Clustering Tutorial

What is Clustering?

Clustering is the use of multiple computers, typically PCs or UNIX workstations, multiple storage devices, and redundant interconnections, to form what appears to users as a single highly available system. Cluster computing can be used for load balancing as well as for high availability. It is used as a relatively low-cost form of parallel processing machine for scientific and other applications that lend themselves to parallel operations.

Computer cluster technology puts clusters of systems together to provide better system reliability and performance. Cluster server systems connect a group of servers together in order to jointly provide processing service for the clients in the network.

Cluster operating systems divide the tasks amongst the available servers. Clusters of systems or workstations, on the other hand, connect a group of systems together to jointly share a critically demanding computational task. Theoretically, a cluster operating system should provide seamless optimization in every case.

At the present time, cluster server and workstation systems are mostly used in High Availability applications and in scientific applications such as numerical computations.

16 node Linux cluster at SCFBio

Advantages of clustering

- High performance
- Large capacity
- High availability
- Incremental growth

Applications of Clustering

- Scientific computing
- Making movies
- Commercial servers (web/database/etc)

Getting Started With Linux Cluster

Although clustering can be performed on various operating systems like Windows, Macintosh, Solaris etc. , Linux has its own advantages which are as follows:-

- Linux runs on a wide range of hardware
- Linux is exceptionally stable
- Linux source code is freely distributed.
- Linux is relatively virus free.
- Having a wide variety of tools and applications for free.
- Good environment for developing cluster infrastructure.

The procedure described is based on the concept of a Beowulf cluster, both using LAM and publicly available OSCAR software package.

Index:

- About Linux
- File structure of Linux
- NFS
- NIS
- RPM
- PVM
- PBS
- MPI
- SSH
- IP address
- Cluster Components
- Building Linux Cluster using LAM
- Open source package eg: OSCAR.
- Appendix

I) About Linux

Linux is an open-source operating system like Unix. It has the reputation of a very secure and efficient system. It is used most commonly to run network servers and has also recently started to make inroads into Microsoft dominant desktop business. It is available for wide variety of computing devices from embedded systems to huge multiprocessors, also it is available for different processors like x86, powerpc, ARM, Alpha, Sparc, MIPS, etc. It should be remembered that Linux is essentially the OS Kernel developed by Linus Torvald and is different from the commonly available distributions like RedHat, Caldera,etc (These are Linux Kernel plus GPLed softwares).

1. Common Commands in Linux

1.1 Changing directory

cd without arguments puts the user in the users home directory. With a directory name as argument, the command moves the user to that directory

```
$>cd directorypath
```

1.2 Copy files

cp makes copies of files in two ways.

```
$>cp file1 file2
```

makes a new copy of file1 and names it file2.

```
$>cp [list of files] directory
```

puts copies of all the files listed into the directory named. Contrast this to the mv command which moves or renames a file.

1.3 Making a link

ln creates a link between files. Example:
The following links the existing file example.c to ex.c.

```
$>ln example.c ex.c
```

The following creates symbolic links.

```
$>ln -s /usr/include incl
```

See the online man pages for many other ways to use ln.

1.4 Make a new directory

mkdir makes a new subdirectory in the current directory.

```
$>mkdir directoryname
```

makes a subdirectory called directoryname.

1.5 Move / rename files

mv moves or changes the name of a file.

$>mv file1 file2

changes the name of file1 to file2. If the second argument is a directory, the file is moved to that directory. One can also specify that the file have a new name in the directory ‘direc’:

$>mv file1 direc/file2

would move file1 to directory direc and give it the name file2 in that directory.

1.6 Present working directory

pwd returns the name of the current working directory. It simply tells you the current directory.

1.7 Remove files

rm removes each file in a list from a directory. By default option -i to rm inquires whether each file should be removed or not. Option -r causes rm to delete a directory along with any files or directories in it.

$>rm filename

1.8 Remove directory

rmdir removes an empty directory from the current directory.

$>rmdir directoryname
removes the subdirectory named directoryname (if it is empty of files). To remove a directory and all files in that directory, either remove the files first and then remove the directory or use the rm –r option described above.

1.9 Listing files and directories

ls lists the files in the current directory or the directory named as an argument. There are many options:

ls -a [directory]
lists all files, including files whose names start with a period.

ls -c [directory]
lists files by date of creation.

ls -l [directory]
lists files in long form: links, owner, size, date and time of last change.

ls -p [directory]
subdirectories are indicated by /.

ls -r [directory]
reverses the listing order.

ls -s [directory]
gives the sizes of files in blocks.

ls -C [directory]
lists files in columns using full screen width.

ls -R [directory]
recursively lists files in the current directory and all subdirectories.

2. File Transfer

2.2 Establishing remote connection

To establish a connection to a remote system use the sftp command. After the connection is established provide the valid password.

$>sftp –oPort=44 user@203.90.127.210

2.2 File uploading

Move a file from the local host to remote host

$>put filename

2.3 To put multiple files using wild cards

$>mput pattern*

2.4 File downloading

Move a file from remote host to local host

$>get filename

2.5 To get multiple files using wild cards

$>mget pattern*

2.6 Making a new directory on remote host

$>mkdir directoryname

2.7 Changing directory in local host

$>lcd directorypath

2.8 Changing directory in the remote host

$>cd directorypath

2.9 Closing the connection

$>bye

II) File structure in Linux

Data and programs are stored in files, which are segmented in directories. In a simple way, a directory is just a file that contains other files (or directories). The part of the hard disk where one is authorized to save data is called home directory. Normally all the data that is to be save will be saved in files and directories in the home directory. The symbol ~ can also be used for home directory. The directory structure of Linux is a tree with directories inside directories, several levels .The tree starts at what is called the root directory / (slash).

The following are the list of directories or say branches of the tree.

- **/bin**: contains basic utilities like bash,chmod,chown,date,df,kill,mkdir,mount etc
- **/boot**: a copy of the kernel (Linux) needed for the machine to start up (to boot).
- **/cdrom**: to read CDs .
- **/dev**: in Linux every hardware is essentially a file which resides here.
- **/etc**: the system configuration files and directories like bashrc, init.d, profile.d, yp.conf of the system.
- **/floppy**: to read floppies.
- **/home**: typically has the user directories to store personal files.
- **/initrd**: another set of files needed for the machine to boot.
- **/lib**: files (called libraries) needed for programs to work.
- **/mnt**: a directory for temporarily reading some hardware devices, mount points for temporary mounts by the system administrator.
- **/proc**: a virtual directory created by the currently running kernel to store information about all the running system/user processes. It is deleted when the system is shut down.
- **/sbin**: these files are utility files used for system management .
- **/usr**: a (huge) directory with many programs. The /usr directory is designed to store static, sharable, read-only data. Programs which are used by all users are frequently stored here. Data which results from these programs is usually stored elsewhere.
- **/root**: the directory where the system administrator (root) saves his/her files
- **/tmp**: a temporary directory used by many programs to save things for short periods of time (files here are periodically removed).
- **/var**: contains variable data, mostly stuff needed for the system to work (like PID information) or databases. This directory stores **var**iable data like logs, mail, and process specific files. Most, but not all, subdirectories and files in the /var directory are shared.

III) Network File System (NFS)

A distributed file system that enables users to access files and directories located on remote computers and treat those files and directories as if they were local. NFS is independent of machine types, operating systems, and network architectures through the use of remote procedure calls (RPC).

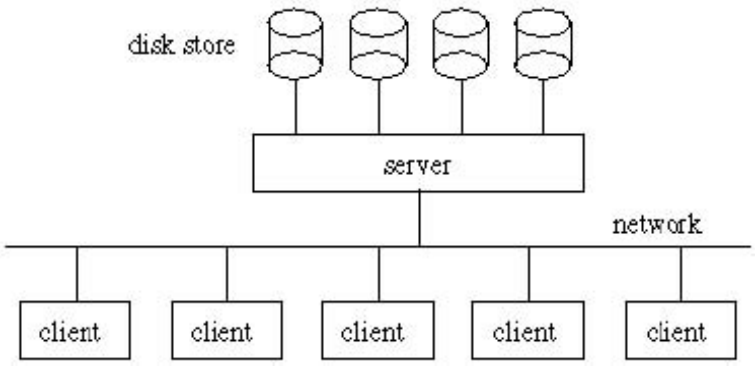


Figure: The division of NFS between client and server

IV) Network Information System (NIS)

Network Information Service has to be known throughout the network to all machines on the network. NIS is a distributed database that provides a repository for storing information about hosts, users, and mailboxes in the UNIX environment. It was originally developed by Sun Microsystems and called YP (Yellow Pages). NIS is used to identify and locate objects and resources that are accessible on a network.

V) Red Hat Package Manager (RPM)

Linux files are generally RPMs. RPM also stands for Red Hat Package Manager. Red Hat Linux uses the RPM technology of software installation and upgrades. Using RPM, either from the shell prompt or through Gnome-RPM, is a safe and convenient way to upgrade or install software.

VI) Parallel Virtual Machine (PVM)

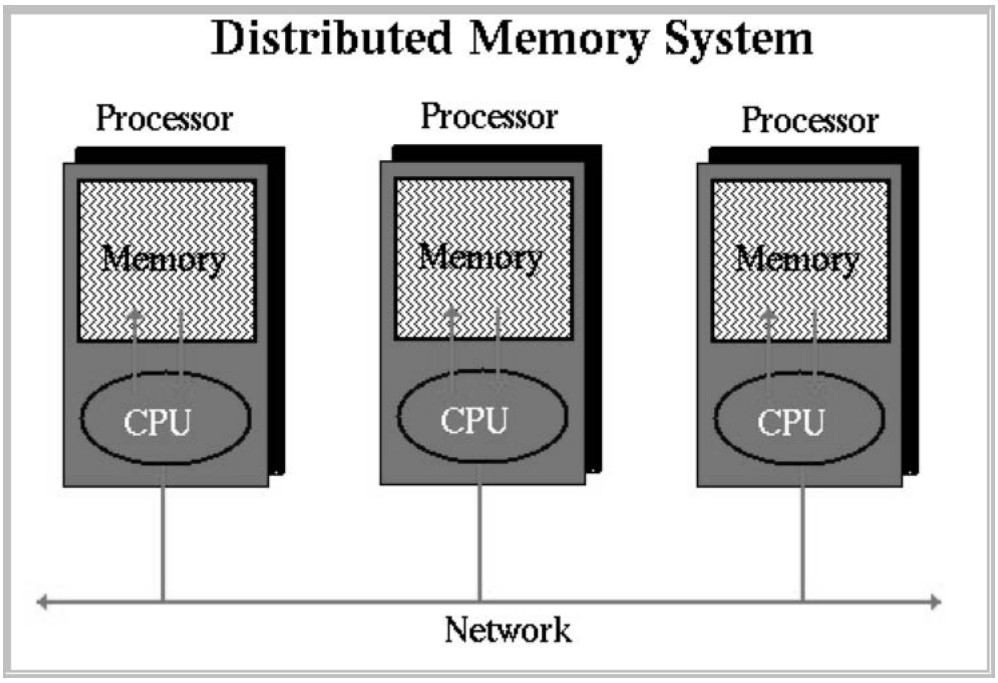
PVM (Parallel Virtual Machine) is a software package that permits a heterogeneous collection of Unix and/or Windows computers hooked together by a network to be used as a single large parallel computer. The individual computers may be shared- or local-memory multiprocessors, vector supercomputers, specialized graphics engines, or scalar workstations, that may be interconnected by a variety of networks ,such as ethernet , FDDI.

VII) Portable Batch System (PBS)

OpenPBS is the original version of the Portable Batch System. It is required for the scheduling of the jobs. It is a flexible batch queuing system. It operates on networked, multi-platform UNIX environments. OpenPBS consists of three primary components—a job server (pbs_server) handling basic queuing services such as creating and modifying a batch job and placing a job into execution when it's scheduled to be run. The executor (pbs_mom) is the daemon that actually runs jobs. The job scheduler (pbs_sched) is another daemon. pbs_server and pbs_sched are run only on the front end node(server node), while pbs_mom is run on every node of the cluster that can run jobs, including the front end node(server).

VIII) Message Passing Interface (MPI)

MPI is a widely accepted standard for communication among nodes that run a parallel program on a distributed-memory system. The standard defines the interface for a set of functions that can be used to pass messages between processes on the same computer or on different computers. MPI can be used to program shared memory or distributed memory computers. Hence MPI is a library of routines that can be called from Fortran and C programs There are a large number of implementations of MPI, two open-source versions are MPICH and LAM.



IX) Secure Shell (SSH)

A packet-based binary protocol that provides encrypted connections to remote hosts or servers. Secure Shell is a program to log into another computer over a network, to execute commands in a remote machine, and to move files from one machine to another. It provides strong authentication

and secure communications over insecure channels. It is a replacement for rlogin, rsh, rcp, and rdist, telnet, ftp.

X) IP(Internet Protocol) Address

Internet protocol is an identifier for a computer or device on a TCP/IP network. Networks using the TCP/IP protocol route messages based on the IP address of the destination. The format of an IP address is a 32-bit numeric address written as four numbers separated by periods. Each number can be zero to 255. For example, 10.90.78.45 could be an IP address.

Within an isolated network, you can assign IP addresses at random as long as each one is unique. However, connecting a private network to the Internet requires using registered IP addresses (called Internet addresses) to avoid duplicates

XI) Cluster Components

The cluster consists of four major parts. These parts are: 1) Network, 2) Compute nodes, 3) Master server, 4) Gateway. Each part has a specific function that is needed for the hardware to perform its function.

1. Network:

    - Provides communication between nodes, server, and gateway
    - Consists of fast ethernet switch, cables, and other networking hardware

2. Nodes:

    - Serve as processors for the cluster
    - Each node is interchangeable, there are no functionality differences between nodes
    - Consists of all computers in the cluster other than the gateway and server

3. Server:

    - Provides network services to the cluster
    - DHCP
    - NFS (Node image and shared file system)
    - Actually runs parallel programs and spawns processes on the nodes
    - Should have minimum requirement.

4. Gateway:

    - Acts as a bridge/firewall between outside world and cluster
    - Should have two ethernet cards

XII) Building a Linux Cluster using Local Area Multicomputer (LAM)

Introduction

LAM is a high-quality open source implementation of the Message Passing Interface specification. It is a development system for heterogeneous computers on a network which can be used to solve compute intensive problems. LAM is an effective way for fast client-to-client communication and is portable to all UNIX machines. It includes standard support for SUN (SunOS and Solaris), SGI IRIX, IBM AIX, DEC OSF/1, HPUX, and LINUX.

Requirements

HARDWARE:

- Ethernet switch for physical connection between the nodes.
- CPUs (Central Processing Units) depending on the number of nodes to be clustered.
- Monitor
- Network Cables
- LAN (local area network) card
- Optional- back up power supply, racks for computers

SOFTWARE:

- LINUX OS - http://www.linuxiso.org/
- LAM package - http://www.lam-mpi.org/7.0/download.ph

Installation Procedure

I. Configuring the nodes
II. Installing and Running LAM
III. Compiling and Executing MPI program

I. Configuring the nodes

Once Linux has been installed (preferably the same version) on all the nodes that need to be clustered, rsh needs to be configured on all nodes such that one can connect to any other nodes in the cluster without password. Following steps need to be done for the same:

1. Login with root and add same username on all the nodes, preferably with the same password. In this document we are assuming that the username is “try”.

   [root@root]$ useradd try
   [root@root]$ passwd try

2. Type “setup” at command prompt and click on system services. Make sure the following are checked: rsh, rlogin, rexec, nfs.

3. Edit the file /etc/hosts. It contains a list of IP addresses and the hostnames of the nodes that need to be clustered,

[root@root]$ vi /etc/hosts
The file should have the following format:

```
#  IP address     Hostname      alias
   10.96.6.1       scfbio01       node1
   10.96.6.2       scfbio02       node2
    .
    .
   Respectively
```

(The word “alias” refers to the different nodes to be clustered and “Hostname” can be checked by the following command on all the nodes: $ hostname)

4. Copy the file /etc/hosts to /etc/hosts.equiv,

   [root@root]$ cp /etc/hosts /etc/hosts.equiv

5. Create/Edit the file /home/try/.rhosts to allow trusted access to given host/user combinations without the need for a password.

   [root@root]$ vi /home/try/.rhosts

   The file should have the following format:

```
#  Hostname/Alias    Username
   node1               try
   node2               try
    .
    .
   Respectively
```

6. Change the permissions of /home/try/.rhosts,

   [root@root]$ chmod 755 /home/try/.rhosts

7. Edit the file /etc/securetty to enable system services like rsh, rlogin and rexec,

   [root@root]$ vi /etc/securetty

   The file should have the following format:

```
.......
tty10
tty11
# add three more lines to file /etc/securetty
rsh
rlogin
rexec
```

   Save and quit.

8. Configuration of the nodes is now complete. We can check the connectivity between different nodes by switching to "try" on node1, using the following command:

   [try@node1]$ rsh node2
   ..... similarily you can switch between various nodes without password.

II. Installing and Running LAM

1. Check whether LAM (preferably same version) exists on all the systems,

   [root@root]$ lam

2. If does not exist, download the latest version of LAM (rpm package) from the following link:

   http://www.lam-mpi.org/7.0/download.ph

   and install LAM using the following command:

   [root@root]$ rpm -Uvh lam*.rpm

3. Create a hostfile in "try" directory on the node where MPI programs needs to be run subsequently (we assume it to be node1). It provides a listing of the machines to be booted in a MPI session.

   [try@node1]$ vi hostfile

   The file should have the following format:
   node1
   node2
   .
   .
   respectively

4. To check whether the Lam can be booted successfully,

   [try@node1]$ recon -v hostfile

   If the comment "WooHoo" appears on the screen, lamboot command in the next step will execute successfully.

5. Booting LAM: lamboot starts LAM on the specified cluster to initiate MPI session.

   [try@node1]$ lamboot -v hostfile

   At the end, if the comment "topology done" appears on the screen, this command will execute on different nodes successfully.

6. To check whether the MPI session is running uninterruptly,

   [try@node1]$ tping -c3 N

III. Compiling and Executing MPI programs

For the following steps, make sure you are in "try".

1. Compilers for C, C++ and fortran programs are mpicc, mpiCC (or mpic++) and mpif77 respectively. These compilers include all the relevant files and directories required for running the MPI programs.

Three examples of MPI programs are listed below for demonstrating basic concepts of parallel computing:

(i) Parallel Program to calculate the sum of square of first N numbers:

```
/* the program starts here */

#include <stdio.h>
#include<string.h>
#include <mpi.h>
#include<math.h>

int main(int argc, char *argv[])
{
        int node;
        int size;
        MPI_Status *status;
        int Total=0;


        MPI_Init(&argc,&argv);

        MPI_Comm_rank(MPI_COMM_WORLD, &node);
        MPI_Comm_size(MPI_COMM_WORLD, &size);
        int length=atoi(argv[1]);

        int perNode=length/size;
        if(perNode==0) perNode=1;

        if(length-((size-1)*perNode) > perNode)
                perNode++;


        if(node==0)
        {
                printf("The maximum input number is %5d \n",length);
                int tag;
                int Sum=0;
                int number;
```

```
        int startPosition=(size-1) * perNode+1;


        for(;startPosition<=length;startPosition++)
        {
                Total+=pow(startPosition,2.0);
        }

        printf("The Sum = %5d by the given node = %5d \n",Total,node);
        Sum=Total;


        for(tag=1;tag<size;tag++)
        {

MPI_Recv(&Total,1,MPI_INT,tag,tag,MPI_COMM_WORLD,status);
                Sum+=Total;
        }
                printf("The Sum of square of 1 - %d numbers is %5d
                \n",length,Sum);

}
else
{
        int startPosition=(node-1) * perNode+1;
        int endPosition=node * perNode;

        if(startPosition>length)
        {
                startPosition=length+1;
                endPosition=length+1;
        }

        if(endPosition>length)
        {
                endPosition=length;
        }

        for(;startPosition <= endPosition;startPosition++)
                Total+=pow(startPosition,2.0);


        printf("The Sum = %5d by the given node = %5d \n",Total,node);


        MPI_Send(&Total,1,MPI_INT,0,node,MPI_COMM_WORLD);
}
```

```
MPI_Finalize();
return 0;
}
/* the program ends here */
```

To compile this program,

[try@node1]$ mpicc sumN.c -lm -o sumN.exe

lm is used if math.h library is not found by default.

This compilation will generate an executable. Executable file, “sumN.exe” in this case, should be copied to try directory on all the nodes.

[try@node1]$ scp hello node2:/home/try/

To execute the MPI program in parallel, the following command is used:

[try@node1]$ mpirun -np 6 sumN.exe

where np are the number of processors on which the program needs to run, 6 in this case.

(ii) Parallel Program to count number of base pairs in a given sequence and implement Chargaff's rule of base pairing – *Listed in Appendix*

(iii) Parallel Program to calculate number of times the alphabet "e" appears in the given text – *Listed in Appendix*

2. If another MPI program needs to be executed, the following command can be used. It will remove all the user processes and messages without rebooting.

[try@node1]$ lamclean –v

3. When LAM is no longer needed, lamhalt command removes all traces of LAM session from the network.

[try@node1]$ lamhalt

4. In case one or more lam nodes crash i.e., lamhalt hangs, the following command needs to be run:

[try@node1]$ wipe -v hostfile

This will kill all the processes running on the hosts mentioned in hostfile.

XIII) OSCAR- Open Source Cluster Application Resource

OSCAR (Open Source Cluster Application Resource) software package is a high performance cluster (HPC) used to simplify the complex tasks required to install a cluster. Its advantage is that several HPC-related packages like MPI implementations, LAM, PVM (Parallel Virtual Machine), PBS (Portable Batch Server) etc are installed by default and need not be installed separately. It can be downloaded from the link given below –

http://oscar.openclustergroup.org/download

Supported Distributions

The following is a list of supported Linux distributions for the OSCAR-4.1 package:

- Red Hat Linux 9 (x86)
- Red Hat Enterprise Linux 3 (x86, ia64)
- Fedora Core 3 (x86)
- Fedora Core 2 (x86)
- Mandriva Linux 10.0 (x86)

Each individual machine of a cluster is reffered to as a node. In OSCAR cluster there are two types of nodes: server and client. A server is responsible for serving the requests of client nodes, whereas a client is dedicated to computation. An OSCAR cluster consists of one server node and one or more client nodes,where all the client nodes must have homogeneous hardware. The Cluster which we are having is named as **Linster,** which is originated from the term **Lin**ux Clus**ter**.

Configuration of Linster

- 16 node cluster.
- 1 Server node and 15 Client nodes.

Server

- 1.5 GHz Pentium P4 Processor.
- Two 40 GB Hard disks.
- 256 MB RAM.
- Two network interface cards supporting TCP/IP stack.
- Keyboard and mouse.

Clients

- 1.5 GHz Pentium P4 Processor.
- One 40 GB Hard disk.
- 256 MB RAM.
- One network interface card supporting TCP/IP stack.
- No keyboard or mouse is required.

The main advantage of using OSCAR is that we need not to do any operation on client nodes all the process has to be done on the server node starting from the linux installation to cluster setup. The client's settings can be taken care of by the OSCAR package itself. All we have to do is to network boot the clients and it will automatically copy the images from the Server.

Setting up the cluster

Steps involved in installing the OSCAR distribution on the server are as follows:

- Install RedHat linux on the Server node and make partition accordingly. (We are using RedHat Linux Enterprise 3).
- Download OSCAR distribution package from http://oscar.sourceforge.net/
- Go to the OSCAR directory by using command.

  Suppose the download directory is /root.

  # cd /root/oscar-4.1

  then run the configure script

  # ./configure.

  Now the server is ready for the OSCAR installation.

  Set environment variables like OSCAR_HOME in /etc/profile directory.

  # source /etc/profile

  Run # make install

  The default directory is /opt/oscar in which OSCAR is to be installed.

- Copy distribution installation RPMs to /tftpboot/rpm directory from linux distribution CD by using the command

  # cp /mnt/cdrom/RedHat/RPMS/*.rpm /tftpboot/rpm.

  If any rpm is missing then download it from the site, ftp://ftp.redhat.com/pub/redhat/linux/enterprise/3/en/RPMS

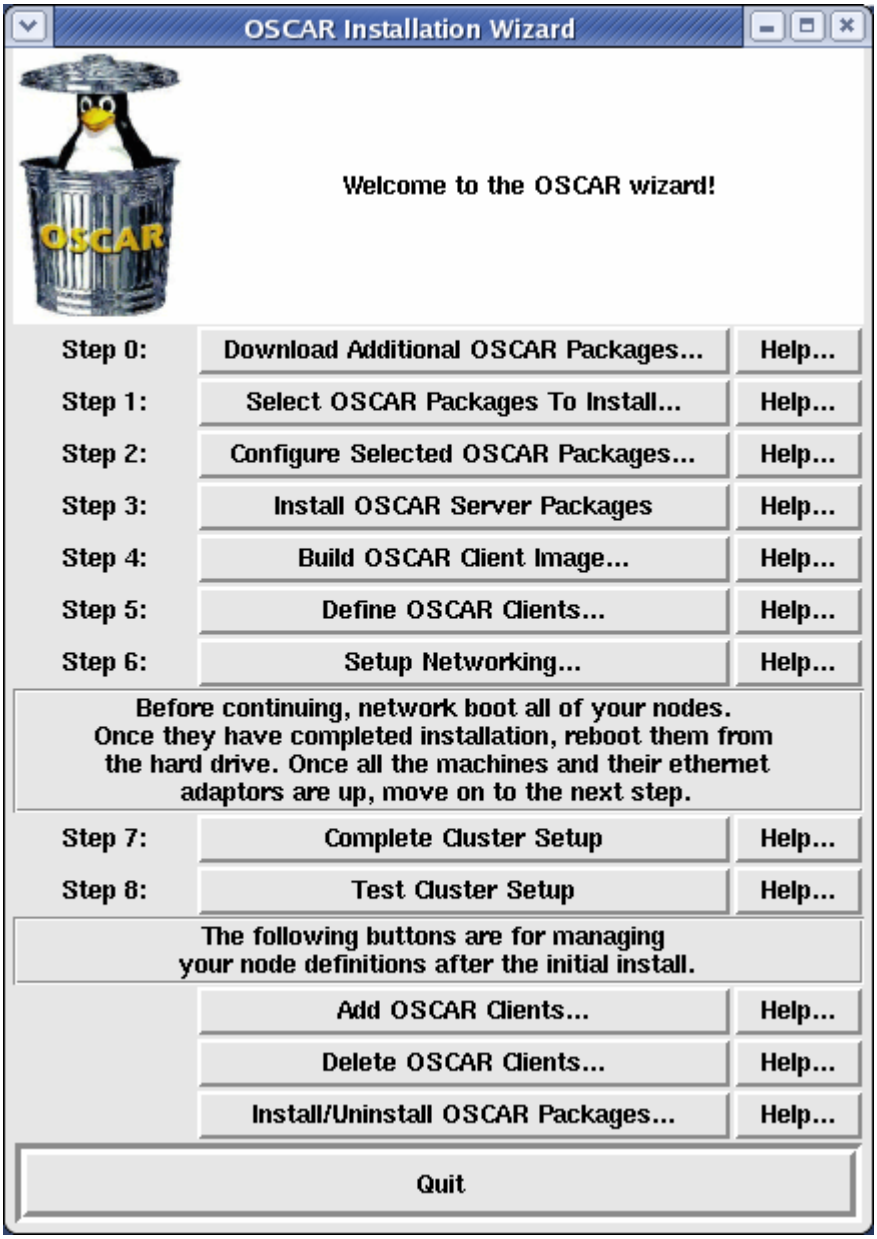
- Change directory to top level OSCAR directory

  # cd $OSCAR_HOME

  # ./install_cluster <device>

  where device is eth0 or eth1.(for the network connections)

Substitute the device name (e.g., eth1) for server's private network Ethernet adapter. While running this command, an OSCAR installation wizard will appear on the screen. Run all the steps which are given in the wizard according to their instructions.



By running this wizard up to the first four steps all things are taken care of by OSCAR itself.

In the 5th step only, define the number of clients to be added to the cluster. Assign an individual IP addresses to all the clients. Also, assign MAC addresses to all the nodes. In the 6th step, network booting is done in order to make the clients synchronize with the servers and linux is installed on the client node with the same configuration as that of the server. Run "Complete Cluster Setup" and "Test Cluster Setup" to check the authenticity of the cluster. With this, clustering is complete in a very simple and easy manner.

Advantages of OSCAR

The main benefit of using this package is that there is no need to configure or install the different file systems or services like NFS, NIS, PBS etc which otherwise will have to be install separately, which saves the user's time. Job can be fired on the server node and the server manages to distribute it to Client nodes. Any number of clients can be added or removed at any instance, even after the setup is done.

Conclusion

The prime concern for making this documentation is to introduce a general idea of utilizing the available resources by making the users aware of the knowledge and concepts of Linux which otherwise would not be explored due to lack of proper training and guidance. Also, while doing clustering one would be well versed with Linux operating system, which is considered as a bit complex and much more technical than other operating systems.

Appendix

I) Parallel Program to count number of base pairs in a given sequence and implement Chargaff's rule of base pairing – ***refer to page 15***

```
/* The program starts here */

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

int main(int argc, char *argv[])
{
  int node;
  int size;
  MPI_Status *status;
  int DNABase[4],BP[4];
  int count=0;

  for(count=0;count < 4;count++)
  {
        DNABase[count]=0;
        BP[count]=0;
  }
  char*Seq="ATGTTGGTGTCCGCAAGGGTAGAGAAACAAAAGCGTGTTGCTTATCAGGGGAAG
GCGACAGTGCTTGCTCTCGGTAAGGCCTTGCCGAGCAATGTTGTTTCCCAGGAGAATCTCGTGG
AGGAGTATCTCCGTGAAATCAAATGCGATAACCTTTCTATCAAAGACAAGCTGCAACACTTGTG
CAAAAGCACAACTGTCAAGACACGCTACACAGTCATGTCACGGGAGACGCTGCACAAATACCC
TGAACTAGCAACCGAGGGTTCCCCAACCATCAAACAGAGGCTTGAGATTGCAAACGATGCGGTT
GTGCAGATGGCATATGAA";

  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &node);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  int length=strlen(Seq);
  int perNode=length/size;

  if(node==0)
  {
        printf("%s \n",Seq);
        printf("The length of the Sequence is %5d \n",length);
        int tag;
        int startPosition=(size-1) * perNode;
        for(;startPosition<length;startPosition++)
              if(Seq[startPosition]=='A')
                    BP[0]++;
              else if(Seq[startPosition]=='T')
                    BP[1]++;
              else if(Seq[startPosition]=='G')
```

```
                BP[2]++;
            else if(Seq[startPosition]=='C')
                BP[3]++;

        for(count=0;count<4;count++)
            DNABase[count]=BP[count];

        for(tag=1;tag<size;tag++)
        {
            MPI_Recv(&BP[0],4,MPI_INT,tag,tag,MPI_COMM_WORLD,status);
            for(count=0;count<4;count++)
                DNABase[count]+=BP[count];
        }

        for(count=0;count<4;count++)
            if(count==0)
                printf("The number of A's in the given sequence is %5d
                \n",DNABase[count]);
            else if(count==1)
                printf("The number of T's in the given sequence is %5d
                \n",DNABase[count]);
            else if(count==2)
                printf("The number of G's in the given sequence is %5d
                \n",DNABase[count]);
            else if(count==3)
                printf("The number of C's in the given sequence is %5d
                \n",DNABase[count]);

        printf("Chargaff's Rule of Base Pairs A+T : G+C ~ %5d : %-5d\n",DNABase[0]

+DNABase[1],DNABase[2]+DNABase[3]);

    }
    else
    {
        int startPosition=(node-1) * perNode;
        int endPosition=node * perNode;

        for(;startPosition < endPosition;startPosition++)
            if(Seq[startPosition]=='A')
                BP[0]++;
            else if(Seq[startPosition]=='T')
                BP[1]++;
            else if(Seq[startPosition]=='G')
                BP[2]++;
            else if(Seq[startPosition]=='C')
                BP[3]++;

        MPI_Send(&BP[0],4,MPI_INT,0,node,MPI_COMM_WORLD);
    }
```

```
MPI_Finalize();
return 0;
}

/* The program ends here */
```

To compile the program,

$ mpicc chargaff.c -o chargaff.exe

This compilation will generate an executable. Execute this program,

$ mpirun -np 2 chargaff.exe

where np are the number of processors, 2 in this case.

II) Parallel Program to calculate number of times the alphabet "e" appears in the given text ***– refer to page 15***

```
/* The program starts here */

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

int main(int argc, char *argv[])
{
    int node;
    int size;
    MPI_Status *status;
    int Total=0;

    char *Text="The Supercomputing Facility for Bioinformatics & Computational Biology
(SCFBio), IIT Delhi, was created in July 2002, with funding from the Department of
Biotechnology (DBT), Govt. of India. (Principal Investigator:Prof.B.Jayaram). The Facility is
committed to making the programs associated with the software suites along with the
computational resources freely accessible at this site over the Web and on Biogrid.\n\n";

    MPI_Init(&argc,&argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &node);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    int length=strlen(Text);
    int perNode=length/size;

    if(node==0)
    {
      printf("%s \n",Text);
      printf("The length of the text is %5d \n",length);
```

```
        int tag;
        int Sum=0;
        int startPosition=(size-1) * perNode;

        for(;startPosition<length;startPosition++)
                if(Text[startPosition]=='e')
                        Total++;

        printf("The number of e's counted = %6d  by the given node = %5d \n",Total,node);
        Sum=Total;

        for(tag=1;tag<size;tag++)
        {
                MPI_Recv(&Total,1,MPI_INT,tag,tag,MPI_COMM_WORLD,status);
                Sum+=Total;
        }
                printf("The number of e's in the given text is  %5d \n",Sum);

      }
      else
      {
        int startPosition=(node-1) * perNode;
        int endPosition=node * perNode;

        for(;startPosition < endPosition;startPosition++)
                if(Text[startPosition]=='e')
                        Total++;
        printf("The number of e's counted = %6d  by the given node = %5d \n",Total,node);

        MPI_Send(&Total,1,MPI_INT,0,node,MPI_COMM_WORLD);
      }

  MPI_Finalize();
  return 0;
  }

  /* The program ends here */
```

To compile the program,

```
$ mpicc text.c  -o text.exe
```

This compilation will generate an executable. Execute this program,

```
$ mpirun -np 6 text.exe
```

where np are the number of processors, 6 in this case.