Laborator nr. 1
MapReduce – Noțiuni introductive

1 Breviar teoretic

MapReduce (***MR***) reprezintă un model de programare destinat organizării, transformării, filtrării și extragerii de informații din masive de date. Avantajul acestui model este de a fi paralelizabil și scalabil nativ, fiind relativ ușor de configurat în cadrul unui sistem de calcul paralel și/sau distribuit.

Componentele unei soluții *MR* sunt:

1. **Procedura de mapare** – în termeni generici, această procedură se referă la transformarea reprezentării unui element dintr-un anumit format într-un alt format, corespunzător rezultatului final. Această transformare se realizează prin intermediul unei funcții : $y_i = f(x_i)$

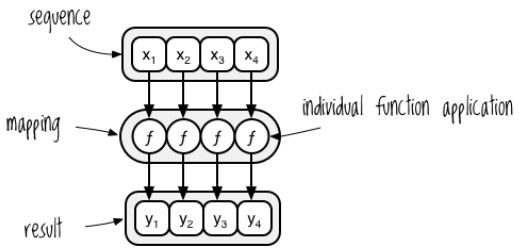

Figura 1: Modelul unei proceduri *map*

2. **Procedura de reducere** – în termeni generici, această procedură aplică o formă de agregare sau de compunere a unui rezultat pe baza unor rezultate intermediare. Această fază se poate implementa prin intermediul unei funcții care realizează pas cu pas această compunere: $s = f(x_i, x_j)$

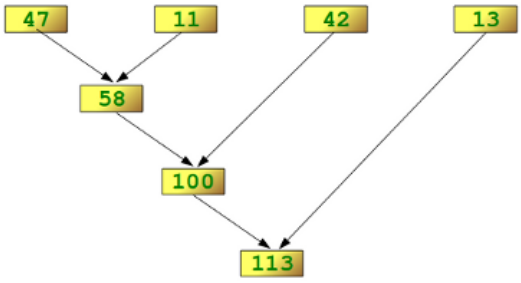

Figura 2: Modelul unei proceduri *reduce*

Aplicațiile *MR* devin utile în mod special atunci când rulează în cadrul unor sisteme distribuite de mari dimensiuni, sisteme care sunt slab cuplate între ele. Apache Hadoop este un astfel de exemplu de aplicație, care poate rula eficient în cadrul unor clustere alcătuite din mii de noduri de calcul. Pentru simplificare, pentru o mai bună înțelegere a conceptelor și pentru a obține un grad de idenpendență de platformă, materialele disciplinei Tehnici BigData sunt axate pe simulări ale MR.

Exemplu Map: Incrementarea elementelor unei liste de întregi

Pentru exemplu, considerăm următoarea listă:

```
data = [1, 2, 3, 4, 5]
```

Abordările tradiționale se bazează pe bucle *for* (Listing 1) sau pe facilități specifice limbajului de programare – *Python list comprehensions* (în mod uzual, mai rapide decât buclele *for* – Listing 2):

```
for i in range(len(data))
        data[i] += 1
```

Listing 1: Incrementare folosind *for*

```
data = [x+1 for x in data]
```

Listing 2: Incrementare folosind *Python list comprehensions*

Utilizarea unei proceduri de tip *map* implică definirea unei funcții care realizează această operație de incrementare și aplicarea acestei funcții asupra fiecărui element din listă. Rezultatul final se poate obține prin transformarea rezultatului în formatul de tip listă original. Vezi Listing 3.

```
def increment(n):
    return n+1

mappedData = map(increment, data)
data = list(mappedData)
```

Listing 3: Incrementare folosind o procedura de tip *Map*

O alternativă mai simplă de a obține același rezultat este prin utilizarea funcțiilor *lambda* – vezi listing-ul următor de cod. Trebuie, însă, reținut faptul că o funcție *lamba* oferă performanțe scăzute față o funcție *clasică*.

```
mappedData = map(lambda n : n+1, data)
```

Sistemele de calcul simple, cu un singur nucleu de procesare, nu oferă un suport foarte bun pentru funcții de tip *map*, motiv pentru care este posibil să nu observăm o creștere a performanțelor în raport cu abordările clasice, precum cele bazate pe bucle *for*. Măsurarea timpului consumat de execuția unei anumite secvențe de cod se poate realiza astfel:

```
import time

startTime = time.perf_counter()
#... your code here ...
endTime = time.perf_counter()
elapsedTime = (endTime - startTime) * 1000 #miliseconds
```

Exemplu Reduce: adunarea elementelor unei liste de întregi

Abordările tradiționale se bazează pe bucle *for*:

```
s = 0
for d in data:
        s += d
```

Soluționarea aceleași probleme folosind o procedură de tip *reduce* implică scrierea unei funcții care să descrie sarcina de lucru la nivel *atomic* și aplicarea acestei funcții peste lista de întregi – *data* (Listing 4).

```
from functools import reduce
# ...

def addition(n, m):
    return n+m

s = reduce(addition, data)
```

Listing 4: *Exemplu Reduce*: adunarea elementelor unei liste de întregi

Observație

Limbajul de programare Python oferă nativ proceduri de tip reduce pentru majoritatea operațiilor de bază: $s = sum(data)$

Exemple complete

1. Dată fiind o listă de întregi, calculați suma valorilor pătratelor acestor elemente.

```
from functools import reduce

def squared(n):
    return n ** 2

def addition(n, m):
    return n+m

data = [1, 2, 3, 4, 5]
mappedData = list(map(squared, data))
reducedData = reduce(addition, mappedData)
print(reducedData)
```

2. Același mod de lucru poate fi utilizat pentru a filtra, de exemplu, în mod eficient, valorile considerate în suma anterioară. Dorim calcularea sumelor pătratelor valorilor numai pentru elementele impare ale listei:

```
from functools import reduce

def squared(n):
    return n ** 2

def addition(n, m):
    return n+m

def isOdd(n):
    return bool(n%2 != 0)

data = [1, 2, 3, 4, 5]
filteredData = list(filter(isOdd, myData))
mappedData = list(map(squared, filteredData))
reducedData = reduce(addition, mappedData)
print(reducedData)
```

2 Exerciții

1. Generați o listă de șiruri și determinați lungimea maximă a acestor șiruri folosind atât abordări clasice, bazate pe bucle *for*, cât și abordări bazate pe *MR*.

2. Măsurați timpul consumat de calcularea sumelor pătratelor valorilor cuprinse într-o listă de întregi atunci când folosim, pe rând, bucle *for*, *list comprehensions* și proceduri de *mapare*. Se vor considera liste de câte 10, 1000, 10000 și 100000 elemente.
   Tip: pentru a genera rapid liste de dimensiuni mai mari, se pot clona liste de câteva elemente. Astfel, secvența de cod:

```
longList = [1, 2, 3] * 1000
```

   va genera o listă de 3000 de elemente, prin repetarea secvenței de bază 1, 2, 3

3. Generați o listă de valori random și aplicați *MR* pentru a determina abaterea standard. Abaterea standard pentru un set de n elemente x_i , $i = 0 \ldots n-1$ se poate calcula pe baza următoarei formule:

$$\sigma = \sqrt{\frac{\sum_{i=0}^{n-1}(x_i - \bar{x})^2}{n}}$$

4. Implementați o soluție clasică pentru a contoriza numărul de apariții ale cuvintelor într-un fișier text. Trebuie să realizati următorii pași:
   (a) împărțiți textul în cuvinte;
   (b) pentru fiecare cuvânt, generați o pereche de tip <*cheie, valoare*>: cheia va fi cuvântul determinat și valoarea inițială va fi valoarea 1 (*prima/următoarea* întâlnire a cuvântului curent);
   (c) pentru fiecare astfel de pereche, reduceți valorile asociate – funcția țintă: *suma* – pentru aceeași cheie.