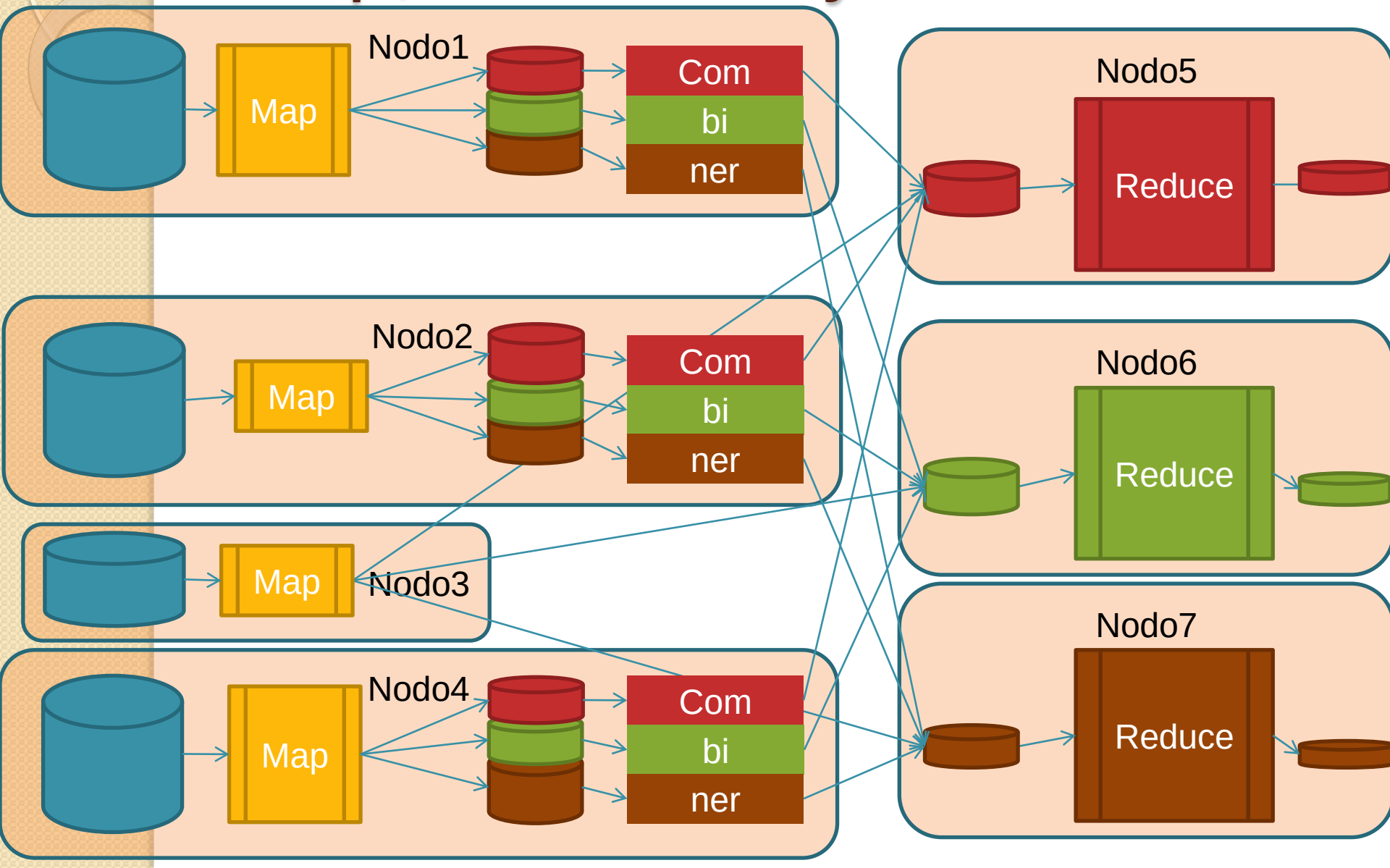
Función combiner

- La función combiner se ejecuta con la salida del map y su resultado es la entrada de la tarea reduce. Por lo tanto la interface de esta función es

<k2,list(v2)> ➔ list(<k2,v2>)



Map, Combiner y Reduce



Ejemplo

- ¿Qué cambios hay que hacer al problema del supermercado si además se desea buscar el bottom B?
- ¿Y si además queremos el promedio?
- ¿Y si además queremos el desvío estándar

$$s^2 = \frac{\sum_{i=1}^{n} (x_i - \overline{x})^2}{n-1}$$

Ejecutando varios jobs

- Muchas veces, resolver un problema complejo representa ejecutar varios jobs, uno detrás de otro de manera secuencial:

Job1 ➔ Job2 ➔ Job3 ➔ Job4

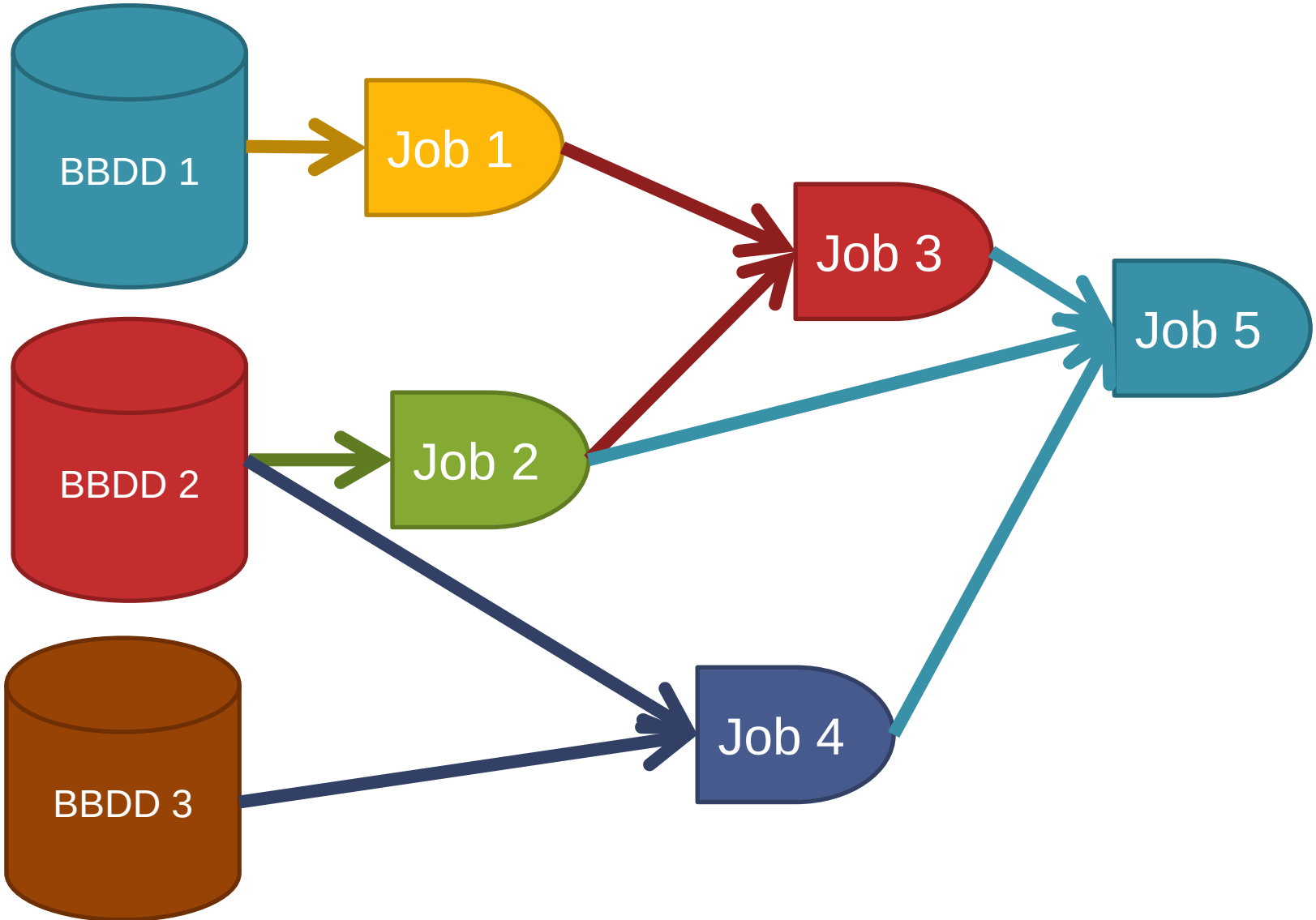
- La salida del Job *i* es entrada para el Job *i*+1. La salida del último Job es el resultado final para el usuario.

Ejecutando varios jobs

- A cada Job se le configura un mapper y un reducer.

- Opcionalmente, dependiendo del problema, dos o más Jobs podrían ejecutar el mismo mapper y/o el mismo reducer (la misma implementación).

Proceso de varios jobs - Ejemplo



Ejecutando varios jobs

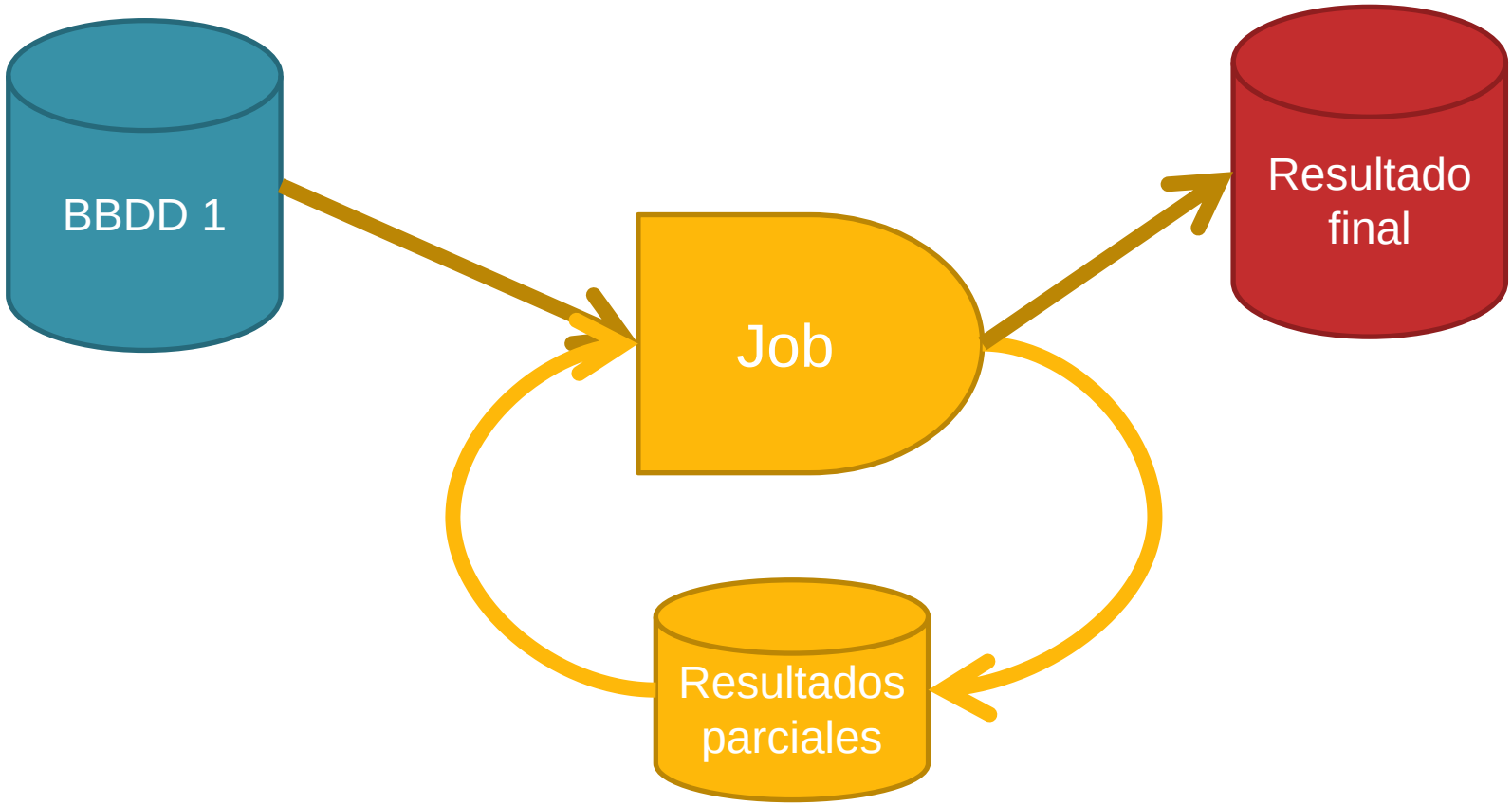
- Se posee un dataset con la siguiente información horaria recolectada durante el último mes: <id_sucursal, id_producto, cantidad>

- Se desea realizar un programa MapReduce que devuelva para cada sucursal, que producto fue el más vendido y cuanto se vendió en total.

Ejecutando varios jobs

- Existen muchos problemas donde se necesita "iterar" sobre el mismo conjunto de datos varias veces.
  - clustering,
  - árboles de clasificación,
  - redes neuronales,
  - algoritmos evolutivos,
  - problemas de optimización,
  - etc.

Proceso de un job iterativo Ejemplo



Ejemplo - Método de Jacobi

- El método de Jacobi es un método iterativo, usado para resolver sistemas de ecuaciones lineales cuadrados (igual cantidad de ecuaciones que de incógnitas):

$$
\begin{aligned}
X - 3Y - \tfrac{1}{2}Z &= 1 \\
-\tfrac{1}{10}X + Y - \tfrac{1}{10}Z &= -4 \\
-\tfrac{1}{2}X - \tfrac{1}{2}Y + Z &= 1
\end{aligned}
$$

Ejemplo - Método de Jacobi

- Se inicia con un valor arbitrario para cada incógnita.
  Ejemplo: ($X_0 = 1$; $Y_0 = 2$; $Z_0 = 3$)

- Se despeja una incógnita diferente en cada ecuación:

$$X_{i+1} = 1 + 3\,Y_i + 1/2\,Z_i$$
$$Y_{i+1} = -4 + 1/10\,X_i + 1/10\,Z_i$$
$$Z_{i+1} = 1 + 1/2\,X_i + 1/2\,Y_i$$

Ejemplo - Método de Jacobi

- Con los valores de la iteración *i* se calculan las variables de la iteración *i*+1:

$$(X_0 = 1; Y_0 = 2; Z_0 = 3)$$

$$\begin{aligned} X_{i+1} &= 1 + 3\,Y_i + 1/2\,Z_i = 8.5 \\ Y_{i+1} &= -4 + 1/10\,X_i + 1/10\,Z_i = -3.6 \\ Z_{i+1} &= 1 + 1/2\,X_i + 1/2\,Y_i = 2.5 \end{aligned}$$

$$(X_1 = 8{,}5; Y_1 = -3{,}6; Z_1 = 2{,}5)$$

Ejemplo - Método de Jacobi

- El método finaliza cuando la diferencia entre los valores de las incógnitas de dos iteraciones consecutivas son menores a un error aceptado:

error = 0.001

dif = 1

while (dif >= error) do

calcular $(X_{i+1}; Y_{i+1}; Z_{i+1})$

$dif = (X_{i+1} - X_i)^2 + (Y_{i+1} - Y_i)^2 + (Z_{i+1} - Z_i)^2$

Ejemplo - Método de Jacobi

Iteración	X	Y	Z	Dif_ant
0	1	2	3	
1	8.5	-3.6	2.5	87.86
2	-8.55	-2.9	3.45	292.095
3	-5.975	-4.51	-4.725	76.05335
4	-14.8925	-5.07	-4.2425	80.06821
5	-16.3313	-5.9135	-8.98125	25.23725
…	…	…	…	…
19	-43.3555	-10.7602	-25.5819	0.950582
…	…	…	…	…
38	-49.2383	-11.8579	-29.4935	0.012495
39	-49.3203	-11.8732	-29.5481	0.009948

Método de Jacobi en MapReduce

- Supongamos tener un Dataset con N ecuaciones de N incógnitas en el formato que espera el método de Jacobi

<i_1, t_i, c_1, c_2, . . ., c_N>
<i_2, t_i, c_1, c_2, . . ., c_N>
<i_3, t_i, c_1, c_2, . . ., c_N>
. . .
<i_N, t_i, c_1, c_2, . . ., c_N>

¿Cómo se implementan el map y el reduce?
¿Cuál es el problema al cambiar de iteración?

Parametrizando jobs

- A veces, en problemas de múltiples jobs, resulta útil pasar parámetros a los diferentes jobs para que estos realicen su tarea.
- Los valores se setean el driver y estos son pasados a los TaskTracker que ejecutan los mappers y los reducers.
- Los mappers y reducers tienen una función *setup* que se ejecuta al momento de crear el TaskTracker.
- También tienen una función *cleanup* que se ejecuta cuando el TaskTracker finaliza su tarea.

Método de Jacobi en MapReduce

- Inicialmente el driver envía un valor arbitrario para cada incógnita.
- Esos valores son enviados a todos los TaskTracker
- Finalizado el job, lee los datos que resultaron del job y los utiliza para:
  - Calcular el error para chequear la condición de fin
  - Enviarlos nuevamente a los TaskTracker en una nueva iteración, si se debe continuar

EJERCICIO 1

- Sea el siguiente dataset dividido en cuatro splits:

Split 1		**Split 2**		**Split 3**		**Split 4**	
Key	Value	Key	Value	Key	Value	Key	Value
34	21	23	45	3	21	30	91
21	34	12	12	15	10	31	32
10	18	36	18	14	18	32	53
32	45	4	97	3	15	19	35

EJERCICIO 1 (CONTINUACIÓN)

Se cuenta con las siguientes funciones:

(la función Reduce también se usa como combiner)

```
def Map(key, value):
    if(value < 30):
        write(1, key)
    else:
        write(2, key)
```

```
def Reduce(key, values):
    return (key,
            max(values))
```

EJERCICIO 1 (CONTINUACIÓN)

Si un TaskTracker en el nodo A ejecuta la etapa map con los splits 1 y 2 y un segundo TaskTracker en el nodo B se encarga de procesar la etapa map para los splits 3 y 4, sabiendo que en ambos nodos se ejecutará la función combiner (Reduce).

Responda:

1. ¿Cuál es la salida de cada TaskTracker?
2. ¿Cuántos TaskTracker ejecutarán la etapa reduce?
3. ¿Qué datos recibe cada reduce?
4. ¿Cuántos archivos habrá en la salida?
5. ¿Qué información tiene cada archivo en la salida?

EJERCICIO 2

- El dataset Ventas.rar tiene información sobre las ventas realizadas durante el último mes por distintas sucursales de una cadena de ferreterías. El formato de los datos del dataset es:
  <id_sucursal, id_producto, cantidad_vendida>
- Calcule:
  - Producto más vendido por sucursal
  - Sucursal que más productos distintos vendió
  - Sucursal que más ítems vendió

EJERCICIO 3

- El dataset Jacobi.txt tiene los coeficientes de un sistema de 15 ecuaciones de 15 incógnitas. Las ecuaciones en el archivo ya están "despejadas" como lo requiere el método de Jacobi.
  Cada línea del archivo posee:
  <var_N, tér_ind, coef_var1, coef_var2, … , coef_ var15>
- Por simplicidad, para cada variable *N* su correspondiente coeficiente es cero.
- Implemente en MapReduce el cálculo del método de Jacobi para la solución del sistema de ecuaciones dado.

EJERCICIO 4

- ¿Si en el dataset original las variables no estuvieran despejadas? ¿Qué habría que modificar?

Ejemplo:

$$X - 3Y - \frac{1}{2}Z = 1$$
$$-\frac{1}{10}X + Y - \frac{1}{10}Z = -4$$
$$-\frac{1}{2}X - \frac{1}{2}Y + Z = 1$$