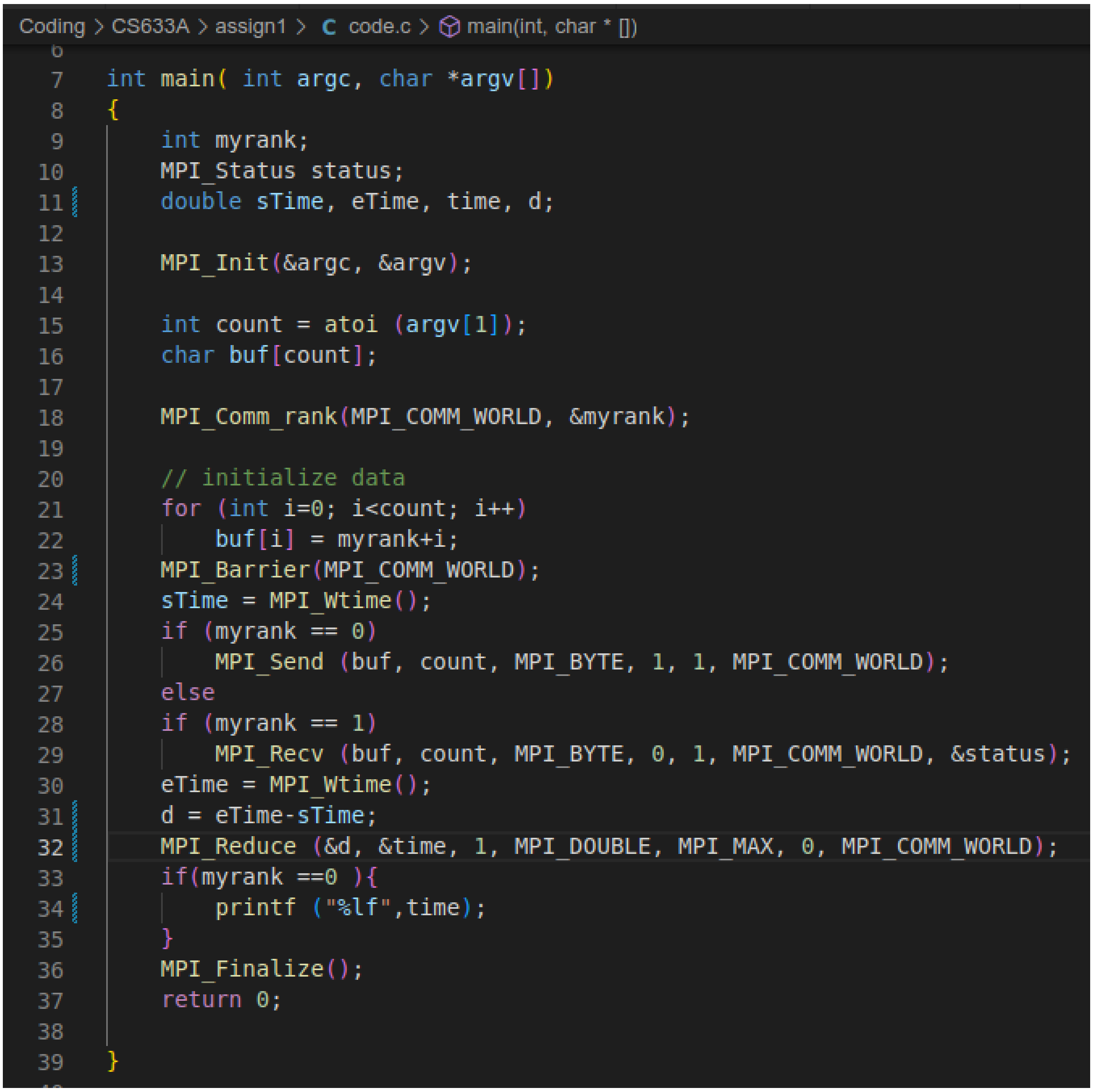
CS633 Assignment 1

Naman Singla(200619)
Nishant Roshan(200643)

Different files and their functions

C Code(MPI Library) used to send data and measure time:

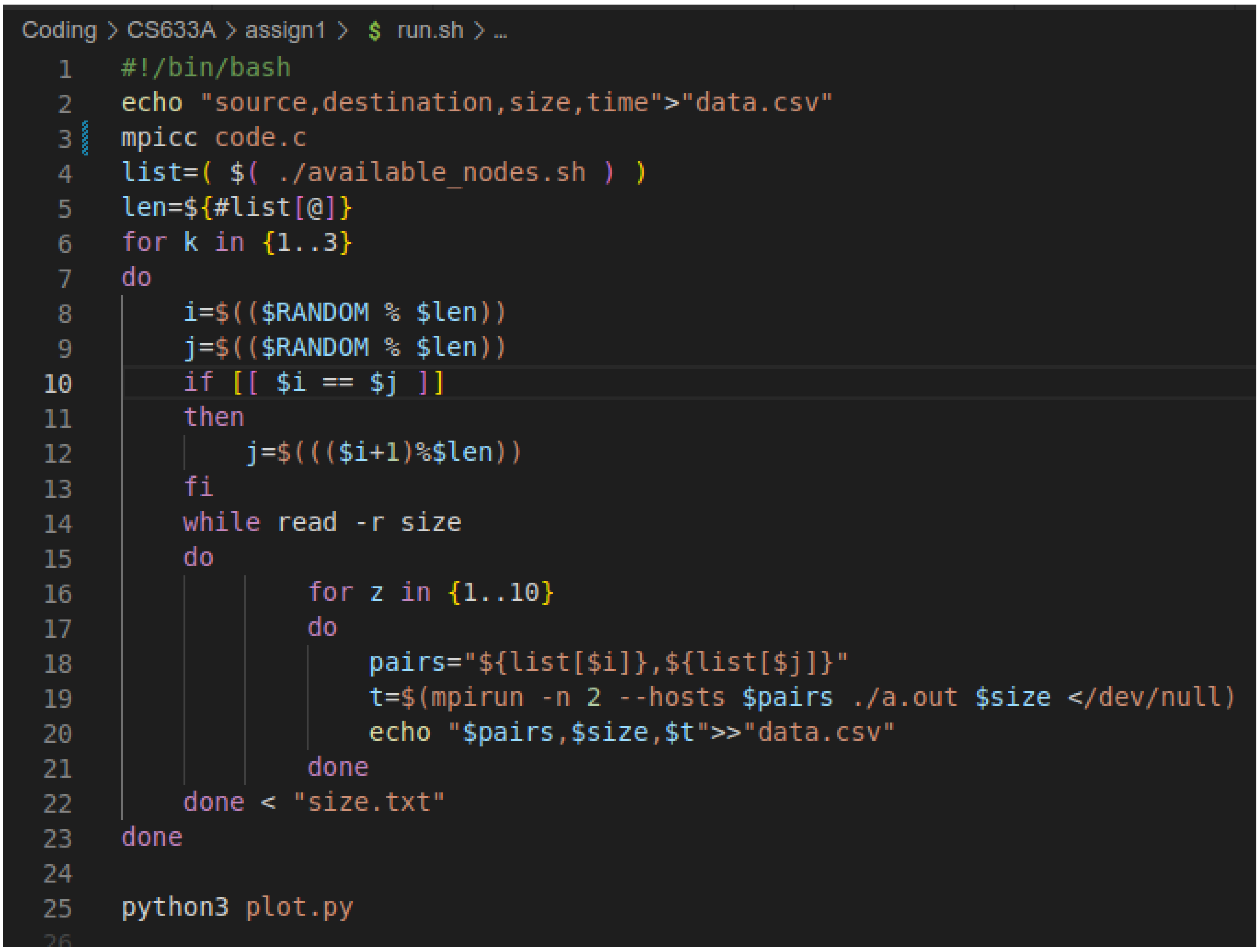


```c
int main( int argc, char *argv[])
{
    int myrank;
    MPI_Status status;
    double sTime, eTime, time, d;

    MPI_Init(&argc, &argv);

    int count = atoi (argv[1]);
    char buf[count];

    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

    // initialize data
    for (int i=0; i<count; i++)
        buf[i] = myrank+i;
    MPI_Barrier(MPI_COMM_WORLD);
    sTime = MPI_Wtime();
    if (myrank == 0)
        MPI_Send (buf, count, MPI_BYTE, 1, 1, MPI_COMM_WORLD);
    else
    if (myrank == 1)
        MPI_Recv (buf, count, MPI_BYTE, 0, 1, MPI_COMM_WORLD, &status);
    eTime = MPI_Wtime();
    d = eTime-sTime;
    MPI_Reduce (&d, &time, 1, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
    if(myrank ==0 ){
        printf ("%lf",time);
    }
    MPI_Finalize();
    return 0;

}
```

Note:

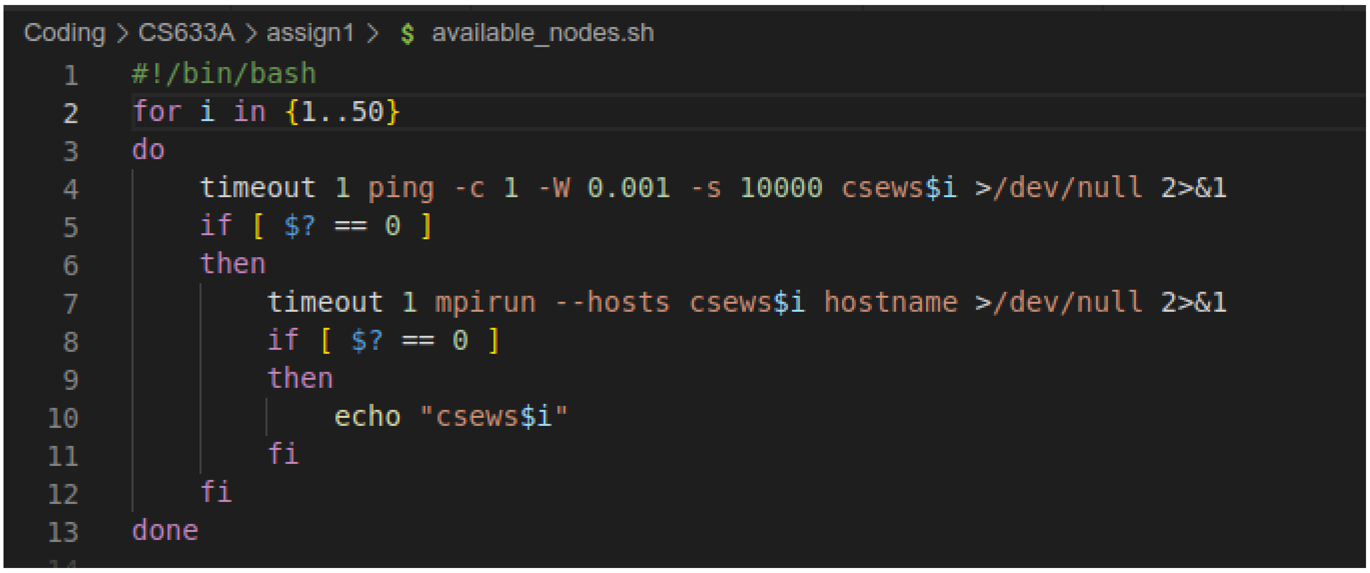
- To measure the communication time, we make sure both processes have same starting time and then one ending last will decide the total communication time. This will calculate the precise total communication time that was required for transfer.
- Rank =0 is taken as root
- Run “ulimit -s unlimited” for sending large data.

Run scripts:



Coding > CS633A > assign1 > $ run.sh > ...

```bash
#!/bin/bash
echo "source,destination,size,time">"data.csv"
mpicc code.c
list=( $( ./available_nodes.sh ) )
len=${#list[@]}
for k in {1..3}
do
    i=$(($RANDOM % $len))
    j=$(($RANDOM % $len))
    if [[ $i == $j ]]
    then
        j=$((($i+1)%$len))
    fi
    while read -r size
    do
            for z in {1..10}
            do
                pairs="${list[$i]},${list[$j]}"
                t=$(mpirun -n 2 --hosts $pairs ./a.out $size </dev/null)
                echo "$pairs,$size,$t">>"data.csv"
            done
    done < "size.txt"
done

python3 plot.py
```

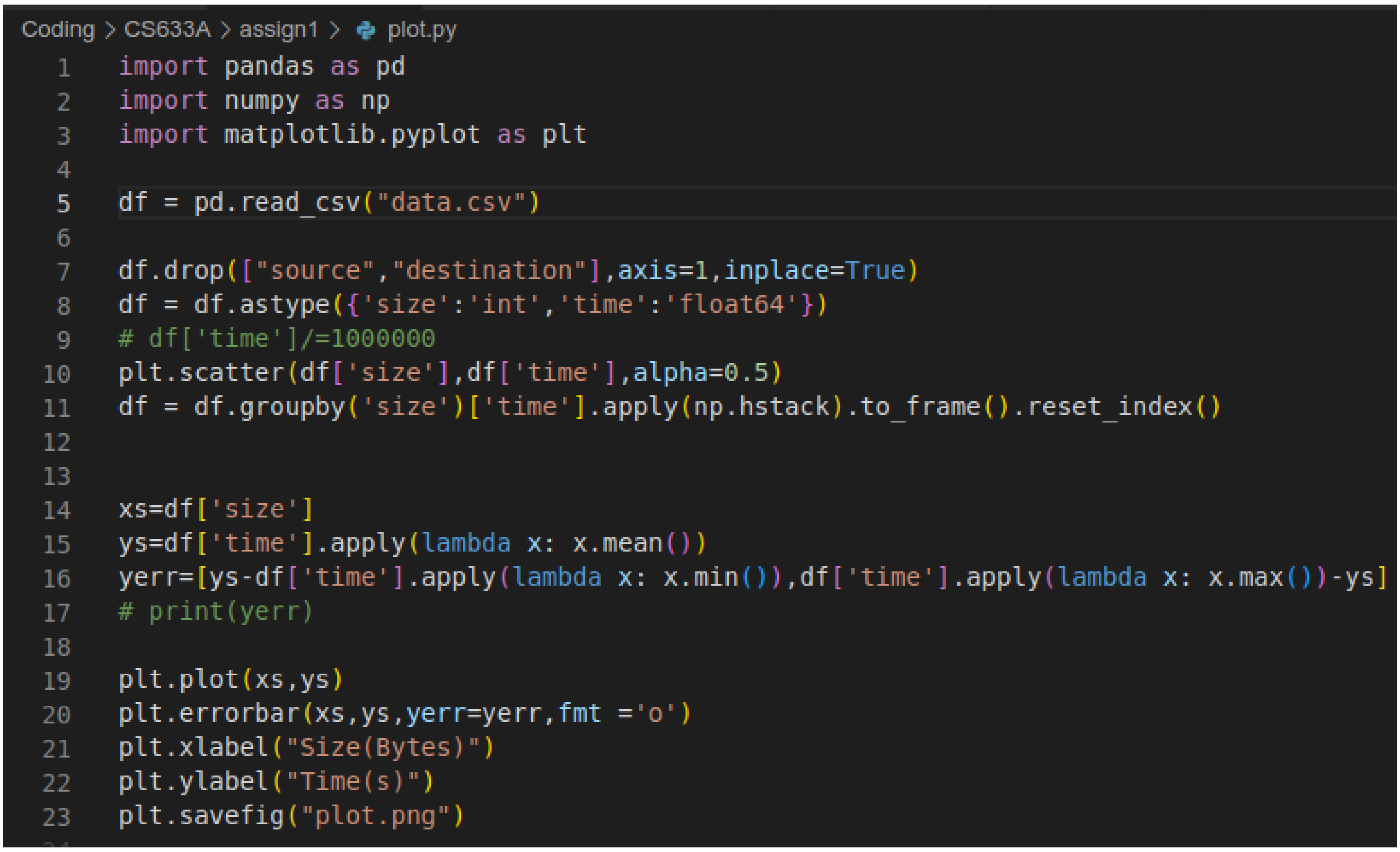


Coding > CS633A > assign1 > $ available_nodes.sh

```bash
#!/bin/bash
for i in {1..50}
do
    timeout 1 ping -c 1 -W 0.001 -s 10000 csews$i >/dev/null 2>&1
    if [ $? == 0 ]
    then
        timeout 1 mpirun --hosts csews$i hostname >/dev/null 2>&1
        if [ $? == 0 ]
        then
            echo "csews$i"
        fi
    fi
done
```

Note:

- available_nodes.sh produces a list that contains online csews pcs that can run mpi
- size.txt contains different sizes for which we want to acquire data.
- run.sh has following operations:
    - Compile the C code(code.c) with mpi.
    - Gets the list of online csews pcs
    - Selects 3 random pair of pc then execute code with different sizes for each pair 10 times.
    - File “data.csv” is generated which contains timings for different configurations.
    - Python script “plot.py” plots the data obtained and produces “plot.png”



```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.read_csv("data.csv")

df.drop(["source","destination"],axis=1,inplace=True)
df = df.astype({'size':'int','time':'float64'})
# df['time']/=1000000
plt.scatter(df['size'],df['time'],alpha=0.5)
df = df.groupby('size')['time'].apply(np.hstack).to_frame().reset_index()


xs=df['size']
ys=df['time'].apply(lambda x: x.mean())
yerr=[ys-df['time'].apply(lambda x: x.min()),df['time'].apply(lambda x: x.max())-ys]
# print(yerr)

plt.plot(xs,ys)
plt.errorbar(xs,ys,yerr=yerr,fmt ='o')
plt.xlabel("Size(Bytes)")
plt.ylabel("Time(s)")
plt.savefig("plot.png")
```

<u>How to run the submission?</u>

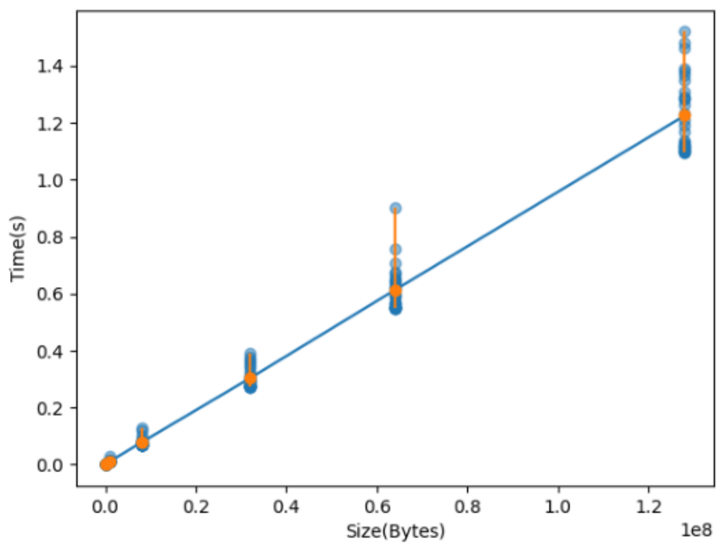
- Make sure all the above-mentioned files are present in the current working directory.
- Run the bash script “run.sh”
- Output plot will be stored in files plot.png

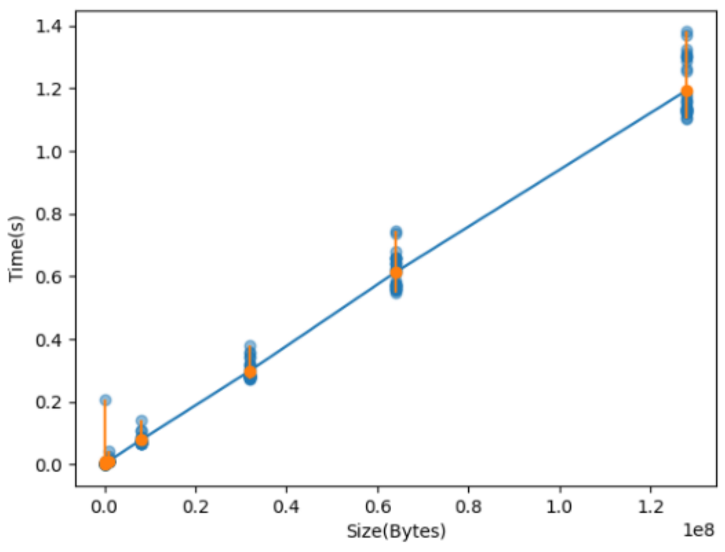
<u>Experimentation Methodology</u>

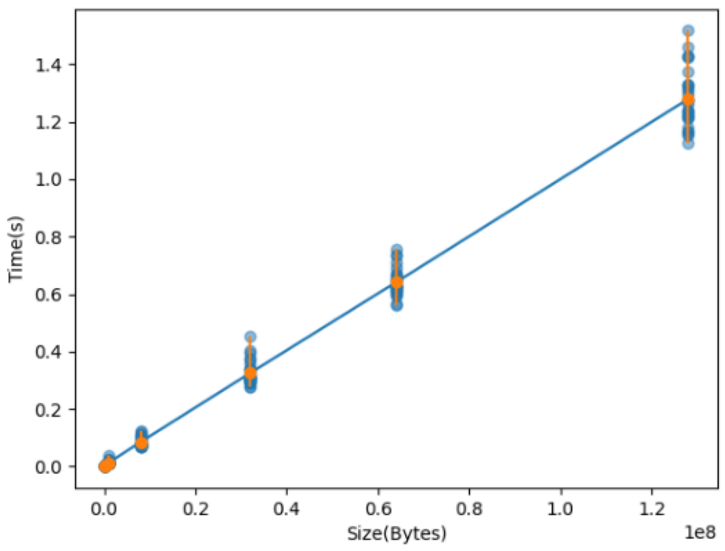
The complete execution of the process is present in “run.sh”.

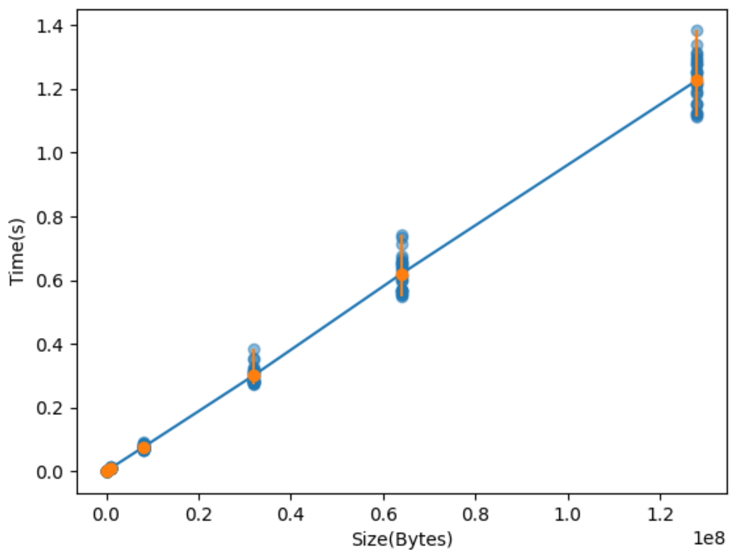
1. Get the list of online nodes the have a reliable connection and can run mpi, using “available_nodes.sh”
2. Select a pair of nodes randomly from above obtained list.
3. Perform MPI_Send and MPI_Recv operations between above selected nodes.
    a. Put a barrier just before send and recv to make sure start time is same for both processes.
    b. Perform the transfer and note down the end time.
    c. Take the difference of start time and the end time of process.
    d. Take maximum of above differences obtained for both processes.
    e. Value obtained above denotes the total time taken for communication, i.e both send and recv start at same time and the one ending last defines the total time taken. This time denotes that the complete data has been sent and received.
4. Perform Step 2,3 for each size mentioned in “size.txt”. Take 10 readings for each size.
5. Perform 2-4 for 3 different pairs of nodes.
6. The data obtained in stored in csv file “data.csv”.
7. We can plot the data obtained using matplotlib, numpy and pandas. The obtained plot is stored in “plot.png”

Different plots obtained during experimentation and observation









1. The average time taken linearly varies with the size of the data sent.
   Explanation: This is evident if we assume a constant data rate. This may not be true for the small amount of data. But for large data sizes, data rates remain constant over the whole transfer span.
2. Error/deflections from expected behavior linearly increase with the amount of data sent(size).
   Explanation: Since data recorded for different sizes come from the same pair of nodes, we can say that different nodes may have different but constant data rates. Hence, we can claim that the deviation observed for a particular size can be scaled to get the approximate deviations of some other size value.
3. Upper deflection(+ve error) increases faster than lower deflection, i.e., the difference between the slope of max-time and mid-time is much more than that between the slopes of mid-time and min-time.
   Explanation: Maximum processes take the expected or near the expected completion time. Some processes experience high latency or lower bandwidth issues, so their time taken spikes high. But due to the number dominance average is only shifted slightly above the min.
4. In some cases, some outliers are observed(Their reading deflects by a large amount from the expected behavior).
   Explanation: These are cases when the daemon might be busy handling other users' requests, or sometimes the initialization cost goes high. These cases usually show a large increase in time. There is very less chance of an outlier being in lower deflection.
5. For each run slope of the line comes out to be approximately the same; only the error bars change.
   Explanation: The slope of the line denotes the data rate of transfer. Now, since all pcs are present on the same network using the same connections to the main network, we can say that their data rate is the same.