Computer Networks Innovative Project

Implementation of a Beowulf Cluster and Analysis of its Performance in Applications with Parallel Programming Using OpenSSH, MPICH2 and Python Scapy Library

A. ABSTRACT

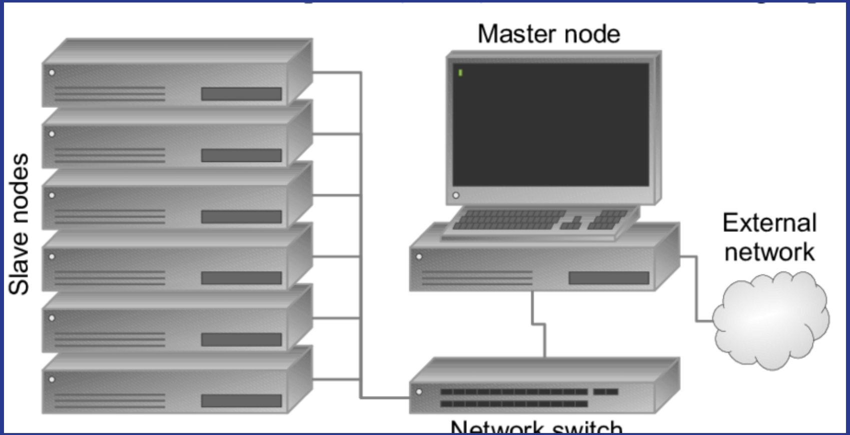
The project aims to establish a local Beowulf Cluster between a Laptop and a Desktop Computer running Ubuntu Desktop and further analysis of the Networking comp. using Scapy (Python Library for Network Monitoring) and MPI (Message Passing Interface) for running Parallel Programmes. We decided to use a laptop as Master and PC as Slave Node in the Local Beowulf Cluster.

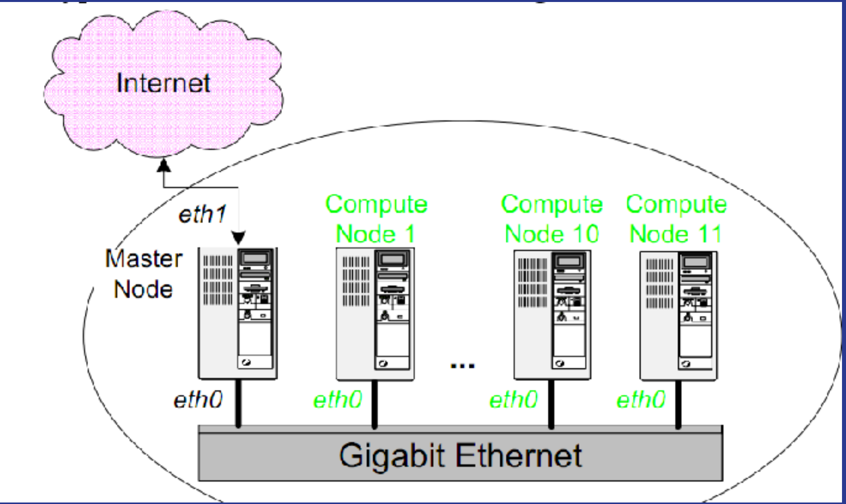
B. What is a Beowulf Cluster ?

Beowulf Clusters are a group of Identical, commercially available computers, that run and operate on Free and Open Source Softwares generally like BSD, Solartis, Unix and in our case we have used GNU/Linux Debian derived Ubuntu Desktop. All the individual nodes are networked into a small TCP/IP LAN and have libraries and programs installed which allow processing to be shared among them.

C. Local Cluster.

A local cluster is a Beowulf Cluster which is connected to public network via Master Node and all other slave nodes are connected to the private network created using LAN and all the control lies with Master Node.



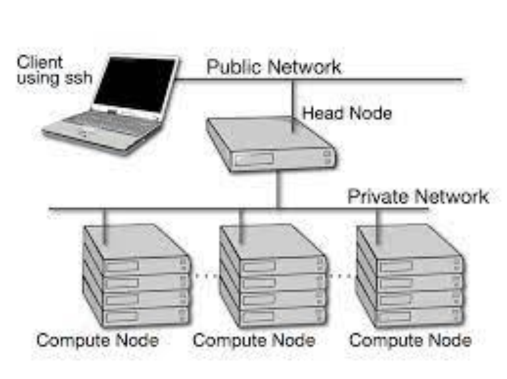


BEOWULF CLUSTER: Methodology in GNU/Linux

A.The most popular ways of building these Beowulf Cluster are using Beowulf Toolkits like **NPACI Rocks Toolkit and OSCAR Toolkit by Red Hat Linux Foundations**, these toolkits are intuitive to use and deploy clusters and require minimal expertise from the end user. So as part of this Innovative MTE Project evaluation, we decided to build one such cluster **from scratch by manually setting up both the machines** by having them run Ubuntu desktop and make them share the same Home Directory NFS and labelling Laptop as Master Node in the Local Cluster and PC as Slave Node..

B.**NFS stands for Network File System** which is a distributed File System Protocol, allowing a user on a client computer to access files over a computer network much like local storage is accessed. NFS, like many other protocols, builds on the **Open Network Computing Remote Procedure Call (ONC RPC) system.** Remote Shells (rsh) are command line computer programs that can execute shell commands as another use and on another computer through the use of a computer network.

1. The cluster consists of the following hardware parts : Network, Master/Server Node, Computer Node and a Gateway. After getting all hardware components ready, we should be configuring the LAN and creation of Nodes, which should add nodes to the hosts files, so that we need not to type in IP addresses each time but only the username.



2.Then we should ping from the master nodes to all nodes to check whether the data and acknowledgement is being received or not.Next step is setting up Secure Remote Shells for communication between nodes and setting up Process Manager and Message Passing Interface Process Running Jobs and analyzing the parallel computations using the clusters instead of a single computer.

BEOWULF CLUSTER: Implementation GNU/Linux

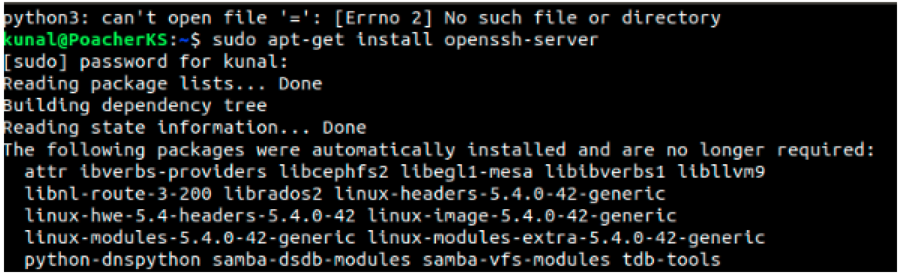
```
milk.txt                                        xaa
music                                           ytrewq.txt
kunal@PoacherKS:~$ sudo -i
root@PoacherKS:~# sudo adduser clusterMaster
adduser: Please enter a username matching the regular expression configured
via the NAME_REGEX[_SYSTEM] configuration variable.  Use the `--force-badname'
option to relax this check or reconfigure NAME_REGEX.
root@PoacherKS:~# sudo adduser 'clusterMaster'
adduser: Please enter a username matching the regular expression configured
via the NAME_REGEX[_SYSTEM] configuration variable.  Use the `--force-badname'
option to relax this check or reconfigure NAME_REGEX.
root@PoacherKS:~# sudo adduser master
Adding user `master' ...
Adding new group `master' (1004) ...
Adding new user `master' (1004) with group `master' ...
Creating home directory `/home/master' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for master
Enter the new value, or press ENTER for the default
        Full Name []: beowulfMasterNode
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n]
root@PoacherKS:~# ls /etc/shadow
/etc/shadow
```

All the required hardware components were set up, Laptop us running Ubuntu Desktop and PC is running Windows Subsystem for Linux Ubuntu Terminal. We created a new user named **clusterMaster** which will control all the working of the cluster as the Master Node

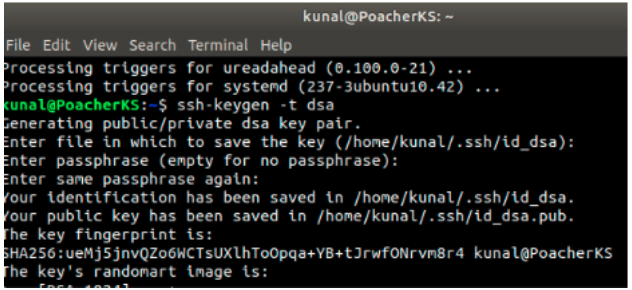
```
kunal:$6$W.ZyNyjs$j0uxlUxEb8xRl5Zqic8oSe3/1r6v.1fVz6CTKXC
5kCop7AFcMcH0:18370:0:99999:7:::
flinuxmaster:!:18459:0:99999:7:::
kos:!:18462:0:99999:7:::
beowulfUser:$6$NV.AlXfk$//xYu6VOUF8K8OsLBMgzCZTGIcWqRdB66
BmJGhxbQ8AOtIsywMf0:18693:0:99999:7:::
sshd:*:18693:0:99999:7:::
master:$6$lFBFwx4c$3tuauV2gAMKcR3pxgJTndMppQ3icPTgKZ7.bec
/C4Iv2haOWJOA1:18731:0:99999:7:::
(END)
```

```
127.0.0.1       localhost
127.0.1.1       PoacherKS

# The following lines are desirable
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

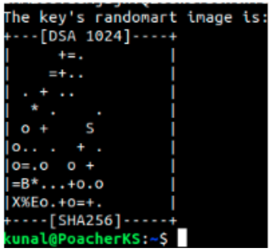
Then we configured the static IP addresses so that we can refer nodes using names instead of their IP addresses. We use YAML pkg manager to configure the Network file and extract the file to know more information about our Ethernet cable which was Cat5 10/100 MBPS unshielded cable. This helped us to issue commands using the computer names.



```
python3: can't open file '=': [Errno 2] No such file or directory
kunal@PoacherKS:~$ sudo apt-get install openssh-server
[sudo] password for kunal:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  attr ibverbs-providers libcephfs2 libegl1-mesa libibverbs1 libllvm9
  libnl-route-3-200 librados2 linux-headers-5.4.0-42-generic
  linux-hwe-5.4-headers-5.4.0-42 linux-image-5.4.0-42-generic
  linux-modules-5.4.0-42-generic linux-modules-extra-5.4.0-42-generic
  python-dnspython samba-dsdb-modules samba-vfs-modules tdb-tools
```



```
kunal@PoacherKS: ~
File  Edit  View  Search  Terminal  Help
Processing triggers for ureadahead (0.100.0-21) ...
Processing triggers for systemd (237-3ubuntu10.42) ...
kunal@PoacherKS:~$ ssh-keygen -t dsa
Generating public/private dsa key pair.
Enter file in which to save the key (/home/kunal/.ssh/id_dsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/kunal/.ssh/id_dsa.
Your public key has been saved in /home/kunal/.ssh/id_dsa.pub.
The key fingerprint is:
SHA256:ueMj5jnvQZo6WCTsUXlhToOpqa+YB+tJrwfONrvm8r4 kunal@PoacherKS
The key's randomart image is:
```



```
The key's randomart image is:
+---[DSA 1024]----+
|       +=.       |
|      =+..       |
|   . + ..        |
|    *     .      |
|  o +     S      |
|o..  .   + .     |
|o=.o   o +       |
|=B*...+o.o       |
|X%Eo.+o=+.       |
+----[SHA256]-----+
kunal@PoacherKS:~$
```

Installation of OpenSSH-Server.

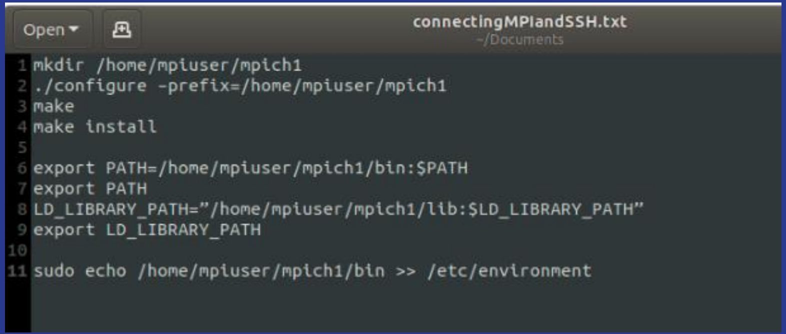
We installed openssh-server to establish remote control and transferring of data over internet options.

Setting Key for OpenSSH-Server Connection

We assigned both the nodes Secure Keys ssh-key-t-dsa commands available in OpenSSH Server options, the keys are added for encrypted data and authentication purposes.

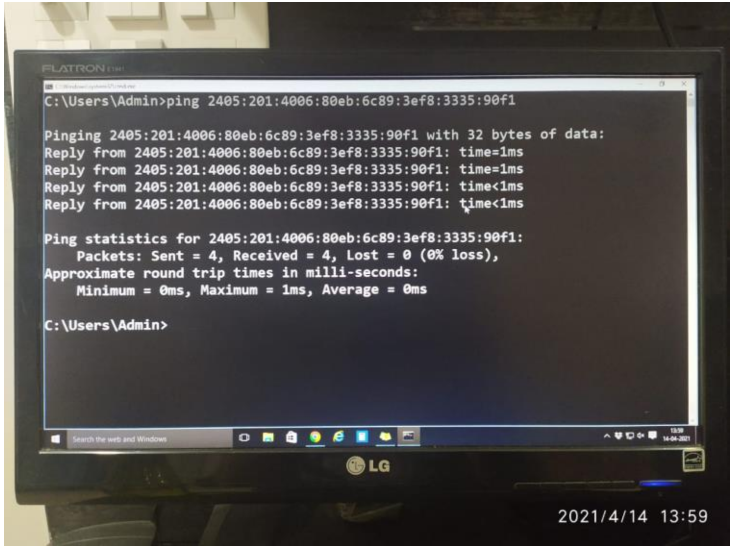
Connection of MPI and SSH

Then we installed MPICH2 which is the MPI-2 Implementation from UNIX labs, the process involved downloading of C compiler (gcc) and the .tar file from UNIX labs website, we downloaded the files and using mpich2-install commands, installed the library and then we ran the script displayed above for connecting MPI and SSH, so that we can use this MPI generated to pass on the messages to other networked node which is using Secure Remote Shells Login we created above. Using VIM text editor in Terminal we went to file in FS in /etc/hosts dir and added the IP of the to be passed node, in our case it is Computer (Slave Node) and then in Computer we used command mpd -h <master> -p 4000 & to login into our master node forms server node. Then any file can be sent over using the terminal command mpiexec -n 4 /bin/hostname.



```
Open ▾   connectingMPIandSSH.txt
         ~/Documents
mkdir /home/mpiuser/mpich1
./configure -prefix=/home/mpiuser/mpich1
make
make install

export PATH=/home/mpiuser/mpich1/bin:$PATH
export PATH
LD_LIBRARY_PATH="/home/mpiuser/mpich1/lib:$LD_LIBRARY_PATH"
export LD_LIBRARY_PATH

sudo echo /home/mpiuser/mpich1/bin >> /etc/environment
```

BEOWULF CLUSTER: Connections in GNU/Linux



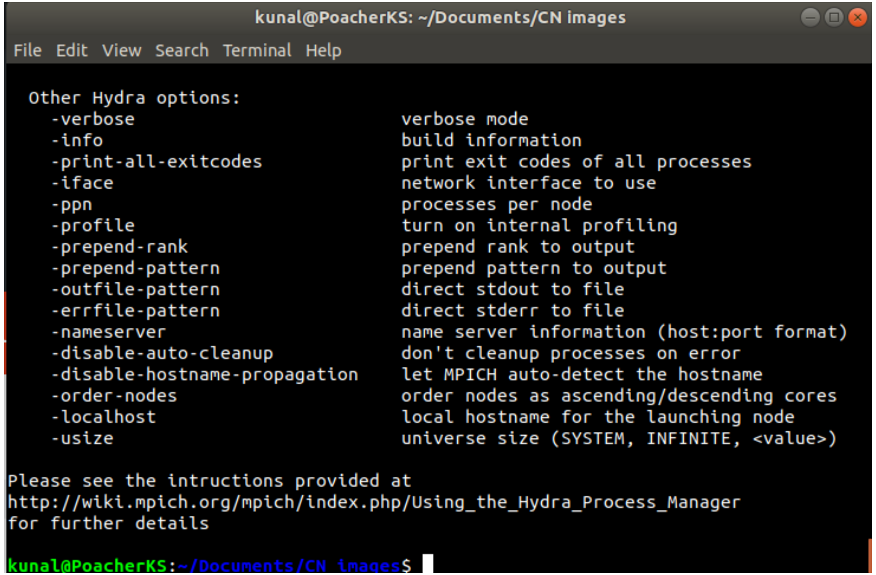
We added new users on both master and slave nodes as “master” and “slave” using [sudo useradd <username>] command and setting up the process for them, here you can see the snapshot of Master Node User creation and setup. Similarly another user as “slave” is created on Windows 10 PC using WSL for GNU/Linux Ubuntu. We issued ping commands like ping <ipv4 address>, and the PC (slave) returned us 52 bytes of data. This confirms that LAN is established and our nodes are interacting.

When we tried issuing commands from MPI and SSH to our Slave Node from Master node data is received in packets of size 32 Bytes, to confirm whether the data being sent over is from Slave node, we tried sending data from Slave node and the result was Successful, indeed the nodes were connected as in LAN and MPI-SSH was working.

```
master:~$ sudo apt-get install nfs-kernel-server
$ sudo apt-get install nfs-common
master:~$ ls -l /home/ | grep mpiuser
master:~$ sudo chown mpiuser:mpiuser /path/to/shared/dir
/home/mpiuser *(rw,sync,no_subtree_check)
master:~$ sudo service nfs-kernel-server restart
master:~$ sudo ufw allow from 192.168.1.0/24
$ ls /home/mpiuser
```

```
python3: can't open file '=': [Errno 2] No such file or directory
kunal@PoacherKS:~$ sudo add-apt-repository ppa:deadsnakes/ppa
[sudo] password for kunal:
 This PPA contains more recent Python versions packaged for Ubuntu.

Disclaimer: there's no guarantee of timely updates in case of security problems or other issues. I
f you want to use them in a security-or-otherwise-critical environment (say, on a production serve
r), you do so at your own risk.

Update Note
===========
Please use this repository instead of ppa:fkrull/deadsnakes.

Reporting Issues
================

Issues can be reported in the master issue tracker at:
https://github.com/deadsnakes/issues/issues

Supported Ubuntu and Python Versions
====================================

- Ubuntu 16.04 (xenial) Python 2.3 - Python 2.6, Python 3.1 - Python3.4, Python 3.6 - Python3.9
- Ubuntu 18.04 (bionic) Python2.3 - Python 2.6, Python 3.1 - Python 3.5, Python3.7 - Python3.9
- Ubuntu 20.04 (focal) Python3.5 - Python3.7, Python3.9
- Note: Python2.7 (all), Python 3.5 (xenial), Python 3.6 (bionic), Python 3.8 (focal) are not prov
ided by deadsnakes as upstream ubuntu provides those packages.
- Note: for focal, older python versions require libssl1.0.x so they are not currently built

The packages may also work on other versions of Ubuntu or Debian, but that is not tested or suppor
ted.
```

BEOWULF CLUSTER: Testing GNU/Linux


```
kunal@PoacherKS: ~/Documents/CN images
File  Edit  View  Search  Terminal  Help

 Other Hydra options:
   -verbose                         verbose mode
   -info                            build information
   -print-all-exitcodes             print exit codes of all processes
   -iface                           network interface to use
   -ppn                             processes per node
   -profile                         turn on internal profiling
   -prepend-rank                    prepend rank to output
   -prepend-pattern                 prepend pattern to output
   -outfile-pattern                 direct stdout to file
   -errfile-pattern                 direct stderr to file
   -nameserver                      name server information (host:port format)
   -disable-auto-cleanup            don't cleanup processes on error
   -disable-hostname-propagation    let MPICH auto-detect the hostname
   -order-nodes                     order nodes as ascending/descending cores
   -localhost                       local hostname for the launching node
   -usize                           universe size (SYSTEM, INFINITE, <value>)

Please see the intructions provided at
http://wiki.mpich.org/mpich/index.php/Using_the_Hydra_Process_Manager
for further details

kunal@PoacherKS:~/Documents/CN images$
```

Now to access files and not just send data we used the above commands as mentioned in the Blog Post on Building Cluster by Serrano Pereira in order to share the NFS /home directory.

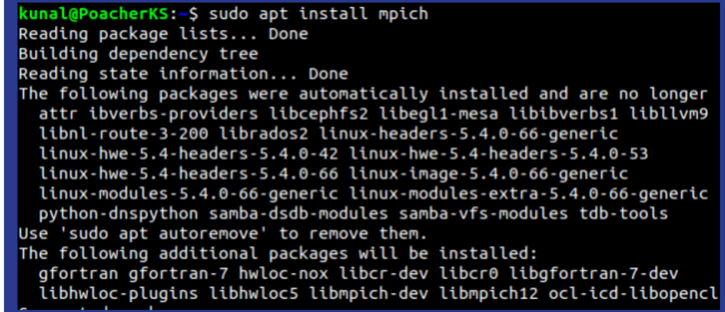
Since both nodes are having their independent processes being run, we need some sort of process manager so we used Hydra Process manager which is configured for uses of MPICH.

The penultimate step now is to run a parallel process and test whether the parallel processing has helped us to get better in computation and solve more challenging tasks by creating a low-cost supercomputer. For selecting of what process we needed, we wrote a small C++ Source Code which runs and does the same thing from different sources that finally prints the rank of the same process executed from different sources, we took 5 cases out of which the 3rd case was when we passed the file using MPI, that processes executed the fastest and took the least amount of time as compared to the time taken by the same process when executed solely on Master or Slave Node.

To run the above mentioned code, we installed from the help of package manager mpich code runner which builds and executes the C++ Code using Mingw g++ compiler and MPI.

> mpicc ParallelProcess.cpp -o ParallelProcess

> mpirun -np 5 ./ParallelProcess


```
kunal@PoacherKS:~$ sudo apt install mpich
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer
  attr ibverbs-providers libcephfs2 libegl1-mesa libibverbs1 libllvm9
  libnl-route-3-200 librados2 linux-headers-5.4.0-66-generic
  linux-hwe-5.4-headers-5.4.0-42 linux-hwe-5.4-headers-5.4.0-53
  linux-hwe-5.4-headers-5.4.0-66 linux-image-5.4.0-66-generic
  linux-modules-5.4.0-66-generic linux-modules-extra-5.4.0-66-generic
  python-dnspython samba-dsdb-modules samba-vfs-modules tdb-tools
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  gfortran gfortran-7 hwloc-nox libcr-dev libcr0 libgfortran-7-dev
  libhwloc-plugins libhwloc5 libmpich-dev libmpich12 ocl-icd-libopencl
```

~/Documents/CN images/ParallelProcess.cpp • - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

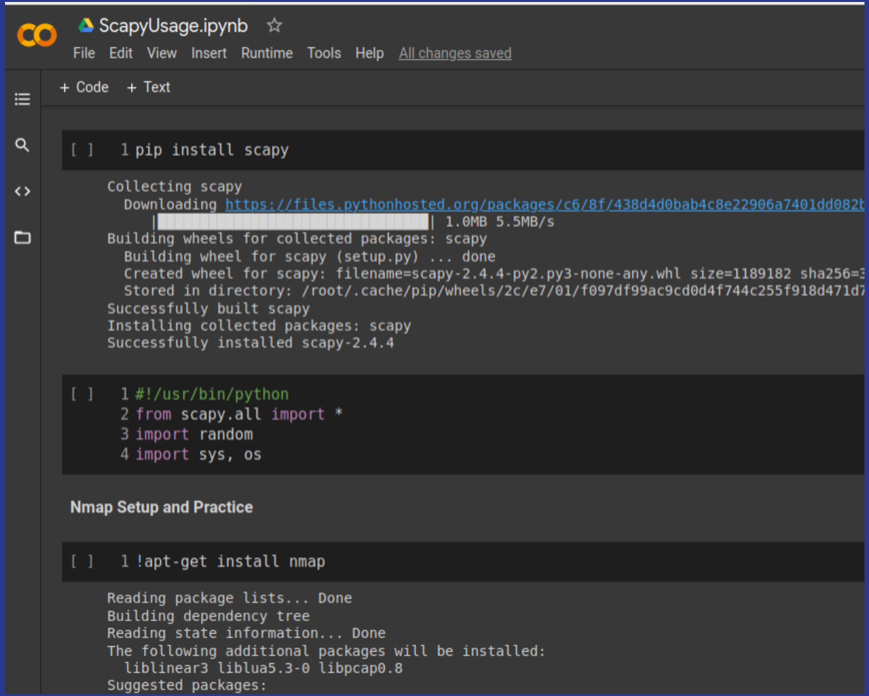
ParallelProcess.cpp

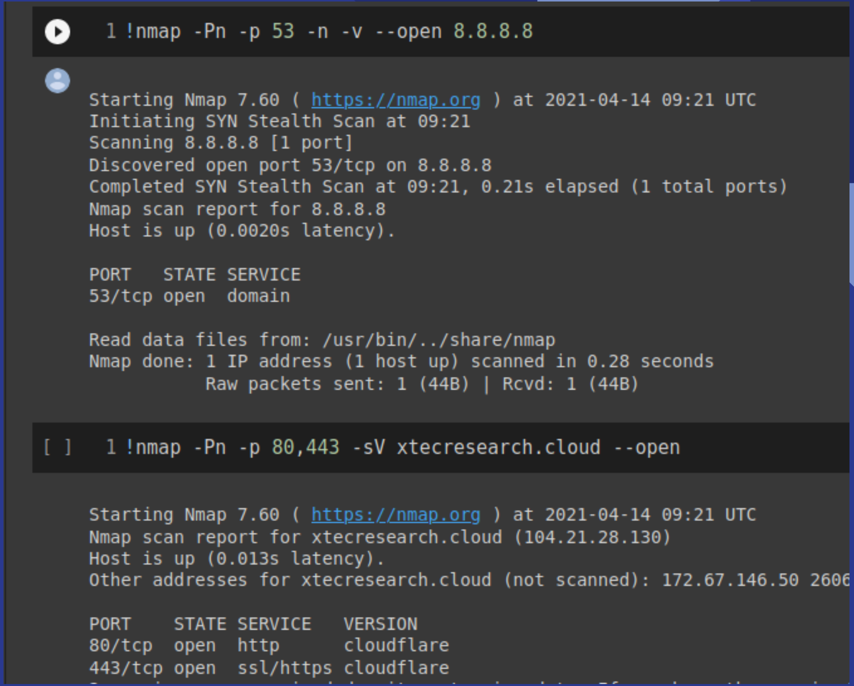
```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    // Initialize the MPI environment
    MPI_Init(NULL, NULL);

    // Get the number of processecs
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // Get the rank of the process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    // Get the name of the processor
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name, &name_len);

    // Print off a hello world message
    printf("Computer Networks Innovative Project %s, rank %d out of %d process
           processor_name, world_rank, world_size);

    // Finalize the MPI environment.
    MPI_Finalize();
}
```

```
/* Process Creation and Management */
int MPI_Close_port(char *);
int MPI_Comm_accept(char *, MPI_Info, int, MPI_Comm, MPI_Comm *);
int MPI_Comm_connect(char *, MPI_Info, int, MPI_Comm, MPI_Comm *);
int MPI_Comm_disconnect(MPI_Comm *);
int MPI_Comm_get_parent(MPI_Comm *);
int MPI_Comm_join(int, MPI_Comm *);
int MPI_Comm_spawn(char *, char *[], int, MPI_Info, int, MPI_Comm,
                   int []);
int MPI_Comm_spawn_multiple(int, char *[], char **[], int [], MPI_
                            MPI_Comm, MPI_Comm *, int []);
int MPI_Lookup_name(char *, MPI_Info, char *);
int MPI_Open_port(MPI_Info, char *);
int MPI_Publish_name(char *, MPI_Info, char *);
int MPI_Unpublish_name(char *, MPI_Info, char *);

/* One-Sided Communications */
int MPI_Accumulate(void *, int, MPI_Datatype, int, MPI_Aint, int,
                   MPI_Datatype,  MPI_Op, MPI_Win);
int MPI_Get(void *, int, MPI_Datatype, int, MPI_Aint, int, MPI_Dat
            MPI_Win);
int MPI_Put(void *, int, MPI_Datatype, int, MPI_Aint, int, MPI_Dat
            MPI_Win);
int MPI_Win_complete(MPI_Win);
int MPI_Win_create(void *, MPI_Aint, int, MPI_Info, MPI_Comm, MPI_
int MPI_Win_fence(int, MPI_Win);
int MPI_Win_free(MPI_Win *);
int MPI_Win_get_group(MPI_Win, MPI_Group *);
int MPI_Win_lock(int, int, int, MPI_Win);
int MPI_Win_post(MPI_Group, int, MPI_Win);
int MPI_Win_start(MPI_Group, int, MPI_Win);
int MPI_Win_test(MPI_Win, int *);
int MPI_Win_unlock(int, MPI_Win);
int MPI_Win_wait(MPI_Win);
```

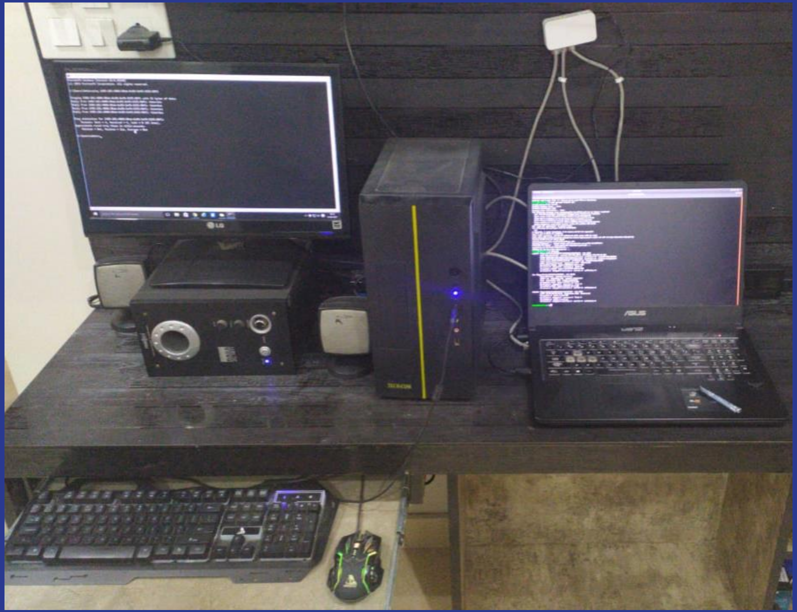
```
kunal@PoacherKS:~/Documents/CN images$ mpicc ParallelProcess.cpp -o ParallelProc
ess
kunal@PoacherKS:~/Documents/CN images$ mpirun -np 5 ./ParallelProcess
Computer Networks Innovative Project PoacherKS, rank 3 out of 5 processors
Computer Networks Innovative Project PoacherKS, rank 4 out of 5 processors
Computer Networks Innovative Project PoacherKS, rank 0 out of 5 processors
Computer Networks Innovative Project PoacherKS, rank 1 out of 5 processors
Computer Networks Innovative Project PoacherKS, rank 2 out of 5 processors
kunal@PoacherKS:~/Documents/CN images$ mpiexec -help

Usage: ./mpiexec [global opts] [local opts for exec1] [exec1] [exec1 args] : [lo
cal opts for exec2] [exec2] [exec2 args] : ...

Global options (passed to all executables):

  Global environment options:
    -genv {name} {value}             environment variable name and value
    -genvlist {env1,env2,...}        environment variable list to pass
    -genvnone                        do not pass any environment variables
    -genvall                         pass all environment variables not managed
                                          by the launcher (default)

  Other global options:
    -f {name}                        file containing the host names
```

We can see the outcome of compiling the previously mentioned codes, the C++ Source Code is passed to 5 different channels, some of them involve testing on only one single node at a time, some of them involved partial sharing some of them involve equal sharing. As we can see here that the call number 3rd comes out as Rank 0 which means minimum time amongst all of the processes, that shows that our Locally created Beowulf Cluster has the fastest execution of the processes involved.





The final step of Project is to use the Python Scapy library to inspect the network elements. In this step we worked using Google Colab Notebook to do the network tests. We used Wireshark which is a famous Network Protocol Analyzer Tool. We analyzed that the ports we opened specifically for the operation of MPI are the only ports that are opened alongside standard HTTP makes sure that we are in connection with the Slave node and all the packets going to Slave nodes through the DLink Router are actually going to the node, and are returned once the node processes its amount of program.

Results
In this project we aimed to establish a local Beowulf Cluster and using MPI and SSH aimed at creating a low-cost supercomputer that can run the processes faster then what single machines can do, after the implementation phase we have completed the procedure and achieved faster computation speed. Finally We ran a parallel process that prints a simple String literal in C++, in it we found that by using Cluster’s computational power, the execution was way faster as compared to what happens when we ran the program individually on nodes. Hence the creation of our low-cost Supercomputer like Beowulf Cluster was a success.

Conclusion

1.To conclude the project, we can say that the execution of programmes that require parallel computations is much faster on a clustered system, like in the case of the above constructed Beowulf Cluster.

2. The programmes run much faster on them as compared to their time requirements on single node computers. Beowulf clusters can communicate among themselves using Open-SSH and Message Passing Interface which is a special tool for communication within nodes in a Private Network. Hence to increase productivity we can use 2 simple computers, merge them in a cluster, a workable Supercomputer.

References

1. What's a Beowulf for ?

2. Beowulf Cluster Computing with Linux. Book

3. Building Your own Beowulf Cluster, Wired Article.

4. Implementation of a Beowulf Cluster. Research Paper.

Thanks Everyone