

MapReduce



# **MÚLTIPLES JOBS PARAMETRIZACIÓN DE TASKS**

Prof. Waldo Hasperué  
[whasperue@lidi.info.unlp.edu.ar](mailto:whasperue@lidi.info.unlp.edu.ar)

# Ejemplo

¿Cómo calculamos el desvío estándar?

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

# Ejecutando varios jobs

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

Job1 → Job2 → Job3 → Job4

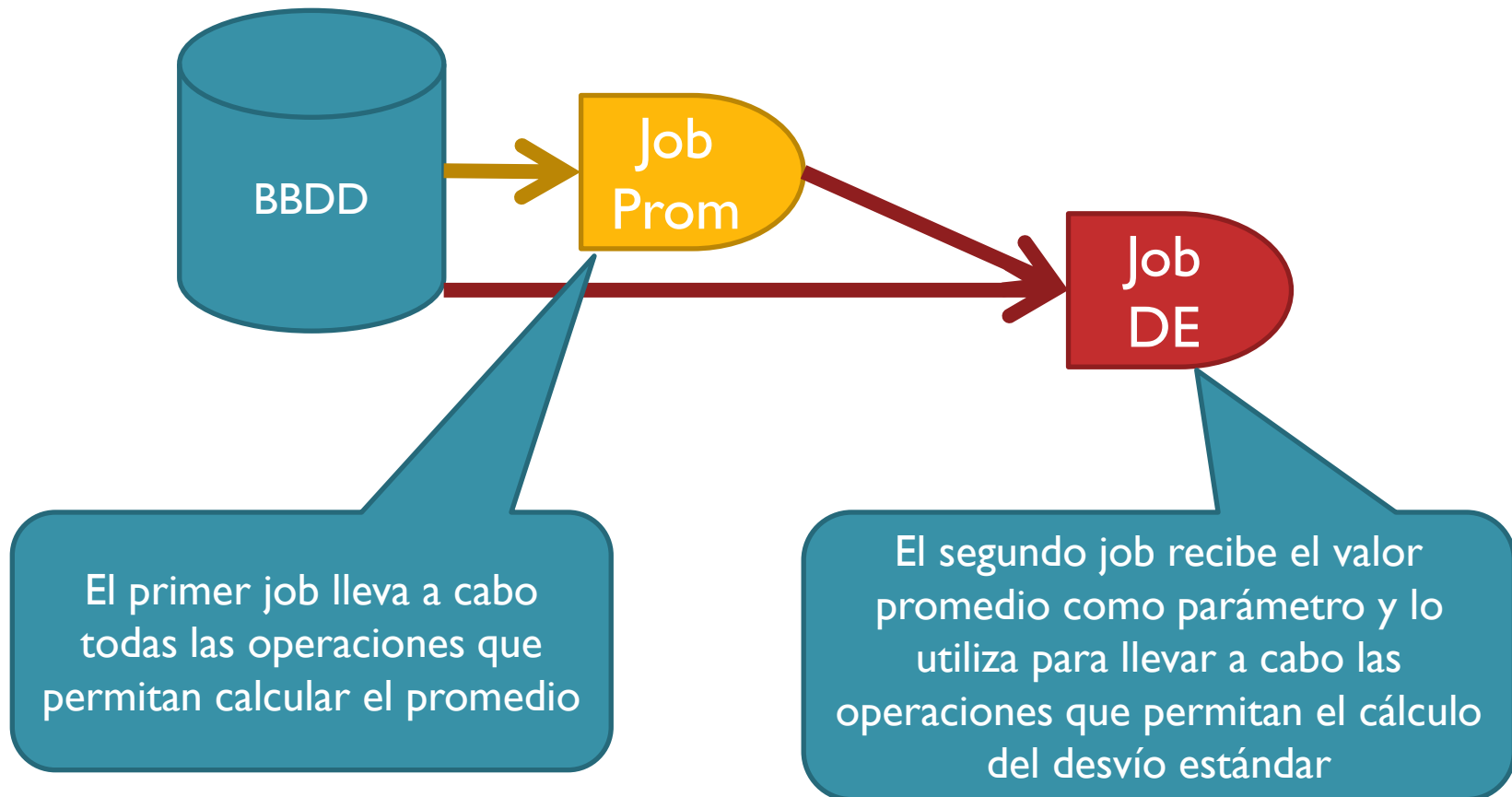
- La salida del Job  $i$  es entrada para el Job  $i+1$ . La salida del último Job es el resultado final para el usuario.

# Ejecutando varios jobs

- A cada Job se le configura un *mapper* y un *reducer* (y eventualmente un *combiner*).
- Opcionalmente, dependiendo del problema, dos o más Jobs podrían ejecutar el mismo *mapper* y/o el mismo *reducer* (la misma implementación).

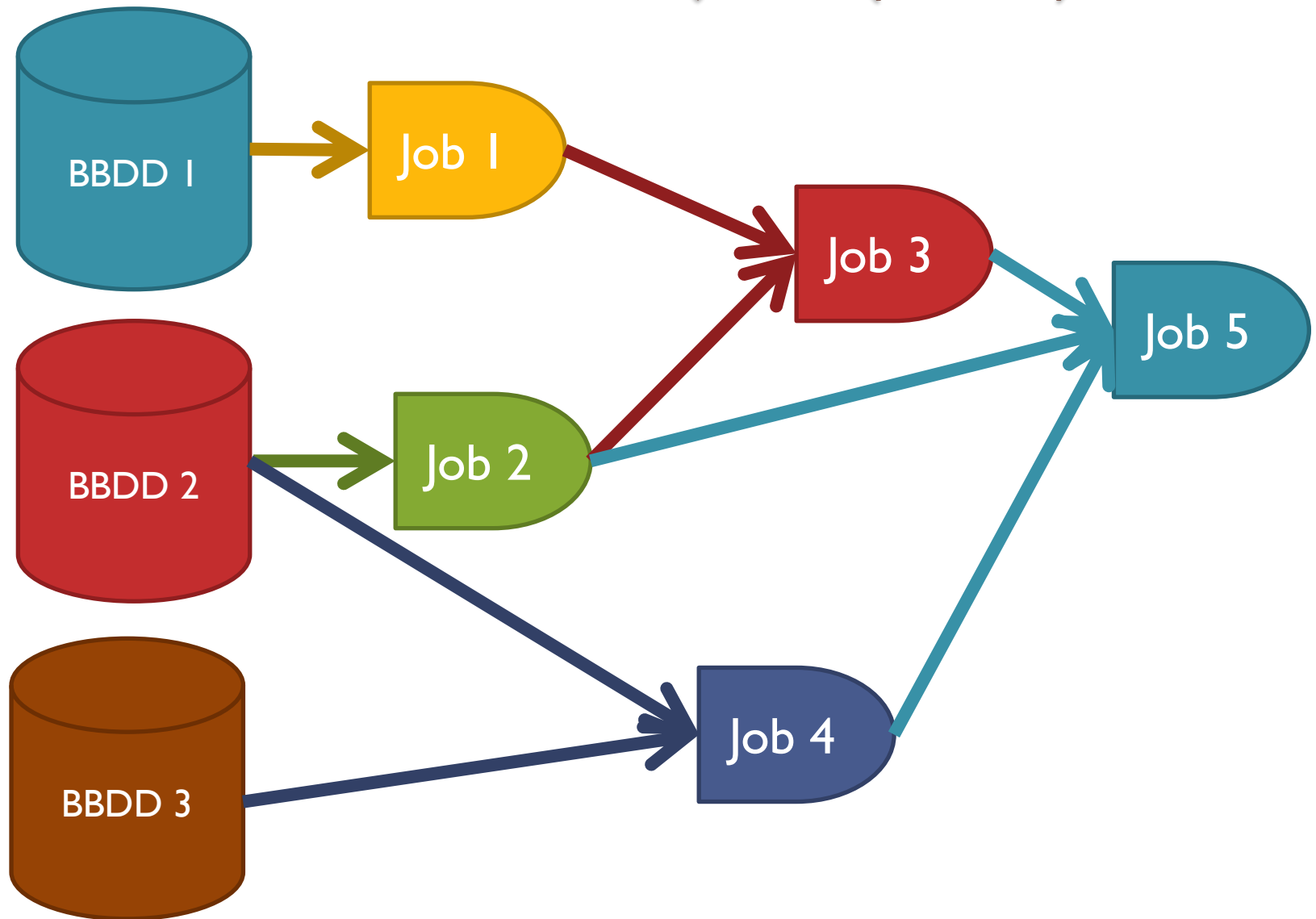
# Ejemplo - Desvío estándar

## Proceso de varios jobs (DAG)



# Ejemplo

## Proceso de varios jobs (DAG)



# Ejemplo de varios jobs

- Se posee un dataset de un sitio web con el siguiente log de la actividad de sus usuarios:

`<id_user, id_page, time>`

- Se desea realizar un programa MapReduce que devuelva para cada usuario, que página fue la más visitada (la página en la que más tiempo permaneció).

# Ejemplo de varios jobs

Id_user	Id_page	time
1	3	10
1	5	45
1	3	23
2	2	12
2	3	20
2	2	13
2	4	15



# Ejemplo de varios jobs

## Solución I

```
def fmap (key, value, context):  
    id_user = key  
    data = value.split("\t")  
    id_page = data[0]  
    time = data [1]  
  
    context.write(id_user, (id_page, time))
```

El reducer debería llevar un acumulado para cada página,  
para luego buscar el máximo

# Ejemplo de varios jobs

## Solución 2

```
def fmap (key, value, context):  
    id_user = key  
    data = value.split("\t")  
    id_page = data[0]  
    time = data [1]  
  
    context.write(id_page, (id_user, time))
```

El reducer recibe todos los usuarios de una misma página.  
Pero luego no podemos determinar la más visitada para cada  
usuario.

# Ejemplo de varios jobs

## Solución 3

```
def fmap (key, value, context):  
    id_user = key  
    data = value.split("\t")  
    id_page = data[0]  
    time = data [1]  
  
    context.write((id_user, id_page), time)
```

Para cada usuario y cada página se puede sumar el tiempo total

# Ejemplo de varios jobs

## Solución 3

```
def fred (key, values, context):  
    id_user, id_page = key  
    total = 0  
    for v in values:  
        total+= v  
  
    context.write((id_user, id_page), total)
```

# Ejemplo de varios jobs

## Solución 3

Id_user, id_page		time
1	3	33
1	5	45
2	2	25
2	3	20
2	4	15

# Ejemplo de varios jobs

## Job 2

```
def fmap (key, value, context):  
    id_user = key  
    data = value.split("\t")  
    id_page = data[0]  
    time_acum = data[1]  
  
    context.write(id_user, (id_page, time_acum))
```

Para cada usuario , el *reducer* puede determinar el máximo

# Ejemplo de varios jobs

```
job1 = Job(inputDir, tmpDir, fmap1, fred1)  
success = job1.waitForCompletion()
```



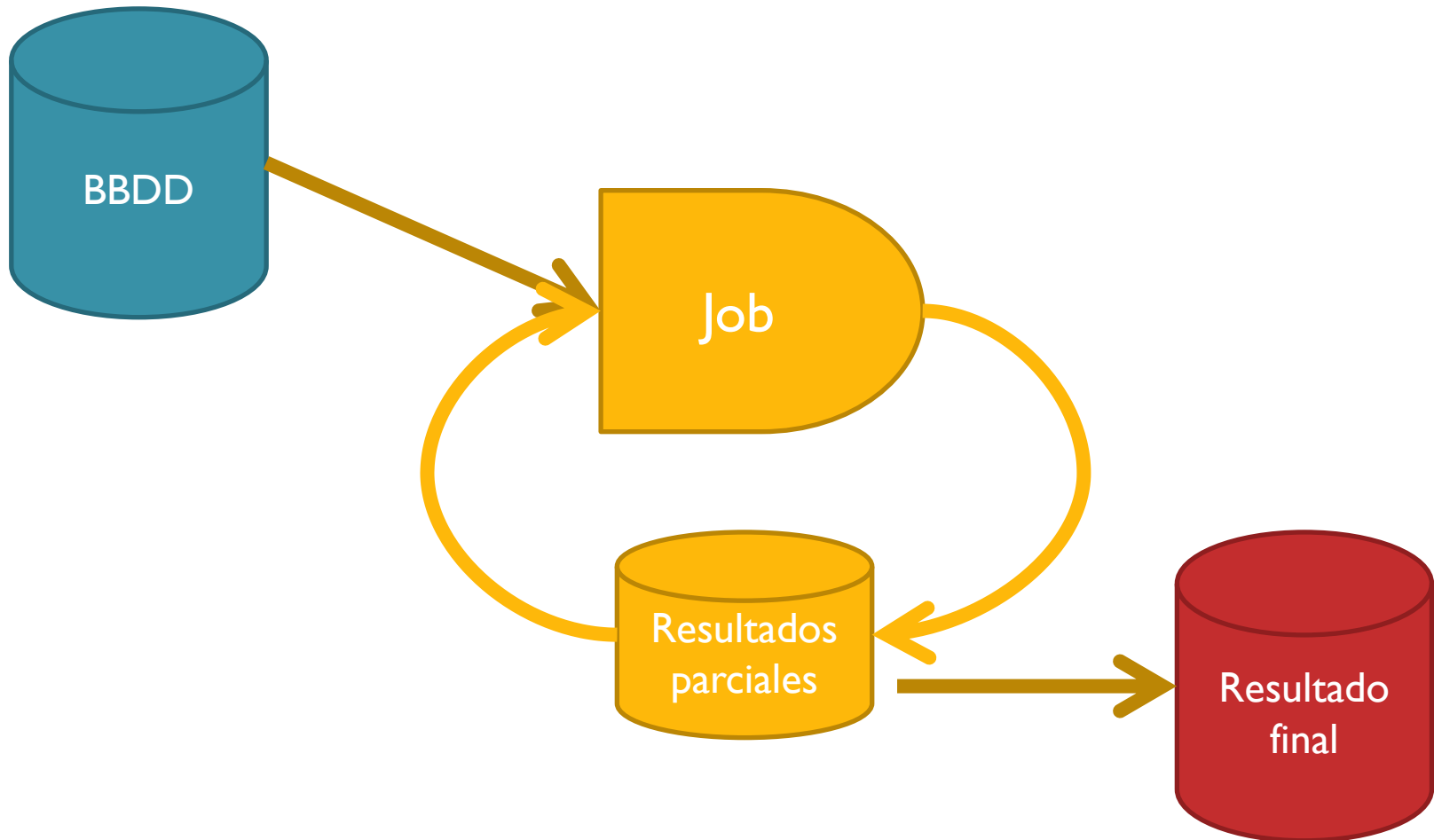
```
job2 = Job(tmpDir, outputDir, fmap2, fred2)  
success = job2.waitForCompletion()
```

# Ejecutando varios jobs

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



# DAG ejemplo de un proceso iterativo



# 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):

$$\begin{array}{rclclclcl} X & - & 3Y & - & 1/2 Z & = & 1 \\ -1/10 X & + & Y & - & 1/10 Z & = & -4 \\ -1/2 X & - & 1/2 Y & + & Z & = & 1 \end{array}$$

# 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:

$$X_1 = 1 + 3Y_0 + 1/2 Z_0$$

$$Y_1 = -4 + 1/10 X_0 + 1/10 Z_0$$

$$Z_1 = 1 + 1/2 X_0 + 1/2 Y_0$$

# Ejemplo - Método de Jacobi

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

$$(X_i = 1; Y_i = 2; Z_i = 3)$$

$$X_{i+1} = 1 + 3Y_i + 1/2 Z_i$$

$$Y_{i+1} = -4 + 1/10 X_i + 1/10 Z_i$$

$$Z_{i+1} = 1 + 1/2 X_i + 1/2 Y_i$$

$$(X_{i+1} = 8,5; Y_{i+1} = -3,6; Z_{i+1} = 2,5)$$

$$(X_{i+2} = -8,55; Y_{i+2} = -2,9; Z_{i+2} = 3,45)$$

# 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

# Ejemplo - Método de Jacobi

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

```
error = 0.01
```

```
dif = 1
```

```
incog = {"X": 1, "Y": 2, "Z": 3}
```

```
while (dif >= error):
```

```
    calcular (incogi, incogi+1)
```

```
    dif = (Xi+1 - Xi)2 +
```

```
          (Yi+1 - Yi)2 + (Zi+1 - Zi)2
```

# Método de Jacobi en MapReduce

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

$\langle \text{var\_1}, t_i, c_1, c_2, \dots, c_N \rangle$

$\langle \text{var\_2}, t_i, c_1, c_2, \dots, c_N \rangle$

$\langle \text{var\_3}, t_i, c_1, c_2, \dots, c_N \rangle$

...

$\langle \text{var\_N}, t_i, c_1, c_2, \dots, c_N \rangle$

¿Cómo se implementan el map y el reduce?

¿Cuál es el problema al cambiar de iteración?

# Método de Jacobi – Fase map

< X	1	0	3	1/2 >
< Y	-4	1/10	0	1/10 >
< Z	1	1/2	1/2	0 >

```
def map(key, value, context):  
    vars = (1, 1, 2, 3)  
    coefs = value.split("\t")  
    res = 0  
    for v in range(4):  
        res = res + vars[i] * coefs[i]  
    context.write(key, res)
```



# Método de Jacobi – Fase reduce

< X            8.5 >

< Y            -3.6 >

< Z            2.5 >

```
def reduce(key, values, context):  
    res = 0  
    for v in values:  
        res = v  
    context.write(key, res)
```

# Método de Jacobi – ¿Problema?

```
def map(key, value, context):  
    vars = (1, 1, 2, 3)
```

Estos valores deberían ser  
reemplazados por (1, 8.5, -3.6, 2.5)

```
    coefs = value.split("\t")  
    res = 0  
    for v in range(4):  
        res = res + vars[i] * coefs[i]  
    context.write(key, res)
```

# Parametrizando jobs

- A veces, en algunos problemas, resulta útil pasar parámetros a los diferentes jobs para que estos realicen su tarea.
- Los valores se setean en el driver y estos son pasados a los TaskTracker que ejecutan los mappers y los reducers.

# 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

# Método de Jacobi – Fase map

```
def map(key, value, context):  
    vars = context["incognitas"]  
    coefs = value.split("\t")  
    res = 0  
    for v in range(4):  
        res = res + vars[i] * coefs[i]  
    context.write(key, res)
```

# Método de Jacobi – Driver

```
job = Job(inputDir, outputDir, fmap, fred)
```

```
coefs = {"incognitas": [1, 1, 2 , 3]}
```

```
job.setParams(coefs)
```

```
success = job.waitForCompletion()
```