Exercise 0: Explain your system:

Hardware and Software	**Specification**
Processor	Intel®core™ i3-5005U CPU @2.00GHz
Number of cores	2
Logical processors	4
RAM	4.00 GB
OS	Windows 10
Python	3.6

Exercise 1: **Distributed K-means Clustering 20 Points:**

a) **Implement distributed K-means for 20-news group dataset:**

For this exercise I have used my previous Lab 2 assignment tf.idf results to perform the distributed k-means clustering algorithm for the 20 news group dataset. I have loaded the data files from the local file system. Each tf-idf results are of counter object type. I have not used any scikit to convert it to sparse matrix and instead continued to perform the experiment using my Counter objects. Now moving on to the exercise. I have initialized my cluster size by some k value and choose k random cluster centres as global mean using "np.random.choice()". My master node is responsible for scattering the data among all the workers and also it broadcasts this global mean across all workers. Now the data will be shared equally across all the workers and each worker would have a copy of this global mean. Now next step was to calculate Euclidean distance of each data point wrt to this global cluster centers using the formula shown in (1) and assign the data point to the cluster which has the least distance. Then i have calculated my new local cluster means by taking the average of each cluster group and have made this local mean as global mean for further iterations. The problem i have faced using this is that there was no way that my local cluster and global cluster to remain same. So in order to make sure that my algorithm converges. I have calculated the distortion in each iteration by using the formula shown in (2) and my all individual workers would send their respective distortion to master node. i have done this using "comm.reduce". My master node will calculate the complete distortion in each iterations and compares the value with previous iterations. My total distortions values throughout the exercise were decreasing at each iteration and i have put

convergence condition to stop if my total distortion value increases from current value. For this dataset i have used partial dataset to save my computation cost.

Euclidean distance:

$$d(\mathbf{p},\mathbf{q}) = d(\mathbf{q},\mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$ ..... (1)

Distortion calculation:

$$J(c,\mu) = \sum_{i=1}^{m} ||x^{(i)} - \mu_{c^{(i)}}||^2$$ ..... (2)

Where:

μ_c = cluster mean

$x^{(i)}$ = each data point belonging to that cluster μ_c

Code:

```
"""
Created on Sun Apr 29 17:41:17 2018

@author: saikiran
"""


from mpi4py import MPI
import math
import json
import numpy as np
import os
from collections import Counter


# method to convert my initial random global clusters to a list of counter objects of size = cluster size
def mean_array(path,mean):
    y = len(mean)
    cnt = [Counter() for i in range (y)]
    i=0
    for x in mean:
        filename=(path+x)
        mean_x = open(filename,'rt').read()
        mean_score = Counter(json.loads(mean_x))
        cnt[i] += mean_score
        if(i<y):
            i=i+1
    return cnt
```

```
def euclidean_distance(mean,data):
    ed_list=[]
    for i in mean:
        c= math.sqrt(sum((i.get(d,0) - data.get(d,0))**2 for d in set(i) | set(data)))
        ed_list = np.append(ed_list,c)
    return ed_list

# method to avoid zero by error exception if the divisor is zero my value will be returned as zero.
def safe_div(x,y):
    if y == 0:
        rem = 0
    else:
        rem = x/y
    return rem

# method to calculate the distortion value of each cluster in my list of counter objects.
def distortion_calculation(a):
    lm_list=[]
    for lm in a:
        lm = sum(lm.values())
        lm_sqr = (lm)**2
        lm_list =np.append(lm_list,lm_sqr)
    sum_list= lm_list.tolist()
    return sum(sum_list)

comm = MPI.COMM_WORLD
rank = comm.rank
size =comm.Get_size()
print("my rank is:", rank)
start_time = MPI.Wtime()
print("start time is:",start_time)
k=2
```

```
dist_list=[]
count= np.zeros(k)
count = count.tolist()

if rank == 0:
    dir_list = os.listdir('C:\\Users\\saikiran\\Downloads\\20_newsgroups_test\\tf.idf')
    a = int(len(dir_list)/size)
    np.random.seed(0)
    mean = np.random.choice(dir_list, k, replace=False)
    mean_lst = mean_array('C:\\Users\\saikiran\\Downloads\\20_newsgroups_test\\tf.idf\\',mean)
    bcast_variable = mean_lst
    chunks = [dir_list[x:x+a] for x in range(0, len(dir_list), a)]
    scat_variable = [(chunks[i])for i in range(len(chunks))]
else:
    bcast_variable = None
    scat_variable = None
receive1 = comm.scatter(scat_variable, root=0)
receive2 = comm.bcast(bcast_variable,root=0)
global_mean=receive2
iteration = 0
while iteration<15:
    mean_update = [Counter() for n in range (k)]
    local_mean=[Counter() for n in range (k)]
    c1 =Counter()
    m=0
    n=0
```

```
    for each_file in receive1:
        filename = 'C:\\Users\\saikiran\\Downloads\\20_newsgroups_test\\tf.idf\\' +each_file
        tf_idf_result = open(filename, 'rt').read()
        tf_idf_score = Counter(json.loads(tf_idf_result))
        ed_distance = euclidean_distance(global_mean,tf_idf_score)
        min_pos = np.argmin(ed_distance)
        for i in range(k):
            if(i==min_pos):
                count[i]+=1
                mean_update[i] += tf_idf_score

    distortion = distortion_calculation(mean_update)
    dist_list = np.append(dist_list,distortion)
    root=0
    distortion_count = comm.reduce(dist_list,root=root,op=MPI.SUM)
    end_time = MPI.Wtime()
    execution_time = end_time-start_time
    print("total execultion time is:",execution_time)
    total_parallel_time = comm.reduce(execution_time,root=root,op=MPI.SUM)
    dist_diff=0
    if rank==root:
        data = distortion_count
        for i in range (len(data)):
            if i==0:
                print("iteration={} and total distortion={}".format(i,data[i]))
            else:
                dist_diff =data[i]-data[i-1]
                if dist_diff>0:
                    print("converged at iteration={}".format(i))
```

```
                break
            else:
                print("iteration={} and total distortion={}".format(i,data[i]))
    total_time = total_parallel_time
    print("total_parallel_execution time is=:",total_time)
if dist_diff>0:
    print("converged at iteration={}".format(i))
    break
for j in mean_update:
    for x in j:
        c1[x] = safe_div(j[x],count[m])
    local_mean[n]+=c1
    c1=Counter()
    if n<k:
        n=n+1
    if m<k:
        m=m+1
iteration=iteration+1
global_mean = local_mean
```

O/p for: mpiexec –n 4 python k-means_mpi.py (for cluster size k=2)

```
iteration=0 and total distortion=2804084.576684464
iteration=1 and total distortion=2650872.929284554
iteration=2 and total distortion=2647413.3692610553
iteration=3 and total distortion=2597245.874484931
iteration=4 and total distortion=2504759.2214887817
iteration=5 and total distortion=2348238.9488516767
iteration=6 and total distortion=2278421.0744834905
converged at iteration=7
total_parallel_execution time is=: 435.0195271926641
converged at iteration=7
```

b) <u>Speed up graph and parallel efficiency:</u>

In this exercise, I have calculated the speedup graph and parallel efficiency graph for the 20 newsgroup data using my implemented distributed k-mean algorithm in previous exercise. Speed up graph can be calculated as follows:

Speedup: $S = \frac{T_s}{T_p}$

Where T_s = Best serial execution time

P = # of processes

T_p = Execution time on P processes

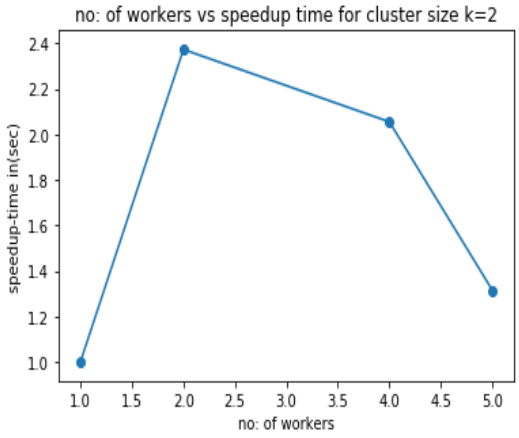
S_p = Speed up on P processes

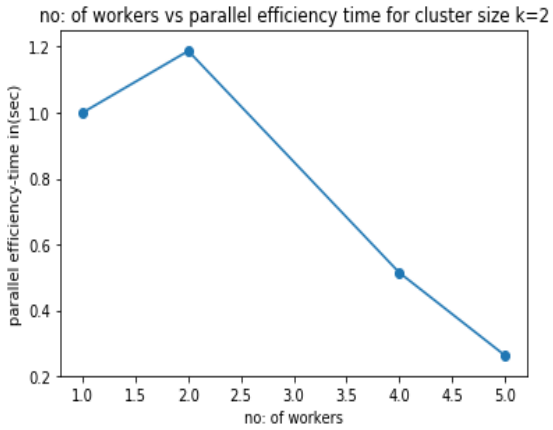
And parallel efficiency is calculated as: $E_p = \frac{S_p}{P}$

Note: I have calculated my results for just 2000 data since calculating T_s on large dataset using one single process takes a lot of time. As my computer can't handle calculation of such big dataset, Results are noted based on this 2000 data for different cluster sizes of k = 2,5,10 each among different workers p = 1,2, 4, 5. And results and graphs are noted as shown below:

For k = 2:

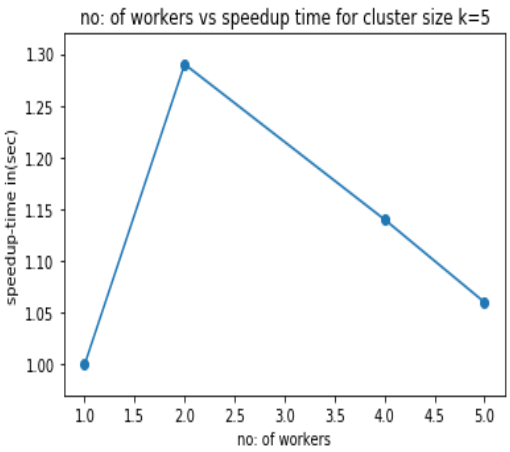
Data size	Cluster size:(k)	Serial Process Execution(Ts): In (sec)	No: parallel processes (p)	Total parallel execution time(Tp)in (sec)	Speed up(Sp) : Ts/Tp in(sec)	Parallel Efficiency (Ep) = (Sp/p) in (sec)
2000	2	853.437	1	853.437	1	1
2000	2	853.437	2	359.685	2.373	1.187
2000	2	853.437	4	415.268	2.055	0.514
2000	2	853.437	5	648.805	1.315	0.263

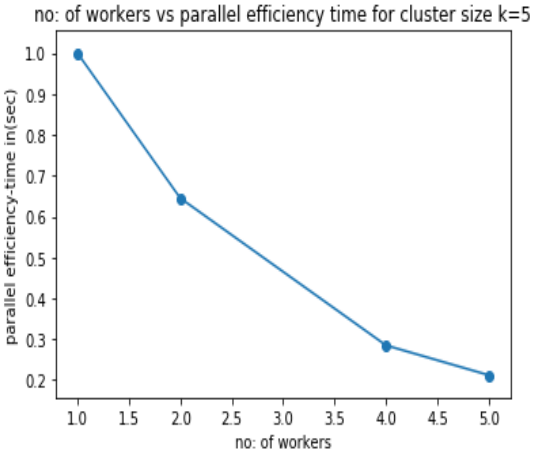




For k=5:

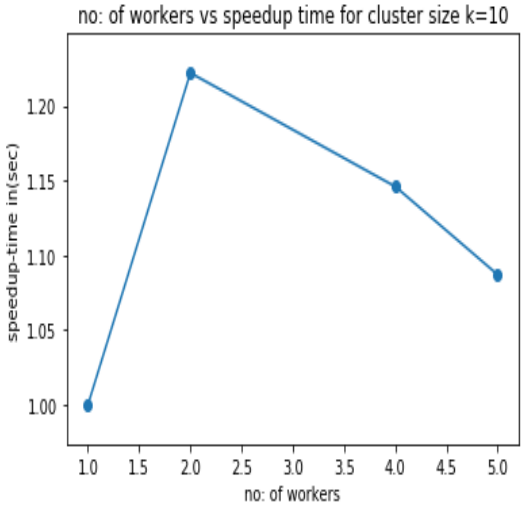
Data size	Cluster size:(k)	Serial Process Execution(Ts): In (sec)	No: parallel processes (p)	Total parallel execution time(Tp)in (sec)	Speed up(Sp) : Ts/Tp in(sec)	Parallel Efficiency (Ep) = (Sp/p) in (sec)
2000	5	679.154	1	679.154	1	1
2000	5	679.154	2	526.092	1.290	0.645
2000	5	679.154	4	591.928	1.14	0.285
2000	5	679.154	5	639.518	1.06	0.212

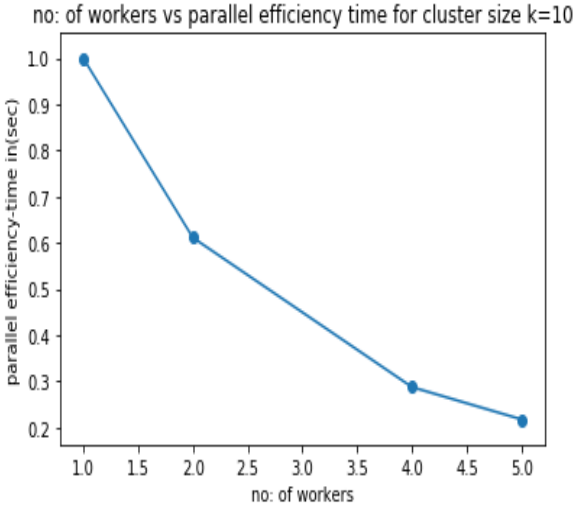




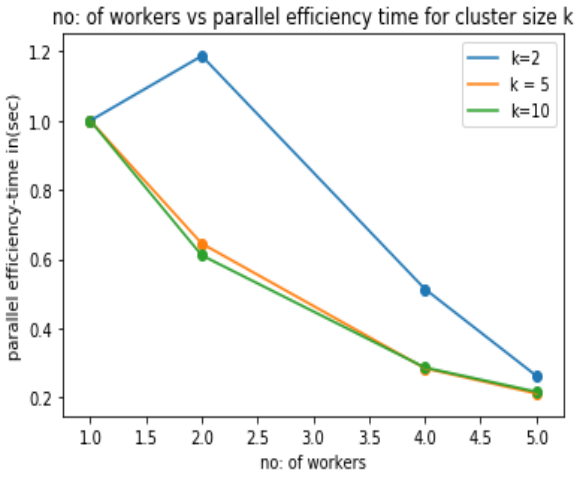
For k=10:

Data size	Cluster size:(k)	Serial Process Execution(Ts): In (sec)	No: parallel processes (p)	Total parallel execution time(Tp)in (sec)	Speed up(Sp) : Ts/Tp in(sec)	Parallel Efficiency (Ep) = (Sp/p) in (sec)
2000	10	766.977	1	766.977	1	1
2000	10	766.977	2	627.678	1.222	0.611
2000	10	766.977	4	669.416	1.146	0.287
2000	10	766.977	5	705.558	1.087	0.217





Overall performance comparison among all the clusters sizes:



My results have shown that parallel process takes less time when compared to single process and my performance is showing good results for p=2(i.e. two parallel workers) when compared to p=4 or p=5 as my lap top specification is capable of 2 cores as mentioned in my system specification above. My code could provide better execution time if ran on a better specification laptop.