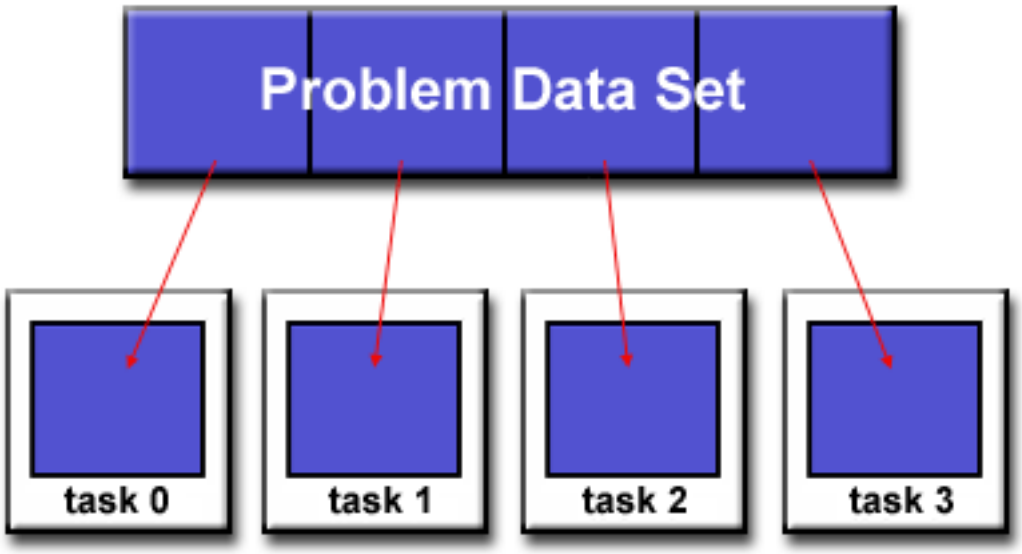
05-MapReduce

August 11, 2020

1 Map Reduce



credits: https://computing.llnl.gov/tutorials/parallel_comp

1.1 `map` function example

The `map(func, seq)` Python function applies the function func to all the elements of the sequence seq. It returns a new list with the elements changed by func

[1]:
```
def f(x):
    return x * x

rdd = [2, 6, -3, 7]
res = map(f, rdd )
res  # Res is an iterator
```

[1]:
```
<map at 0x7fcf2c5b30d0>
```

[2]:
```
print(*res)
```

```
4 36 9 49
```

[3]:
```
from operator import mul
rdd1, rdd2 = [2, 6, -3, 7], [1, -4, 5, 3]
res = map(mul, rdd1, rdd2 ) # element wise sum of rdd1 and rdd2
```

[4]:
```
print(*res)
```

```
2 -24 -15 21
```

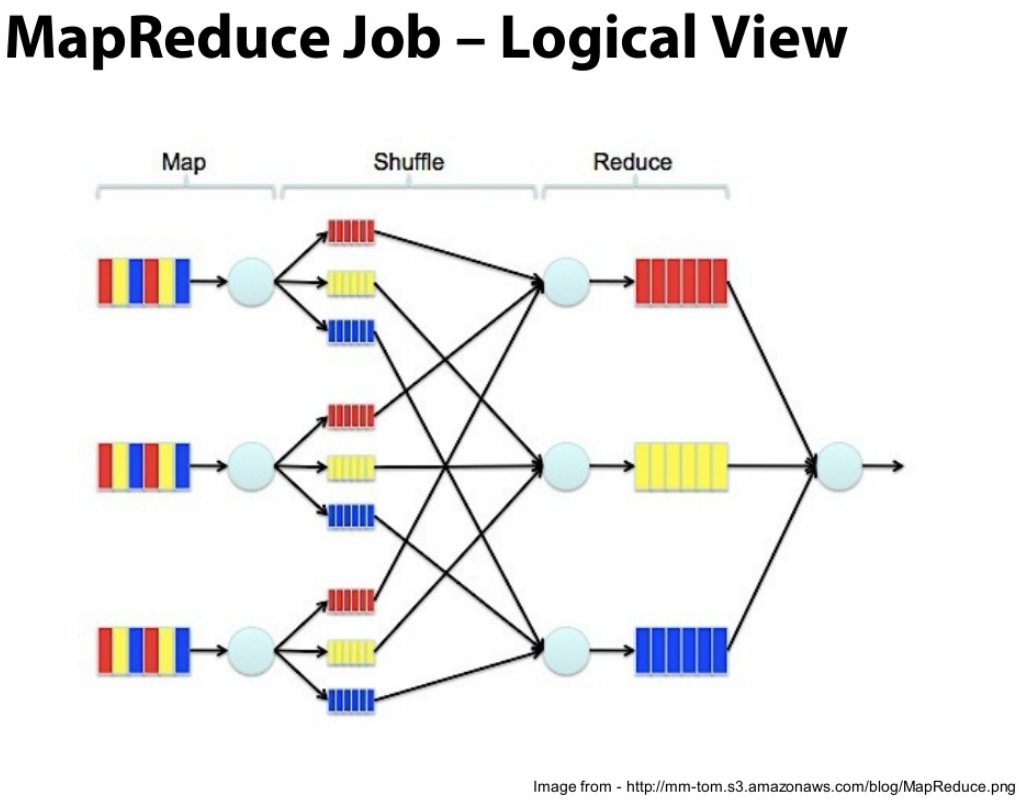


Image from - http://mm-tom.s3.amazonaws.com/blog/MapReduce.png

1.2 `functools.reduce` example

The function `reduce(func, seq)` continually applies the function func() to the sequence seq and return a single value. For example, reduce(f, [1, 2, 3, 4, 5]) calculates f(f(f(f(1,2),3),4),5).

[5]:
```
from functools import reduce
from operator import add

reduce(mul, rdd) # computes ((((1+2)+3)+4)+5)
```

[5]:
```
-252
```

[6]:
```
p = 1
for x in rdd:
    p *= x
p
```

[6]: -252

[7]:
```
def g(x,y):
    return x * y

reduce(g, rdd)
```

[7]: -252

[8]:
```
p = 1
for x in rdd:
    p *= x
p
```

[8]: -252

1.3 Weighted mean and Variance

If the generator of random variable X is discrete with probability mass function $x_1 \mapsto p_1, x_2 \mapsto p_2, \ldots, x_n \mapsto p_n$ then

$$\operatorname{Var}(X) = \left(\sum_{i=1}^{n} p_i x_i^2\right) - \mu^2,$$

where μ is the average value, i.e.

$$\mu = \sum_{i=1}^{n} p_i x_i.$$

[9]:
```
X = [5, 1, 2, 3, 1, 2, 5, 4]
P = [0.05, 0.05, 0.15, 0.05, 0.15, 0.2, 0.1, 0.25]
```

Example of `zip`

[10]:
```
for x, p in zip(X, P):
    print(f" x = {x} ..... p = {p}")
```

```
 x = 5 … p = 0.05
 x = 1 … p = 0.05
 x = 2 … p = 0.15
```

```
x = 3 … p = 0.05
x = 1 … p = 0.15
x = 2 … p = 0.2
x = 5 … p = 0.1
x = 4 … p = 0.25
```

[11]:
```
from itertools import zip_longest

for x, p in zip_longest(X, [0.1], fillvalue=0.0):
    print(f" x = {x} ..... p = {p}")
```

```
x = 5 … p = 0.1
x = 1 … p = 0.0
x = 2 … p = 0.0
x = 3 … p = 0.0
x = 1 … p = 0.0
x = 2 … p = 0.0
x = 5 … p = 0.0
x = 4 … p = 0.0
```

[12]:
```
sum(P)
```

[12]:
```
0.9999999999999999
```

1.3.1 Exercise 5.1

- Write functions to compute the average value and variance using for loops

[13]:
```
def weighted_mean( X, P):
    return sum([x*p for x, p in zip(X,P)])

weighted_mean(X,P)
```

[13]:
```
2.8
```

[14]:
```
def variance(X, P):
    mu = weighted_mean(X,P)
    return sum([p*x*x for x,p in zip(X,P)]) - mu**2

variance(X, P)
```

[14]:
```
1.9600000000000017
```

1.3.2 Exercise 5.2

- Write functions to compute the average value and variance using `map` and `reduce`

[15]:
```
from operator import add, mul
from functools import reduce

def weighted_mean( X, P):
    return reduce(add,map(mul, X, P))

weighted_mean(X,P)
```

[15]: 2.8

[16]:
```
def variance(X, P):
    mu = weighted_mean(X,P)
    return reduce(add,map(lambda x,p:p*x*x, X, P)) - mu**2

variance(X, P)
```

[16]: 1.9600000000000017

1.3.3 Examples with filter

[17]:
```
res = filter( lambda p: p > 0.1, P)  # select p > 0.1
print(*res)
```

```
0.15 0.15 0.2 0.25
```

[18]:
```
res = filter( lambda x: x % 3 == 0, range(15)) # select integer that can be␣
 ↪divided by 3
print(*res)
```

```
0 3 6 9 12
```

NB: Exercises above are just made to help to understand map-reduce process. This is a bad way to code a variance in Python. You should use Numpy instead.

1.4 Wordcount

We will modify the `wordcount` application into a map-reduce process.

The `map` process takes text files as input and breaks it into words. The `reduce` process sums the counts for each word and emits a single key/value with the word and sum.

We need to split the wordcount function we wrote in notebook 04 in order to use map and reduce.

In the following exercices we will implement in Python the Java example described in Hadoop documentation.

1.5 Map - Read file and return a key/value pairs

1.5.1 Exercise 5.3

Write a function `mapper` with a single file name as input that returns a sorted sequence of tuples (word, 1) values.

```
mapper('sample.txt')
[('adipisci', 1), ('adipisci', 1), ('adipisci', 1), ('adipisci', 1), ('adipisci', 1), ('adipisc
```

[19]:
```
import lorem
with open('sample.txt','w') as f:
    f.write(lorem.text())
```

[20]:
```
def mapper(filename):
    with open(filename) as f:
        data = f.read()

    data = data.strip().replace(".","").lower().split()

    return sorted([(w,1) for w in data])

mapper("sample.txt")
```

[20]:
```
[('adipisci', 1),
 ('adipisci', 1),
 ('adipisci', 1),
 ('adipisci', 1),
 ('adipisci', 1),
 ('adipisci', 1),
 ('adipisci', 1),
 ('adipisci', 1),
 ('adipisci', 1),
 ('aliquam', 1),
 ('aliquam', 1),
 ('aliquam', 1),
 ('aliquam', 1),
 ('aliquam', 1),
 ('aliquam', 1),
 ('aliquam', 1),
 ('aliquam', 1),
 ('amet', 1),
 ('amet', 1),
 ('amet', 1),
 ('amet', 1),
 ('amet', 1),
 ('amet', 1),
 ('amet', 1),
```

```
('amet', 1),
('consectetur', 1),
('consectetur', 1),
('consectetur', 1),
('consectetur', 1),
('consectetur', 1),
('consectetur', 1),
('dolor', 1),
('dolor', 1),
('dolor', 1),
('dolor', 1),
('dolor', 1),
('dolor', 1),
('dolor', 1),
('dolor', 1),
('dolor', 1),
('dolor', 1),
('dolor', 1),
('dolor', 1),
('dolor', 1),
('dolor', 1),
('dolor', 1),
('dolore', 1),
('dolore', 1),
('dolore', 1),
('dolore', 1),
('dolore', 1),
('dolore', 1),
('dolore', 1),
('dolore', 1),
('dolorem', 1),
('dolorem', 1),
('dolorem', 1),
('dolorem', 1),
('dolorem', 1),
('dolorem', 1),
('dolorem', 1),
('dolorem', 1),
('dolorem', 1),
('dolorem', 1),
('dolorem', 1),
('dolorem', 1),
('eius', 1),
('eius', 1),
('eius', 1),
('eius', 1),
('eius', 1),
('eius', 1),
('eius', 1),
```

```
('eius', 1),
('eius', 1),
('eius', 1),
('est', 1),
('est', 1),
('est', 1),
('est', 1),
('est', 1),
('etincidunt', 1),
('etincidunt', 1),
('etincidunt', 1),
('etincidunt', 1),
('etincidunt', 1),
('etincidunt', 1),
('etincidunt', 1),
('etincidunt', 1),
('etincidunt', 1),
('ipsum', 1),
('ipsum', 1),
('ipsum', 1),
('ipsum', 1),
('ipsum', 1),
('ipsum', 1),
('ipsum', 1),
('ipsum', 1),
('ipsum', 1),
('ipsum', 1),
('ipsum', 1),
('labore', 1),
('labore', 1),
('labore', 1),
('labore', 1),
('labore', 1),
('labore', 1),
('labore', 1),
('labore', 1),
('labore', 1),
('labore', 1),
('labore', 1),
('magnam', 1),
('magnam', 1),
('magnam', 1),
('magnam', 1),
('magnam', 1),
('magnam', 1),
('magnam', 1),
('magnam', 1),
```

```
('magnam', 1),
('modi', 1),
('modi', 1),
('modi', 1),
('modi', 1),
('modi', 1),
('modi', 1),
('modi', 1),
('modi', 1),
('modi', 1),
('modi', 1),
('modi', 1),
('neque', 1),
('neque', 1),
('neque', 1),
('neque', 1),
('neque', 1),
('neque', 1),
('neque', 1),
('neque', 1),
('neque', 1),
('neque', 1),
('neque', 1),
('neque', 1),
('neque', 1),
('neque', 1),
('neque', 1),
('non', 1),
('non', 1),
('non', 1),
('non', 1),
('non', 1),
('non', 1),
('non', 1),
('non', 1),
('non', 1),
('non', 1),
('non', 1),
('non', 1),
('non', 1),
('non', 1),
('numquam', 1),
('numquam', 1),
('numquam', 1),
('numquam', 1),
('numquam', 1),
('numquam', 1),
('numquam', 1),
('numquam', 1),
('numquam', 1),
```

```
('numquam', 1),
('numquam', 1),
('porro', 1),
('porro', 1),
('porro', 1),
('porro', 1),
('porro', 1),
('porro', 1),
('porro', 1),
('porro', 1),
('porro', 1),
('porro', 1),
('porro', 1),
('porro', 1),
('porro', 1),
('quaerat', 1),
('quaerat', 1),
('quiquia', 1),
('quiquia', 1),
('quiquia', 1),
('quiquia', 1),
('quiquia', 1),
('quisquam', 1),
('quisquam', 1),
('quisquam', 1),
('quisquam', 1),
('quisquam', 1),
('quisquam', 1),
('quisquam', 1),
('quisquam', 1),
('quisquam', 1),
('sed', 1),
('sed', 1),
('sed', 1),
('sed', 1),
('sit', 1),
('sit', 1),
('sit', 1),
('sit', 1),
('sit', 1),
('sit', 1),
('sit', 1),
('sit', 1),
('sit', 1),
('sit', 1),
('tempora', 1),
('tempora', 1),
('tempora', 1),
('tempora', 1),
('tempora', 1),
```

```
 ('tempora', 1),
 ('ut', 1),
 ('ut', 1),
 ('ut', 1),
 ('ut', 1),
 ('ut', 1),
 ('ut', 1),
 ('ut', 1),
 ('velit', 1),
 ('velit', 1),
 ('velit', 1),
 ('velit', 1),
 ('velit', 1),
 ('voluptatem', 1),
 ('voluptatem', 1),
 ('voluptatem', 1),
 ('voluptatem', 1)]
```

1.6 Partition

1.6.1 Exercise 5.4

Create a function named `partitioner` that stores the key/value pairs from `mapper` that group (word, 1) pairs into a list as:

```
partitioner(mapper('sample.txt'))
[('adipisci', [1, 1, 1, 1, 1, 1, 1]), ('aliquam', [1, 1, 1, 1, 1, 1, 1]), ('amet', [1, 1, 1, 1
```

[21]:
```
from collections import defaultdict


def partitioner(mapped_values):

    res = defaultdict(list)
    for w, c in mapped_values:
        res[w].append(c)

    return res.items()


partitioner(mapper('sample.txt'))
```

[21]:
```
dict_items([('adipisci', [1, 1, 1, 1, 1, 1, 1, 1, 1]), ('aliquam', [1, 1, 1, 1,
1, 1, 1, 1]), ('amet', [1, 1, 1, 1, 1, 1, 1, 1]), ('consectetur', [1, 1, 1, 1,
1, 1]), ('dolor', [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]), ('dolore', [1,
1, 1, 1, 1, 1, 1]), ('dolorem', [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]), ('eius', [1,
1, 1, 1, 1, 1, 1, 1, 1]), ('est', [1, 1, 1, 1, 1]), ('etincidunt', [1, 1, 1,
```

```
1, 1, 1, 1, 1, 1]), ('ipsum', [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]), ('labore', [1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1]), ('magnam', [1, 1, 1, 1, 1, 1, 1, 1, 1]),
('modi', [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]), ('neque', [1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1]), ('non', [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]), ('numquam',
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]), ('porro', [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1]), ('quaerat', [1, 1]), ('quiquia', [1, 1, 1, 1, 1]), ('quisquam', [1, 1, 1,
1, 1, 1, 1, 1]), ('sed', [1, 1, 1, 1]), ('sit', [1, 1, 1, 1, 1, 1, 1, 1, 1]),
('tempora', [1, 1, 1, 1, 1, 1]), ('ut', [1, 1, 1, 1, 1, 1, 1]), ('velit', [1, 1,
1, 1, 1]), ('voluptatem', [1, 1, 1, 1])])
```

1.7 Reduce - Sums the counts and returns a single key/value (word, sum).

1.7.1 Exercice 5.5

Write the function `reducer` that read a tuple (word,[1,1,1,..,1]) and sum the occurrences of word to a final count, and then output the tuple (word,occurences).

```
reducer(('hello',[1,1,1,1,1])
('hello',5)
```

[22]:
```
from operator import itemgetter


def reducer( item ):
    w, v = item
    return (w,len(v))

reducer(('hello',[1,1,1,1,1]))
```

[22]:
```
('hello', 5)
```

1.8 Process several files

Let's create 8 files `sample[0-7].txt`. Set most common words at the top of the output list.

[23]:
```
from lorem import text
for i in range(8):
    with open("sample{0:02d}.txt".format(i), "w") as f:
        f.write(text())
```

[24]:
```
import glob
files = sorted(glob.glob('sample0*.txt'))
files
```

[24]:
```
['sample00.txt',
 'sample01.txt',
```

```
 'sample02.txt',
 'sample03.txt',
 'sample04.txt',
 'sample05.txt',
 'sample06.txt',
 'sample07.txt',
 'sample08.txt',
 'sample09.txt']
```

1.8.1 Exercise 5.6

- Use functions implemented above to count (word, occurences) by using a for loops over files and partitioned data.

[25]:
```
from itertools import chain

def wordcount(files):

    mapped_values = [mapper(file) for file in files]

    partioned_values = partitioner(chain(*mapped_values))

    return sorted([reducer(val) for val in partioned_values],
                  key = itemgetter(1),
                  reverse = True)

wordcount(files)
```

[25]:
```
[('tempora', 94),
 ('aliquam', 92),
 ('sed', 92),
 ('porro', 91),
 ('dolore', 89),
 ('est', 86),
 ('neque', 86),
 ('non', 86),
 ('consectetur', 85),
 ('ipsum', 85),
 ('modi', 85),
 ('quisquam', 85),
 ('velit', 85),
 ('quiquia', 84),
 ('sit', 82),
 ('ut', 82),
 ('numquam', 81),
 ('quaerat', 79),
```

```
 ('dolorem', 78),
 ('eius', 78),
 ('labore', 75),
 ('magnam', 74),
 ('dolor', 73),
 ('etincidunt', 69),
 ('amet', 67),
 ('adipisci', 66),
 ('voluptatem', 63)]
```

1.8.2 Exercise 5.7

- This time use `map` function to apply mapper and reducer.

[26]:
```
def wordcount(files):

    mapped_values = map(mapper, files)
    partioned_values = partitioner(chain(*mapped_values))

    return sorted( map(reducer, partioned_values),
                  key=itemgetter(1),
                  reverse=True)


wordcount(files)
```

[26]:
```
[('tempora', 94),
 ('aliquam', 92),
 ('sed', 92),
 ('porro', 91),
 ('dolore', 89),
 ('est', 86),
 ('neque', 86),
 ('non', 86),
 ('consectetur', 85),
 ('ipsum', 85),
 ('modi', 85),
 ('quisquam', 85),
 ('velit', 85),
 ('quiquia', 84),
 ('sit', 82),
 ('ut', 82),
 ('numquam', 81),
 ('quaerat', 79),
 ('dolorem', 78),
 ('eius', 78),
```

```
('labore', 75),
('magnam', 74),
('dolor', 73),
('etincidunt', 69),
('amet', 67),
('adipisci', 66),
('voluptatem', 63)]
```