

Conceptos y aplicaciones en Big Data

2do semestre 2021

Práctica 2 - Hadoop MapReduce

- 1) ¿En el dataset del ejercicio 1 de la práctica 1 indique para cada Job, si se vería beneficiado por una función combiner? En caso afirmativo, ¿cuál es la implementación de dicha función? ¿Qué datos recibe cada reduce, al utilizar la función combiner?
- 2) Implemente una función combiner para el problema del WordCount.
- 3) Implemente un job MapReduce para calcular el máximo, mínimo, promedio y desvío standard de las ocurrencias de todas las palabras del dataset Libros.
- 4) Utilice el dataset Libros para implementar una aplicación MapReduce que devuelva como salida todos los párrafos que tienen una longitud mayor al promedio.
- 5) El dataset website tiene información sobre el tiempo de permanencia de sus usuarios en cada una de las páginas del sitio. El formato de los datos del dataset es:
`<id_user, id_page, time>`
Implemente una aplicación MapReduce, utilizando combiners en los casos que considere necesario, que calcule
 - a. La página más visitada (la página en la que más tiempo permaneció) para cada usuario
 - b. El usuario que más páginas distintas visitó
 - c. La página más visitada (en cuanto a cantidad de visitas, sin importar el tiempo de permanencia) por todos los usuarios.

Indique como queda el DAG del proceso completo (las tres consultas)
- 6) El dataset Jacobi 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, term_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.

cuántas veces debo iterar?
supongo 15, porque no conozco
los resultados de la ecuación

- 7) Resuelva el problema del método de Jacobi con el dataset jacob2.

Este dataset tiene los datos almacenados de la siguiente forma:

<incognita_i, coef_i, valor>

Donde para cada incógnita, los términos de la ecuación correspondiente aparecen en distintas tuplas, inclusive el término independiente. El valor de `coef_i` es, o bien el nombre de la incógnita afectada por el coeficiente `valor`, o bien el string “`TI`”, haciendo referencia a que `valor` es el término independiente de dicha ecuación. Ejemplo:

| incognita_i | coef_i | valor |
|-------------|--------|-------|
| X | TI | 1 |
| X | Y | 3 |
| X | Z | 0.5 |
| Y | TI | -4 |
| Y | X | 1/10 |
| Y | Z | 1/10 |
| Z | TI | 1 |
| Z | X | 1/2 |
| Z | Y | 1/2 |

Nota: Continúe enviando los valores de las incógnitas por parámetros a los *mappers* y *reducers*, según corresponda.

- 8) Cómo plantearía una solución MapReduce a los siguientes algoritmos secuenciales:

a.

i. entrada

`textos: array [1..N] of string (dataset libros)`

ii. algoritmo

```
a={} ; b={} ; N = len(textos)
```

```
for l in textos:
```

```
    words = l.split()
```

```
    for w in words:
```

```
        a[w] = a[w]+1
```

```
for w in a.keys():
```

```
    for l in lines:
```

```
        words = l.split()
```

```
        if w in words:
```

```
            b[w]=b[w]+1
```

```
for k in a.keys():
```

```
    print(k + " = " + str(a[w] * (N / b[w])))
```

Job 1: cuenta las ocurrencias de cada palabra en el dataset

Job 2: cuenta los párrafos donde aparece cada palabra del dataset

Job 4: hace una cuenta con los resultados de Job 1 y 2 para cada palabra del dataset

Job 3: obtener N?

b.

i. entrada
datos: array [1..N] of <bool₁, bool₂, ..., bool_M>

ii. algoritmo

```
for t in lines:  
    v = t.split("\t")  
    c = v[-1]  
    for a in range(len(v)-1):  
        x= v[a]  
        m[a][x][c] = m[a][x][c] + 1  
  
max=[[0,0,0], [0,0,0]]  
for x in range(len(m)):  
    for y in range(len(m[0])):  
        for z in range(2):  
            if(m[x][y][z] > max[z][0]):  
                max[z][0] = m[x][y][z]  
                max[z][1]=x  
                max[z][2]=y  
  
for z in range(2):  
    print(z ";" + max[z][1] ";" + max[z][2])
```

c.

i. entrada
datos: array [1..N] of <int₁, int₂, ..., int_M>

ii. algoritmo

```
error = 0.001; dif = 1; K=5; M=5  
  
prom = [ 0, 0, 0, 0, 0 ]; N=len(lines)  
  
for t in lines:  
    v = t.split("\t")  
    for m in range(M):  
        prom[m] = prom[m] + float(v[m])  
  
C=[]  
  
for m in range(K):  
    e=[]  
    for m in range(M):  
        e.append( prom[m] / N + random.random() )  
    C.append(e)
```

```

while dif > error:
    S=[]
    for m in range(K):
        S.append([ [0,0,0,0,0], 0 ])

    for t in lines:
        v = t.split("\t")
        min=9999999
        for q in range(K):
            a=0
            for m in range(M):
                a=a + (C[q][m] - float(v[m]))**2
            if(a < min):
                min = a
                Q = q
        for m in range(M):
            S[Q][0][m]= S[Q][0][m] + float(v[m])
        S[Q][1] = S[Q][1] + 1

    for q in range(K):
        if S[q][1] > 0:
            for m in range(M):
                S[q][0][m]= S[q][0][m] / S[q][1]

    dif=0
    for q in range(K):
        for m in range(M):
            dif=dif + (S[q][0][m] - C[q][m])**2
        if(S[q][1] > 0):
            C[q][m] = S[q][0][m]

```