Practical 2: MPI setup and sample code

GAHAN SARAIYA (18MCEC10)

18mcec10@nirmauni.ac.in

I. AIM

- Setup MPI
  - Write steps for MPI setup
  - List problem observed during setup and execution
  - Specify cause and solution of each problem
- print Computer machine name and rank id for each process.
- MPI time function to measure time taken by given fragment.
- Communication function (send/receive or scatter or broadcast etc.)
- Find command to set fix process per node.

II. PREREQUISITE

Install MPICH2 in each of the machine

III. MPI SETUP

I. Configure hosts file

You are gonna need to communicate between the computers and you don't want to type in the IP addresses every so often. Instead, you can give a name to the various nodes in the network that you wish to communicate with. hosts file is used by your device operating system to map hostnames to IP addresses.

```
$ sudo nano /etc/hosts

10.1.3.4        master
10.1.3.5        node
```

II. Create a new user

Though you can operate your cluster with your existing user account, I'd recommend you to create a new one to keep our configurations simple. Let us create a new user mpi.

Create new user accounts with the same username in all the machines to keep things simple.

```
$ sudo adduser mpi
```

Follow prompts and you will be good. Please don't use $useradd$ command to create a new user as that doesn't create a separate home for new users.

III. Setting up SSH

machines are gonna be talking over the network via SSH and share data via NFS.

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

And right after that, login with your newly created account

```
$ su - mpi
```

Since the ssh server is already installed, you must be able to login to other machines by ssh username@hostname, at which you will be prompted to enter the password of the username. To enable more easier login, we generate keys and copy them to other machines' list of authorized_keys.

```
$ ssh-keygen -t rsa
```

Now, add the generated key to each of the other computers. In our case, the client machine.

```
$ ssh-copy-id client #ip-address may also be used
```

Do the above step for each of the client machines and your own user (localhost).

This will setup openssh-server for you to securely communicate with the client machines. ssh all machines once, so they get added to your list of known_hosts. This is a very simple but essential step failing which passwordless ssh will be a trouble.

Now, to enable passwordless ssh,

```
$ eval `ssh-agent`
$ ssh-add ~/.ssh/id_dsa
```

Now, assuming you've properly added your keys to other machines, you must be able to login to other machines without any password prompt.

IV. Setting up NFS

You share a directory via NFS in **master** which the **mpi** mounts to exchange data.

IV.1 NFS-Server

Install the required packages by

```
$ sudo apt-get install nfs-kernel-server
```

Now export directory to share among nodes as below:

```
$ sudo nano /etc/exports
/home/mpiuser/cloud *(rw,sync,no_root_squash,no_subtree_check)
```

- Here, instead of $*$ you can specifically give out the IP address to which you want to share this folder to. But, this will just make our job easier.
- **rw**: This is to enable both read and write option. ro is for read-only.
- **sync**: This applies changes to the shared directory only after changes are committed.
- **no_subtree_check**: This option prevents the subtree checking. When a shared directory is the subdirectory of a larger filesystem, nfs performs scans of every directory above it, in order to verify its permissions and details. Disabling the subtree check may increase the reliability of NFS, but reduce security.
- **no_root_squash**: This allows root account to connect to the folder.

After you have made the entry, run the following.

```
$ sudo exportfs -a
```

Run the above command, every time you make a change to $/etc/exports$.

If required, restart the nfs server

```
$ sudo service nfs-kernel-server restart
```

IV.2 NFS-Client

Install the required packages

```
$ sudo apt-get install nfs-common
```

mount the shared directory like

```
$ sudo mount -t nfs master:/home/mpi ~/
```

To make the mount permanent so you don't have to manually mount the shared directory everytime you do a system reboot, you can create an entry in your file systems table - i.e., $/etc/fstab$ file like this:

```
$ sudo nano /etc/fstab
#MPI CLUSTER SETUP
master:/home/mpiuser/cloud /home/mpiuser/cloud nfs
```

IV. Troubleshooting Setup

- Make sure all the machines you are trying to run the executable on, has the same version of MPI. Recommended is MPICH2.
- The hosts file of master should contain the local network IP address entries of master and all of the slave nodes. For each of the slave, you need to have the IP address entry of master and the corresponding slave node.

- Whenever you try to run a process parallely using MPI, you can either run the process locally or run it as a combination of local and remote nodes. You cannot invoke a process only on other nodes.

  To make this more clear, from master node, this script can be invoked.

```
$ mpirun -np 10 --hosts master ./main
# To run the program only on the same master node
```

  So can this be. The following will also run perfectly.

```
$ mpirun -np 10 --hosts master,node ./main
# To run the program on master and slave nodes.
```

  But, the following is not correct and will result in an error if invoked from master.

```
$ mpirun -np 10 --hosts node ./cpi
# Trying to run the program only on remote slave
```

- **Architecture Issue**: Try using same hardware architecture for master and slaves (otherwise you need to learn how to run mpi file on cross platform) and will generate either error with using library with either ELFCLASS64 or ELFCLASS32
- Avoid mount entry in $/etc/fstab$
  The reason is if your server is not going to be live 24 hours the system tries to mount that partition and you might face login loop for the user.
- DO NOT TURN OFF master node before un mounting partition which is mounted in home directory of existing system

I. Sequential Code

```
1  //
   ↪ ------------------------------------------------------------------------------------------
2  // Author: Gahan Saraiya
3  // GiT: http://github.com/gahan9/
4  // StackOverflow: https://stackoverflow.com/users/story/7664524
5  // Website: http://gahan9.github.io/
6  //
   ↪ ------------------------------------------------------------------------------------------
7  // Setup MPI and execute sample code given in MPI src.
8  // Code should print Computer machine name and rank id for each process.
9  // Find command to set fix process per node.
10
11 #include "mpi.h"
12 #include <stdio.h>
```

```
13
14  int main(int argc, char** argv) {
15      // Initialize the MPI environment. The two arguments to MPI Init are not
16      // currently used by MPI implementations, but are there in case future
17      // implementations might need the arguments.
18      MPI_Init(NULL, NULL);
19      char processor_name[MPI_MAX_PROCESSOR_NAME];
20      int name_len;
21
22      // Get the number of processes
23      int world_size;
24      MPI_Comm_size(MPI_COMM_WORLD, &world_size);
25
26      // Get the rank of the process
27      int world_rank;
28      int data;
29      MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
30
31      // Task 3: Broadcast Data
32      if (world_rank == 0){
33          // Master Process
34          // printf("***I'm The master!! Kneel before your master***\n");
35          // for detail refer
            ↪ http://mpitutorial.com/tutorials/mpi-broadcast-and-collective-communication/
36          data = 100;
37          MPI_Get_processor_name(processor_name, &name_len);
38          // Task 1: printing processor name
39          printf("[%s-process:%d] broadcasting data %d\n", processor_name,
            ↪ world_rank, data);
40          double start = MPI_Wtime();
41          MPI_Bcast(
42              &data,          // void* data               data variable
43              1,              // int count,
44              MPI_INT,        // MPI_Datatype datatype,
45              world_rank,     // int root                 process zero
46              MPI_COMM_WORLD  // MPI_Comm communicator
47              );
48          double end = MPI_Wtime();
49          // Task 2: Time measure
50          printf("[%s-process:%d] Time Elapsed: %lf\n", processor_name, world_rank,
            ↪ end - start);
51      }
```

```
52      else {
53          // Node/Slave Processes
54          double start = MPI_Wtime();
55          MPI_Bcast(&data, 1, MPI_INT, 0, MPI_COMM_WORLD);
56          double end = MPI_Wtime();
57          MPI_Get_processor_name(processor_name, &name_len);
58          // Task 1: printing processor name
59          printf("[%s-process:%d] received data %d from master\n", processor_name,
            ↪  world_rank, data);
60          printf("[%s-process:%d] Time Elapsed: %lf\n", processor_name, world_rank,
            ↪  end - start);
61      }
62
63      // Get the name of the processor
64      MPI_Get_processor_name(processor_name, &name_len);
65
66      // Task 1: print Computer machine name and rank id for each process.
67      printf("Gratitude!!! from ``%s``, rank %d out of %d processes\n",
68          processor_name, world_rank, world_size);
69
70      // The times are local; the attribute MPI_WTIME_IS_GLOBAL may be used
71      // to determine if the times are also synchronized with each other for
72      // all processes in MPI_COMM_WORLD.
73
74      // Finalize the MPI environment. No more MPI calls can be made after this
75      MPI_Finalize();
76  }
```

I.1 Compiling and Executing File!!!

```
read processes

mpicc main.c -o main

mpirun -np $processes --hosts master,node ./main > output.txt
```

I.2 Output

```
[master-process:0] broadcasting data 100
[master-process:0] Time Elapsed: 0.000450
```

```
Gratitude!!! from ``master``, rank 0 out of 8 processes
[master-process:2] received data 100 from master
[master-process:2] Time Elapsed: 0.000534
Gratitude!!! from ``master``, rank 2 out of 8 processes
[master-process:4] received data 100 from master
[master-process:4] Time Elapsed: 0.000524
Gratitude!!! from ``master``, rank 4 out of 8 processes
[master-process:6] received data 100 from master
[master-process:6] Time Elapsed: 0.000556
Gratitude!!! from ``master``, rank 6 out of 8 processes
[node-process:1] received data 100 from master
[node-process:1] Time Elapsed: 0.000570
Gratitude!!! from ``node``, rank 1 out of 8 processes
[node-process:3] received data 100 from master
[node-process:3] Time Elapsed: 0.000588
Gratitude!!! from ``node``, rank 3 out of 8 processes
[node-process:5] received data 100 from master
[node-process:5] Time Elapsed: 0.000595
Gratitude!!! from ``node``, rank 5 out of 8 processes
[node-process:7] received data 100 from master
[node-process:7] Time Elapsed: 0.000610
Gratitude!!! from ``node``, rank 7 out of 8 processes
```

V. Fix Process per node

According to documentation of mpirun here are the list of flags helpful to set fix number of processes per node.

- **-npersocket, –npersocket <#persocket>** On each node, launch this many processes times the number of processor sockets on the node. The -npersocket option also turns on the -bind-to-socket option. (deprecated in favor of –map-by ppr:n:socket)
- **-npernode, –npernode <#pernode>** On each node, launch this many processes. (deprecated in favor of –map-by ppr:n:node)
- **-pernode, –pernode** On each node, launch one process – equivalent to -npernode 1. (deprecated in favor of –map-by ppr:1:node)

The number of processes launched can be specified as a multiple of the number of nodes or processor sockets available.

For example,

```
mpirun -H aa,bb -npersocket 2 ./a.out
```

launches processes 0-3 on node aa and process 4-7 on node bb, where aa and bb are both dual-socket nodes. The -npersocket option also turns on the -bind-to-socket option, which is discussed in a later section.

```
mpirun -H aa,bb -npernode 2 ./a.out
```

launches processes 0-1 on node aa and processes 2-3 on node bb.

```
mpirun -H aa,bb -npernode 1 ./a.out
```

launches one process per host node.

```
mpirun -H aa,bb -pernode ./a.out
```

is the same as -npernode 1.