DALCO Cluster Manual

The following manual will help you to operate and use your DALCO Cluster. Most tasks are the same for single workstations and servers. A cluster is basically just a collection of single systems with a common look and feel for the users and for the applications. This manual will describe the differences and the tools which allows you to distribute software and your work.

The manual is compiled to describe your personal system, with all installed options and covers the daily tasks usually done. You find separated parts for the system administrator and for the users, feel free to distribute the appropriate pages to your team.

The document is also available for download on the webinterface of your cluster, in the *Cluster documentation* section.

In case you have still open questions: Don't hesitate to call our support on +41 44 908 38 38 or write an email to support@dalco.ch!

Have fun with your DALCO Cluster!

DALCO Cluster Manual

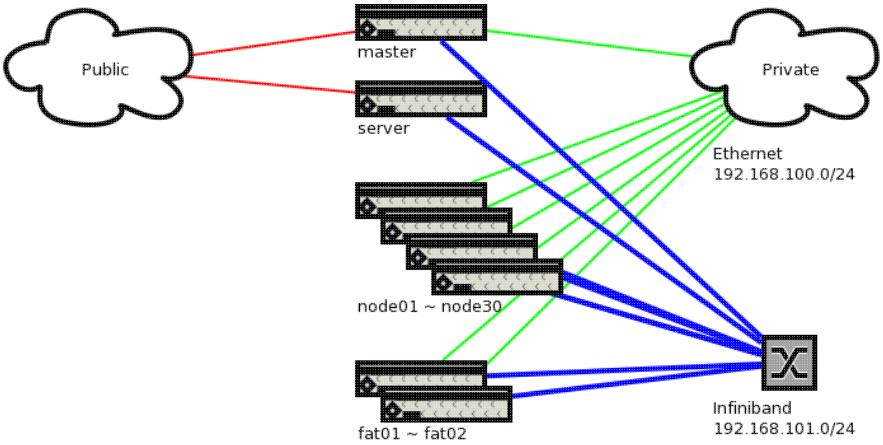
DALCO Cluster Manual

DALCO Cluster Manual

Cluster Uni Fribourg

One master, one fileserver, thirty compute nodes and two fat nodes, Intel Nehalem based. Infiniband interconnect. DALCO Cluster Suite based on CentOS 5.4.

Network overview



Private LAN with IPs out of 192.168.100.0/24 are attached to `eth0`. On the master and the server the public LAN is attached to `eth1`. Management interfaces are sharing the physical LAN connection of `eth0`. The Infiniband interconnect is using IPs out of 192.168.101.0/24.

Filesystems

The master is using a RAID1 of 250G for it's system and the installation sources. The server is using a RAID6 devided into two volumes of 250G and 13T for the homes and applications of the users. `/home`, `/apps`, `/opt/cluster` and `/opt/sge` are shared using NFSv4 across the cluster.

The master and the nodes have a local `/scratch` available for the users.

Base installation (Kickstart)

The master of your cluster is preloaded with our DALCO Cluster Suite. It contains all user data, the cluster wide configuration, the installsource for the nodes and prepacked software. It is the place where users log in and do their interactive work like preparing and postprocessing data, compiling and debugging code and submit their jobs.

The master also operates all basic services for the cluster. It act's as timeserver for all the nodes, a caching only DNS server and the queue manager for managing distributed batch jobs. The system is designed to operate mostly independend of your existing infrastructure - even it is fully integrated into your lan. The master monitors the state of all nodes and provides a central webinterface containing the current state and historical data of the system.

Operating system

The base installation of the master is done using Anaconda. Fell free to use adjust the configuration using the standard tools or by simply editing the configuration files. There is no automatism which destroys your changes. We took care to select a good choice of software for building and running typical cluster applications.

All our scripts and software packages are located in `/opt/cluster`, which is shared over all nodes. The directory `/opt/cluster/admin` is in the path to allow you the usage of our tools without entering the full path.

Kickstart Setup

The configuration of the Kickstart config files is automated by scripts. There are also some configuration files on the master itself, we'll describe them in the order of execution:

Installation source

A complete copy of the installation media is found in `/opt/cluster`. This repository is used during installation, use it also when you have to install additional packages on the master ot on the nodes.

DHCP

The DHCP server on the master is configured to offer the nodes their stable IP addresses. The server is configured to not disturb an existing DHCP server. Only cluster nodes will get an IP address.

The mapping from the MAC address of each node to it's IP number is defined in `/etc/dhcpd.conf`. In case of a node replacement, you have to adjust the file with the provided MAC address.

TFTP

After getting an IP address, the node requests the PXE boot loader over TFTP. The TFTP server is accessible for the cluster nodes only. There is no information leaked out to other systems in your LAN. All files are located in `/tftpboot`.

We use PXElinux as a network aware boot loader for the installer.

PXE

PXElinux requests it's configuration file which is served out of `/tftpboot/pxelinux.cfg` and it's menu in `/tftpboot/message`. These files are generated using the script `generatePXE` which is located in `/opt/cluster/spool/admin`. Feel free to adjust the script and rerun it to regenerate the configuration files in case you like to boot different floppy images or linux kernels.

The default option for the next boot of the nodes is controlled by the parameter of `generatePXE`. Use `generatePXE install` to force a reinstallation of all nodes.

Kickstart

Anaconda out of the loaded kernel and initrd will ask for the personal configuration file over NFS. This is prepared using the script `generateKS` in `/opt/cluster/admin`. Adjust and rerun this script for changes in the filesystem layout, the network settings or for additional packages provided by the distribution.

We'll set a static IP number for each node to give you a stable network operation even during an outage of the DHCP server located on the master.

Postinstallation

At the end of the installation done by the distributions installer, the script `/opt/cluster/admin/postinstall.sh` is called. We do some basic tasks like copying over our set of configuration files, switching of services which are not needed in a cluster environment and installing additional packages.

In the postinstall stage, the system isn't yet fully up and still running the installer's kernel. We have some limits as most of the daemons are not yet running and some esential kernel modules are not loadable. We introduced a second stage to do advanced tasks, after the first successful boot of the node.

First boot

The script `/opt/cluster/admin/firstboot.sh` is run during the first boot of the node. All services are up and running, the final kernel is installed. We use this script to install all software packages, the job scheduler and the monitoring.

Adjust this script to add additional software during the installation of your nodes.

Configration files

All adjusted configuration files are located in `/opt/cluster/spool/node-config`. They are copied over to the node in the postinstallation process. You may add additional files or adjust them.

The script `clusterSyncFiles` out of `/opt/cluster/admin` takes care to synchronize the most changed configuration files, push them out to the nodes and keeps `/opt/cluster/spool/node-config` up to date.

Prepared services

A cluster is always a trusted system. Whenever possible we separate the cluster LAN with the nodes from your cooperate LAN. A firewall permit only the cluster LAN to access the essential services. This allows the secure usage of the potentional insecure IP based authentication used by several services. Using this approach, we gain a lot of performance from avoiding cryptographic authentication of the communication.

NFS

`/home` and `/opt/cluster` are shared to all nodes using NFS. To add other shared directories, add them on the master in `/etc/exports`. On the nodes add it to `/etc/fstab`. Take care to add them also to `/opt/cluster/admin/generateAY`. Rerun the script to avoid reinstalled nodes won't get the mount.

All exported filesystems are located in `/export` on the master. To provide a convenient access on the master, they are bind mounted to the right place. You'll see a bind mount from `/export/home` to `/home` in `/etc/fstab`. On the nodes a NFS mount from the master to `/home`. This approach is a best practice for NFS shared directories. We recommend you to add addtional filesystems using the same way.

SSH

Inside the cluster all users have password free ssh access. We use RSH based host authentication. There is no need to create a personal keypair. root use ssh-key-auth, as host based authentication is not allowed for root by OpenSSH.

X11 is fully forwarded when using ssh. Access the cluster with X11 tunneling, all nodes will then be able to open a display on your workstation.

R-Services

Some applications relies on rsh / rlogin to distribute their work. We enabled password free rsh / rlogin access inside the cluster.

Timeserver

The master is operating a NTP server for the nodes. This avoids unneeded load on the upwards time source. It will synchronize to the public available timeservers or your local timeserver.

Caching DNS

To avoid unneeded load on your cooperatre DNS server, the master operates a caching only DNS server. There are no additional zones inside the cluster. We use exclusively `/etc/hosts` for the IP to name mapping.

Logserver

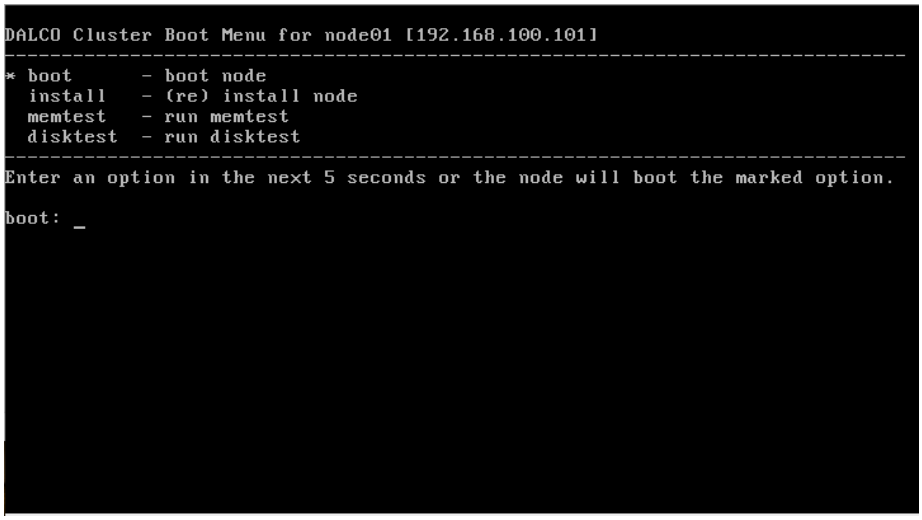
The master is logserver for all the nodes. You do not have to login to a node to access it's

`/var/log/messages`. Just use the one of the master which consolidates all messages.

Node installation

There are as many philosophies how to operate a cluster as there are administrators in the world. We use the approach to install all needed software locally. Only data directories like `/home` are shared. This avoids trafic during boot and startup of your programs. The installation process is fully automated. A typical node (re-)installation runs completly unattendend.

While booting, you'll get a small boot menu:



Without interaction, the node will boot the default option. The default option is marked with an asterisk. During operation, this is `boot` which boots up the system from the local disk. The default is controlled by `generatePXE` and resetted during the installation back to `boot`.

The option `install` will reinstall the node. `disktest` and `memtest` are provided for our service staff.

The unattendend installation is fully controlled by the AutoYast file generated by `generateAY`, the `postinstall.sh` and the `firstboot.sh` scripts. All those steps are logged. You find the logfiles after installation in the `/root` directory.

Accessing the cluster

Your main interface will be the Linux command line. Of course, there are also many programs running a graphical interface. But most computational tasks are just running in batch mode. Don't worry, after the first steps you'll become familiar with the Linux shell and start to like it!

You need the name or the IP address of the master of your cluster. In the following examples, we use `master`. Additional, you need a personal login and password. We use `dalco` for all the examples.

Command line and X11

Depending on your workstation, there are some different programs used to get access to the command line. All of them are using the `ssh` protocol. `ssh` is a secure and encrypted protoccol especially written for shell access. Everything is done to keep the communication secure so nobody will be able to listen to it.

X11 is the classic graphic environment for UNIX systems. A big advantage is the network capability. A program running on one system is able to open a window on a different system. X11 itself is unencrypted and has a sophisticated access scheme which is not easy to understand. `ssh` once again provides a secure and simple way to tunnel X11 data.

You are using a UNIX like operating system

All flavors of Linux, UNIX and BSD provides `ssh` out of the box. Also MacOS X is such a system. Open e terminal and call `ssh`:

```
$ ssh master
```

The first time you connect, `ssh` will ask you if the key presented by the master should be accepted. Just enter yes. This key will be stored on your workstation and compared to the key presented by the server in future connections. This ensures nobody is able to replace your master with a hacked system.

In case you are really replacing or reinstalling your cluster, you need to remove the local copy of the server key. Run the folling command:

```
$ rm ~/.ssh/known_hosts
```

In case your local login don't match with the login on the master, pass the master's login to `ssh`:

```
$ ssh dalco@master
```

`ssh` provides a convenient way to tunnel X11. Just enable the tunneling in the `ssh` call:

```
$ ssh -X master
```

Start your application. Every graphical window will be opened directly on your desktop.

`ssh` protects your local X11 server from unfriendly access. There are some rare applications which needs additional rights and presents you the following error message:

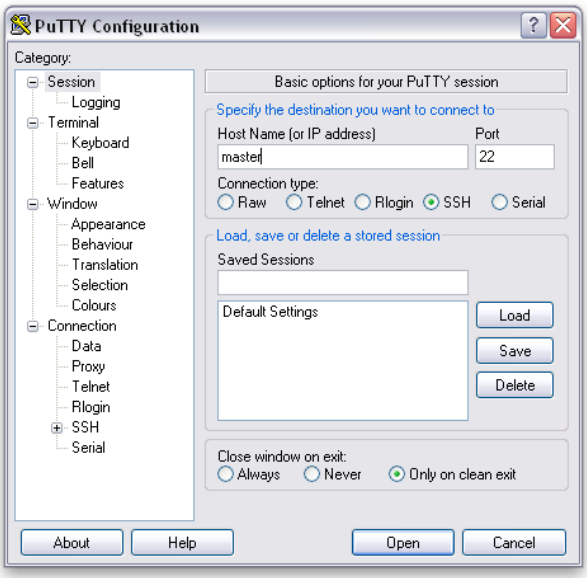
```
$ xxx
X Error of failed request:  BadAtom (invalid Atom parameter)
  Major opcode of failed request:  20 (X_GetProperty)
  Atom id in failed request:  0x17
  Serial number of failed request:  3
  Current serial number in output stream:  3
```

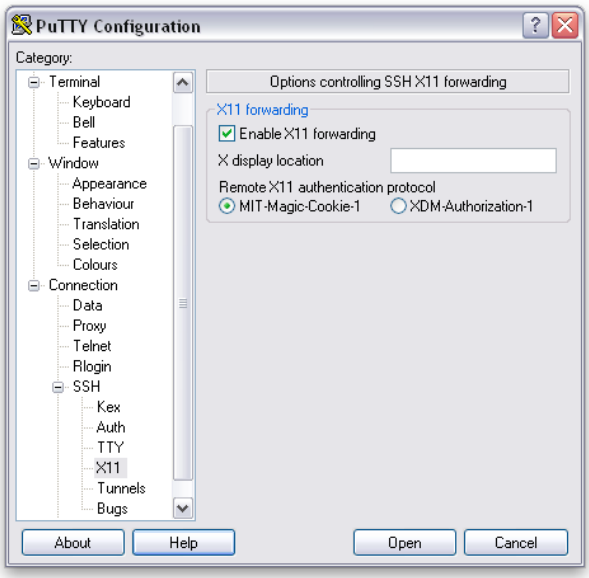
In this case, use the *trusted X11 forwarding* of ssh:

```
$ ssh -Y master
```

You are using a Windows based workstation

Users of a Windows workstation should install Putty 1), a local copy is available on the webinterface of the cluster, section *Downloads*.





To use X11 enabled applications, you have to install an X11 server like Xming 2) *on your local workstation* and enable the X11 tunneling on the connections option of Putty.

Large commercial X11 servers like ReflexionX or Exceed have built in support for `ssh` and you don't need to start Putty in advance. Be warned - even commercial X11 servers may give you headache with applications which requests features not available in the server.

VNC

Another approach to access the cluster is the *Virtual Network Computing*. VNC is a quite old development to share graphical desktops and we see several different implementations today. You start the server part of VNC on the cluster and connect using a VNC viewer. You are able to detach the session and let all your programs continue to run. Reattach the session from the same or a different workstation. Even simultanous connections are allowed.

Start the VNC server

All our clusters have the VNC server allready installed. RedHat based installations have *RealVNC* 3), SuSE uses *TightVNC* 4). Both servers uses the same protocol. A *RealVNC* client talks to a *TightVNC* server and vice versa.

Check with your system administrator that the ports 5800-5999/tcp are open in the firewall on the master.

Login on the master using `ssh` or Putty and start your personal VNC server. The first time you do this, you will be asked for a password:

```
$ vncserver

You will require a password to access your desktops.

Password: secret
Verify:   secret
Would you like to enter a view-only password (y/n)? n

New 'X' desktop is master:1

Creating default startup script /home/dalco/.vnc/xstartup
Starting applications specified in /home/dalco/.vnc/xstartup
Log file is /home/dalco/.vnc/master:1.log
```

The password is remembered, next time you start the VNC server it doesn't ask you again:

```
$ vncserver

New 'master:1 (dalco)' desktop is master:1

Starting applications specified in /home/brubischon/.vnc/xstartup
Log file is /home/brubischon/.vnc/scoop.cluster:1.log
```

One line is important: `desktop is master:1` Remember the number, it's your personal display number for the time the server runs. Each session get it's unique number to avoid clashes.

The VNC server tries to figure out the perfect size of your virtual screen. Sometimes this works, somtimes you have to adjust the size during launch of the server:

```
$ vncserver -geometry 1024x768
```

Stopping the server

Use the following command to get rid of the running VNC session on the server. Select your display number as parameter:

```
$ vncserver -kill :1
```

Download the client

On a common Linux workstation, the `vncviewer` is allready installed or available on the distributions CD ROM.

On a Windows workstation, you have to download the viewer. You don't need the server, the viewer only binary is all you need. Get it from one of the following sites:

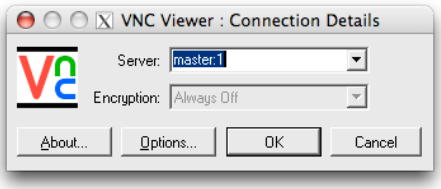
- RealVNC: http://www.realvnc.com/products/free/4.1/index.html
- TightVNC: http://www.tightvnc.com/download.html

MacOS X has a VNC viewer integrated. Use *Connect to server* and enter the URL vnc://master:5901 while the last digit is your display number. For more speed and comfort we recommend an alternative viewer:

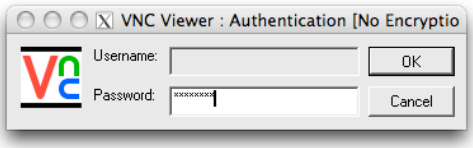
- Chicken of the VNC http://sourceforge.net/projects/cotvnc/

Access your session

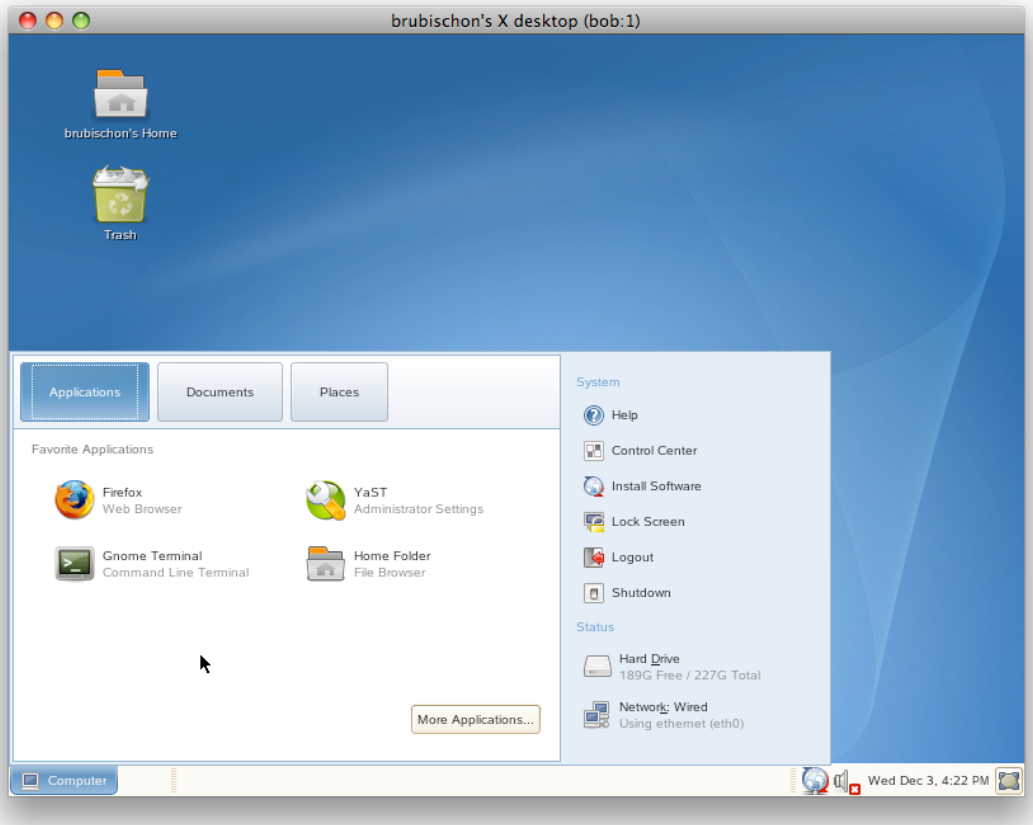
Launch the appropriate viewer and connect to the hostname and the display number of your personal session:



You will be propted for your password and get the desktop:



Close the window anytime you like and reconnect later. The programs running on the cluster will continue their work.



Improve your session setup

The VNC server starts a basic environment which is available on any X11 installation. The look and feel is somewhat old fashioned and only suitable for an experienced user.

You may improve the session by adjusting the file `~/.vnc/xstartup`. It's original content starts up `twm` as window manager:

```
xrdb $HOME/.Xresources
xsetroot -solid grey
xterm -geometry 80x24+10+10 -ls -title "$VNCDESKTOP Desktop" &
twm &
```

An example which starts up a full Gnome or KDE session including a fallback in case both are not available:

```
#!/bin/bash

GSESSION="$(which gnome-session 2>/dev/null)"
STARTKDE="$(which startkde 2>/dev/null)"
TWM="$(which twm 2>/dev/null)"

if [ -n "$GSESSION" ]; then
    # by default, we run GNOME.
    exec "$GSESSION"
elif [ -n "$STARTKDE" ]; then
    # if GNOME isn't installed, try KDE.
    exec "$STARTKDE"
else
```

```
    # Still no desktop? Start twm
    exec "$TWM"
fi
```

Kill and restart your server to get the adjusted environment.

Securing your VNC session

In a typical cooperate LAN there is no need to do encryption of the network traffic between your workstation and the cluster. This looks different when you connect through a typical university network or through the internet.

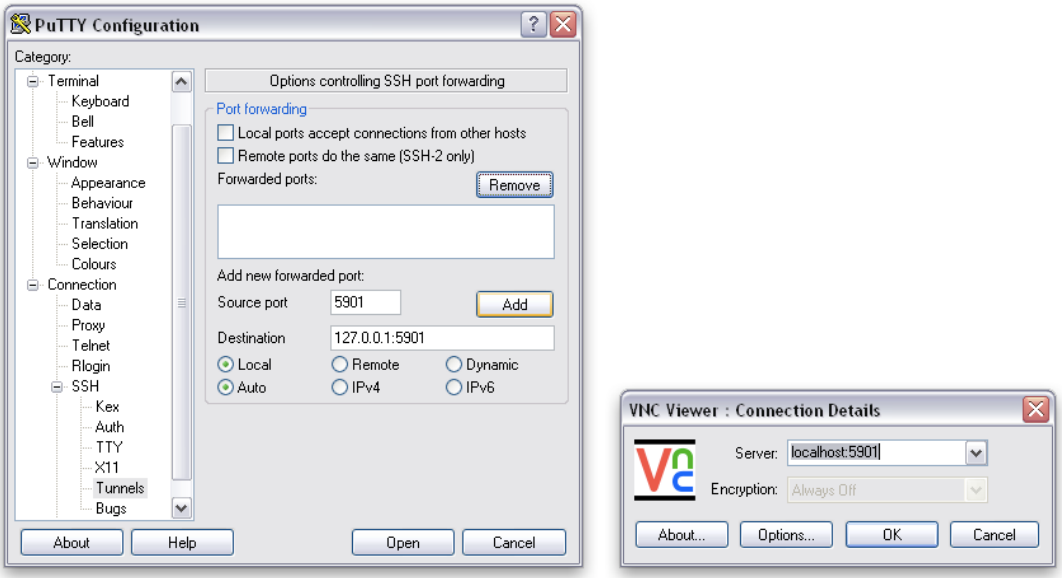
Once again `ssh` provides a simple solution. You remember the screen number from your VNC server? Add 5900 to it. This is the port we will forward through ssh.

Using a comand line `ssh`, add the paramter -L. We use display :1 which results in port 5901:

```
$ ssh -L 5901:127.0.0.1:5901 master
```

Now start your vnc server and connect to `localhost:1`. The whole traffic will be encrypted and tunneled through `ssh`.

Putty is also able to do forwarding. Before connecting, go into the appropriate configuration option and add a tunnel. Afterwards, start your VNC viewer:



As you see, the displaynumber (:1 in this case) and the portnumber (:5901 in this case) are both accepted.

NX

NX is an alternative to VNC to access clusters. It provides good performance using efficient compressing and caching algorithm and asynchronous screen updates. The traffic is automatically tunneled through `ssh` which encrypts everything between the cluster and your workstation. No additional open ports are needed on the firewall. We use FreeNX 5) on the cluster side which is an

OpenSource implementation of the NX server. Look for the directory `/usr/NX` to check if NX is installed on your system. Our support will assisst you in installing FreeNX if it is not allready preloaded.

Download the client

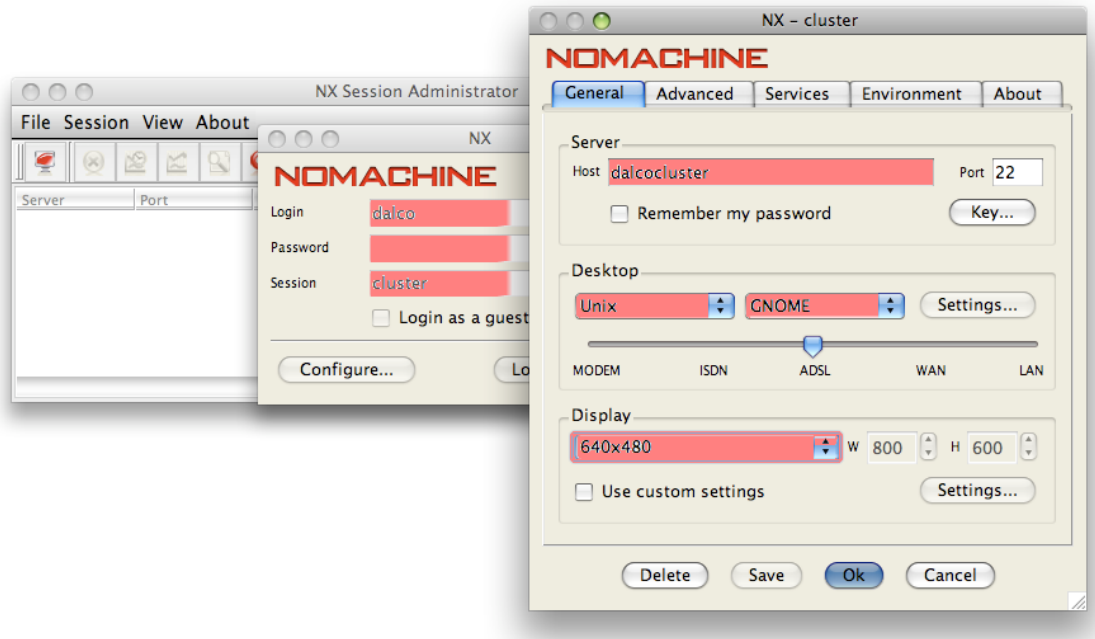
Get the appropriate client for your workstation from the cluster's webinterface or from Nomachine 6). It is closed source, but free of charge.

Start a session

There are two ways to access your cluster. It depends on your personal preference which one you will choose. For both approaches, start the *NX Session Administrator* and choose *File*, *New NX Session*.

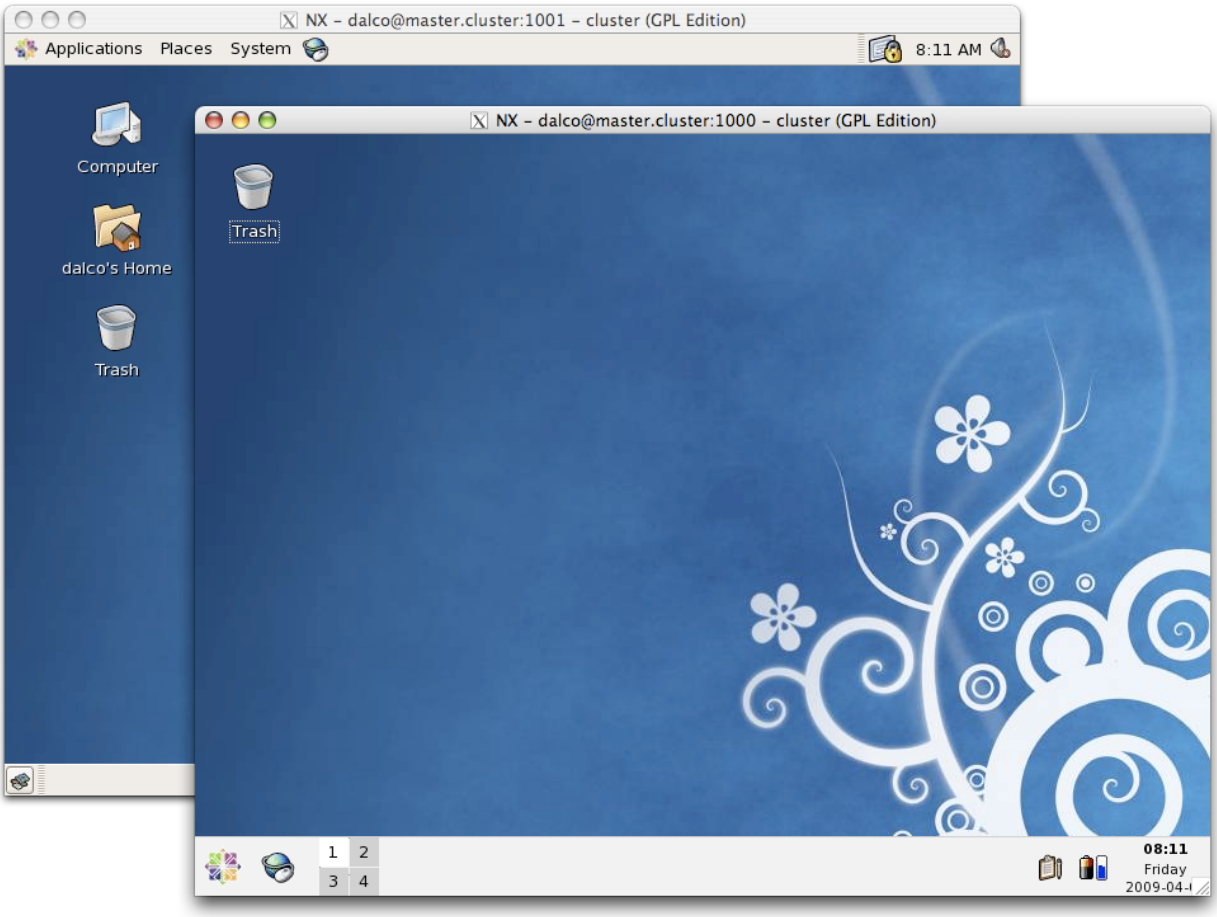
Enter your login and password in the box. Choose a nice name for your session with the cluster. All the settings of the session are stored under this name. First time press the *Configure* button to customize your session.

Full desktop



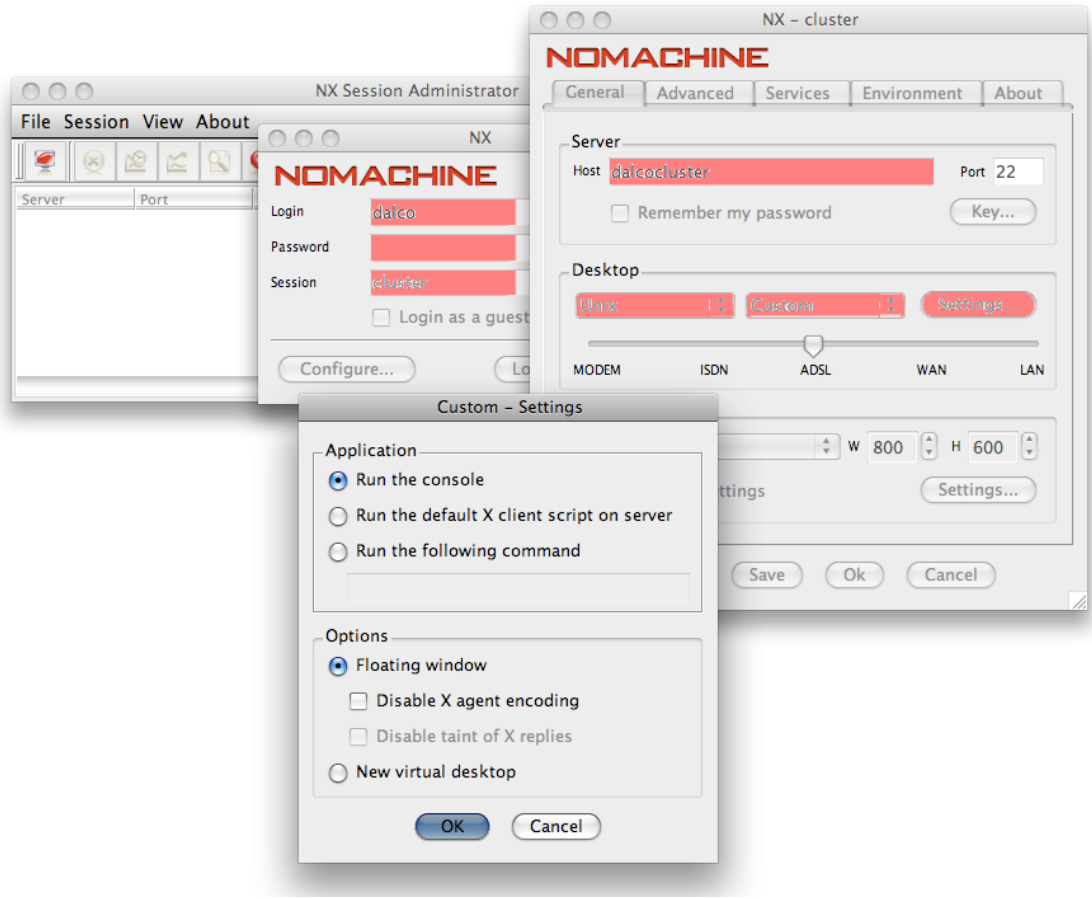
Enter the hostname or IP number of your cluster's master. Choose a *UNIX* Desktop, select *Gnome* or *KDE*. Not all clusters have both desktops installed, so start first with Gnome. Choose an appropriate window size which fits to your local screen. Using *Fullscreen* you will get a real Linux Workstation look & feel.

Save this configuration and log in. You will get a window or a full screen view of the system running on the cluster:



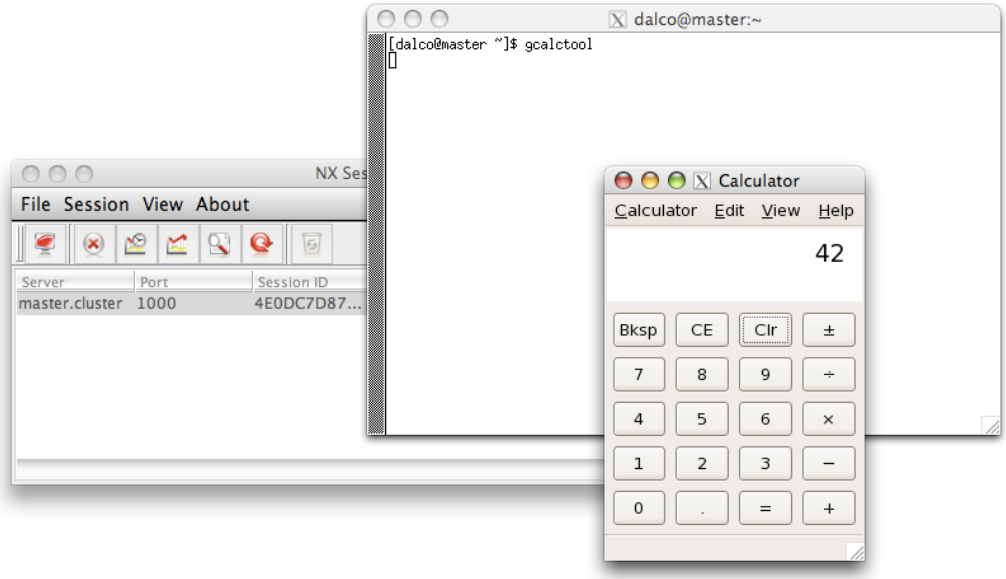
This session is detachable. When you close your window, you are able to reconnect to your session anytime later. All open applications will continue to run.

Terminal only



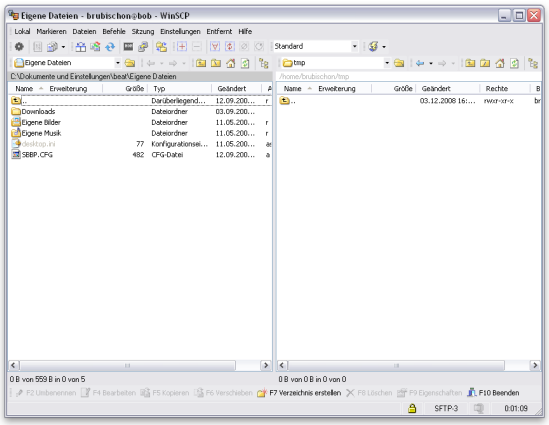
Enter the hostname or IP number of your cluster's master. Choose a *UNIX* Desktop, *Custom*. Click the *Settings…* button and check if the options *Run the console* and *Floating window* are selected.

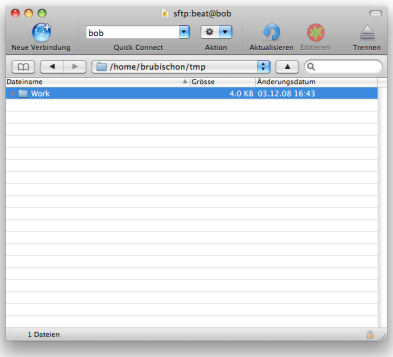
Save the configuration and log in. You will get a terminal with full X11 forwarding. All windows from the cluster are rootless integrated into your desktop:



Up- and download of data

Using `sftp` or the various graphical frontends the up- and download of data is quite simple. We recommend WinSCP 7) for Windows users or Cyberduck 8) for MacOS X users. The typical Linux desktops Gnome and KDE have built in support for sftp. Open a filemanager window and go to sftp://master/ in Gnome and fish://master/ in KDE.





As a Linux user, you have probably also the possibility to access the cluster wide filesystems on your workstation using NFS.

Your homedirectory in the cluster is shared accross all nodes. Any file you place in `/home/<your login>` is visible on the master and on all nodes. It's the starting point for all of your work. There may be additional shared data directories available.

On the nodes, you'll find a public writeable `/scratch` directory. It's primary for applications using local disk storage. Don't expect data placed in `/scratch` will survive a long time, the directory is designed for *temporary* stuff only. Please clean up your temporary stuff and keep the nodes clean. When not doing so, you'll get sooner or later real troubles - with your system administrator or with your cluster.

Command line basics

After logging in, you get a command prompt. We will now describe the most common commands used in this environment. Not all as a typical Linux installation provides you more then 3000 commands!

Navigating in the tree

All disk and network shares are mounted in a single tree. There are no separate drives, everything is in a *single namespace*. The most used directories and their usage:

```
/bin                  Basic commands, in your ''PATH''
/sbin                 System commands, only used by ''root''
/lib, /lib64          Basic libraries, 32 and 64 bit variants
/usr/bin              More commands, in your ''PATH''
/usr/sbin             More system commands, only used by ''root''
/usr/lib, /usr/lib64 More libraries, 32 and 64 bit variants
/opt                  Software packages
/home/.../            Your personal home
```

```
/root                System administrators home
/tmp                 Temporary stuff, may be small!
/scratch             Temporary stuff
```

When logging in, you'll start usually in your Home. This directory is abbreviated with a ~.

- `ls` shows you all files and directories in the current directory
- `ls -l` shows additional information like size and last modification time
- `cd <directory>` dive into the directory
- `cd ..` goes one directory up
- `cd ~` brings you back into your home
- `rm <file>` removes a file
- `mkdir <directory>` creates a new directory
- `rmdir <directory>` removes an **emtpy** directory
- `rm -Rf <directory>` removes a directory including it's content
- `less <file>` look into a file
- `vi <file>` edit a file

`<file>` and `<directory>` may be replaced by *wildcards*. Use a `*` to say *all files and directories in this directory*. Useful for `rm`, but also really dangerous.

Edit a file

There are many, many different editors for Linux. Usually the hardcore Linux users, called geeks, have quite intensive discussions which one is the best. They call this *editor wars*. But only one editor is available on every Linux, BSD, MacOS and UNIX system, it's name is `vi`. We choose this one for our manual even we understand why you will prefere another one for your daily work.

Start your editor session by typing `vi <filename>` in your shell. In case you choose an existing file, it will be loaded. Otherwise the file will be created as soon as you quit `vi`.

`vi` knows two modes. First you beginn in the navigation mode. Use your cursor keys and PgUp / PgDown to navigte in the file. Press **i** to start **i**nsert mode.

In the insert mode, you are able to write text. Press **<ESC>** to come back into command mode.

In the command mode, two keys are useful: **x** will delete the character under the current cursor position, **u** will undo your last change.

To leave `vi` without saving the file, type `:q!` in the navigation mode. To leave `vi` with saving the file, type `:wq`.

There are many, many additional options. But belief us: The described ones are enough to start and edit your files.

Alternatives to `vi` are `emacs`, `joe`, `nedit`, `nano`, `pico`, `gedit`... There are thousends, but none of them will be usable in any situation. Feel free to ask us for recommondations applicable to your cluster!

Pack and unpack archives

Software and data is usually distributed in a compressed tar file. Use the following matrix to get the right commands to create, view and unpack such files:

Filename	Create	List	Unpack	Description
`*.tar`	`tar cvf <tarfile> <directories and files>`	`tar tvf <tarfile>`	`tar xvf <tarfile>`	Uncompressed
`*.tar.gz`	`tar cvfz <tarfile> <directories and files>`	`tar tvfz <tarfile>`	`tar xvfz <tarfile>`	Portable compression
`*.tar.bz2`	`tar cvfj <tarfile> <directories and files>`	`tar tvfj <tarfile>`	`tar xvfj <tarfile>`	Only for modern Linux systems

Manage your processes

When starting a program, it usually builds a process. Multiple processes could run at the same time. Most processes are just waiting that something happens. You get an overview about your processes using the following commands:

- `top` interactive process watcher
- `ps aux` list all processes
- `ps uax | grep dalco` list processes of user dalco
- `kill <PID>` send a termination signal to process
- `kill -9 <PID>` force termination of a process, be careful
- `killall <program>` stop all processes running program
- `killall -9 <program>` same with force, be careful

Adjust your environment

While the basic installation on a Linux cluster just works, most additional applications needs some special environment. There are some variables which must be adjusted. Other applications requires that you include a global configuration file in your personal one. See the manual of the application or ask our support in case of questions.

All adjustements must be done in `~/.bashrc`. Take the file in a editor, do the changes, logout and login to activate the changes. Jobs submitted by `mpirun`, `ssh` / `rsh` or through the job scheduler are also using those settings.

Add a directory to your program searchpath:

```
export PATH=/opt/fluent/6.3.26/Fluent.Inc/bin:$PATH
```

This allows you to start all programs out of the added directory without typing the full path to them.

Add a directory to your library searchpath:

```
export LD_LIBRARY_PATH=/opt/openmpi/lib:$LD_LIBRARY_PATH
```

This step is needed for applications searching dynamic loadad libraries.

Add a license server:

```
export LM_LICENSE_FILE=1234@server
```

You will get the port and the servername from your system administrator.

Include an additional file provided from the application:

```
. /opt/intel/cce/10.1.017/bin/iccvars.sh
```

Change your password

Login on the master of your cluster and call `passwd`. You will be asked once for your existing password and twice for the new one. There are some checks that you won't choose a too simple one.

root is able to change the passwords of other users. Call `passwd <user>`. You will be asked twice for the new password. The checks against simple passwords are also done, but not enforced.

Accessing a node

You are able to use `ssh`, `rlogin` and `rsh` without a password inside the cluster. You find the full list of nodes in `/etc/hosts`. X11 is enabled automatically inside the cluster. Each node is able to open a window on your desktop.

Distributing commands and files

Quite often you have to execute a command on all nodes or copy out a file to them. The DALCO Cluster Suite provides the C3 tools 9) with the commands `cexec` and `cpush`. Some examples:

Executing a command on all nodes

```
$ cexec uptime
************************ cluster ************************
--------- node01---------
 10:00:30 up 1 min,  0 users,  load average: 1.87, 0.73, 0.26
--------- node02---------
 10:00:31 up 1 min,  0 users,  load average: 2.84, 1.24, 0.46
```

You are able to cexecing anything which doesn't read from your terminal - so avoid programs like `vi`.

Pushing out a file

```
$ cpush /scratch/mybiginputfile
building file list ... done
building file list ... done
mybiginputfile
mybiginputfile

sent 91 bytes  received 60 bytes  100.67 bytes/sec
```

```
total size is 1862  speedup is 12.33

sent 91 bytes  received 60 bytes  100.67 bytes/sec
total size is 1862  speedup is 12.33
```

`cpush` accepts an additional parameter as destination directory. Use `cpush /home/file /scratch` as an example.

cpush doesn't allow to copy a directory, only files. We create usually a tar of the specific directory, copy it to the nodes using cpush and unpack them in a cexec call.

Finding a process

```
$ cexec "ps aux | grep myprog | grep -v grep"
************************* cluster **************************
--------- node01---------
dalco     3871  0.0  0.2   1716   648 ?        Ss   09:59   0:00 ./myprog
--------- node02---------
dalco     3870  0.0  0.2   1720   652 ?        Ss   09:59   0:00 ./myprog
```

Listing your processes

```
$ cexec "ps aux | grep ^dalco"
************************* cluster **************************
--------- node01---------
dalco     4738  0.0  0.7   3476  1844 ?        Rs   10:03   0:00 orted --bootproxy 1 --name 0.0.1
dalco     4739 93.7  1.1   6464  3036 ?        R    10:03   0:07 ./mping 0 1k
--------- node02---------
dalco     4737  0.2  0.7   3476  1844 ?        Ss   10:03   0:00 orted --bootproxy 1 --name 0.0.2
dalco     4738 97.7  1.2  14668  3284 ?        R    10:03   0:07 ./mping 0 1k
```

1) http://www.chiark.greenend.org.uk/~sgtatham/putty/
2) http://www.straightrunning.com/XmingNotes/
3) http://www.realvnc.com/
4) http://www.tightvnc.com/
5) http://freenx.berlios.de/
6) http://www.nomachine.com/download.php
7) http://winscp.net/
8) http://cyberduck.ch/
9) http://www.csm.ornl.gov/torc/C3/

Administrators Tasks

In daily practise we see some tasks which must be done frequently. Most of them are the same as for administrating a stand alone Linux workstation or a server. Some are a bit special as you need to keep all nodes in sync.

Your first step will be a terminal connected to the cluster. You need the name or the IP address of the master. Most tasks should be done as a regular user, for administrative tasks you need to have the `root` password.

Booting and shutdown

There are some dependencies between the master and the nodes. This enforce you to fully boot the master before booting the nodes. In of case one or more nodes are booted before the master finish it's boot process, you'll find shared directories like `/home` empty. The nodes won't appear in the job scheduler. There is no danger jobs will be started on nodes missing their NFS mounts.

To fix such an issue, reboot the nodes again:

```
# cexec /sbin/reboot
```

They will start, reach all resources and integrate themself to the job scheduler.

To shutdown the cluster first shutdown the nodes:

```
# cexec /sbin/poweroff
```

After successful shutdown of all nodes, power off the master:

```
# poweroff
```

Adding users

Add users with the distribution's tools like YaST, `system-config-users` or simply by editing the files in `/etc` and creating the home directory. Distrbute the changes using `clusterSyncFiles`. This updates all nodes and also the file used during installation.

The homedirectory created in `/home` is always distributed over all nodes using NFS.

Be sure to have all nodes up and running when using `clusterSyncFiles`. A node which is offline won't get the new file. One workaround is to set the node in the installation mode by using `setDefault <nodename> install`. Another possibility is to run `clusterSyncFiles` once again when the node is back in production.

Installing software

We see several different ways how software vendors packs their programs. There is no common way as the distributions itself are different and the programmers often prefere their own installation mechanism. Instead of relaying on a single way, we implemented different hooks to automate the installation process during the reinstallation of a node. Choose the appropriate way to install your

software:

Software used by a single user

The easiest way is to follow the install instructions by the software distributor and install the package into the personal home. Probably you have to set some environment variables in `~/.bashrc`.

This is the only usable way when the user actively develops his package.

Packages provided by the distribution

Locate the software package in the local installation source. It's somewehere in in `/opt/cluster`. Install it on the master and on the nodes:

RedHat based installations:

```
# rpm -i /opt/cluster/redhat/.../mycoollib-1.0.rpm
# cexec rpm -i /opt/cluster/redhat/.../mycoollib-1.0.rpm
```

or when RHN is configured:

```
# yum install mycoollib
# cexec yum install mycoollib
```

SLES11, OpenSuSE >= 10.3 based installation:

```
# zypper install mycoollib
# cexec zypper install mycoollib
```

SLES10, OpenSuSE 10.1/10.2 based installation:

```
# rug install mycoollib
# cexec rug install mycoollib
```

Debian based installation:

```
# apt-get install mycoollib
# cexec apt-get install -y mycoollib
```

Now add the package to the automated installation process. Edit `/opt/cluster/admin/generateKS` / `/opt/cluster/admin/generateAY`, add the package to the list and rerun the script.

RPM packages provided by the software vendor

Put the packages into `/opt/cluster/spool`. Install them on the master and on the nodes:

```
# rpm -i /opt/cluster/spool/mycoolpackage.rpm
# cexec rpm -i /opt/cluster/spool/mycoolpackage.rpm
```

To autoinstall the package add it to `/opt/cluster/admin/firstboot.sh`, near our own examples:

```
#-----------------------------------------------------
# MyCoolPackage
rpm -i /opt/cluster/spool/mycoolpackage.rpm
```

Check `/root/firstboot.log` after a fresh installation for errors.

DEB packages provided by the software vendor

Put the packages into `/opt/cluster/spool`. Install them on the master and on the nodes:

```
# dpkg -i /opt/cluster/spool/mycoolpackage.rpm
# cexec dpkg -i /opt/cluster/spool/mycoolpackage.rpm
```

To autoinstall the package add it to `/opt/cluster/admin/firstboot.sh`, near our own examples:

```
#-----------------------------------------------------
# MyCoolPackage
dpkg -i /opt/cluster/spool/mycoolpackage.rpm
```

Check `/root/firstboot.log` after a fresh installation for errors.

Software distributed with an installer

Run the installer on the master. Using `/opt` is not a bad idea. In this example, I install MyCoolSoftware in `/opt/mycoolsoft`. Now pack and distribute the software over the nodes:

```
# cd /
# tar cfj /opt/cluster/spool/mycoolsoftware.tar.bz2 opt/mycoolsoft
# cexec "cd /; tar xfj /opt/cluster/spool/mycoolsoftware.tar.bz2 opt/mycoolsoft
```

To autoinstall the package add it to `/opt/cluster/admin/firstboot.sh`, near our own examples:

```
#-----------------------------------------------------
# MyCoolSoftware
cd /
tar xfj /opt/cluster/spool/mycoolsoftware.tar.bz2
cd $OLDPWD
```

There are some software packages which doesn't support this approach. You'll find the software not running, usually missing some files, when started on a node. This is the worst case. You have to find a way to automate the installer.

Do a manual installation on each node individually or use the automated installer in a cexec call.

We place the installer files in `/opt/cluster/spool` and add the installer call to `/opt/cluster/admin/firstboot.sh`:

```
#-----------------------------------------------------
# MyNotSoCoolSoftware
cd /opt/cluster/spool
./install -auto -yes -reallyyes
cd $OLDPWD
```

Check `/root/firstboot.log` after a fresh installation for errors.

Software distributed as source

Compile and place the software in a central directory. GNU Autoconf based packages allows the usage of the `-prefix` paramter. In the following example, we build a hello 1):

```
# mkdir /root/kits/hello
# cd /root/kits/hello
# wget http://ftp.gnu.org/gnu/hello/hello-2.3.tar.gz
# tar xvfz hello-2.3.tar.gz
# cd hello-2.3
# ./configure --prefix=/opt/hello
# make
# make install
# cd /
# tar cvfj /opt/cluster/spool/hello.tar.bz2 opt/hello
# cexec "cd /; tar xfj /opt/cluster/spool/hello.tar.bz2"
```

Now add the installation of hello to `/opt/cluster/admin/firstboot.sh`:

```
#-----------------------------------------------------------
# GNU Hello
cd /
tar xfj /opt/cluster/spool/hello.tar.bz2
cd $OLDPWD
```

Advice the user how to use the software. Usually they'll have to add some environment variables to their `~/.bashrc`:

```
PATH=/opt/hello/bin:$PATH
LD_LIBRARY_PATH=/opt/hello/lib:$LD_LIBRARY_PATH
export PATH LD_LIBRARY_PATH
```

Backup

In the initial design, the nodes dont't contain any user data. As long as your users dont't place data on the nodes itself there is no need to do backups of the nodes. If you change the installation, you have to keep the installation process in sync.

It's different on the master. It contains all the users data, the configuration and the software needed for your daily work. Even we did our best to protect your data with a RAID controller and reliable filesystems. There are cases where you need a recent backup of the data and the system.

In case of questions contact our support. We have several different solutions for backups. We are sure there is also the perfect solution for your needs.

1) http://www.gnu.org/software/hello/

Managing Infiniband

Infiniband is a specialized networking technology for short area networks. Think about a *cluster wide network* or a *data center wide network*. Compared with Ethernet Infiniband offers you more ways to transmit data. Not only TCP/IP, but also reliable stream, message and block passing. Most important for HPC is the ability to do RDMA, *remote direct memory access*. This is used by MPI to pass messages between nodes without the help of the kernel.

On top of Infiniband you find a software stack called OFED, the *OpenFabrics Enterprise Distribution*. Depending on your environment, we preinstalled the official OFED distribution 1), some vendor specific OFED package or the base support which is included into the Linux kernel.

Subnet manager

While Ethernet only allows simple Fat tree setups, Infiniband allows near any network topology. Fat trees, clos networks and even rings are possible. The data is routed in a way which provides you the best possible bandwith of your network.

Distibution the LID, the *local id* and the routing tables is done by the *Subnet Manager*. This is a daemon running on one or more of the nodes attached to the Infiniband fabric. It polls the current network once every 10 seconds and modifies the routing tables in the switches. You may have more then one Subnet Manager in your network, but only one of them is active.

As long as no Subnet Manager is running, the fabric doesn't forward any data. In this case, check for the process `opensmd`, start it using `/etc/init.d/opensmd start` and check it's logfile in `/var/log/opensm.log`.

Checking connectivity

We configured on each node equipped with an Infiniband card an IP using IPoIB, *IP over Infiniband*. Use a simple `ping` to check the connectivity:

```
node01:~ # ping node02i
PING node02i (192.168.1.102) 56(84) bytes of data.
64 bytes from node02i (192.168.1.102): icmp_seq=1 ttl=64 time=0.061 ms
64 bytes from node02i (192.168.1.102): icmp_seq=2 ttl=64 time=0.063 ms
64 bytes from node02i (192.168.1.102): icmp_seq=3 ttl=64 time=0.059 ms

--- node02i ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1999ms
rtt min/avg/max/mdev = 0.059/0.061/0.063/0.001 ms
```

Checking for errors

Infiniband uses the highest available data rates on the wire. This results in constraints in the cable lenght and gives errors in case of badly plugged cables. We preinstalled tools to help you find bad connections.

Display the current error counters:

```
node01:~ # ibcheckerrors
```

```
#warn: counter XmtDiscards = 1505       (threshold 100) lid 3 port 1
Error check on lid 3 (node02 HCA-1) port 1:  FAILED

## Summary: 2 nodes checked, 0 bad nodes found
##          2 ports checked, 1 ports have errors beyond threshold
```

Reset the error counters:

```
node01:~ # ibclearerrors

## Summary: 2 nodes cleared 0 errors
```

It's not uncommon that the counters will show you bad values after a boot of a node or a power cycle of a switch. Reset them and recheck the counters some time later.

1) http://www.openfabrics.org/

Monitoring the cluster

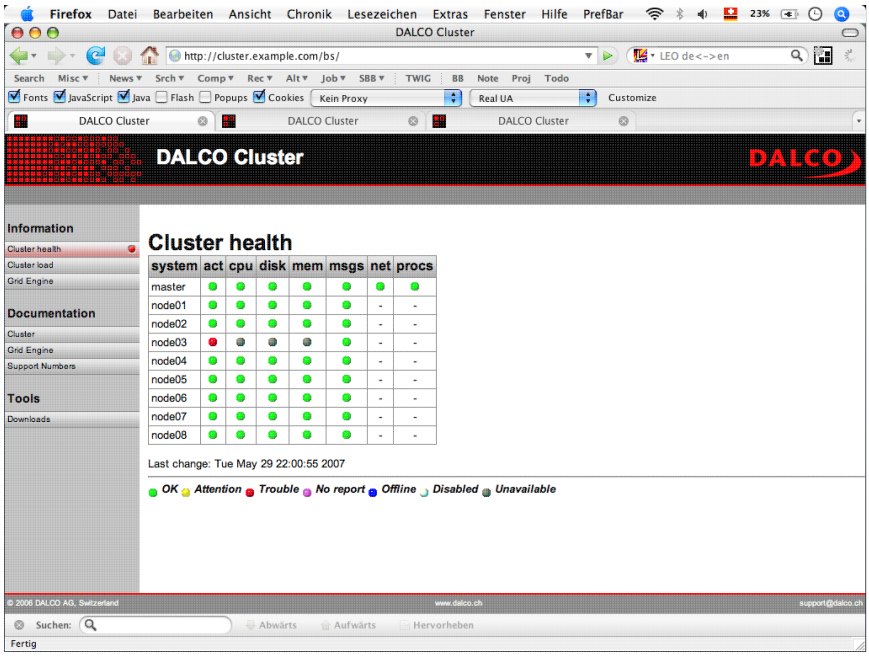
Several tools are installed to provide an overview about the cluster and it's load. You will get a feeling how your cluster behave during the daily usage. Each system is a bit different and a situation which is a problem for one cluster is a normal condition on another.

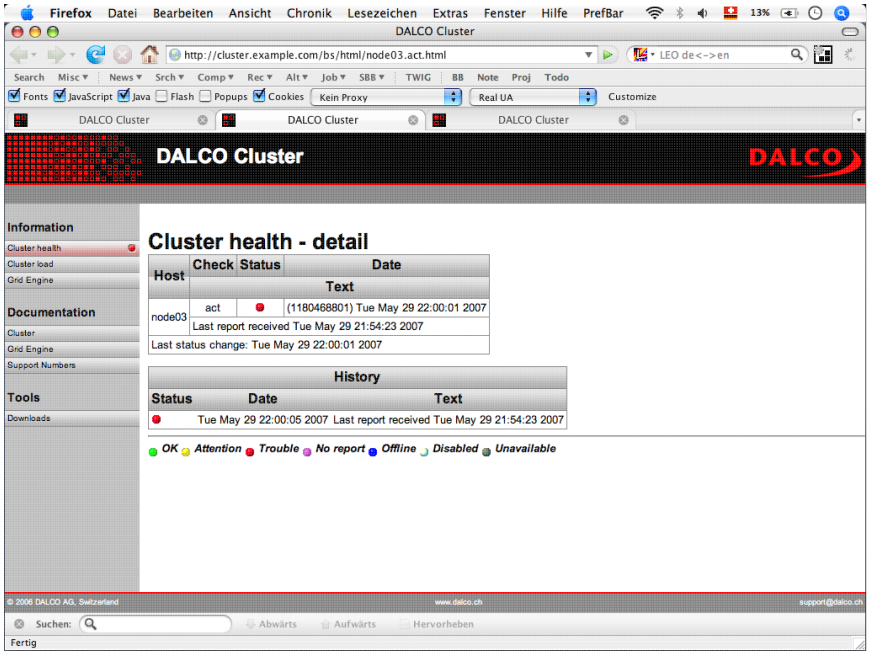
Worst things are usually orphan processes which consumes CPU but don't do any useful work and overbooking of memory or disk. Recongizing such situations helps you to maximize the output of the system.

Webinterface

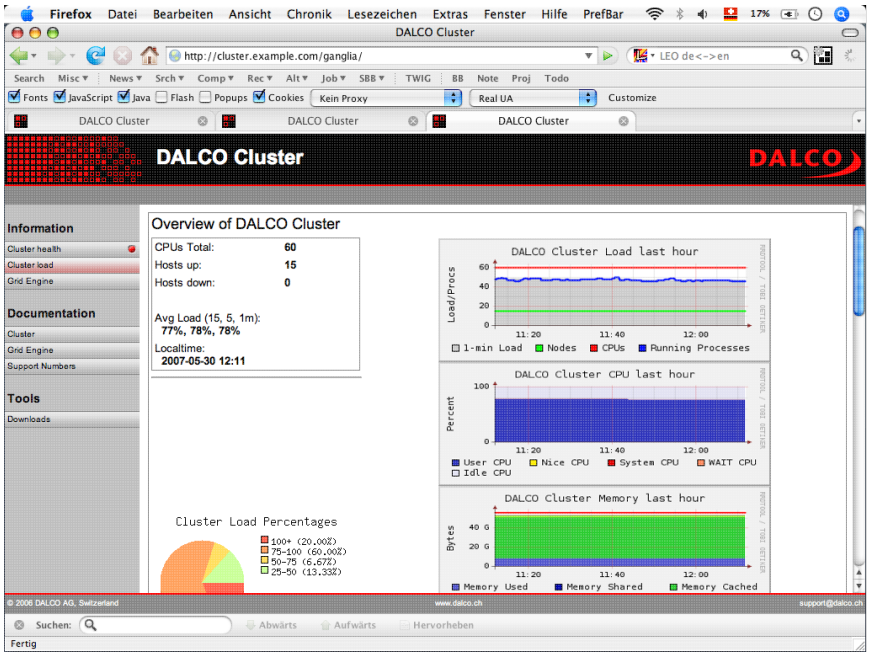
A simple webinterface offers a current view of the running nodes and alerts you in case of an overload. It collects load statistics over the last year. There are no access restrictions. It is completly read only. You can't destroy anything by trying it out!

Cluster health informs you about the state of the master an the nodes. Each row is a system. The columns are the services. Our monitoring knows about the typical usage of a cluster. It don't alert you when you completly fill a node with processes. But you'll get informed as soon as two jobs are running the same time:





Cluster load shows you the long term statistics of various metrics like load, memory and cpu usage:



Command line

While logged in on the master, there are some simple commands to see what's going on. Get a list of working nodes:

```
# cexec uptime
************************ cluster ************************
--------- node01---------
 13:50:40 up  3:51,  0 users,  load average: 0.00, 0.01, 0.00
--------- node02---------
 13:50:40 up  3:51,  0 users,  load average: 0.00, 0.00, 0.00
```

This system is currently idle. The load is calculated for the last one, five and fiteen minutes. It counts all processes waiting for or running on CPU and waiting for disk access. A load of the same value as you have CPUs in the node is a good sign.

In case you have an overloaded node, login by ssh and run `top`:

```
top - 13:55:11 up  3:56,  1 user,  load average: 0.15, 0.04, 0.01
Tasks:  53 total,   3 running,  50 sleeping,   0 stopped,   0 zombie
Cpu(s): 10.3%us, 85.4%sy,  0.0%ni,  0.0%id,  0.0%wa,  4.2%hi,  0.0%si,  0.0%st
Mem:    255604k total,   134284k used,   121320k free,    24584k buffers
Swap:  1052248k total,        0k used,  1052248k free,    73508k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 5680 dalco     25   0  6472 3036 2228 R 99.5  1.2   0:08.49 mping
  296 root      10  -5     0    0    0 S  0.5  0.0   0:00.23 kjournald
    1 root      15   0  2060  652  560 S  0.0  0.3   0:00.89 init
    2 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 migration/0
```

It's the same node as before. There is a shortly started process and the load goes slowly up. We see a process `mping` run by user `dalco`. It consumes near 100% of the available CPU. It is a small program as it consumes just 1.5% memory.

To get the amount of usable memory, take the total amount (255604k) minus the amount of used memory (134284k) plus the buffers (24584k) plus the cache (73508k). This node would be able to run processes up to 219MBytes without swapping.

Use `kill <PID>` to kill a process. When it doesn't disappears use `kill -9 <PID>` to force the exit. You are able to kill your own processes. root is able to kill any. A second command called `killall <programname>` allows you to kill all processes running a specific program.

Another useful source of information is `/var/log/messages`. Use `less /var/log/messages` to navigate through the file. `tail -f /var/log/messages` to get a realtime view.

Using the cluster

Most of our customers uses the cluster like a service, a mailserver or an intranet server. You log in, probably you'll transfer data to it, compile and debug code and submit your programs to the batch scheduler.

Your first step will be a terminal connected to the cluster. You need the name or the IP address of the master. Additional, you need a personal login and password provided by your system administrator.

Compiling and installing software

Usually common software is installed for all users of the cluster. There are also specific needs to build and install your own software. Mostly in cases where you are actively developing your own code. We advise to put anything in your home. Using this approach you have the code available on all the nodes. But take care to not overfill your home as your administrator (or yourself) won't be happy when all available diskspace is used by your home.

In the base installation, you'll find all GNU development tools. Sometimes we also install additional compilers which creates more efficient code. Compilation and debugging is usually done on the master, using small datasets.

Parallel code should be compiled using `mpicc`, `mpiccx`, `mpif70` or `mpif90`. Don't try to link the needed libraries using standard `gcc` and `ld` as this approach creates non portable build scripts and makefiles.

Probably there are more then one MPI implementation installed on your cluster. Use `mpi-selector-menu` to check which one is system wide default. Or to change your personal setting. Log out and in to activate a change. Be sure to run your code with the same MPI as you used for building.

Running your code

For debugging purposes, you usually run your programs directly on the master. Using a debugger like `gdb` or simply `strace` is near impossible when submitting jobs to batch.

You have also the possibility to run your code on a node. Choose a free one and login there using `ssh`. The full list of nodes is available in `/etc/hosts`.

Runing MPI jobs

The following approach works with all MPICH1 based MPI implementations, with OpenMPI and also with HP-MPI.

To run MPI jobs, you'll have to prepare a list of nodes running your job. Take care your list has an entry for each CPU you'll allocate using `mpirun`. Take care you'll never allocate more CPUs as available on a node. A simple example looks like this:

```
$ cat machinefile
node01
```

```
node01
node02
node02
```

The command to start your program looks like this:

```
$ mpirun -np 4 -machinefile machinefile ./yourprogram your paramters
```

Running MPICH2 jobs

MPICH2 brought a new concept for starting MPI jobs. The idea was to support also systems like Windows without `rsh`/`ssh`. When using MPICH2, LAM-MPI and also Intel MPI you need to prepare some stuff before starting your jobs.

First create a file containing a secret for the communication:

```
$ cat ~/.mpd.conf
MPD_SECRETWORD=secret
```

This step is only needed once per user. Choose a good secret, different to your login password on the cluster.

Create a list containing the nodes you will use for your job. One line per node even it has more then one CPU:

```
$ cat mpd.hosts
node01
node02
```

Now it's time to fire up your mpds. The nodecount `-n` it the number of nodes incremented by one. The master also runs an instance:

```
$ mpdboot -f mpd.hosts -n 3
```

To check if the mpds are running, submit a small command to them:

```
$ mpiexec -np 3 hostname
master.cluster
node01.cluster
node02.cluster
```

For the job execution itself you need a second file containing one line per CPU:

```
$ cat machinefile
node01
node01
node02
node02
```

The command to start your program looks like this:

```
$ mpiexec -machinefile machinefile -n 4./yourprogram your paramters
```

After running your jobs, please clean up your mpds:

```
$ mpdallexit
```

Running PVM jobs

PVM uses a slightly different approach then MPI to distribute workload. You start up a PVM daemon on all nodes you plan to use and your application spawns children using the PVM API. It is not uncommon that the controlling program and the workers are not the same binary. PVM supports different architectures, even we see today a monoculture of AMD64/EM64T capable nodes. Workers may be spawned or ended during calculation.

PVM needs to find it's libraries and binaries. You have to set the environment vairable PVM_ROOT to the installation base of your PVM directory:

```
$ cat .bashrc
...
export PVM_ROOT=/usr/lib/pvm3
...
```

Create a list of nodes you plan to use in your PVM job:

```
$ cat pvmhosts
node01
node02
```

Start the PVM environment by calling `pvm`. You get a prompt. Check for the availability of the nodes and quit `pvm`:

```
$ pvm pvmhosts
pvm> conf
conf
2 hosts, 1 data format
                     HOST     DTID     ARCH   SPEED       DSIG
          node01.cluster    40000  LINUX64   1000   0x00408841
          node02.cluster    40000  LINUX64   1000   0x00408842
pvm> quit
quit

Console: exit handler called
pvmd still running.
```

The PVM daemons are now running. Call your application and tell it how many processes should be started.

During the job run, `pvm` allows you to track the state of your processes:

```
$ pvm
pvm> ps
ps
                     HOST       TID   FLAG 0x COMMAND
          node01.cluster    40002       4/c -
          node01.cluster    40003     6/c,f
                    /home/dalco/povray/bin/pvmpov-3.50c
          node02.cluster    40004     6/c,f
                    /home/dalco/povray/bin/pvmpov-3.50c
pvm> quit
quit
```

```
Console: exit handler called
pvmd still running.
```

After calculation, please stop all the `pvmd`s:

```
$ pvm
pvm> halt
halt
Terminated
```

PVM uses TCP/IP only for the communication between the workers. To take the advantage of a high speed interconnect, you have to reroute the traffic over the provided alternative IP addresses. The argument `ip` in the pvmhosts must be set to the IP address of the interconnect interface:

```
$ cat pvmhosts
node01 ip=192.0.1.1
node02 ip=192.0.1.2
```

Some interconnects like Myrinet provides additional improvements by using an alternative Socket interface. Please contact our support for help to set up your environamnt.

Use the job scheduler!

Now comes the problem: You'll never know if your node or your nodes are really free or if someone else is allready using them. For this reason, you should use the job scheduler which knows about the used nodes and keep it free for you until your job is finished.

Or at least you'll have to discuss with the other cluster users, that you are planning to run jobs.

Running two jobs at the same time an a CPU consumes much more time then running the jobs one after the other. The switching between processes takes several thousends of CPU cycles and happens several times per second.

Using the Grid Engine

When sharing a cluster between several people, the risk of running jobs concurrent is high. This is a big disadvantage for the performance. Two concurrent jobs needs much longer as the same jobs one after the other. A lot of CPU cycles is wasted by the kernel for switching between both processes.

Additional you get the benefit of a queue, which allows you to submit jobs and enjoy your free time while your cluster is busy finishing it's work.

Listing available nodes and CPUs

The command `qhost` displays you a list of all available nodes, the amount of CPUs and memory in each one of them:

```
[dalco@master ~]$ qhost
HOSTNAME                ARCH         NCPU  LOAD  MEMTOT  MEMUSE  SWAPTO  SWAPUS
----------------------------------------------------------------------------------
global                  -               -     -       -       -       -       -
node01                  lx24-x86        2  0.03  249.6M   33.8M    1.0G     0.0
node02                  lx24-x86        2  0.06  249.6M   33.7M    1.0G     0.0
```

In this case, we have two nodes, each one with two CPUs and 1Gig of memory. Both are mostly idle.

Running interactive work

The master of your cluster is typically the location to do the pre- and postprocessing of your data and to develop and debug your code. From time to time it makes sense to allocate a CPU or a node and do processing intensive work there. The Grid Engine offers a command called `qrsh` which allocates the resources and logs you in in the reserved node.

Start such a session using `qrsh`:

```
master$ qrsh
node01$
```

You get a slot aka one cpu reserved until you log out of the shell. Check your allocation by using `qstat`:

```
$ qstat
job-ID  prior   name    user    state submit/start at  queue                    slots
-------------------------------------------------------------------------------------
      6 0.55500 QRLOGIN dalco   r     11/26/2008  10:46:58  all.q@node02.cluster  1
```

For larger computations, you should allocate a full node. Since the version 6.2u3 the Grid Engine offers the possibility to use a node in exclusive mode:

```
master$ qrsh -l excl=true
node01$
```

Another way is to explicitly allocate all the CPUs in a node by using the parallel environment `smp`. Using this approach it is also possible to use a part of a node and left the other ressources free for different jobs. Get the number of CPUs per node from `qhosts`:

```
master$ qrsh -pe smp 2
node01$
```

`qstat` shows the allocated amount of CPUs:

```
$ qstat
job-ID  prior   name    user    state submit/start at  queue                    slots
-----------------------------------------------------------------------------------
      8 0.55500 QRLOGIN dalco   r     11/26/2008  10:46:58  all.q@node02.cluster  2
```

`qstat` by default only lists your jobs. Use `qstat -u '*'` to display the jobs of all users. To make this behaviour default, create a file in your homedirectory containing one line:

```
$ cat .sge_qstat
-u *
```

`qrsh` is limited. In cases the cluster is fully loaded, you'll get an error message:

```
$ qrsh -pe smp 2
Your "qrsh" request could not be scheduled, try again later.
```

Submit a serial job

The real power of the Grid Engine is to process batch jobs. You have to create a job file and submit it to the batch scheduler. A simple example:

```
$ cat example.sub
#!/bin/sh
#$ -N test
#$ -S /bin/sh
#$ -cwd
#$ -j y
#$ -m eas

hostname
```

The lines starting with `#$` are parameters for the Grid Engine. You find all of them in the manpage of `qsub`. We use the following in our example:

- `-N test` is the name of the job
- `-S /bin/sh` selects the shell used to execute the jobfile, default is `csh` which is probably not the one you like
- `-cwd` means that the job is started in the current working directory, default is the home
- `-j y` merges STDOUT and STDERR in the same file
- `-m eas` sends you a mail as soon as the job is finished or crashed

Submit this job using `qsub`:

```
$ qsub example.sub
Your job 9 ("test") has been submitted
```

Monitor your job using `qstat` as described before. Once your job is finished, you'll get the output of our job in the logfiles:

```
$ ls
example.sub  test.o9

$ cat test.o9
node01.cluster
```

Submit a job using a full node

Using the simple approach, you get a *slot*, which typically means a CPU and not a full node. This is great for simple serial applications. From time to time it makes sense to allocate a full node to run a program. Since Grid Engine 6.2u3 we have the possibility to run a job on a node in exclusive mode.

Your job script gets an additional line:

```
$ cat full.sub
#!/bin/sh
#$ -N test
#$ -S /bin/sh
#$ -cwd
#$ -j y
#$ -m eas
#$ -l excl=true

hostname
```

The new parameter:

- `-l excl=true` forces the allocation of a full node for your job

Submit this job using `qsub`:

```
$ qsub full.sub
Your job 10 ("test") has been submitted
```

Monitor your job using `qstat` as described before. Once finished, you'll get the output of our job in the logfiles:

```
$ ls
full.sub  test.o10

$ cat test.010
node01.cluster
```

Submit a job using more then one CPU

For threaded programs like OpenMP codes you should allocate the amount of used CPUs by your program. In this case, use the parallel environment `smp` and allocate more then one or all CPUs you have in the node. Check the amount of available CPUs by using `qhost`.

So your job file gets an additional line:

```
$ cat large.sub
```

```
#!/bin/sh
#$ -N test
#$ -S /bin/sh
#$ -cwd
#$ -j y
#$ -m eas
#$ -pe smp 2

hostname
```

The new parameter:

- `-pe smp 2` will allocate two CPUs

Be sure you don't allocate more CPUs then a single node has. It won't be executed.

Submit this job using `qsub`:

```
$ qsub large.sub
Your job 11 ("test") has been submitted
```

Monitor your job using `qstat` as described before. Once finished, you'll get the output of our job in the logfiles:

```
$ ls
large.sub  test.o11

$ cat test.011
node01.cluster
```

Submit MPI jobs

Each MPI flavor has some differencies in the way how to start a job. The usual cases are allready configured in your Grid Engine installation:

Submit MPICH jobs

MPICH needs a *machinefile* with a line per process. The Grid Engine prepare this file and you only need to pass it to your `mpirun` command:

```
$ cat mpich.sub
#!/bin/sh
#$ -N test
#$ -S /bin/sh
#$ -cwd
#$ -j y
#$ -m eas
#$ -pe mpich 2

mpirun -np $NSLOTS -machinefile $TMPDIR/machines ./yourprogram your paramters
```

You see a new paramter:

- `-pe mpich 2` tells the Grid Engine to use the scripts for MPICH and you want to allocate two CPUs

You have the possibility to choose a range of CPUs. Using `-pe mpich 2-4` would try to allocate four CPUs, but adjust it down to two in case four are not available. The Grid Engine pass the amount of allocated CPUs in the environment variable `$NSLOTS` which is then passed to `mpirun`. Of course, your code must be able to work on a dynamic amount of CPUs before you use it this way.

Submit this job using `qsub`:

```
$ qsub mpich.sub
Your job 12 ("test") has been submitted
```

Monitor your job using `qstat` as described before. Once finished, you'll get the output of our job in the logfiles:

```
$ ls
mpich.sub  test.o12  test.po12
```

The .o file contains the output of your program, the .po file the messages produced during the setup of the parallel environment.

Submit OpenMPI jobs

OpenMPI understands the environment passed by the Grid Engine and use back the Grid Engine features to start it's children. We have prepared a dedicated parallel environment for OpenMPI, your job script needs to look a bit different:

```
$ cat openmpi.sub
#!/bin/sh
#$ -N test
#$ -S /bin/sh
#$ -cwd
#$ -j y
#$ -m eas
#$ -pe orte 2

mpirun -np $NSLOTS ./yourprogram your paramters
```

In this case, we use the parallel environment `orte` and we do not pass the machinefile as OpenMPI uses the internal data from the Grid Engine.

Submit this job using `qsub`:

```
$ qsub openmpi.sub
Your job 13 ("test") has been submitted
```

Monitor your job using `qstat` as described before. Once finished, you'll get the output of our job in the logfiles:

```
$ ls
openmpi.sub  test.o13  test.po13
```

The .o file contains the output of your program, the .po file the messages produced during the setup of the parallel environment and is always empty in this case.

Submit generic MPI jobs

The former two examples were *tight integrated* into the Grid Engine. The advantage is the full control of the Grid Engine over all processes of the job. This allows a clean shutdown of the job and full accounting.

Several commercial applications use their own wrappers. A tight integration of MPICH2, LAM-MPI or Intel MPI is technically possible, but requires dedicated setup for each flavor and version of the used MPI impelementation.

We provide a generic parallel environment called `mpi`. It simply reserves the nodes for you and provides a machinefile. No additional tasks are done by the Grid Engine until the job is finished. You have the full control over your job in your sumbission script.

An example of a script for a MPICH based job:

```
#!/bin/sh
#$ -N test
#$ -S /bin/sh
#$ -cwd
#$ -j y
#$ -m eas
#$ -pe mpi 2

mpirun -np $NSLOTS -machinefile $TMPDIR/machines ./yourprogram your paramters
```

In this case, we use the parallel environment `mpi` and allocate two slots. Any MPICH, OpenMPI or HP-MPI based application which requests a list of nodes could be started by this approach. An exmaple how to start Fluent which uses a script starting all the parts of a Fluent calculation:

```
#!/bin/sh
#$ -N test
#$ -S /bin/sh
#$ -cwd
#$ -j y
#$ -m eas
#$ -pe mpi 2

fluent 3d -g -nosge -t$NSLOTS -cnf=$TMPDIR/machines -i job.in
```

MPICH2 and Intel MPI needs an `mpd` started in advance. We must take care that each job has it's own mpd ring by providing an environment variable `MPD_CON_EXT` with a unique value for each job. Otherwise the `mpdallexit` at the end of the job may kill an mpd ring of a different job from the same user.

```
#!/bin/sh
#$ -N test
#$ -S /bin/sh
#$ -cwd
#$ -j y
#$ -m eas
#$ -pe mpi 2

export MPD_CON_EXT="sge_$JOB_ID.$TASK_ID"
sort -u < $TMPDIR/machines > $TMPDIR/mpdhosts
mpdboot -n `wc -l < $TMPDIR/mpdhosts` -f $TMPDIR/mpdhosts
mpiexec -machinefile $TMPDIR/machines -n $NSLOTS ./yourprogram your paramters
```

```
mpdallexit
rm -f $TMPDIR/mpdhosts
```

Based on this code snippets, you should be able to integrate any parallel application into the Grid Engine. Don't hesitate to ask us in case of troubles.

Submit PVM jobs

The Grid Engine takes care to start the needed `pvmd`s and your initial program. During job run, your program is responsible to spwan and end the workers. After running your program, the `pvmd`s are cleaned up. In case of a kill of your job, the Grid Engine will take care to kill all orphan processes. Each job gets an individual *Virtual Machine ID* to allow multiple jobs from the same user.

A typical submission script for a PVM job:

```
#!/bin/sh
#$ -N test
#$ -S /bin/sh
#$ -cwd
#$ -j y
#$ -m eas
#$ -pe pvm 2

/path/to/your/binary -numproc $NSLOTS
```

Most difficult part in a PVM job is to find the worker binaries. When using the same binary for controlling the job and doing the calculation, it is usually enough to specify the full path to the binary. It is able to use ARGV[0] as an argument to spawn the children. When using different binaries, ask your system administrator to add the path to the pvm parallel environment.

Additional changes to the parallel environment are needed to use the adavantage of a high speed interconnect. Please ask your system administrator or contact our support.

Kill a job

Getting rid of a job is quite simple. Use `qstat` to get the job id and use it in `qdel`:

```
$ qdel 1234
```

The system administrator is able to kill any job. You are only able to kill your own. As long as the job is not yet in the execution phase, it will be thrown out of the queue. When the job is allready running all children are killed using a `kill -9`.

Troubleshooting

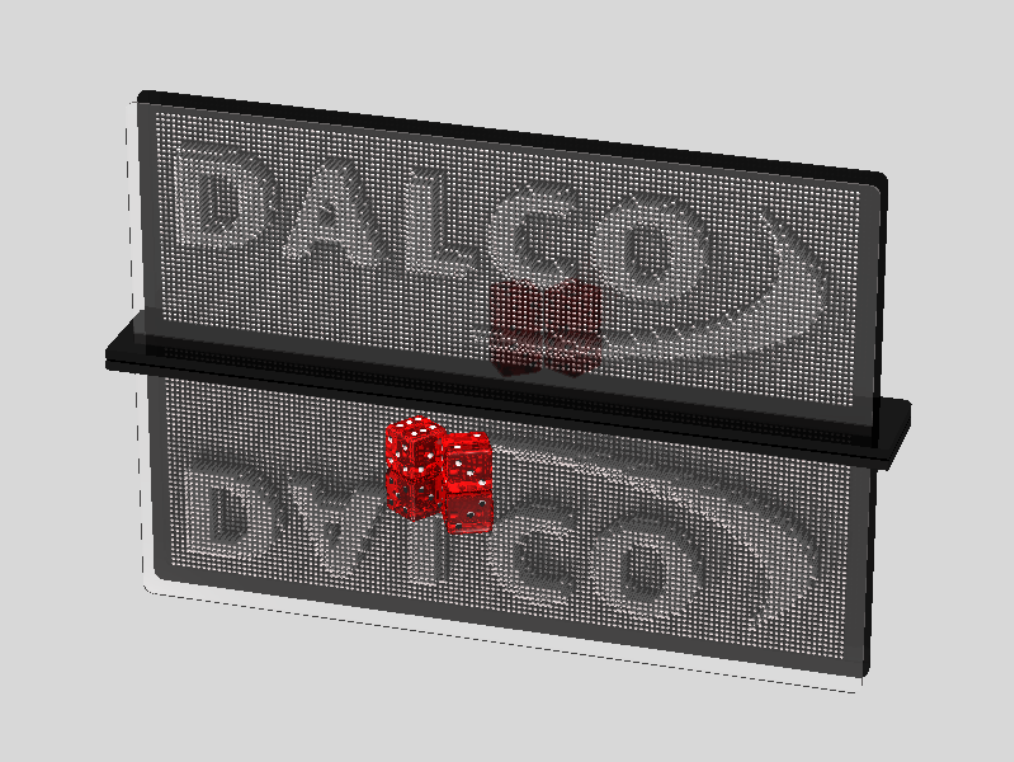
Probably you'll see jobs which were never executed. Use `qstat -j jobid` to see why the scheduler is not happy with your job:

```
$ qstat -j 13
job_number:                 13
...
scheduling info:            cannot run in PE "smp" because it only offers 2 slots
```

In this case, the parallel environment `smp` was used with more slots then a node has available.

Examples

There are may parallel applications available which uses clusters to improve speed and accurancy of the results. For simple examples, we looked for a small application which uses several ways of parallelizing and also produces some nice results. We found a great example in POV-Ray 1), a raytraycer available in source. The community provided several patches allowing to spread the calculation using threads, MPI and PVM. A small scene, made of a glass desk, a pin board and two cubes gives you a nice result:



Everything needed to reproduce the result is kept in `dalco`'s account. Login as this user, start the examples and have fun!

Serial jobs

A large amount of calculation problems knows a natural parallelizing. Have a look on a movie: Each second of a movie is made up of 24 individual frames, 86'400 frames per hour. It is usually not worth to split up the calculation of a single frame. The easier approach is to see each frame as an individual job. A similar parallelism is used in Monte Carlo simulations used in physics. Processing data from a collider, an earthquake or an oil epxploration is also such a case.

The Grid Eninge knows about array jobs which helps you to simplify the creation of your submission files. One single submission may process several thousends of data patterns in a efficient way.

Slighty different is the real time processing of small data sets in a financial application. It's usually also a single task, but it must be processed as soon as possible. For such clusters, we use a different set up of the Gird Engine to increase the responsiveness of your application.

Building POV-Ray

There is no need to adjust anything in the code of the program. It is only necessairy that the program doesn't need any interactive input during the calculation.

You find two versions of POV-Ray in the `dalco` account. 3.50c is the one we use also for MPI and PVM, 3.6.1 is the latest stable release. Rebuilding them is straithtforward:

```
cd ~/povray/src/povray-3.50c
./configure --prefix=$HOME/povray --program-suffix=-3.50c
make
make install
make clean
```

and

```
cd ~/povray/src/povray-3.6.1
./configure --prefix=$HOME/povray --program-suffix=-3.61 \
COMPILED_BY="Beat Rubischon <beat.rubischon@dalco.ch>"
make
make install
make clean
```

We add the path to the binaries to our `.bashrc` for a convenient access to them:

```
export PATH=~/povray/bin:$PATH
```

Testing

Change into the directory containing the scene and run POV-Ray:

```
cd ~/pinart
povray-3.50c pinart.pov

cd ~/pinart
povray-3.61 pinart.pov
```

Download the resulting `pinart.png` or view it by using `display pinart.png`.

Submitting to batch

Now we set up a submission file and send the job to the Grid Engine. Create the file `pinart-serial.sub`:

```
#!/bin/bash

#$ -N Pinart
#$ -S /bin/bash
#$ -cwd
#$ -j y
#$ -m eas

# Run a simple serial job on one core
# Output is a file using the GE job id in it's name

POVRAY=povray-3.50c
```

```
WIDTH=1280
HEIGHT=1024
IN=pinart.pov
OUT=Pinart-$JOB_ID.png

$POVRAY -w$WIDTH -h$HEIGHT -O$OUT $IN
```

Sunmit this file:

```
$ qsub pinart-serial.sub
```

Track your job using `qstat`. As soon it is executed, change to the node listet in `qstat` and watch the processing using `top`. It will use a single core.

When submitting multiple jobs, the Grid Engine will fill up the nodes with the same amount of jobs as cores are available.

Shared memory

Using the SMP approach, you get simple applications with an improved calculation speed. Several CPUs are doing their work on the same set of data. No worry about distibuting your data and passing messages between the jobs.

At least three approaches are common. Using shared memory between independend processes is the first. SystemV shared memory or `mmap() /dev/null` before `fork()` are common. We see this approach in several commercial applications with an old codebase.

Threading is the most widely used approach. Using POSIX threads, you mark subroutines in your code which should be run by a different CPU. You must take care of protecting your data from concurrent access. POSIX threads offers several tools like semaphores or queues. All libraries used by your program must be *thread save*. This means they are using private memory spaces for temporary data.

A relatively new approach is OpenMP. It uses POSIX threads, but you have not to write explicit code to use them. OpenMP use compiler directives to mark loops which could be run in parallel. The benefit of using OpenMP is not so big compared with a implicit threading, otherwise it is really simple to improve existing code.

All SMP approaches are limited by the size of a single node. Your application can't be spreaded over more then one system. We see often so called *fat nodes* in a cluster. This are dedicated nodes without an interconnect, but with a hughe amount of memory and typically more CPU cores.

Building POV-Ray

The current development versions of POV-Ray supports SMP using a threading helper library called BOOST 2). Your cluster provides all needed development headers and building is simple:

```
cd ~/povray/src/povray-3.7.0.beta.32
./configure --prefix=$HOME/povray --program-suffix=-3.70.beta.32 \
COMPILED_BY="Beat Rubischon <beat.rubischon@dalco.ch>" \
--with-boost-thread=boost_thread-mt
make
make install
```

```
make clean
```

Testing

Change into the directory containing your scene and call POV-Ray:

```
povray-3.70.beta.32 pinart.pov
```

Download the resulting `pinart.png` or view it by using `display pinart.png`.

Submitting to batch

Now we set up a submission file and send the job to the Grid Engine. Create the file `pinart-smp.sub`:

```
#!/bin/bash

#$ -N Pinart
#$ -S /bin/bash
#$ -cwd
#$ -j y
#$ -m eas
#$ -pe smp 8

# Run a threaded job using all cores in a node
# Output is a file using the GE job id in it's name

POVRAY=povray-3.70.beta.32
WIDTH=1280
HEIGHT=1024
IN=pinart.pov
OUT=Pinart-$JOB_ID.png

$POVRAY -w$WIDTH -h$HEIGHT -O$OUT $IN
```

Sunmit this file:

```
$ qsub pinart-smp.sub
```

Track your job using `qstat`. As soon as it is executed, change to the node listet in `qstat` and watch the processing using `top`. You will see a process consuming all available CPU power.

POV-Ray detects the amount of available CPU cores and runs the same amount of threads. You will see the process running at 400% CPU load on a four core system and on 800% on a eight core system. Take care to run only a single job on a node to avoid overbooking. This will result in a poor performance. The task switching between two processes or threads consumes a lot of CPU cycles.

MPI Jobs

MPI is the de facto standard for parallel applications. It is a basic library providing a standardized interface for sending and receiving messages. Additional, MPI will take care to start all the processes needed for the job.

You won't get a parallel application by simply linking your code to the MPI library. It is only the

message passing layer. Not more, not less. Your first step in building a parallel code is to think about the distribution of data and how it will be processed by the CPUs.

In the case of a raytracer, each process needs the full information of the scene. The master process then gives out the job of calculation a scanline to a worker. This approach is quite simple, but results in a a lot of redundant data spreaded over your cluster. For large problems, you have to split your input data and use messages for the interface between the processes.

MPI is an open standard with several implementations. More precise it is currently diveded into two standards, MPI1 and MPI2. The reference implementations, which is GPL licensed and free, are MPICH and MPICH2. Each interconnect vendor has an enhanced MPI based on those implementations: MPICH-GM, MPICH-MX and MPICH2-MX for Myrinet, Quadrics MPI, MVAPICH and MVAPICH2 for Infiniband. OpenMPI is a free MPI implentation which supports near any kind of underlaying transport. Not always at the best performance, even the community does it's best to improve it. There are also transport independend commercial MPIs which may give you good results: Intel MPI, HP-MPI and Platform MPI.

As MPI is a standard, your application may use any of the available implementations. There is no need to rewrite your code when changing the underlaying MPI. You just need to recompile your code. A multiprotocoll MPI like OpenMPI or the commercial ones will help you to build binaries for multiple different clusters. They are also often used by ISVs providing binary only commercial applications.

Building POV-Ray

We use POV-Ray 3.50c together with the parpov patch 3). It will use some features unique in the MPI2 standard, so you have to use OpenMPI or a similar MPI implementation. Check your current used MPI by calling `mpi-selector-menu`. Building is stright forward:

```
cd ~/povray/src/parapov-3.50c
./configure --prefix=$HOME/povray --program-suffix=-3.50c
make
cp src/povray $HOME/povray/bin/parpov-3.50c
make clean
```

Testing

Change into the directory containing your scene. Create a `machinefile` containing your nodes, one line per CPU you will use:

```
node01
node01
node02
node02
```

Run POV-Ray:

```
mpirun -np 4 parpov-3.50c pinart.pov
```

Download the resulting `pinart.png` or view it by using `display pinart.png`.

Submitting to batch

Now we set up a submission file and send the job to the Grid Engine. Create the file `pinart-mpi.sub`:

```
#!/bin/bash

#$ -N Pinart
#$ -S /bin/bash
#$ -cwd
#$ -j y
#$ -m eas
#$ -pe orte 16

# Run a OpenMPI job
# Output is a file using the GE job id in it's name

POVRAY=parpov-3.50c
WIDTH=1280
HEIGHT=1024
IN=pinart.pov
OUT=Pinart-$JOB_ID.png

mpirun -np $NSLOTS $POVRAY -w$WIDTH -h$HEIGHT -O$OUT $IN
```

Sunmit this file:

```
$ qsub pinart-mpi.sub
```

Track your job using `qstat -g t`. As soon as it is executed, change to the nodes listet in `qstat` and watch the processing using `top`. You will see a process for each CPU used by the calculation.

You shuld adjust the amount of used CPUs to the available ones. Using less CPUs will spread your jobs over more nodes and increase the load on the interconnect. Overbooking a node with more processes then CPUs are available will decrease the performance. The task switching between two processes consumes a lot of CPU cycles.

PVM Jobs

Today PVM is mostly superseded by MPI. We still se several PVM based applications. Even more, we see at least one commercial application performing better when using PVM then MPI.

Like MPI, PVM offers a distrbuted memory model using message passing. The handling is somewhat different. Your controlling program and the workers must not be the same binary. You have to spawn additional worker processes yourself.

Compared with MPI PVM has some advantages: It allows the usage of nodes from different architectures and sizes. PVM takes care to choose the right binary for the workers and it converts the different format of integers and floats. It allows to add and remove nodes dynamically. Typically, an outage of a node doesn't crash your job. Nodes of different speed are allowed. In MPI the slowest nodes dictates the speed of your job.

These advantages are bought by a slightly lower performance. Even worse, PVM is using plain TCP/IP and is not interconnect aware. In clusters where PVM is used, we set up a standard network

interface over the interconnect. This is IP over Myrinet, IP over Quadrics or IP over Infiniband. In a Myrinet cluster, we have also the possibility to use the Myrinet Socket interface to increase the performance of PVM.

Building POV-Ray

Before you start to use PVM, you have to set the root of your PVM installation. It is typically `/usr/lib/pvm3` or `/usr/share/pvm3`. It may be that you have your personal PVM installation in your home. Add the following line to your `.bashrc`:

```
export PVM_ROOT=/usr/lib/pvm3
```

Logout and login to activate the change. Now you are able to build and run any PVM application.

We use POV-Ray 3.50c with the pvmpov patch 4). The building must be adjusted according your `PVM_ROOT`:

```
cd ~/povray/src/pvmpov-3.50c
./configure --prefix=$HOME/povray --program-suffix=-3.50c \
--enable-pvm --with-pvm-incs=/usr/include \
--with-pvm-libs=/usr/lib/pvm3/lib/LINUX64
make
cp src/povray $HOME/povray/bin/pvmpov-3.50c
make clean
```

Testing

Create a file called `pvmhosts` in your home. Enter one line per node you will use for your calculation:

```
node01
node02
```

No start up the `pvmd`s which will build a starting environment for your workers. Check if eerything is up and running:

```
$ pvm pvmhosts
pvm> conf
conf
2 hosts, 1 data format
                    HOST     DTID     ARCH   SPEED       DSIG
          node01.cluster    40000  LINUX64   1000   0x00408841
          node02.cluster    40000  LINUX64   1000   0x00408842
pvm> quit
quit

Console: exit handler called
pvmd still running.
```

Change to the directory containing your scene and start POV-Ray:

```
pvmpov-3.50c pinart.pov +N +NT4
```

Download the resulting `pinart.png` or view it by using `display pinart.png`.

After calculation, shut down your `pvmd`s:

```
$ pvm
pvm> halt
halt
Terminated
```

Submitting to batch

Now we set up a submission file and send the job to the Grid Engine. Create the file `pinart-pvm.sub`:

```
#!/bin/bash

#$ -N Pinart
#$ -S /bin/bash
#$ -cwd
#$ -j y
#$ -m eas
#$ -pe pvm 16

# Run a PVM job
# Output is a file using the GE job id in it's name

POVRAY=`which pvmpov-3.50c`
WIDTH=1280
HEIGHT=1024
IN=pinart.pov
OUT=Pinart-$JOB_ID.png

$POVRAY -w$WIDTH -h$HEIGHT +N +NT$NSLOTS -O$OUT $IN
```

Sunmit this file:

```
$ qsub pinart-pvm.sub
```

Track your job using `qstat -g t`. As soon it is executed, change to the nodes listet in `qstat` and watch the processing using `top`. You will see a process for each CPU used by the calculation.

You should adjust the amount of used CPUs to the available ones. Using less CPUs will spread your jobs over more nodes and increase the load on the network. Overbooking a node with more processes then CPUs are available will decrease the performance. The task switching between two processes consumes a lot of CPU cycles.

Which model should I use?

We saw several approaches to distribute work in a cluster. Natural parallelizing, shared memory (SMP) using SysV SHM, POSIX threads or OpenMP, distributed memory (MPP) using MPI or PVM. Which one is the right approach?

You will dicover that the different approaches performs different. For an example, performance dosen't hurt: POV-Ray using MPI or PVM doesn't scale well. This usually looks different on other kind of mathematical problems.

Quite often you have no choice. You are using an application, provided by an ISV. Several ISVs provides their code using different memory models. When you have the choice between SMP and MPP, it's the best approach to run a case using both binaries and compare the used walltime. It is not uncommon that an MPP based code performs better, even on a single system. There are also applications providing binaries built for MPI and PVM. Also here, a benchmark with your data is the best approach to find the better performing one.

When you maintain your own code, you are free to choose the best approach. Our experience: Simpler code performs better. As long as you have the choice of using a single process, do it. In case you need more then one node and CPU to do the job, the prefered solution is MPI. Avoid hybrid applications using MPI and threading. They are hard to debug. Even more the performance is usually less then the same code using MPI only. We see also a tendence that hybrid applications overbooks the available CPU cores.

OpenMP is a nice approach to enable some parallelizing in an existing code. But it will never outperform a clean MPP design. The same rule applies for threaded libraries like GotoBLAS.

License

Even Povray is provided as source, is is not public domain or free software. Keep the copyright in mind whenever you use this great piece of software! We use an inofficial, custom versions of Povray which is copyrighted by:

Copyright © 1991-2003, Persistence of Vision Team.
Copyright © 2003-2004, Persistence of Vision Raytracer Pty. Ltd.

The program is *provided on an "AS IS" basis, without warranty of any kind, express or implied, including without limitation, any implied warranties of merchantability, fitness for a particular purpose and non-infringement of intellectual property of any third party. The Licensed Program has inherent limitations including design faults and programming bugs.*

See `povlegal.doc` and `source-license.txt` for additional information. All sources are kept in `~dalco/povray/src`.

The scene used in our examples is Copyright © by Marius Rieder marius.rieder@durchmesser.ch and licensed under the *Creative Commons - Attribution / Share Alike* license.

1) http://www.povray.org/
2) http://www.boost.org/
3) http://www.verrall.demon.co.uk/mpipov/
4) http://pvmpov.sourceforge.net/