

DISTRIBUTED SYSTEMS LABORATORY

SEMESTER – VI - CSE3261

CONTENTS

LAB NO	TITLE
1	PYTHON BASIC PRACTICE - I
2	PYTHON BASIC PRACTICE - II
3	PYTHON BASIC PRACTICE - III
4	SOCKET PROGRAMMING USING PYTHON
5	MAP REDUCE PROGRAMS AND PRACTICE
6	CLOCK SYNCHRONIZATION
7	MUTUAL EXCLUSION (ELECTION ALGORITHM)
8	HADOOP INSTALLATION
9	PROGRAM EXECUTION ON HADOOP -I
10	PROGRAM EXECUTION ON HADOOP-II
11	MINI PROJECT
12	MINI PROJECT

LAB – 1
PYTHON BASIC PRACTICE - I

1. Assigning Values to Variables

counter = 100 # An integer assignment

miles = 1000.0 # A floating point

name = "John" # A string

print (counter)

print (miles)

print (name)

2. Multiple Assignment

Python allows you to assign a single value to several variables simultaneously.

For example:

a = b = c = 1

a, b, c = 1, 2, "john"

Standard Data Types

Python has five standard data types:

☐ Numbers

☐ String

☐ List

☐ Tuple

☐ Dictionary

Python Numbers

a = 5 # integer assignment

b = 4.56 #floating point assignment

#mathematical operations with scalar variables

print (5*a), would give the result 25

print (a/2) , would give the result 2.5

`print(a**2)`, is the power (squaring operation) would give the result 25

import numpy as np

numpy includes various scientific functions like log 10, log2, natural log, exp, floor, ceil, rounding and all statistical summary functions.

Python Strings

```
str = 'Hello World!'
print (str) # Prints complete string
print (str[0]) # Prints first character of the string
print (str[2:5]) # Prints characters starting from 3rd to 5th
print (str[2:]) # Prints string starting from 3rd character
print (str * 2) # Prints string two times
print (str + "TEST") # Prints concatenated string
```

1. Updating a string

```
var1 = 'Hello World!'
print ("Updated String :", var1[:6] + 'Python')
```

Ans: Updated String :Hello Python

2. String formatting operator

One of Python's coolest features is the string format operator `%`. This operator is unique to strings and makes up for the pack of having functions from C's `printf()` family. Following is a simple example:

```
Print( "My name is %s and weight is %d kg!" % ('Anitha', 55))
```

My name is Anitha and weight is 55 kg!

3. Built-in String methods

`capitalize()`, the first character of the string is converted to upper case.

```
str = "this is string example....wow!!!";
```

```
print (str.capitalize())
```

Ans: This is string example....wow!!!

count(), counts the number of times a specific 'substring', occurrence in the main string

```
str = "this is string example....wow!!!";
```

```
str.count('s')
```

Ans: 3 # three times 's', appears in str.

find(), will locate the position of searching 'substring', (index)

```
Str.find('example')
```

Ans: 15 #at 15th index, the substring 'example' is placed.

lower(), returns a copy of the string in which all case-based characters have been lowercased.

```
str = "THIS IS STRING EXAMPLE....WOW!!!";  
print (str.lower())
```

this is string example....wow!!!

replace(), this method returns a copy of the string with all occurrences of substring old replaced by new.

```
str = "this is string example....wow!!! this is really string";  
print (str.replace("is", "was"))
```

Ans: thwas was string example....wow!!! thwas was really string

swapcase(), this method returns a copy of the string in which all the case-based characters have had their case swapped.

```
str = "this is string example....wow!!!";  
print (str.swapcase())
```

Ans: THIS IS STRING EXAMPLE....WOW!!!

title(), returns a copy of the string in which first characters of all the words are capitalized.

```
str = "this is string example....wow!!!";  
print (str.title())
```

Ans: This Is String Example....Wow!!!

Python LIST

Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]). To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.

The values stored in a list can be accessed using the slice operator ([] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator. For example:

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylst = [123, 'john']
print (list) # Prints complete list
print (list[0]) # Prints first element of the list
print (list[1:3]) # Prints elements starting from 2nd till 3rd
print (list[2:]) # Prints elements starting from 3rd element
print (tinylst * 2) # Prints list two times
print (list + tinylst) # Prints concatenated lists
```

Output will be like this

```
['abcd', 786, 2.23, 'john', 70.200000000000003]
abcd
[786, 2.23]
[2.23, 'john', 70.200000000000003]
[123, 'john', 123, 'john']

['abcd', 786, 2.23, 'john', 70.200000000000003, 123, 'john']
```

Note: LIST is mutable data type

Functions & Methods in LIST

```
list = ['physics', 'chemistry', 1997, 2000];
```

```
list.append('maths')
```

```
Ans: ['physics', 'chemistry', 1997, 2000, 'maths'];
```

To delete an element in a list

`del list[2]`, will remove 1997 record from the list.

To check a data in a list,

`'physics' in list`, will return `'True'`

`'english' in list`, will return `'False'`

`len(list)`, will return total items in list

`list.count('physics')`

Ans: 1

`list.pop()`, will remove and return the last item from the list.

`list = ['physics', 'chemistry', 1997, 2000];`

`list.pop()`

Ans: `['physics', 'chemistry', 1997]`

`list.insert()`, will insert an item in the specified index

`list = ['physics', 'chemistry', 1997, 2000];`

`list.insert(2, 'maths')`

Ans: `['physics', 'chemistry', 'maths', 1997, 2000];`

`list = ['physics', 'chemistry', 1997, 2000];`

`list.remove('chemistry')`, will remove the item specified.

Ans: `['physics', 1997, 2000];`

`list = ['physics', 'chemistry', 1997, 2000];`

`list.reverse()`, will reverse the objects of the list in place.

Ans: `[2000, 1997, 'chemistry', 'physics']`

Python TUPLE

A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.

The main differences between lists and tuples are: Lists are enclosed in brackets ([]) and their elements and size can be changed, while tuples are enclosed in parentheses (()) and cannot be updated. Tuples can be thought of as **read-only** lists.

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tuple[2] = 1000 # Invalid syntax with tuple
list[2] = 1000 # Valid syntax with list
```

Looping & Conditional Branches in Python

Eg.1

```
num=float(input('Enter a number:'))
if num>0:
    print('pos number')
elif num==0:
    print('zero')
else:
    print('Neg number')
```

Eg.2

```
x=float(input('Enter a number:'))
if x<10:
    print('smaller')
if x>20:
    print('bigger')
print('Finished')
```

Eg.3

```
x=5
print('Before 5')
if x==5:
    print ('this is 5')
    print('still 5')
print('After 5')
print('Before 6')
if x==6:
    print('this is 6')
```



```
print ('After 6')
```

Eg.4: which will never print?

```
x=float(input('Enter a number:'))
if x<20:
    print('Below 20')
elif x<10:
    print('Below 10')
else:
    print('something else')
```

Ans: Below 10

Eg.5: Nested Decisions

```
x=42
if x>1:
    print('above one')
    if x<100:
        print('less than 100')
print('All done')
```

Eg.6: Ternary operator

```
age=15
b=('kid' if age<18 else 'adult')
print(b)          #this will print 'kid'
```

Usage of For-loop

Eg.1

```
for val in [5,4,3,2,1]:
    print(val)
print ('Done')
```

Eg.2

```
stud=['Ram','Vijay','Nithya','Anu','Ramesh','suja']
for k in stud:
    print('Hello:', k)
print('done')
```

Eg.3

```

for i in range(5):
    print(i)
    If i>2:
        print('Bigger than 2')
    print('Done with i',i)

```

Eg.4: Calculate factors of a number

```

x=int(input('Enter a number:'))
for i in range(1,x+1):
    if x%i ==0:
        print(i)
x=10
1,2,5,10

```

Eg.5: Calculate largest number in an array

```

from math import *
Let x= [9, 41, 12, 3, 74, 15]
Largest=-inf
for i in x:
    if i>Largest:
        Largest=i
Print(Largest)

```

Eg.6: Calculate smallest number in an array

```

from math import *
Let x= [9, 41, 12, 3, 74, 15]
smallest=inf
for i in x:
    if i<smallest:
        smallest=i
Print(smallest)

```

Eg.7: Calculate the count, sum and average of numerical array

```

Let x= [9, 41, 12, 3, 74, 15]

count=sum=avg=0

for i in x:
    count=count+1
    sum=sum+i
avg=sum/count
print(count)
print(sum)
print(avg)

```

Eg.8: Filtering in a loop (print all numbers >20)

Let x= [9, 41, 12, 3, 74, 15]

```
for i in x:
    if i>20:
        print (i)
```

Eg.9: For the above problem, instead of printing the result, store the elements in a variable (object)

Let x= [9, 41, 12, 3, 74, 15]

```
res=[]
for i in x:
    if i>20:
        res.append(i)
```

Eg.10: For the above x, replace all elements <20 into zero. Store the result in different variable (object)

```
y=np.zeros(len(x))
for i in range(len(x)):
    if x[i]>20:
        y[i]=x[i]
print(y)
```

LAB – 2

PYTHON BASIC PRACTICE - II

Introduction to PANDAS

```
import pandas as pd
import numpy as np
PANDAS : Series data structure
s=pd.Series([3,9,-2,10,5])
s.sum()
25
s.min()
-2
s.max()
10
```

Creating a Data Frame

```
import pandas as pd
data = [['Dinesh',10],['Nithya',12],['Raji',13]]
df = pd.DataFrame(data,columns=['Name','Age'])
Name Age
0 Dinesh 10
1 Nithya 12
2 Raji 13
```

Indexed Data Frame

```
data = {'Name':['Kavitha', 'Sudha', 'Raju','Vignesh'],'Age':[28,34,29,42]}
df = pd.DataFrame(data, index=['rank1','rank2','rank3','rank4'])
Age Name
rank1 28 Kavitha
rank2 34 Sudha
rank3 29 Raju
rank4 42 Vignesh
```

Creating a DataFrame using Dictionary

```
df1=pd.DataFrame({'A':pd.Timestamp('20130102'),'B':np.array([3]*4,dtype='int32'),  
'C':pd.Categorical(['Male','Female','Male','Female'])})
```

```
A          B      C  
0 2013-01-02 3 Male  
1 2013-01-02 3 Female  
2 2013-01-02 3 Male  
3 2013-01-02 3 Female
```

```
df1.shape  
(4,3)
```

```
df1.dtypes  
A datetime64  
B int32  
C category  
dtype: object
```

```
df1.head()  
#will display first 5 records
```

```
df1.tail()  
#will display last 5 records
```

```
df.summary()
```

	A	B	C	D
count	6.000000	6.000000	6.000000	6.000000
mean	-0.175649	-0.208576	0.019053	-0.145193
std	1.286893	1.446490	0.678454	0.722313
min	-2.567050	-1.968611	-0.782803	-0.962028
25%	-0.345430	-1.435541	-0.472918	-0.609707
50%	0.174139	0.179390	-0.058210	-0.306675
75%	0.447666	0.557708	0.519626	0.230624
max	1.144652	1.648411	0.912454	1.005213

```
df.T    # will transpose the data frame
```

```
import pandas as pd  
import numpy as np
```

```
dates=pd.date_range('20130101', periods=100)
```

```
df = pd.DataFrame(np.random.randn(100,4), index=dates, columns=list('ABCD'))
```

To view first 5 records

```
>>> df.head()
```

To view last 5 records

```
>>> df.tail()
```

To view the index

```
>>> df.index
```

To view the column names

```
>>> df.columns
```

To transpose the df

```
>>> df.T
```

Sorting by Axis

```
>>> df.sort_index(axis=1, ascending=False)
```

Sorting by Values

```
>>> df.sort_values(by='B')
```

Slicing the rows

```
>>> df[0:3], which slice first 3 records (rows)
```

Slicing with index name

```
>>> df['20130105':'20130110']
```

Slicing with row and column index (like 2D Matrix)

```
>>> df.iloc[0], will fetch entire 1st row
```

```
>>> df.iloc[0,:2], will fetch 1st row, first 2 columns
```

```
>>> df.iloc[0,0], will fetch 1st row, 1st column element (single element)
```

Selecting a single column

```
>>> df['A'], which yields a Series
```

Selecting more than one column

```
>>> df[['A','B']], entire 2 columns
```

Selecting more than one column, with selected number of records

```
>>> df[['A','B']][:5], first 5 records
```

[OR]

```
>>> df.loc['20130101':'20130105',['A','B']][:5], first 5 records
```

Boolean Indexing

df[df.A>0], will fetch all positive values of A column

Include a 6th column (a categorical) character data

```
df['F']=['Male','Female','Female','Male','Female','Female']
```

Setting by assigning with a numpy array

```
df.loc[:, 'D']=np.array([5]*len(df))
```

Which will replace the 'D', column with all 5

Deleting a row or column

```
df.drop ('col_name', axis =1, inplace=True)
```

will drop the column name specified in col_name

```
df.drop ('row_index', axis =0, inplace=True)
```

will drop the row specified in row_index

Concatenation of two Data Frames

Let df1 be of size 10 x 5 and df2 of size 10 x 3 if concatenated horizontally (as a new column insertion)

```
Df_new= pd.concat (df1, df2, axis=1)
```

```
Df_new.shape
```

```
10 x 8
```


Let A dataframe of size 10 x 5 and B dataframe of size 15 x 5 if concatenated Vertically (as a new row insertion)

```
D= pd.concat (A, B, axis=0)
```

```
D.shape
```

```
25 x 5
```

**** Sorting and Ordering a DataFrame:****

For the given DataFrame let us sort the age column

Let the dataframe be "A"

	Age	Name
rank1	28	Kavitha
rank2	34	Sudha
rank3	29	Raju
rank4	42	Vignesh

```
A.sort_values(by = 'Age')
```

	Age	Name
rank1	28	Kavitha
rank3	29	Sudha
rank2	34	Raju
rank4	42	Vignesh

I/O Operations with external files

Reading a CSV file & XLS file format

```
df = pd.read_csv('xyz.csv',header=None)
```

```
df.head() #This will display 1st 5 records
```

```
df.tail() #This will display last 5 records
```

The above dataset doesn't have header, we shall attach our own header.

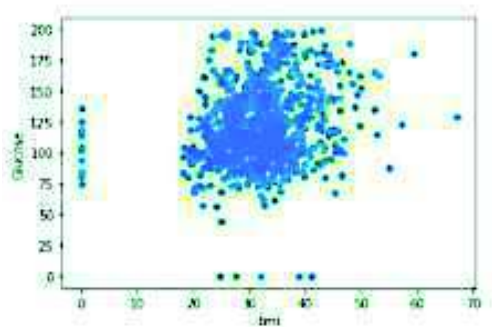
```
df.columns=['preg','glu','bp','sft','ins','bmi','dpf','age','class']
```

#Let us visualize the scatter plot of two continuous variable.

```
plt.scatter(df['bmi'],df['glu'])
```

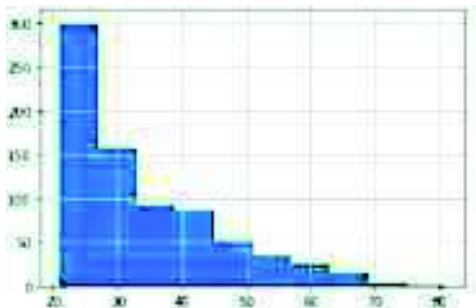
```
plt.xlabel('bmi')
```

```
plt.ylabel('Glucose')
```

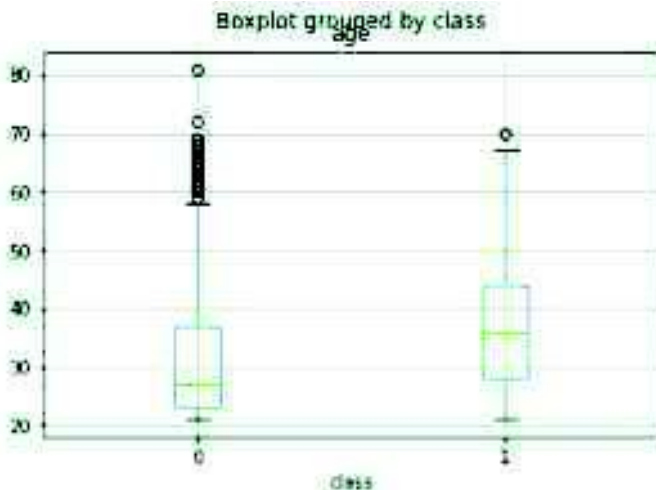


#Let us visualize the histogram of another continuous variable 'Age'

```
df['age'].hist()
```



#Let us visualize the distribution 'Age' with respect to Categories: Label-0(Healthy), Label-1 (Diabetes)



#We can observe the median age of diabetes is slightly greater than median age of healthy

```
W = pd.read_csv('xyz.xls',header=None)
```

W.head() **#XLS file format also, we can read using pd.read_csv**

```
D= np.loadtxt('xyz.data',delimiter=",")
```

D[:5,:] **# this file is loaded in Numpy 2D array format**

Reading a XLSX file format

```
G=pd.read_excel('xyz.xlsx',sheet_name='Sheet1')
```

```
G.head()
```

Here additionally we need to pass the sheet_name. If not specified, it will read the first page by default.

Reading a HTML file format

Pandas can read table tabs off of html. For example:

```
B = pd.read_html('http://www.fdic.gov/bank/individual/failed/banklist.html')
```

Reading a TXT file format

```
H = pd.read_table('HR.txt')
```

```
H.head()
```

```
f=H["Department"].value_counts()
```

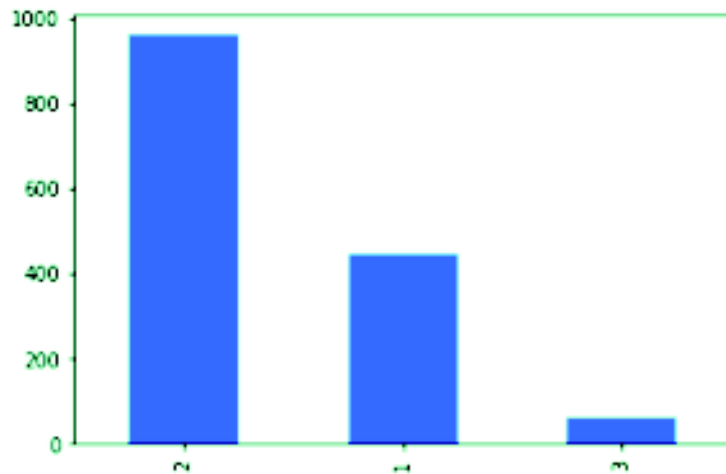
```
f
```

```
2 961
```

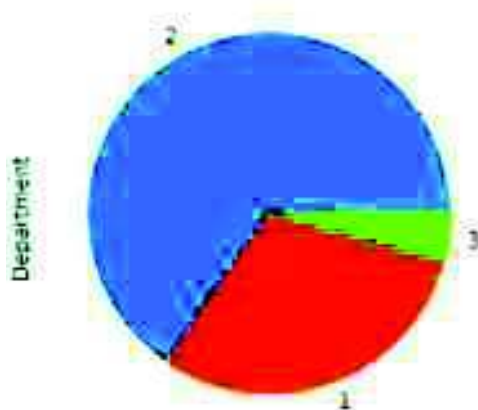
```
1 446
```

```
3 63
```

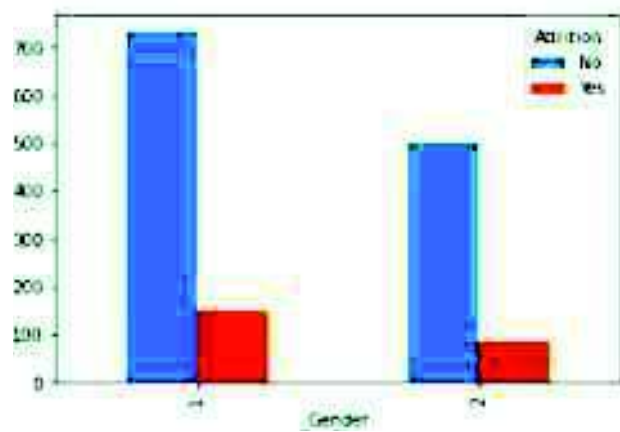
#We can visualize the distribution of categorical values using
bar plot and pie chart
f.plot(kind='bar')



The above bar plot can be perceived in terms of Pie chart
which would give % percentage information
f.plot(kind='pie')



#We can visualize two categorical variables at a time
fa=pd.crosstab(H['Gender'],H['Attrition'])
fa.plot(kind='bar')



Lab Exercise Programs for Week 1 & Week 2:

1. Write a program to find the area of rectangle. Take input from user.
Eg. `x= int(input('Enter number:'))`
2. Write a program to swap the values of two variables.
3. Write a program to find whether a number is even or odd.
4. Write a program to check the largest among the given three numbers.
5. Write a program to demonstrate while loop with else.
6. Write a program to demonstrate List functions and operations.
7. Consider the tuple(1,3,5,7,9,2,4,6,8,10). Write a program to print half its values in one line and the other half in the next line.
8. Consider the tuple (12, 7, 38, 56, 78). Write a program to print another tuple whose values are even number in the given tuple.
9. *Write a Python program to print negative Numbers in a List using for loop.* Eg. [11, -21, 0, 45, 66, -93].
10. Write a program to print negative Numbers in a List using while loop.
11. Write a Python program to count positive and negative numbers in a List.
12. Write a Python program to remove all even elements from a list .
13. Define a dictionary containing Students data {Name, Height, Qualification}.
 - a) Convert the dictionary into DataFrame
 - b) Declare a list that is to be converted into a new column (Address)
 - c) Using 'Address' as the column name and equate it to the list and display the result.
14. Define a dictionary containing Students data {Name, Height, Qualification}.
 - a) Convert the dictionary into DataFrame
 - b) Use DataFrame.insert() to add a column and display the result.

LAB – 3

PYTHON BASIC PRACTICE - III

Usage of Numpy Data Structure

Import numpy as np

1. Array creation

```
A = np.array ([2,5,10])
```

A.dtype

Will show, int64 data type

```
B=np.array ([2.4,10.6,5.2])
```

B.dtype

Will show, float64 data type

Creating sequence of sequence will create 2-dimensional array.

```
A=np.array([(3,4,5),(12,6,1)])
```

```
Z=np.zeros((2,4)) # will create zero matrix of dimension 2x4
```

Similarly, np.ones((3,3)) # will create one's matrix of dimension 3x3

To create a sequence of data,

```
S=np.arange(10,30,5)
```

Print (S) will give (10,15,20,25,30), with step size of 5

```
np.arange( 0, 2, 0.3 ) # it accepts float arguments
```

```
array([ 0. , 0.3, 0.6, 0.9, 1.2, 1.5, 1.8])
```

#Instead of step-size, we can specify total number of elements in the array using

```
S1=np.linspace(0,2,9) # produce 9 numbers starting 0 & ends with 2
```



```
array([ 0. , 0.25, 0.5 , 0.75, 1. , 1.25, 1.5 , 1.75, 2. ])
```

#usage of Random Number functions

```
import random
```

```
random.choice([1,2,3,4,5]), this will pick one number from the list randomly
```

```
random.choice('python'), will pick one character from the string randomly
```

```
random.randrange(25,50), will pick one integer between 25 to 50
```

```
random.randrange(25,50,2), will pick one integer between 25 to 50 with step size of 2
```

```
random.random(), will pick a random number between 0 to 1
```

```
random.uniform(5,10), will pick a floating point number between 5 to 10
```

```
random.shuffle([1,2,3,4,5]), will shuffle the list elements
```

```
random.seed(10), to get same random value during every execution
```

2-Dimensional array (Matrix)

```
a = np.arange(15).reshape(3, 5)
```

```
array([[ 0,  1,  2,  3,  4],  
       [ 5,  6,  7,  8,  9],  
       [10, 11, 12, 13, 14]])
```

```
#to check the dimension
```

```
a.shape
```

```
(3,5)
```

```
a.size # will return total elements in matrix (here 15)
```

```
# to transpose a matrix
```

```
a.T # transposed to 5x3 matrix
```

3-Dimensional array

```
c = np.arange(24).reshape(2,3,4) # 1st value indicates (no of planes) (3,4) is the dimension
```

```
print(c)
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]
 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
```

c.shape will return (2,3,4)

c[1,...] is equal to c[1,:,:] # will fetch all elements of 2nd plane

c[...,2] is equal to c[:, :,2] [[3,7,11],[15,19,23]]

Array operations

```
a = np.array( [20,30,40,50] )
b = np.arange( 4 )
b
array([0, 1, 2, 3])
c = a-b
c
array([20, 29, 38, 47])
b**2
array([0, 1, 4, 9])
10*np.sin(a)
array([ 9.12945251, -9.88031624,  7.4511316 , -2.62374854])
a<35
array([ True,  True, False, False], dtype=bool)
```

Matrix operations

```
A = np.array( [[1,1],[0,1]] )
B = np.array( [[2,0],[3,4]] )
A*B          # elementwise product
array([[2, 0],
       [0, 4]])
A.dot(B)      # matrix product
array([[5, 4],
       [3, 4]])
```

(OR)

```
np.dot(A, B)  # another matrix product
array([[5, 4],
       [3, 4]])
```

```

b = np.arange(12).reshape(3,4)
b
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])

b.sum(axis=0)          # sum of each column
array([12, 15, 18, 21])

b.sum(axis=1)          # sum of each row
array([ 6, 22, 38])

```

Indexing, Slicing & Iterating Array

```

a = np.arange(10)**3
a
array([ 0,  1,  8, 27, 64, 125, 216, 343, 512, 729])

a[2:5]
array([ 8, 27, 64])

a[0:6:2]
array([0,8,64,216])

```

Let 'b', is an input matrix of size 5x4

```

array([[ 0,  1,  2,  3],
       [10, 11, 12, 13],
       [20, 21, 22, 23],
       [30, 31, 32, 33],
       [40, 41, 42, 43]])

b[2,3] #will fetch 23

b[0:5,1] or b[:5,1] or b[:,1] #will fetch [1,11,21,31,41]

b[-1,:] # will fetch last row
b[:, -1] # will fetch last col

```

```

for row in b:
    print (row) # will print every row

```

for element in b.flat:

print (element) # will show all elements of b in 1-D array

Changing the shape of a matrix

b.ravel() # returns the array flattened to (1x 20)

Later, we can convert 5x4 matrix into 4x 5 matrix using

B1=b.reshape(4,5)

Stacking together different arrays

A1=np.array([(3,4,5),(12,6,1)])

```
3  4  5
12 6  1
```

A2=np.array([(1,2,6),(-4,3,8)])

```
1  2  6
-4  3  8
```

D1=np.vstack((A1,A2))

```
3  4  5
12 6  1
1  2  6
-4  3  8
```

D2=np.hstack((A1,A2))

```
3  4  5  1  2  6
12 6  1 -4  3  8
```

Stacking 1-D array into 2-D array (column wise)

a = np.array([4.,2.])

b = np.array([3.,8.])

np.column_stack((a,b)) # returns a 2D array

```
array([[ 4.,  3.],
       [ 2.,  8.]])
```

```
np.hstack((a,b))      # the result is different
array([ 4., 2., 3., 8.])
```

```
np.hstack((a[:,newaxis],b[:,newaxis])) # the result is the same
array([[ 4., 3.],
       [ 2., 8.]])
```

Indexing with array of indices

```
a = np.arange(12)**2          # the first 12 square numbers
i = np.array([ 1,1,3,8,5 ])   # an array of indices
a[i]                          # the elements of a at the positions i
array([ 1, 1, 9, 64, 25])
```

```
j = np.array([ [ 3, 4], [ 9, 7 ] ]) # a bidimensional array of indices
a[j]                                # the same shape as j
```

```
array([[ 9, 16],
       [81, 49]])
```

Usage of for-loop (Mapping by Value)

Calculate sum of all the elements in a 2D Numpy Array (iterate over elements)

```
import numpy as np
a=np.array([(3,2,9),(1,6,7)])

s1=0
for row in a:
    for col in row:
        s1+=col
print(s1)
```

Usage of for-loop (Mapping by Index)

Calculate sum of all the elements in a 2D Numpy Array (iterate over range)

```
import numpy as np
a=np.array([(3,2,9),(1,6,7)])
s=0
for i in range(a.shape[0]):
    for j in range(a.shape[1]):
        s+=a[i,j]
print(s)
```

Lab Exercise Programs for Week -3:

1. Write a program to find the factors of a given number (get input from user) using for loop.
2. Find the sum of columns and rows using axis.
3. Operations on Arrays (use numpy wherever required):
 - a) Create array from list with type float
 - b) Create array from tuple
 - c) Creating a 3X4 array with all zeros
 - d) Create a sequence of integers from 0 to 20 with steps of 5
 - e) Reshape 3X4 array to 2X2X3 array
 - f) Find maximum and minimum element of array, Row wise max and min, column wise max and min and sum of elements. (Use functions max(), min(), sum())
4. Write a program to transpose a given matrix.
5. Write a program to add two matrices.
6. Write a program to find element wise product between two matrices.

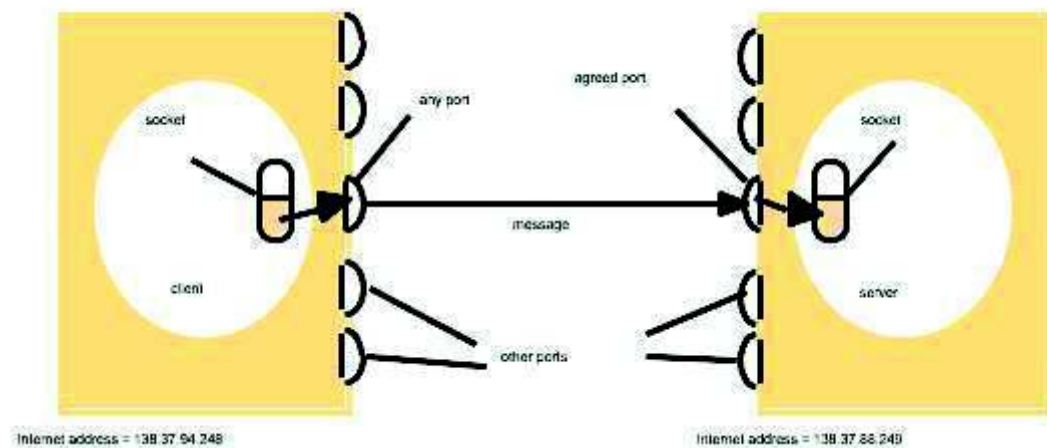
LAB – 4

SOCKET PROGRAMMING USING PYTHON

Socket Programming in Python

Socket Basics:

A *network socket* is an endpoint of an inter-process communication flow across a computer network. Sockets may communicate within a process, between processes on the same machine, or between processes on different continents. Today, most communication between computers is based on the internet protocol; therefore most network sockets are *internet sockets*. To create a connection between machines, Python programs import the **socket** module, create a socket object, and call the object's methods to establish connections and send and receive data. Sockets are the endpoints of a bidirectional communications channel.



Socket in Python:

Python provides two levels of access to network services. At a *low level*, you can access the basic socket support in the underlying operating system, which allows you to implement clients and servers for both connection-oriented and connectionless protocols. Python also has libraries that provide *higher level* access to specific application level network protocols, such as FTP, HTTP, SMTP, and so on. Sockets may be implemented over a number of different channel types: UNIX domain sockets, TCP, UDP, and so on. The socket library provides specific classes for handling the common transports as well as a generic interface for handling the rest.

Vocabulary of Sockets

Term	Description
domain	The family of protocols that will be used as the transport mechanism. These values are constants such as AF_INET , PF_INET , PF_UNIX , PF_X25 , and so on.
type	The type of communications between the two endpoints, typically SOCK_STREAM for connection-oriented protocols and SOCK_DGRAM for connectionless protocols.
protocol	Typically zero, this may be used to identify a variant of a protocol within a domain and type.
hostname	The identifier of a network interface: <ul style="list-style-type: none">• A string, which can be a host name, a dotted-quad address, or an IPV6 address in colon (and possibly dot) notation• A string "<broadcast>", which specifies an INADDR_BROADCAST address.• A zero-length string, which specifies INADDR_ANY, or• An Integer, interpreted as a binary address in host byte order.
port	Each server listens for clients calling on one or more ports. A port may be a Fixnum port number, a string containing a port number, or the name of a service.

The *socket* Module:

To create a socket, you must use the ***socket.socket()*** function available in *socket* module, which has the general syntax:

```
s = socket.socket (socket_family, socket_type, protocol=0)
```

Here is the description of the parameters:

- **socket_family**: This is either **AF_UNIX** or **AF_INET**, as explained earlier.
- **socket_type**: This is either **SOCK_STREAM** or **SOCK_DGRAM**.
- **protocol**: This is usually left out, defaulting to 0.

Once you have *socket* object, then you can use required functions to create your client or server program.

Server Socket Methods

Method	Description
s.bind()	This method binds address (hostname, port number pair) to socket.
s.listen()	This method sets up and start TCP listener.
s.accept()	This passively accept TCP client connection, waiting until connection arrives (blocking).

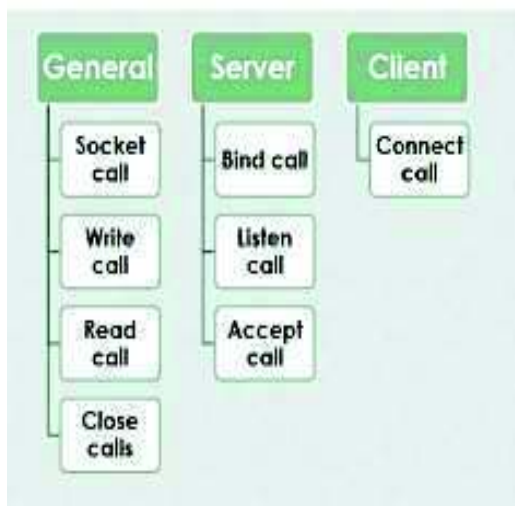
Client Socket Methods

Method	Description
s.connect()	This method actively initiates TCP server connection.

General Socket Methods

Method	Description
s.recv()	This method receives TCP message
s.send()	This method transmits TCP message
s.recvfrom()	This method receives UDP message
s.sendto()	This method transmits UDP message
s.close()	This method closes socket
socket.gethostname()	Returns the hostname.

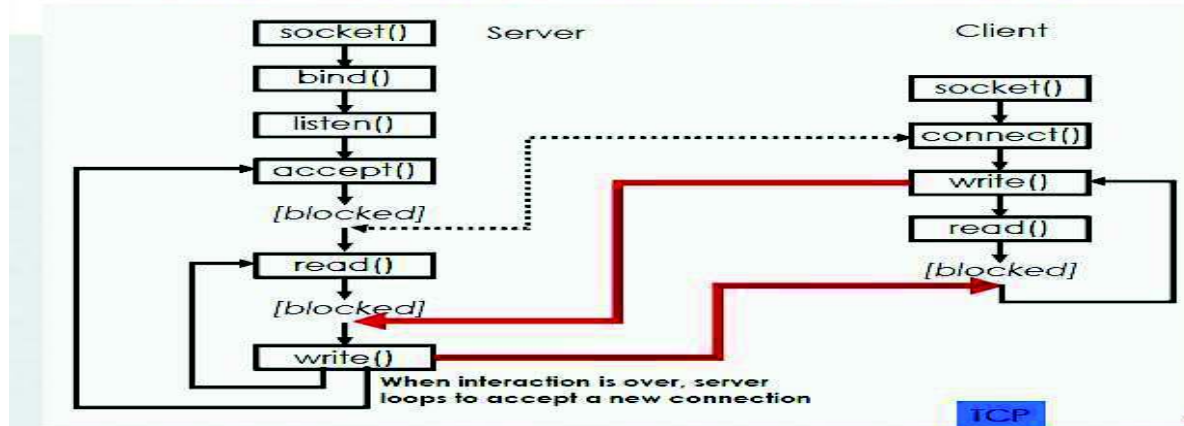
MAJOR SYSTEM CALLS



MAJOR SYSTEM CALLS (SUMMARY)

Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Attach a local address to a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Block the caller until a connection attempt arrives
CONNECT	Actively attempt to establish a connection
SEND	Send some data over the connection
RECEIVE	Receive some data from the connection
CLOSE	Release the connection

CONNECTION-ORIENTED SERVICES



A Simple Server:

To write Internet servers, we use the **socket** function available in socket module to create a socket object.

A socket object is then used to call other functions to setup a socket server. Now call **bind(hostname, port)** function to specify a *port* for your service on the given host. Next, call the *accept* method of the returned object. This method waits until a client connects to the port you specified, and then returns a *connection* object that represents the connection to that client.

```

# Echo server program
import socket
host = socket.gethostname()
port = 12345
s = socket.socket()
s.bind((host, port))
s.listen(5)
conn, addr = s.accept()
print('Got connection from ', addr[0], '(', addr[1], ')')
print('Thank you for connecting')
while True:
    data = conn.recv(1024)
    if not data: break
    conn.sendall(data)
conn.close()
    
```

A Simple Client:

Now we will write a very simple client program which will open a connection to a given port 12345 and given host. This is very simple to create a socket client using Python's *socket* module function.

The `socket.connect(hostname, port)` opens a TCP connection to *hostname* on the *port*. Once you have a socket open, you can read from it like any IO object. When done, remember to close it, as you would close a file.

The following code is a very simple client that connects to a given host and port, reads any available data from the socket, and then exits:

```
# Echo client program
import socket
host = socket.gethostname()
port = 12345
s = socket.socket()
s.connect((host, port))
s.sendall(b'Welcome User!')
data = s.recv(1024)
s.close()
print(repr(data))
```

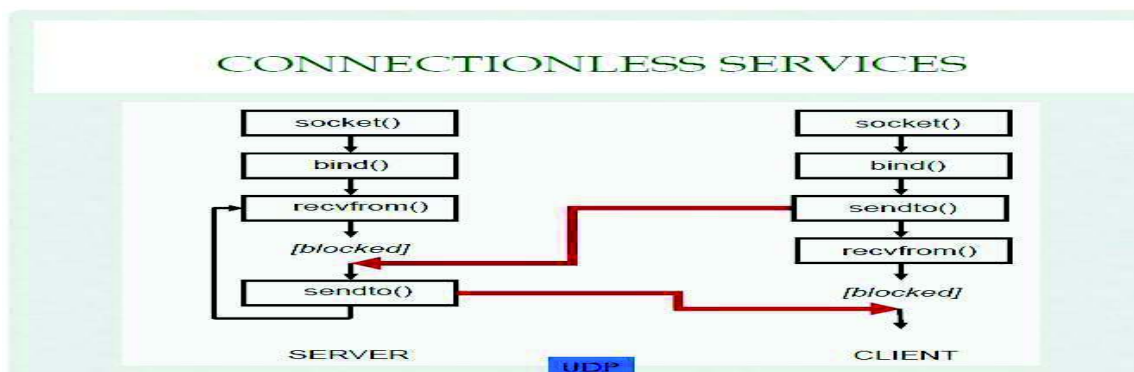
Note: The `repr()` function returns a printable representational string of the given object.

Now run this *server.py* in background and then run above *client.py* to see the result.

Output:

Step 1: Run *server.py*. It would start a server in background.

Step 2: Run *client.py*. Once server is started run client.



Simple Connectionless Server

```
import socket
```

```

sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)    # For UDP

udp_host = socket.gethostname()                          # Host IP
udp_port = 12345                                         # specified port to connect

sock.bind((udp_host, udp_port))

while True:
    print "Waiting for client..."
    data,addr = sock.recvfrom(1024)                    #receive data from client
    print "Received Messages:",data.decode()," from",addr

```

Simple Connectionless Client:

```

import socket

sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)    # For UDP

udp_host = socket.gethostname()                          # Host IP
udp_port = 12345                                         # specified port to connect

msg = "UDP Program!"
print "UDP target IP:", udp_host
print "UDP target Port:", udp_port

sock.sendto(msg.encode(),(udp_host,udp_port))

```

Practice Programs:

1A) Write a program where client can send a message to the server and the server can receive the message and send, or echo, it back to the client.

Echo Client:

```
import socket

HOST = '127.0.0.1' # The server's hostname or IP address
PORT = 2053        # The port used by the server

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
    s.sendall(b'Hello, world')
    data = s.recv(1024)
    print('Received Connection')
    print('Server:', data.decode())
```

Echo Server:

```
import socket

HOST = '127.0.0.1' # Standard loopback interface address (localhost)
PORT = 2053        # Port to listen on (non-privileged ports are > 1023)

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()
    conn, addr = s.accept()

    with conn:
        print('Connected by', addr)
        while True:
            data = conn.recv(1024)
            if data:
                print("Client: ",data.decode())
```

```
data = input("Enter message to client:");
if not data:
    break;
# sending message as bytes to client.
conn.sendall(bytearray(data, 'utf-8'));

conn.close()
```

2A) Write a program to create TCP time server in Python

Time Client:

```
#client.py
import socket

# create a socket object
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# get local machine name
host = socket.gethostname()
port = 9991
# connection to hostname on the port.
s.connect((host, port))
# Receive no more than 1024 bytes
tm = s.recv(1024)
print(' Current time from Sever :', tm.decode())
s.close()
```

Time Server:

```
# server.py
import socket
```

```
import time

# create a socket object
serversocket = socket.socket(
    socket.AF_INET, socket.SOCK_STREAM)

# get local machine name
host = socket.gethostname()

port = 9991

# bind to the port
serversocket.bind((host, port))

# queue up to 5 requests
serversocket.listen(5)

while True:
    # establish a connection
    clientsocket, addr = serversocket.accept()

    print("Got a connection from %s" % str(addr))
    currentTime = time.ctime(time.time()) + "\r\n"
    clientsocket.send(currentTime.encode('ascii'))
    clientsocket.close()
```

3A) Write a TCP chat server in python using socket programming.

Client Chat:

```
import socket

HOST = '127.0.0.1' # Standard loopback interface address (localhost)
PORT = 31621 # Port to listen on (non-privileged ports are > 1023)
s = socket.socket()
name = input(str("\nEnter your name: "))

print("\nTrying to connect to ", HOST, "(" , PORT, ") \n")

s.connect((HOST, PORT))
print("Connected...\n")

s.send(name.encode())
s_name = s.recv(1024)
s_name = s_name.decode()
print(s_name, "has joined the chat room\nEnter [e] to exit chat room\n")
while True:
    message = s.recv(1024)
    message = message.decode()
    print(s_name, ":", message)
    message = input(str("Me : "))
    if message == "[e]":
        message = "Left chat room!"
        s.send(message.encode())
        print("\n")
        break
    s.send(message.encode())
```

Server Chat:

```
# server.py
import socket

HOST = '127.0.0.1' # Standard loopback interface address (localhost)
PORT = 31621 # Port to listen on (non-privileged ports are > 1023)

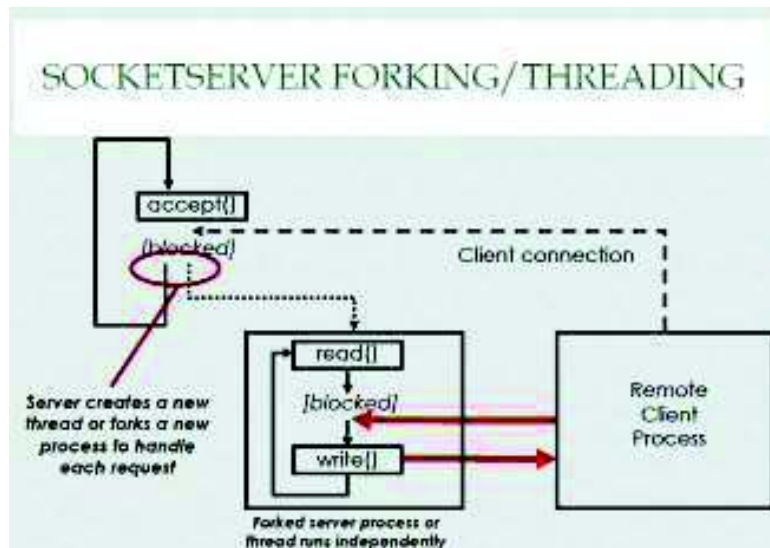
s = socket.socket()

s.bind((HOST, PORT))

s.listen()
print("\nWaiting for incoming connections...\n")
conn, addr = s.accept()
print("Received connection from ", addr[0], "(", addr[1], ")")

s_name = conn.recv(1024)
s_name = s_name.decode()
print(s_name, "has connected to the chat room\nEnter [e] to exit chat room\n")
name = input(str("Enter your name: "))
conn.send(name.encode())

while True:
    message = input(str("Me : "))
    if message == "[e]":
        message = "Left chat room!"
        conn.send(message.encode())
        print("\n")
        break
    conn.send(message.encode())
    message = conn.recv(1024)
    message = message.decode()
    print(s_name, ":", message)
```



4A. Forking/ Threading (Concurrent Server)

```
import socket
```

```
ClientSocket = socket.socket()
```

```
host = '127.0.0.1'
```

```
port = 11596
```

```
print('Waiting for connection')
```

```
try:
```

```
    ClientSocket.connect((host, port))
```

```
except socket.error as e:
```

```
    print(str(e))
```

```
Response = ClientSocket.recv(1024)
```

```
while True:
```

```
    Input = input('Client Say Something: ')
```

```
    ClientSocket.send(str.encode(Input))
```

```
Response = ClientSocket.recv(1024)
print('From Server : ' + Response.decode())
```

```
ClientSocket.close()
```

4B. Server Program

```
import socket
import os
from _thread import *

ServerSocket = socket.socket()
host = '127.0.0.1'
port = 11596
ThreadCount = 0
try:
    ServerSocket.bind((host, port))
except socket.error as e:
    print(str(e))

print('Waiting for a Connection..')
ServerSocket.listen(5)

def threaded_client(connection):
    connection.send(str.encode('Welcome to the Server'))
    while True:
        data = connection.recv(2048)
        print('Received from client : ' + str(ThreadCount) + data.decode())
        Inputs = input('Server Says: ')
        if not data:
            break
        connection.sendall(Inputs.encode())
    connection.close()
```

while True:

```
    Client, address = ServerSocket.accept()
    print('Connected to: ' + address[0] + ':' + str(address[1]))
    start_new_thread(threaded_client, (Client, ))
    ThreadCount += 1
    print('Thread Number: ' + str(ThreadCount))
```

```
ServerSocket.close()
```

Lab Exercise:

1. Write a UDP time server to display the current time and day.
2. Write a UDP simple chat program for message send and receive.

Sample Output:

Client Side

Do Ctrl+c to exit the program !!

Type some text to send =>Hi

1. Client Sent : Hi

2. Client received : Hello

Server Side:

Do Ctrl+c to exit the program !!

Server is listening

2. Server received: Hi

Type some text to send => Hello

1. Server sent : Hello

Server is listening

3. Write a TCP/UDP peer to peer chat system between two different machines.
-

4. Try to debug the error in the code and execute it.

Client:

```
import socket

serverIP = 'localhost'
serverPort = 16000

clientSock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
clientSock.connect((serverIP, serverPort))

message = raw_input("Input integers with space in between: ")
message2 = raw_input("Enter the length of the set: ")

clientSock.send(message)
clientSock.send(message2)

data = clientSock.recv(1024)

temp = [float(x) for x in data.split(' ')]

print("The total of all numbers is: " + str(temp[0]))
print("The lowest number is: " + str(temp[1]))
print("The highest number is: " + str(temp[2]))
print("The mean is: " + str(temp[3]))

clientSock.close()
```

```
import socket

serverIP = 'localhost'
serverPort = 16000

serverSock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
serverSock.bind((serverIP, serverPort))
serverSock.listen(1)

print("TCP server has started and is ready to receive")

while 1:
    connection, addr = serverSock.accept()
    data = connection.recv(1024)

    if not data: break

    temp = [float(x) for x in data.split(' ')]
    print "Received data:", temp

    length = len(temp)
    maximum = max(temp)
    minimum = min(temp)
    total = sum(temp)
    mean = total/length

    msg = str(total) + " " + str(minimum) + " " + str(maximum) +
    " " + str(mean)
```

```
connection.send(str(msg))
```

- Below is an example input from clientTCP side:

- Input integers with space in between: 5 4 6 9 3.4

- Next, the user will enter the length of the above set when prompted from clientUDP side:

- Enter the length of the set: 5

- Next, you will notice a screen output on the serverTCP side as follows:

- TCP server has started and is ready to receive

- Received data: [5.0, 4.0, 6.0, 9.0, 3.4]

- Finally you will should see an output on the clientTCP side as follows:

- The total of all numbers is: 27.4

- The lowest number is: 3.4

- The highest number is: 9.0

The mean is: 5.48

LAB – 5

MAP REDUCING PROGRAMS USING PYTHON

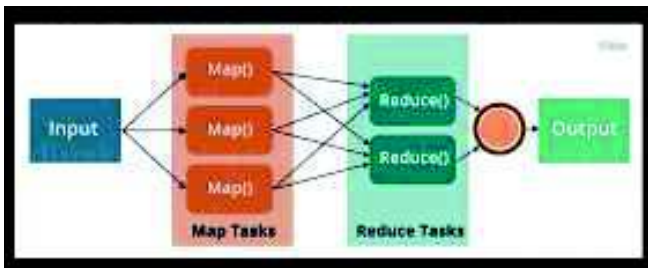
Lab – 5

MapReduce Programming using Python

MapReduce: Programming Model and Implementations:

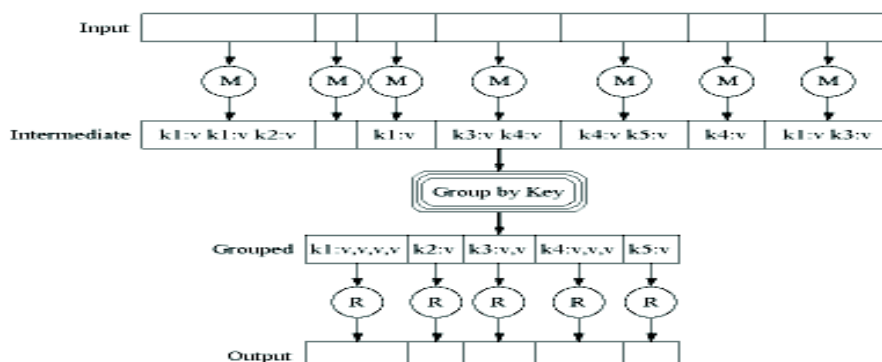
Hadoop is a framework that allows to process and store huge data sets. Basically, Hadoop can be divided into two parts: processing and storage. So, MapReduce is a programming model which allows you to process huge data stored in Hadoop. When you install Hadoop in a cluster, we get MapReduce as a service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



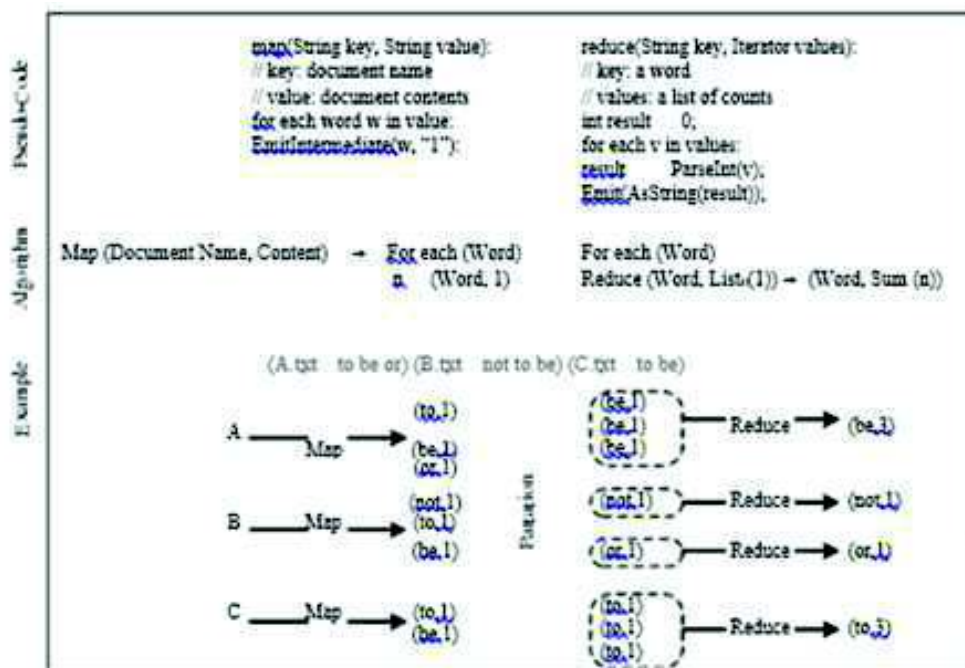
- To generate a set of output key-value pairs from a set of input key-value pairs
 - $\{ \langle k_i, v_i \rangle \} \rightarrow \{ \langle k_o, v_o \rangle \}$
- Expressed using two abstractions:
 - Map task
 - $\langle k_i, v_i \rangle \rightarrow \{ \langle k_{int}, v_{int} \rangle \}$
 - Reduce task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

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).



Another Example:

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

```
reduce(String output_key, Iterator intermediate_values):  
    // output_key: a word      <"Sam", ["1","1","1"]>, <"Apple", ["1"]>, <"Mom", ["1", "1"] >  
    // output_values: a list of counts  
    int result = 0;           "3"  
    for each v in intermediate_values: "1"  
        result += ParseInt(v);      "2"  
    Emit(AsString(result));
```

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

```
#!/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 ("{}{}\t{}".format(location, cost))
```

```
#can try with different instances.....
```

```
    #print ("{}{}\t{}".format(payment, cost))
```

```
    #print ("{}{}\t{}".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 ["<current_word>", "<count>"] 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
```

```
Boston397.21
```

```
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%.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 *****

LAB – 6

CLOCK SYNCHRONIZATION

CLOCK SYNCHRONIZATION

I. Introduction to Clock Synchronization

Time notion: Each computer is equipped with a physical (hardware) clock.

At time t , the operating system (OS) of a process i reads the hardware clock $H(t)$ of the processor, generates the software clock $C = aH(t) + b$, as a counter incremented by ticks of an oscillator.

Time in Distributed System (DS): Time is a key factor in a DS to analyse how distributed execution evolve.

Problems: Lacking of a global reference time: it's hard to know the state of a process during a distributed computation. However, it's important for processes to share a common time notion.

The techniques used to coordinate a common time notion among processes is known as Clock Synchronization.

Clock Synchronization: The hardware clock of a set of computers may differ because they count time with different frequencies.

Clock Synchronization faces this problem by means of synchronization algorithms

- Standard communication infrastructure
- No additional hardware

Clock synchronization algorithms

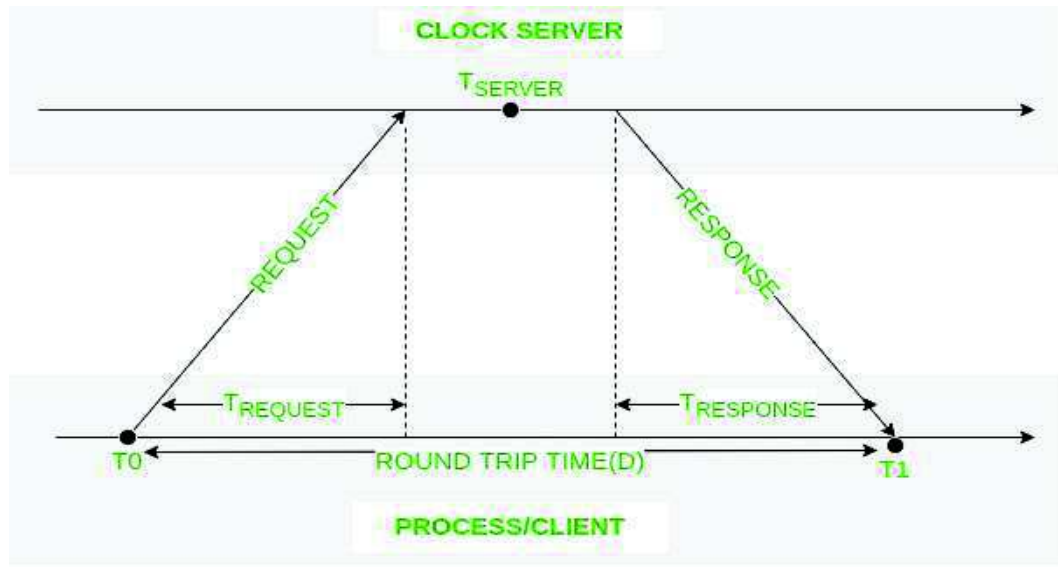
- External
- Internal
- Heartbeat (pulse)

External clock synchronization:

Is the one in which an external reference clock is present. It is used as a reference and the nodes in the system can set and adjust their time accordingly.

Internal clock synchronization: Is the one in which each node shares its time with other nodes and all the nodes set and adjust their times accordingly.

- II. **Cristian's algorithm:** Cristian's Algorithm is a clock synchronization algorithm is used to synchronize time with a time server by client processes. This algorithm works well with low-latency networks where Round Trip Time is short as compared to accuracy while redundancy prone distributed systems/applications do not go hand in hand with this algorithm. Here Round Trip Time refers to the time duration between start of a Request and end of corresponding Response.



Algorithm:

- 1) The process on the client machine sends the request for fetching clock time (time at server) to the Clock Server at time.
- 2) The Clock Server listens to the request made by the client process and returns the response in form of clock server time.
- 3) The client process fetches the response from the Clock Server at time and calculates the synchronised client clock time using the formula given below.

$$T_{CLIENT} = T_{SERVER} + (T_1 - T_0)/2$$

T_{CLIENT} = synchronised clock time

T_{SERVER} = clock time returned by the server

T_0 = time at which request was sent by the client process

T_1 = time at which response was received by the client process

- III. **Berkeley's algorithm:** Berkeley's Algorithm is a clock synchronization technique used in distributed systems. The algorithm assumes that each machine node in the network either doesn't have an accurate time source or doesn't possess an UTC server.

Algorithm:

- 1) An individual node is chosen as the master node from a pool nodes in the network. This node

is the main node in the network which acts as a master and rest of the nodes act as slaves. Master node is chosen using a election process/leader election algorithm.

2) Master node periodically pings slaves nodes and fetches clock time at them using Cristian's algorithm.

3) Master node calculates average time difference between all the clock times received and the clock time given by master's system clock itself. This average time difference is added to the current time at master's system clock and broadcasted over the network.

Pseudocode for above step:

```
# receiving time from all slave nodes
```

```
repeat_for_all_slaves:
```

```
    time_at_slave_node = receive_time_at_slave()
```

```
# calculating time difference
```

```
    time_difference = time_at_master_node - time_at_slave_node
```

```
# average time difference calculation
```

```
average_time_difference = sum(all_time_differences) / number_of_slaves
```

```
synchronized_time = current_master_time + average_time_difference
```

```
# broadcasting synchronized to whole network
```

```
broadcast_time_to_all_slaves(synchronized_time)
```

Diagram below illustrates how the master sends request to slave nodes.

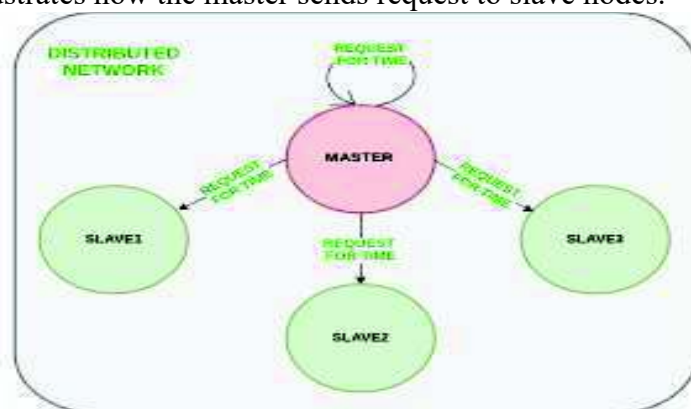


Diagram below illustrates how slave nodes send back time given by their system clock.

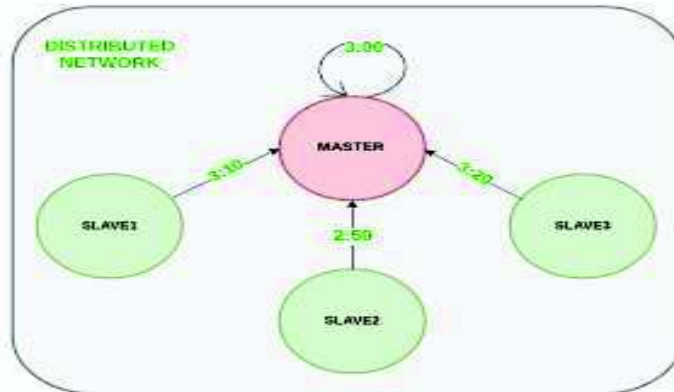
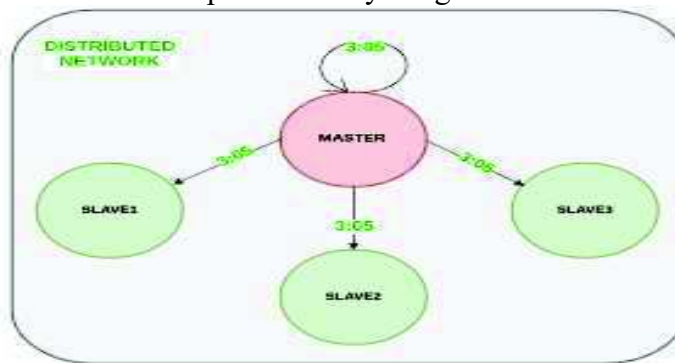


Diagram below illustrates the last step of Berkeley's algorithm.



IV. Solved examples:

A. Cristian's algorithm

1) To initiate a prototype of a clock server on local machine:

Server:

Python3 program imitating a clock server

import socket

import datetime

import time

function used to initiate the Clock Server

def initiateClockServer():

 s = socket.socket()

 print("Socket successfully created")

 # Server port

 port = 8011

```

s.bind(('', port))
# Start listening to requests
s.listen(5)
print("Socket is listening...")
# Clock Server Running forever
while True:
# Establish connection with client
    connection, address = s.accept()
    print('Server connected to', address)
# Respond the client with server clock time
    connection.send(str(datetime.datetime.now()).encode())
# Close the connection with the client process
    connection.close()
# Driver function
if __name__ == '__main__':
    # Trigger the Clock Server
    initiateClockServer()

```

Output Sever Side:

```

Ubuntu@ubuntu:~$ python3 cris_ser.py
Socket successfully created
Socket is listening...
Server connected to ('127.0.0.1', 42310)

```

- II) **Code below is used to initiate a prototype of a client process on local machine:**
Python3 program imitating a client process

```

import socket
import datetime
from dateutil import parser
from timeit import default_timer as timer

```

```
# function used to Synchronize client process time
def synchronizeTime():

    s = socket.socket()

    # Server port
    port = 8011

    # connect to the clock server on local computer
    s.connect(('127.0.0.1', port))

    request_time = timer()

    # receive data from the server
    server_time = parser.parse(s.recv(1024).decode())
    response_time = timer()
    actual_time = datetime.datetime.now()

    print("Time returned by server: " + str(server_time))

    process_delay_latency = response_time - request_time

    print("Process Delay latency: " + str(process_delay_latency) + " seconds")

    print("Actual clock time at client side: " + str(actual_time))

    # synchronize process client clock time
    client_time = server_time + datetime.timedelta(seconds = (process_delay_latency) / 2)
```



```

print("Synchronized process client time: " + str(client_time))

# calculate synchronization error
error = actual_time - client_time
print("Synchronization error : " + str(error.total_seconds()) + " seconds")

s.close()

# Driver function
if __name__ == '__main__':

    # synchronize time using clock server
    synchronizeTime()

```

Output Client Side:

```

Time returned by server: 2021-05-27 17:41:36.223936
Process Delay latency: 0.000858066999995799 seconds
Actual clock time at client side: 2021-05-27 17:41:36.224587
Synchronized process client time: 2021-05-27 17:41:36.224365
Synchronization error : 0.000222 second

```

B. Berkeley's algorithm:

Server Side:

Python3 program imitating a clock server

```
from functools import reduce
from dateutil import parser
import threading
import datetime
import socket
import time

# datastructure used to store client address and clock data
client_data = {}

''' nested thread function used to receive
    clock time from a connected client '''
def startRecieveingClockTime(connector, address):

    while True:
        # recieve clock time
        clock_time_string = connector.recv(1024).decode()
        clock_time = parser.parse(clock_time_string)
        clock_time_diff = datetime.datetime.now() - \
            clock_time

        client_data[address] = {
            "clock_time" : clock_time,
            "time_difference" : clock_time_diff,
            "connector" : connector
        }

    print("Client Data updated with: "+ str(address),
```



```
sum_of_clock_difference = sum(time_difference_list, \
                               datetime.timedelta(0, 0))
```

```
average_clock_difference = sum_of_clock_difference \
                             / len(client_data)
```

```
return average_clock_difference
```

```
def synchronizeAllClocks():
```

```
    while True:
```

```
        print("New synchroniztion cycle started.")
        print("Number of clients to be synchronized: " + \
              str(len(client_data)))
```

```
        if len(client_data) > 0:
```

```
            average_clock_difference = getAverageClockDiff()
```

```
            for client_addr, client in client_data.items():
```

```
                try:
```

```
                    synchronized_time = \
                        datetime.datetime.now() + \
                            average_clock_difference
```

```
                    client['connector'].send(str(
                        synchronized_time).encode())
```

```

        except Exception as e:
            print("Something went wrong while " + \
                  "sending synchronized time " + \
                  "through " + str(client_addr))

        else :
            print("No client data." + \
                  " Synchronization not applicable.")

        print("\n\n")

        time.sleep(5)

# function used to initiate the Clock Server / Master Node
def initiateClockServer(port = 8080):

    master_server = socket.socket()
    master_server.setsockopt(socket.SOL_SOCKET,
                             socket.SO_REUSEADDR, 1)

    print("Socket at master node created successfully\n")

    master_server.bind(("", port))

    # Start listening to requests
    master_server.listen(10)
    print("Clock server started...\n")

    # start making connections
    print("Starting to make connections...\n")

```

```
master_thread = threading.Thread(  
    target = startConnecting,  
    args = (master_server, ))  
master_thread.start()  
  
# start synchroniztion  
print("Starting synchronization parallely...\n")  
sync_thread = threading.Thread(  
    target = synchronizeAllClocks,  
    args = ())  
sync_thread.start()
```

Driver function

```
if __name__ == '__main__':
```

Trigger the Clock Server

```
initiateClockServer(port = 8080)
```

Output:

New synchroniztion cycle started.

Number of clients to be synchronized: 2

Client Data updated with: 127.0.0.1:37624

Client Data updated with: 127.0.0.1:37626

Client Side :

Python3 program imitating a client process

```
from timeit import default_timer as timer
```

```
from dateutil import parser
```

```
import threading
```

```
import datetime
```

```
import socket
```

```
import time
```

```
# client thread function used to send time at client side
```

```
def startSendingTime(slave_client):
```

```
    while True:
```

```
        # provide server with clock time at the client
```

```
        slave_client.send(str(
            datetime.datetime.now()).encode())
```

```
        print("Recent time sent successfully",
              end = "\n\n")
```

```
        time.sleep(5)
```

```
# client thread function used to receive synchronized time
```

```
def startReceivingTime(slave_client):
```

```
    while True:
```

```
        # receive data from the server
```

```
        Synchronized_time = parser.parse(
            slave_client.recv(1024).decode())
```

```
print("Synchronized time at the client is: " + \
      str(Synchronized_time),
      end = "\n\n")
```

```
# function used to Synchronize client process time
def initiateSlaveClient(port = 8080):
```

```
    slave_client = socket.socket()
```

```
    # connect to the clock server on local computer
```

```
    slave_client.connect(('127.0.0.1', port))
```

```
    # start sending time to server
```

```
    print("Starting to receive time from server\n")
```

```
    send_time_thread = threading.Thread(
```

```
        target = startSendingTime,
```

```
        args = (slave_client, ))
```

```
    send_time_thread.start()
```

```
    # start receiving synchronized from server
```

```
    print("Starting to receiving " + \
```

```
          "synchronized time from server\n")
```

```
    receive_time_thread = threading.Thread(
```

```
        target = startReceivingTime,
```

```
        args = (slave_client, ))
```

```
    receive_time_thread.start()
```



```
# Driver function
if __name__ == '__main__':

    # initialize the Slave / Client
    initiateSlaveClient(port = 8080)
```

Output:

Synchronized time at the client is: 2021-05-27 17:45:29.885369

Recent time sent successfully

Synchronized time at the client is: 2021-05-27 17:45:34.888644

Recent time sent successfully

Lab Exercises:

1. The Manipal Foodie is a renowned automated food processing outlet known for its tiffin service to students. The various processes involved are food production, filling and packing. Every day more than 3000 orders are received on an average from the students in manipal. There are total of 4 production lines for orders received from KMC, MIT, TAPMI and SOLS students, each of them has a digital clock which needs to be in synchronization with the master clock. The master clock mounted in the testing lab controls the entire clock system. Design an appropriate solution using Berkeley's algorithm for the above scenario. Assume that the clocks at the institutes are slave/clients.
2. Manipal Buddy is a banking and education application for the students and staff of MIT, Manipal. Mr Vinay, a sixth semester student wants to pay the end semester exams fees for a re-registered course. He simultaneously wishes to register for a course on NPTEL through the app. To register for exam he uses the mobile app whereas to register for NPTEL course he uses his laptop to log in. As he needs to finish both the registrations on the same day, he tries to do both the tasks simultaneously. Analyse and demonstrate using a program how Cristian's algorithm can be used in the above case to synchronize the clocks. Assume the relevant parameters.

References:

1. Mahajan, S. and Shah, S., 2015. *Distributed Computing*. 2nd ed. Oxford University Press, pp.141-209.
2. GeeksforGeeks. 2020. *Geeksforgeeks | A Computer Science Portal For Geeks*. [online] Available at: <<https://www.geeksforgeeks.org/>> [Accessed 3 April 2020].
3. American-time.com. 2020. *Food Manufacturing Clock Synchronization Case Study | American Time*. [online] Available at: <<https://www.american-time.com/industries/manufacturing/case-studies/food-manufacturing>> [Accessed 3 April 2020].

LAB – 7

MUTUAL EXCLUSION (ELECTION ALGORITHM)

MUTUAL EXCLUSION (ELECTION ALGORITHM)

ELECTION ALGORITHMS

Theory:

Many algorithms used in the distributed system require a coordinator that performs functions needed by other processes in the system. Election algorithms are designed to choose a coordinator.

Election Algorithms

Election algorithms choose a process from a group of processors to act as a coordinator. If the coordinator process crashes due to some reasons, then a new coordinator is elected on another processor. E.g. coordinator process is picked as a master in Berkeley clock synchronization algorithm. Election algorithm determines, where a new copy of a coordinator should be restarted. Election algorithm assumes that every active process in the system has a unique priority number. The process with the highest priority will be chosen as a new coordinator. Hence, when a coordinator fails, this algorithm elects that active process which has the highest priority number. The active highest selected number is then sent to every active process in the distributed system.

We have two election algorithms for two different configurations of the distributed system.

1. The Bully Algorithm

This algorithm applies to a system where every process can send a message to every other process in the system.

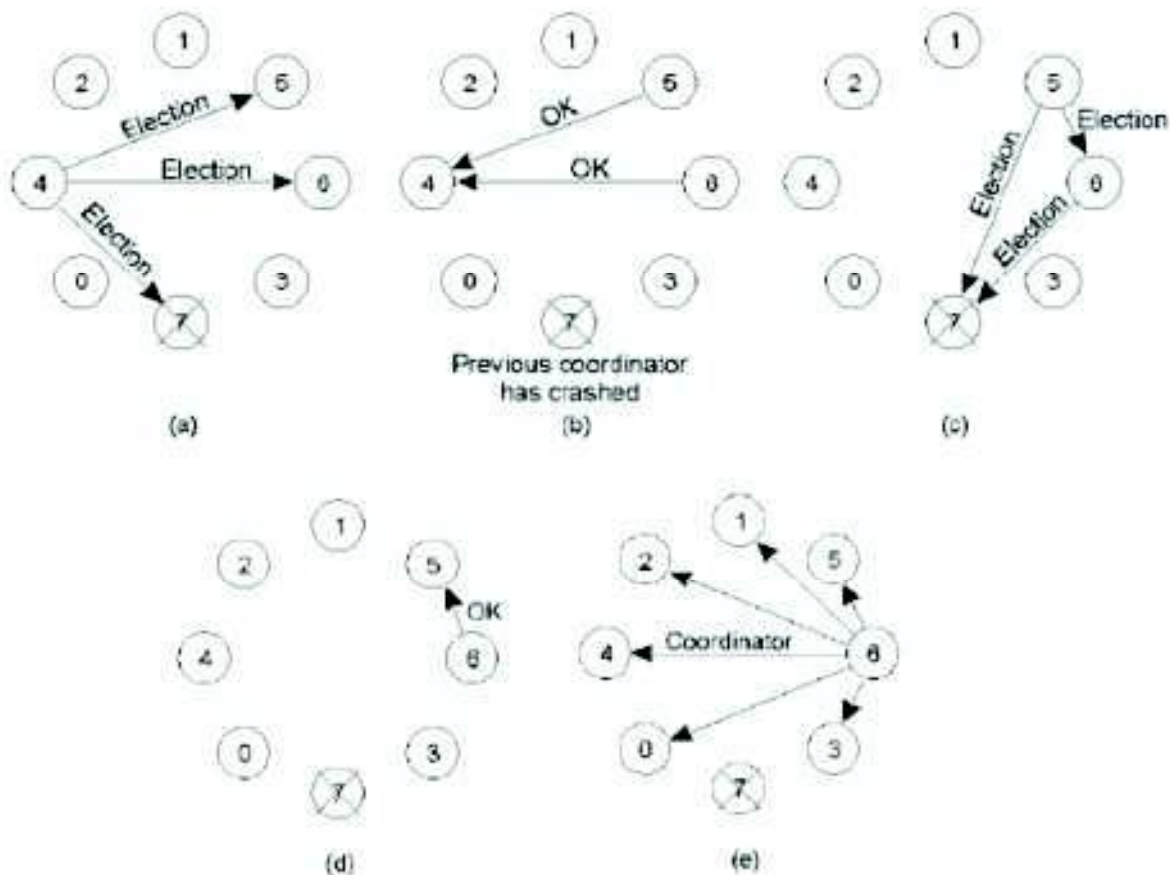
Assumptions

- Each process knows the ID and address of every other process
- Communication is reliable
- A process initiates an election if it just recovered from failure or it notices that the coordinator has failed
- Three types of messages: Election, OK, Coordinator
- Several processes can initiate an election simultaneously
- Need consistent result

Algorithm – Suppose process P sends a message to the coordinator node.

1. If the coordinator node does not respond to it within a time interval T, then it is assumed that the coordinator node has failed.
2. P sends Election messages to all process with higher IDs and awaits OK messages
3. If no OK messages, P becomes coordinator and sends Coordinator messages to all processes with lower IDs

4. If it receives an OK, it drops out and waits for a Coordinator message
5. If a process receives an Election message
 - Immediately sends Coordinator message if it is the process with the highest ID
 - Otherwise,
 - Returns an OK and starts an election
6. If a process receives a Coordinator message, it treats a sender as the coordinator



Bully Algorithm Example

2. The Ring Algorithm

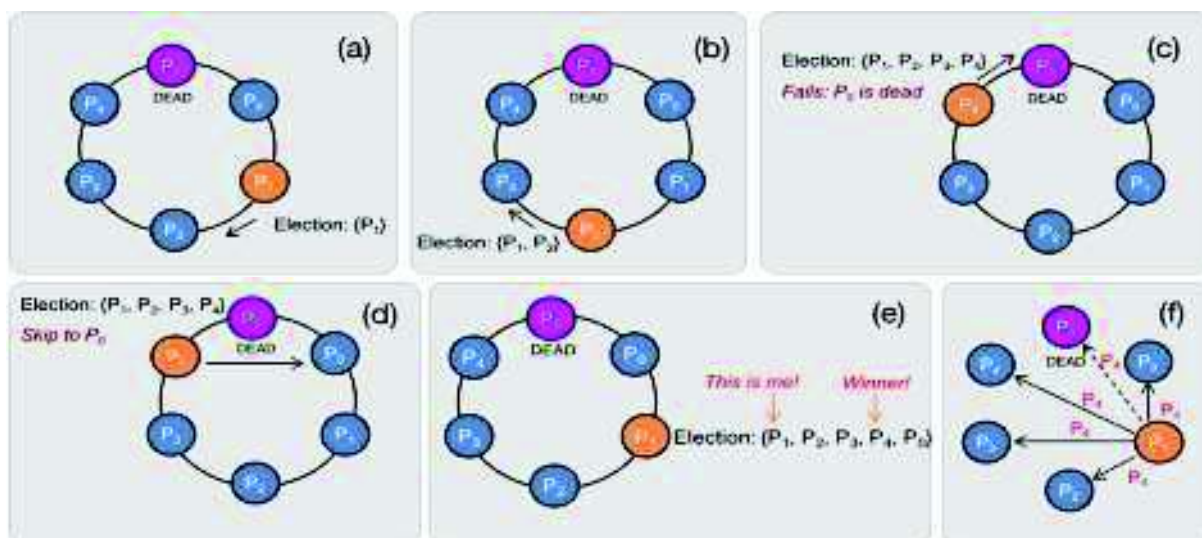
This algorithm applies to systems organized as a ring (logically or physically). In this algorithm, we assume that the link between the processes are unidirectional, and every process can message to the process on its right only.

Assumes that processes are logically ordered in some fashion, and that each process knows the order and who is the coordinator. No token is involved. When a process notices that the coordinator is not responding it sends an ELECTION message with its own id to its downstream neighbour. If

that neighbour doesn't acknowledge, it sends it to its neighbour's neighbour, etc. Each station that receives the ELECTION message adds its own id to the list. When the message circulates back to the originator, it selects the highest id in the list and sends a COORDINATOR message announcing the new coordinator. This message circulates once and is removed by the originator. If two elections are held simultaneously (say because two different processes notice simultaneously that the coordinator is dead) then each comes up with the same list and elects the same coordinator. Some time is wasted, but this really hurts nothing.

Algorithm:-

1. Processes are arranged in a logical ring; each process knows the structure of the ring
2. A process initiates an election if it just recovered from failure or it notices that the coordinator has failed
3. The initiator sends Election message to the closest downstream node that is alive
 - Election message is forwarded around the ring
 - Each process adds its own ID to the Election message
4. When Election message comes back, initiator picks node with highest ID and sends a Coordinator message specifying the winner of the election
 - Coordinator message is removed when it has circulated once.



. Ring Algorithm Example

Lab Exercises:

1. Simulate a scenario in distributed systems to implement the Bully Algorithm for choosing a coordinator node amongst the participative nodes of the system after the collapse of the existing coordinator node in the system.

2. Simulate a scenario in distributed systems to implement the Ring Algorithm for choosing a coordinator node amongst the participative nodes of the system after the collapse of the existing coordinator node in the system.

References:

1. Mahajan, S. and Shah, S., 2015. *Distributed Computing*. 2nd ed. Oxford University Press, pp.141-209.
 2. GeeksforGeeks. 2020. *Geeksforgeeks | A Computer Science Portal For Geeks*. [online] Available at: <<https://www.geeksforgeeks.org/>> [Accessed 3 April 2020].
 3. American-time.com. 2020. *Food Manufacturing Clock Synchronization Case Study | American Time*. [online] Available at: <<https://www.american-time.com/industries/manufacturing/case-studies/food-manufacturing>> [Accessed 3 April 2020].
-

LAB – 8

INSTALLATION OF HADOOP ON SINGLE NODE

INSTALLATION OF HADOOP ON SINGLE NODE

Steps to be performed on Ubuntu OS:

1. Update Ubuntu

```
$sudo apt-get update
```

2. Check for Java version:

```
$ java -version
```

If Java is not installed, then install it using

```
$ sudo apt install java-default
```

3. Add dedicated user for hadoop

```
$ sudo addgroup hadoop  
$ sudo adduser --ingroup hadoop hduser
```

4. Assign super user permissions to hduser from

```
$pico /etc/sudoers
```

5. Shift from current user to hduser

```
$ su - hduser
```

6. Generate RSA keys for password less access among nodes

```
ssh-keygen -t rsa -P ""  
cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
```

7. Install Openssh server if not exists:

```
$ sudo apt install openssh-server
```

8. Connect to localhost:

```
$ ssh localhost
```

9. To disable IPv6 on Ubuntu, open /etc/sysctl.conf in the editor of your choice and add the following lines to the end of the file:

```
net.ipv6.conf.all.disable_ipv6 = 1  
net.ipv6.conf.default.disable_ipv6 = 1
```

```
net.ipv6.conf.lo.disable_ipv6 = 1
```

10. **You can check whether IPv6 is enabled on your machine with the following command:**

```
$ cat /proc/sys/net/ipv6/conf/all/disable_ipv6
```

A return value of 0 means IPv6 is enabled, a value of 1 means disabled

11. **Refresh sysctl.conf**

```
$ sudo sysctl -p and repeat step 10.
```

12. **Download hadoop from wget**

<https://www-eu.apache.org/dist/hadoop/core/hadoop-2.7.7/hadoop-2.7.7.tar.gz>

13. **Unzip it:**

```
$ sudo tar xzf hadoop-2.x.x.tar.gz
```

14. **Rename and move it to specific folder:**

```
$ sudo mv hadoop-2.7.7 /usr/local/Hadoop
```

15. **Change directory**

```
$ cd /usr/local/hadoop
```

16.

```
$ sudo chown -R hduser:hadoop hadoop
```

17.

```
$ sudo chmod 777 hadoop
```

18. **update bashrc: Open bashrc using \$ pico ~/.bashrc and append following at the end of the file**

```
# $HOME/.bashrc
# Set Hadoop-related environment variables
export HADOOP_HOME=/usr/local/hadoop

# Set JAVA_HOME (we will also configure JAVA_HOME directly for Hadoop later
on)
export JAVA_HOME=/usr/lib/jvm/default-java

# Some convenient aliases and functions for running Hadoop-related commands
unalias fs &> /dev/null
alias fs="hadoop fs"
unalias hls &> /dev/null
alias hls="fs -ls"
```

```
# If you have LZOP compression enabled in your Hadoop cluster and
# compress job outputs with LZOP (not covered in this tutorial):
# Conveniently inspect an LZOP compressed file from the command
# line; run via:
#
# $ lzohead /hdfs/path/to/lzop/compressed/file.lzo
#
# Requires installed 'lzop' command.
#
lzohead () {
    hadoop fs -cat $1 | lzop -dc | head -1000 | less
}

# Add Hadoop bin/ directory to PATH
export PATH=$PATH:$HADOOP_HOME/bin
```

19. set JAVA_HOME in

```
$ pico /usr/local/hadoop/etc/hadoop/hadoop-env.sh
```

20. Create Hadoop tmp directory to store logs

```
$ sudo mkdir -p /app/hadoop/tmp
$ sudo chown hduser:hadoop /app/hadoop/tmp
$ sudo chmod 777 /app/hadoop/tmp
```

21. Configure Hadoop files using following steps:

22. Open \$pico /usr/local/hadoop/etc/hadoop/core-site.xml and add the following lines between <configuration> and </configuration>

```
<property>

<name>hadoop.tmp.dir</name>
<value>/app/hadoop/tmp</value>
<description>A base for other temporary directories.</description>
</property>

<property>
<name>fs.default.name</name>
<value>hdfs://localhost:54310</value>
<description>The name of the default file system. A URI whose
scheme and authority determine the FileSystem implementation. The
uri's scheme determines the config property (fs.SCHEME.impl) naming
the FileSystem implementation class. The uri's authority is used to
determine the host, port, etc. for a filesystem.</description>
</property>
```

23. cp etc/hadoop/mapred-site.xml.template etc/hadoop/mapred-site.xml

24. Open mapred-site.xml and add following lines between <configuration> And </configuration>

```
<property>
  <name>mapred.job.tracker</name>
  <value>localhost:54311</value>
  <description>The host and port that the MapReduce job tracker runs
    at. If "local", then jobs are run in-process as a single map
    and reduce task.
  </description>
</property>
```

25. Create directories for namenode and datanode logs

```
sudo mkdir -p /usr/local/hadoop_store/hdfs/namenode
sudo mkdir -p /usr/local/hadoop_store/hdfs/datanode
sudo chown -R hduser:hadoop /usr/local/hadoop_store
sudo chmod 777 /usr/local/hadoop_store
```

26. Update /usr/local/hadoop/etc/hadoop/hdfs-site.xml

```
<property>
  <name>dfs.replication</name>
  <value>1</value>
  <description>Default block replication.
    The actual number of replications can be specified when the file is created.
    The default is used if replication is not specified in create time.
  </description>
</property>
<property>
  <name>dfs.namenode.name.dir</name>
  <value>file:/usr/local/hadoop_store/hdfs/namenode</value>
</property>
<property>
  <name>dfs.datanode.data.dir</name>
  <value>file:/usr/local/hadoop_store/hdfs/datanode</value>
</property>
```

27. Update yarn-site.xml

```
<property>
  <name>yarn.nodemanager.aux-services</name>
  <value>mapreduce_shuffle</value>
</property>
```

28. Format the namenode

```
$ bin/hadoop namenode -format
```

29. Start the nodes:

```
$ sbin/start-dfs.sh
```

30. List all running nodes using \$jps

(It should list namenode and datanode)

31. Stop the nodes:

```
$ sbin/stop-dfs.sh
```

FILE MANAGEMENT AND HADOOP ADMINISTRATION

1. File Management

1. Create a directory in HDFS at given path(s).

Usage: bin/hdfs dfs -mkdir

Example: bin/hdfs dfs -mkdir /user/hduser/dir1 /user/hduser/dir2

2. List the contents of a directory.

Usage : bin/hdfs dfs -ls

Example: bin/hdfs dfs -ls /user/hduser

3. Upload and download a file in HDFS.

Copy single src file, or multiple src files from local file system to the Hadoop data file system Usage: bin/hdfs dfs -put ...

Example: bin/hdfs dfs -put /home/hduser/Samplefile.txt /user/ hduser/dir3/

Download: Downloads files to the local file system

Usage: bin/hdfs dfs -get

Example: bin/hdfs dfs -get /user/hduser/dir3/Samplefile.txt /home/

4. See contents of a file Same as unix cat command:

Usage: bin/hdfs dfs -cat

Example: bin/hdfs dfs -cat /user/hduser/dir1/abc.txt

5. Copy a file from source to destination This command allows multiple sources as well in which case the destination must be a directory.

Usage: bin/hdfs dfs -cp

Example: bin/hdfs dfs -cp /user/hduser/dir1/abc.txt /user/hduser/ dir2

6. Copy a file from/To Local file system to HDFS copyFromLocal

Usage: bin/hdfs dfs -copyFromLocal

Example: bin/hdfs dfs -copyFromLocal /home/hduser/abc.txt /user/ hduser/abc.txt

Similar to put command, except that the source is restricted to a local file reference. copyToLocal

Usage: bin/hdfs dfs -copyToLocal

Similar to get command, except that the destination is restricted to a local file reference.

7. Move file from source to destination. Note:- Moving files across filesystem is not permitted.

Usage : bin/hdfs dfs -mv

Example: bin/hdfs dfs -mv /user/hduser/dir1/abc.txt /user/hduser/ dir2

8. Remove a file or directory in HDFS. Remove files specified as argument. Deletes directory only when it is empty

Usage : bin/hdfs dfs -rm

Example: bin/hdfs dfs -rm /user/hduser/dir1/abc.txt Recursive version of delete. Usage : bin/hdfs dfs -rmr Example: bin/hdfs dfs -rmr /user/hduser/

9. Display last few lines of a file. Similar to tail command in Unix.

Usage : bin/hdfs dfs -tail

Example: bin/hdfs dfs -tail /user/hduser/dir1/abc.txt

10. Display the aggregate length of a file.

Usage : bin/hdfs dfs -du

Example: bin/hdfs dfs -du /user/hduser/dir1/abc.txt 3.W

2. Run Hadoop MapReduce examples

Use the [WordCount example job](#) which reads text files and counts how often words occur. The input is text files and the output is text files, each line of which contains a word and the count of how often it occurred, separated by a tab.

Steps:

1. Download text files from Gutenberg site.

Example: wget <http://www.gutenberg.org/cache/epub/20417/pg20417.txt>

2. Copy all the files to a local directory, Example input-dir

3. Start the nodes

4. Copy dir1 from local file system to hdfs using put.

5. Use ls and check content of hdfs

6. Run inbuilt wordcount example

\$ bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.6.1.jar WordCount input-dir output-dir

7. Examine the output files:

Copy the output files from the distributed filesystem to the local filesystem and examine them:

```
$ bin/hdfs dfs -get output output
$ cat output/*
```

or

View the output files on the distributed filesystem:

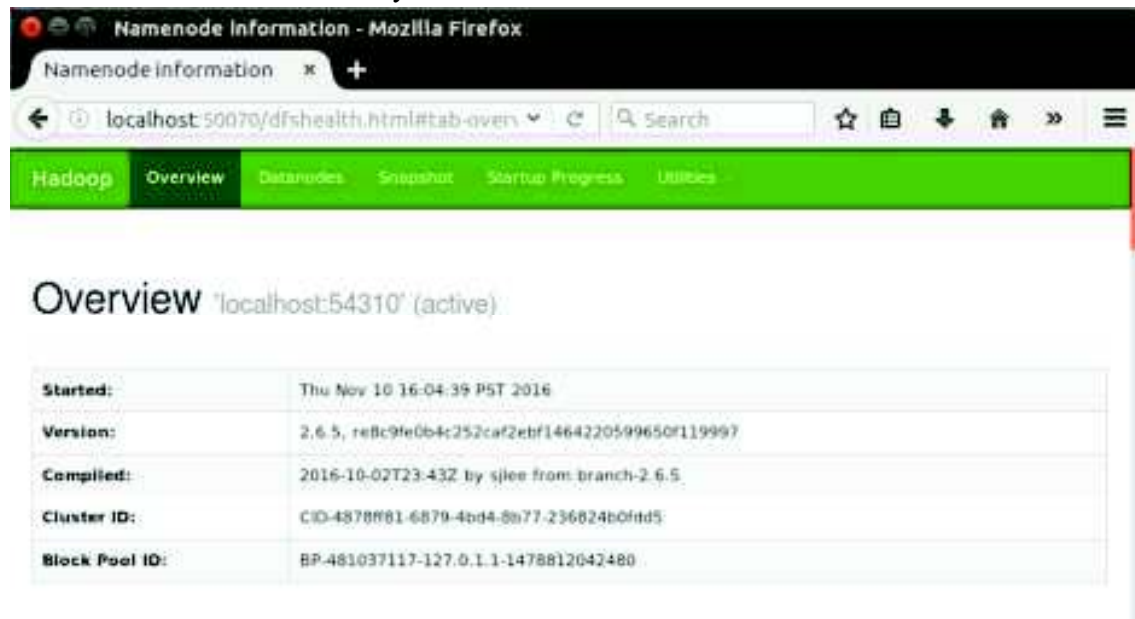
```
$ bin/hdfs dfs -cat output/*
```

8. When you're done, stop the daemons with:

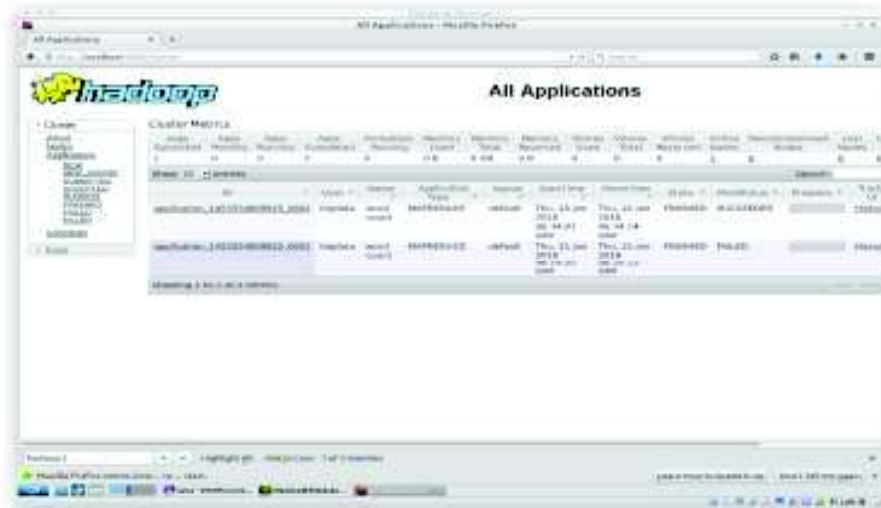
```
$ sbin/stop-dfs.sh
```

3. Hadoop Administration (Web Interfaces)

1. We can see cluster summary at localhost: 50070



2. Status of the MapReduce code available at localhost:8088



TITLE: MAPREDUCE PROGRAMMING USING PYTHON ON HADOOP

Example: Write a basic wordcount program.

Pseudocode:

Mapper:

void Map (key, value)

```
{  
  
    //Where key is offset or NULL and value is a record in a text file.  
  
    for each word x in value:  
  
        emit(x, 1);  
  
}
```

Reducer:

void Reduce (keyword, <list_val>)

```
{  
  
    //Where keyword is a word from mapper and <list_val> is a list of occurrence of 1s  
    from mapper  
  
    for each x in <list_val>:  
  
        sum+=x;  
  
        emit(keyword, sum);  
  
}
```

Python Program:

```
#!/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
```

```
hduser@ubuntu:~$ echo "foo foo quux labs foo bar quux" | /home/hduser/mapper.py
```

```
foo 1
```

```
foo 1
```

```
quux 1
```

```
labs 1
```

```
foo 1
```

```
bar 1
```

```
quux 1
```

```
hduser@ubuntu:~$ echo "foo foo quux labs foo bar quux" | /home/hduser/mapper.py | sort -k1,1 | /home/hduser/reducer.py
```

```
bar 1
```

```
foo 3
```

```
labs 1
```

```
quux 2
```

Run MapReduce job using Hadoop streaming jar.

1. Download input text files from <http://www.gutenberg.org/ebooks/20417>, <http://www.gutenberg.org/ebooks/5000> to HDFS

2.

```
hduser@ubuntu:/usr/local/hadoop$ bin/hadoop jar streaming/hadoop-*streaming*.jar -
file /home/hduser/mapper.py -mapper /home/hduser/mapper.py -file
/home/hduser/reducer.py -reducer /home/hduser/reducer.py -input /user/hduser/input-
dir/* -output /user/hduser/output-dir
```

Reference:

1. <https://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster/>
2. <https://www.youtube.com/watch?v=QNB1SZm2jS4>
3. <https://www.edureka.co/blog/install-hadoop-single-node-hadoop-cluster>