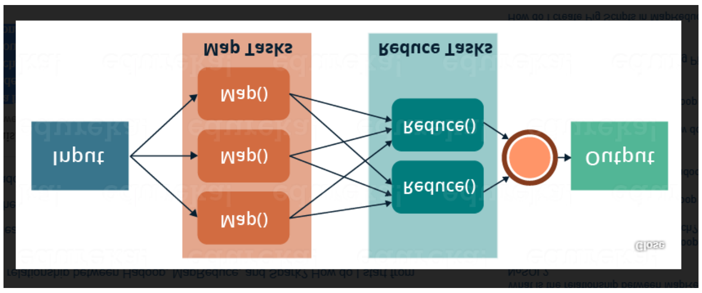
Lab – 5

MapReduce Programming using Python

MapReduce: Programming Model and Implementations:

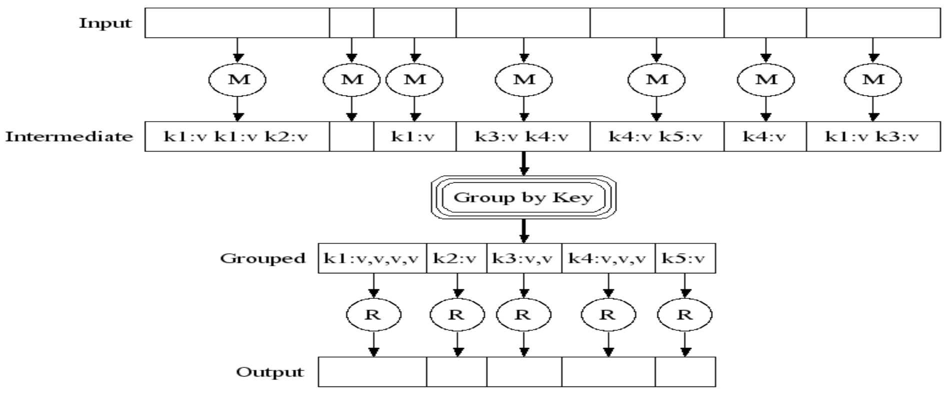
Hadoop is a framework that allows to process and store huge data sets. Basically, Hadoop can be divided into two parts: processing and storage. So, MapReduce is a programming model which allows you to process huge data stored in Hadoop. When you install Hadoop in a cluster, we get MapReduce as a service where you can write programs to perform computations in data in parallel and distributed fashion.

Map – Reduce Implementation:



MapReduce is a programming framework that allows us to perform distributed and parallel processing on large data sets in a distributed environment. MapReduce consists of two distinct tasks – Map and Reduce. As the name MapReduce suggests, reducer phase takes place after mapper phase has been completed. So, the first is the map job, where a block of data is read and processed to produce key-value pairs as intermediate outputs. The output of a Mapper or map job (key-value pairs) is input to the Reducer. The reducer receives the key-value pair from multiple map jobs. Then, the reducer aggregates those intermediate data tuples (intermediate key-value pair) into a smaller set of tuples or key-value pairs which is the final output.

Execution:



- To generate a set of output key-value pairs from a set of input key-value pairs
  - $\{ < k_i, v_i >\} \rightarrow \{ < k_o, v_o >\}$
- Expressed using two abstractions:
  - Map task
    - $< k_i, v_i > \rightarrow \{ < k_{int}, v_{int} > \}$
  - Reduce task
    - $< k_{int}, \{v_{int}\} > \rightarrow < k_o, v_o >$

The Wordcount Example

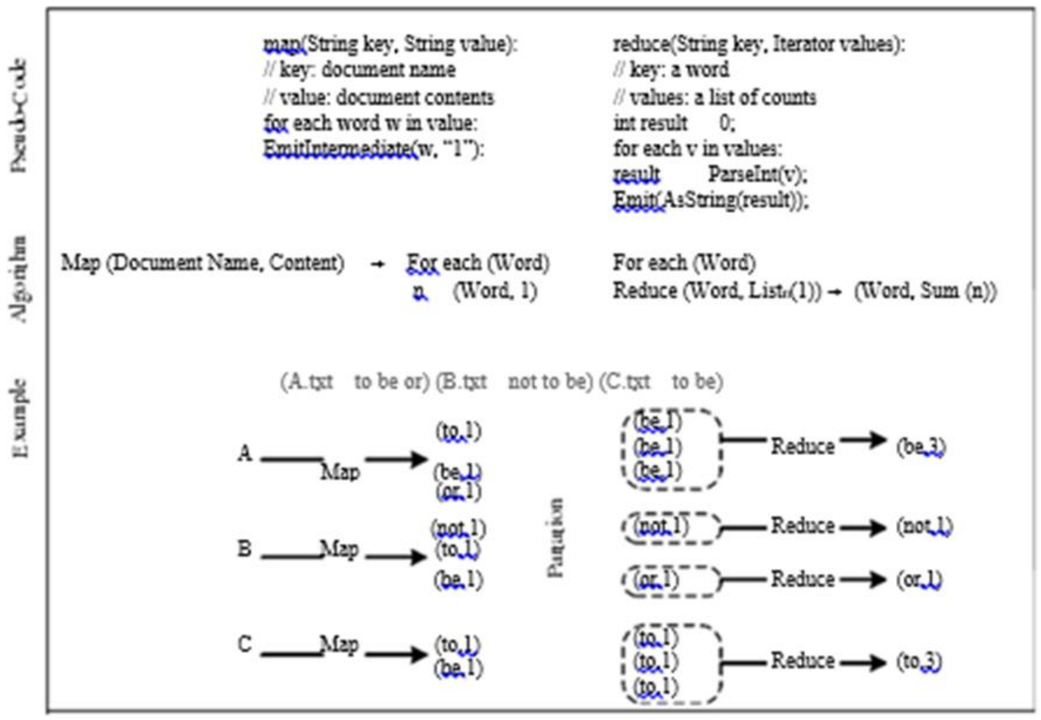
The Wordcount application counts the number of occurrences of each word in a large collection of documents.

The steps of the process are briefly described as follows:

→ The input is read and broken up into key/value pairs (e.g., the Map function emits a word and its associated count of occurrence, which is just “1”).

→ The pairs are partitioned into groups for processing, and they are sorted according to their key as they arrive for reduction.

→ Finally, the key/value pairs are reduced, once for each unique key in the sorted list, to produce a combined result (e.g., the Reduce function sums all the counts emitted for a particular word).



```
map(String key, String value):
// key: document name
// value: document contents
for each word w in value:
EmitIntermediate(w, "1"):
```

```
reduce(String key, Iterator values):
// key: a word
// values: a list of counts
int result    0;
for each v in values:
result      ParseInt(v);
Emit(AsString(result));
```

```
Map (Document Name, Content)  →  For each (Word)
                                  n    (Word, 1)
```

```
For each (Word)
Reduce (Word, List(1)) →  (Word, Sum (n))
```

Another Example:

```
map(String input_key, String input_value):
  // input_key: document name
  // input_value: document contents
  for each word w in input_value:
    EmitIntermediate(w, "1");
```

<“Sam”, “1”>, <“Apple”, “1”>, <“Sam”, “1”>, <“Mom”, “1”>,
<“Sam”, “1”>, <“Mom”, “1”>,

```
reduce(String output_key, Iterator intermediate_values):
  // output_key: a wo
  // output_values: a list of counts
  int result = 0;
  for each v in intermediate_values:
    result += ParseInt(v);
  Emit(AsString(result));
```

<“Sam” , [“1”,”1”,”1”]>, <“Apple” , [“1”]>, <“Mom” , [“1”, “1”] >

“3”
“1”
“2”

1. Write a basic wordcount program.

Sample Pseudocode:

Mapper:

```
void Map (key, value)
{
        for each word x in value:
                emit(x, 1);
}
```

Reducer:

```
void Reduce (keyword, <list_val>)
{
        for each x in <list_val>:
                sum+=x;
                emit(keyword, sum);
}
```

Python Programs:

```
#!/usr/bin/env python
"""mapper.py"""


import sys


# input comes from STDIN (standard input)
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()
    # increase counters
    for word in words:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        #
        # tab-delimited; the trivial word count is 1
        print '%s\t%s' % (word, 1)
```

```
#!/usr/bin/env python
"""reducer.py"""


from operator import itemgetter
import sys
```

```
current_word = None
current_count = 0
word = None

# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()

    # parse the input we got from mapper.py
    word, count = line.split('\t', 1)

    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        # count was not a number, so silently
        # ignore/discard this line
        continue

    # this IF-switch only works because Hadoop sorts map output
    # by key (here: word) before it is passed to the reducer
    if current_word == word:
        current_count += count
    else:
        if current_word:
            # write result to STDOUT
            print '%s\t%s' % (current_word, current_count)
        current_count = count
        current_word = word
# do not forget to output the last word if needed!
if current_word == word:
    print '%s\t%s' % (current_word, current_count)
```

Test your code locally:

```
# Test mapper.py and reducer.py locally first
# very basic test (using only mapper.py)
hduser@ubuntu:~$ echo "a a a a v v f f hh hh fg tg fg  gt nnn ccc ddd nnn ddd"|python3 mapper.py
a	1
a	1
a	1
a	1
v	1
v	1
f	1
f	1
hh	1
hh	1
fg	1
tg	1
fg	1
gt	1
nnn	1
ccc	1
ddd	1
nnn	1
```

```
ddd         1

# very basic test (using mapper.py and reducer.py)

hduser@ubuntu:~$ echo "a a a a v v f f hh hh fg tg fg  gt nnn ccc ddd nnn ddd"|python3
mapper.py|python3 reducer.py

a           4

v           2

f           2

hh          2

fg          1

tg          1

fg          1

gt          1

nnn         1

ccc         1

ddd         1

nnn         1

ddd         1

# very basic test (use mapper.py , sort the output and  use reducer.py)

hduser@ubuntu:~$ echo "a a a a v v f f hh hh fg tg fg  gt nnn ccc ddd nnn ddd"|python3
mapper.py|sort|python3 reducer.py

a           4

ccc         1

ddd         2

f           2

fg          2

gt          1

hh          2

nnn         2

tg          1
```

```
v          2
```

# very basic test (use mapper.py , sort the output and use reducer.py) and write it to text file)

```
hduser@ubuntu:~$ echo "a a a a v v f f hh hh fg tg fg  gt nnn ccc ddd nnn ddd"|python3 mapper.py|sort|python3 reducer.py > out.txt

hduser@ubuntu:~$ cat out.txt
```

To extract words from any dataset or any file…. (use the proper path of file in the command)

```
hduser@ubuntu:~$  cat /home/xxx/Desktop/HR.txt | python3 mapper.py | sort | python3 reducer.py > out_HR.txt

shduser@ubuntu:~$ cat out_HR.txt
```

Exercise 1: Try the above word count program for the Heart Disease dataset, covid_19_data dataset, example dataset and German Credit dataset.

Students can decide their own way of displaying results (can work on any columns in the dataset) on the dataset mentioned.

---

2. MapReduce program to find frequent words

freqmap1.py

```python
#!/usr/bin/env python
# A basic mapper function/program that
# takes whatever is passed on the input and
# outputs tuples of all the words formatted
# as (word, 1)
from __future__ import print_function
import sys

# input comes from STDIN (standard input)
for line in sys.stdin:

    # create tuples of all words in line
    L = [ (word.strip().lower(), 1 ) for word in line.strip().split() ]

    # increase counters
    for word, n in L:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        #
        # tab-delimited; the trivial word count is 1
        print( '%s\t%d' % (word, n) )
```

freqred1.py

```
#!/usr/bin/env python
# reducer.py
from __future__ import print_function
import sys

lastWord = None
sum = 0

for line in sys.stdin:
      word, count = line.strip().split('\t', 1)
      count = int(count)

      if lastWord==None:
          lastWord = word
          sum = count
          continue

      if word==lastWord:
          sum += count
      else:
          print( "%s\t%d" % ( lastWord, sum ) )
          sum = count
          lastWord = word

# output last word
if lastWord == word:
    print( '%s\t%s' % (lastWord, sum ) )
```

freqmap2.py

```
#!/usr/bin/env python
# A basic mapper function/program that
# takes whatever is passed on the input and
# outputs tuples of all the words formatted
# as (word, 1)
from __future__ import print_function
import sys

# input comes from STDIN (standard input)
for line in sys.stdin:

    word, count = line.strip().split('\t', 1)
    count = int(count)
    print( '%d\t%s' % (count, word) )
```

---

freqred2.py

```
#!/usr/bin/env python
# reducer.py
from __future__ import print_function
import sys

mostFreq = []
currentMax = -1

for line in sys.stdin:
      count, word = line.strip().split('\t', 1)
      count = int(count)
      if count > currentMax:
            currentMax = count
            mostFreq = [ word ]
      elif count == currentMax:
            mostFreq.append( word )

# output mostFreq word(s)
for word in mostFreq:
    print( '%s\t%s' % ( word, currentMax ) )
```

```
hduser@ubuntu:~$ echo "foo foo foo labs labs labs quux labs foo bar quux" | python3
freqmap1.py |sort|python3 freqred1.py

bar     1

foo     4
```

```
labs    4

quux    2



hduser@ubuntu:~$ echo "foo foo foo labs labs labs quux labs foo bar quux" | python3
freqmap1.py |sort|python3 freqred1.py|python3 freqmap2.py

1       bar

4       foo

4       labs

2       quux

hduser@ubuntu:~$ echo "foo foo foo labs labs labs quux labs foo bar quux" | python3
freqmap1.py |sort|python3 freqred1.py|python3 freqmap2.py|sort

1       bar

2       quux

4       foo

4       labs

hduser@ubuntu:~$ echo "foo foo foo labs labs labs quux labs foo bar quux" | python3
freqmap1.py    |sort|python3    freqred1.py|python3    freqmap2.py|sort|python3
freqred2.py

foo     4

labs    4
```

Exercise 2: Try the above frequent word count program for the Heart Disease dataset, covid_19_data dataset, example dataset and German Credit data.

Students can decide their own way of displaying results (can work on any columns in the dataset) on the dataset mentioned.

---

3. MapReduce program to explore the dataset and perform the filtering (typically creating key/value pairs) by mapper and perform the count and summary operation on the instances.

Itemmap.py

#!/usr/bin/python

"""

INPUT: Transactions of products in multiple stores and location; these can also be passed to STDIN

```
        Format of each line is: date\ttime\tstore location\titem description\tcost\tmethod of payment
OUTPUT: E.g.
            Las Vegas       208.97
            Miami   84.11
            Tucson  489.93
            San Francisco   388.3
            Dallas  145.63
            Tampa   353.23
            Washington      481.31
            San Jose        492.8
            Newark  410.37
            Memphis 354.44
            Jersey City     369.07
            Plano   4.65
            Buffalo 337.35
            Louisville      213.64
            Miami   154.64
            ...
"""
#import string
import fileinput


for line in fileinput.input():
    data = line.strip().split("\t")
    if len(data) == 6:
        date, time, location, item, cost, payment = data
        print ("{0}\t{1}".format(location, cost))
```

```
#can try with different instances.....
                #print ("{0}\t{1}".format(payment, cost))
        #print ("{0}\t{1}".format(item, cost))
```

itemred.py

```
#!/usr/bin/python

"""
INPUT: Output from mapper.py
        Format of each line is: location\tcost
OUTPUT: E.g.
            50  12268.16
"""
import fileinput

transactions_count = 0
sales_total = 0

for line in fileinput.input():

    data = line.strip().split("\t")

    if len(data) != 2:
        # Something has gone wrong. Skip this line.
        continue

    current_key, current_value = data
    transactions_count += 1

    sales_total += float(current_value)


print (transactions_count, "\t", sales_total)
```

Mapper Output:

```
hduser@ubuntu: ~ $ cat /home/shanthi/Desktop/example.txt | python3 itemmap.py| sort

Atlanta     189.22

Aurora      82.38

Austin      48.09

Birmingham          1.64

Boston      397.21

Buffalo     337.35

Buffalo     386.56

Chicago     364.53

Chicago     431.73

Cincinnati          129.6

Cincinnati          1.41

Cincinnati          288.32

Cincinnati          443.78

Corpus Christi      157.91

…….
```

Reducer Output:

```
hduser@ubuntu:~$ cat /home/shanthi/Desktop/example.txt | python3 itemmap.py |sort| python3 itemred.py

50          12268.159999999996      # displayed total instances and its sum
```

Exercise 3: Try the above ‘Item explore and count program’ for the Heart Disease dataset, covid_19_data dataset, example dataset and German Credit dataset.

Students can decide their own way of displaying results (can work on any columns in the dataset) on the dataset mentioned.

---

4. Write a mapper and reducer program for word count by defining separator instead of using “\t”.

sepmap.py

```
#!/usr/bin/env python

"""A more advanced Mapper, using Python iterators and generators."""


import sys


def read_input(file):

    for line in file:

        # split the line into words

        yield line.split()


def main(separator='\t'):

    # input comes from STDIN (standard input)

    data = read_input(sys.stdin)

    for words in data:

        # write the results to STDOUT (standard output);

        # what we output here will be the input for the

        # Reduce step, i.e. the input for reducer.py

        # tab-delimited; the trivial word count is 1

        for word in words:

            print ('%s%s%d' % (word, separator, 1))
```

```
if __name__ == "__main__":
    main()


sepred.py


#!/usr/bin/env python
"""A more advanced Reducer, using Python iterators and generators."""

from itertools import groupby
from operator import itemgetter
import sys

def read_mapper_output(file, separator='\t'):
    for line in file:
        yield line.rstrip().split(separator, 1)

def main(separator='\t'):
    # input comes from STDIN (standard input)
    data = read_mapper_output(sys.stdin, separator=separator)
    # groupby groups multiple word-count pairs by word,
    # and creates an iterator that returns consecutive keys and their group:
    #   current_word - string containing a word (the key)
    #   group - iterator yielding all ["&lt;current_word&gt;", "&lt;count&gt;"] items
    for current_word, group in groupby(data, itemgetter(0)):
        try:
            total_count = sum(int(count) for current_word, count in group)
            print ("%s%s%d" % (current_word, separator, total_count))
        except ValueError:
            # count was not a number, so silently discard this item
            pass


if __name__ == "__main__":
    main()
```

Note:

Why Separator ?

Eg:

```
print('G','F', sep='', end='')print('G')

#\n provides new line after printing the year

print('09','12','2016', sep= '-', end='\n')
print('prtk','agarwal', sep= '  ' , end='@')
print('manipal')
```

Output:

```
GFG
09-12-2016
prtkagarwal@manipal
```

Output:

```
hduser@ubuntu:~$ echo " Time is gold Time is  Time  gold" | python3 sepmap.py|python3 sepred.py

Time   ->1

is       ->1

gold    ->1

Time   ->1

is       ->1

Time   ->1

gold    ->1

hduser@ubuntu:~$ echo " Time is gold Time is  Time  gold" | python3 sepmap.py|sort|python3 sepred.py
```

```
gold    ->2

is      ->2

Time    →3
```

Exercise 4: Try to include separator using map reducing for the output of Heart Disease dataset, covid_19_data dataset, example dataset and German Credit dataset.

---

5. Write a map reduce program that returns the cost of the item that is most expensive, for each location in the dataset example.txt

itemmap_expensive.py

```
#!/usr/bin/python

“""

INPUT: Transactions of products in multiple stores and location; these can also be passed to STDIN

        Format of each line is: date\ttime\tstore location\titem description\tcost\tmethod of payment

OUTPUT: E.g.

            Las Vegas       208.97

            Miami   84.11

            Tucson  489.93

            San Francisco   388.3

            Dallas  145.63

            Tampa   353.23

            Washington      481.31

            San Jose        492.8

            Newark  410.37

            Memphis 354.44

            Jersey City     369.07

            Plano   4.65
        Buffalo 337.35

        Louisville      213.64

        Miami   154.64

        ...

"""


import fileinput


for line in fileinput.input():

    data = line.strip().split("\t")

    if len(data) == 6:

        date, time, location, item, cost, payment = data

        print ("{0}\t{1}".format(location, cost))
```

itemred_expensive.py

```
#!/usr/bin/python


"""

INPUT: Output from mapper.py

        Format of each line is: location\tcost

OUTPUT: E.g.

        Atlanta     189.22

        Aurora  82.38

        Austin  48.09

        Birmingham  1.64
        Boston  397.21

        Buffalo     386.56

"""

import fileinput


max_value = 0

old_key = None


for line in fileinput.input():


    data = line.strip().split("\t")


    if len(data) != 2:

        # Something has gone wrong. Skip this line.

        continue


    current_key, current_value = data


    # Refresh for new keys (i.e. locations in the example context)

    if old_key and old_key != current_key:

        print (old_key, "\t", max_value)

        old_key = current_key

        max_value = 0


    old_key = current_key

    if float(current_value) > float(max_value):
        max_value = float(current_value)


if old_key != None:

    print (old_key, "\t", max_value)
```

Output:

```
hduser@ubuntu:~$ cat /home/shanthi/Desktop/example.txt | python3 itemmap1.py|sort

Atlanta189.22

Aurora 82.38

Austin 48.09

Birmingham    1.64

Boston 397.21

Buffalo         337.35

Buffalo         386.56    # selects max value

Chicago         364.53

Chicago         431.73   …… # selects max value



hduser@ubuntu:~$ cat /home/shanthi/Desktop/example.txt | python3 itemmap1.py|sort|python3 itemred1.py

Atlanta         189.22

Aurora          82.38

Austin  48.09

Birmingham      1.64

Boston          397.21

Buffalo         386.56   # selected max value

Chicago         431.73   # selected max value
```

Exercise 5: Try to apply finding max value using map reduce concept for the output of Heart Disease dataset, covid_19_data dataset, example dataset and German Credit dataset.

Students can decide their own way of displaying results (can work on any columns in the dataset) on the dataset mentioned.

---

6. Write a mapreduce program to evaluate the PI.

mapper_pi.py

```
!/usr/bin/env python
import sys

def f( x ):
    return 4.0 / ( 1.0 + x*x )


# input comes from STDIN (standard input)
for line in sys.stdin:

    # remove leading and trailing whitespace
    line = line.strip()

    # split the line into words
    words = line.split()
    N = int( words[0] )
    deltaX = 1.0 / N

    for i in range( 0, N ):
        print( "1\t%1.10f" % ( f( i * deltaX )*deltaX ) )
```

reducer_pi.py

```
#!/usr/bin/env python
from __future__ import print_function
from operator import itemgetter

import sys

sum = 0
```

```
# input comes from STDIN
for line in sys.stdin:
      # remove leading and trailing whitespace
      line = line.strip()

      # parse the input we got from mapper.py
      word, count = line.split('\t', 1)

      # convert count (currently a string) to int
      try:
          count = float(count)
      except ValueError:
          # count was not a number, so silently
          # ignore/discard this line
          #print( "--skipping (%s, %s)" % ( str(word), str(count) ) )
          continue

      sum += count

# do not forget to output the last word if needed!
print( '%1.10f\t0' % sum )
```

```
hduser@ubuntu:~$ echo "5" | python3 pimap.py

1        0.8000000000

1        0.7692307692

1        0.6896551724

1        0.5882352941

1        0.4878048780

shanthi@shanthi:~$ echo "5" | python3 pimap.py|python3 pired.py

3.3349261137        0

hduser@ubuntu:~$ echo "3" | python3 pimap.py

1        1.3333333333

1        1.2000000000

1        0.9230769231

hduser@ubuntu:~$ echo "3" | python3 pimap.py|python3 pired.py
```

3.4564102564	**0**

Exercise 6: Write a MapReduce program to generate a report with Number of males, females and total births in each year, number of males, females and total births in each month of a particular year from national birth data.

Exercise 7: Write a MapReduce program to count even or odd numbers in randomly generated natural numbers

*************************************All The Best ************************************