The WTCHG Research Computing Core

Talk 5: The ResComp Managed Applications

Robert Esnouf (robert@strubi.ox.ac.uk; robert@well.ox.ac.uk),
Jon Diprose (jon@well.ox.ac.uk) & Colin Freeman (cfreeman@well.ox.ac.uk)

Generic emails: support rescomp@well.ox.ac.uk
users rescomp-users@well.ox.ac.uk

A series of introductory talks…

A set of six talks of about 45 minutes each (plus questions)

- Talk 1: What is the ResComp Core?
  (Mon 23/1 10:00 Room B; Robert Esnouf)
- Talk 2: A basic introduction to Linux
  (Wed 25/1 10:00 Room B; Robert Esnouf)
- Talk 3: Submitting jobs to the cluster
  (Thu 26/1 10:00 Room B; Robert Esnouf)
- Talk 4: Monitoring and troubleshooting
  (Mon 30/1 11:00 Room B; Robert Esnouf)
- Talk 5: ResComp centrally-managed applications
  (Wed 1/2 10:00 Room B; Jon Diprose)
- Talk 6: Doing your own thing (compiling and customizing)
  (Thu 2/2 10:00 Room B; Jon Diprose)

The story so far…

The ResComp cluster has different types of node:

- 288 "A" cores @ 2.67GB/core; 640 "B" cores @ 8GB/core; 1720 "C" cores & 768 "D" cores @ 16GB/core; 144 "H" cores @ 42.66GB/core
- Login/submission nodes: rescomp[1-2] and the project servers

Files must be on the (chargeable) GPFS filesystems:

- /gpfs0: /users/<*group*>/<*user*>, /apps/well, /mgmt
- /gpfs1 & /gpfs2: /well/<*group*>

Refresher of basic Linux concepts

- User accounts, groups, commands, shells (bash)
- Environments and environment variables: $PATH and $LD_LIBRARY_PATH
- File streams (stdin, stdout and stderr) redirection & pipes ("<", ">", "|")
- Foreground and background jobs ("&"), "nohup" and "screen"
- Writing, executing and sourcing a shell script

The story so far…

Cluster computing is non-graphical and non-interactive!

- Sun Grid Engine (SGE) using a group-based share tree policy

Your job is to submit jobs, our job is to ensure that they start

- Don't wait for free slots, don't expect jobs to start immediately!

Submit scripts to the SGE scheduler using "qsub"

- Embed "qsub" arguments using "#$" lines
- Specify where stdout and stderr should go
- "#$ -pe shmem|mpi <*n*>" for multicore or MPI jobs
- Modify jobs with "qalter", "qrls", "qmod" & "qdel"
- Monitor jobs and SGE with "qstat", "qsum", "qacct", "qload" & "qconf"

Different queues: typically short (<24 hours) & long (<7 days)

- Jobs killed if they exceed time or memory limits
- Special queues for relion jobs and on project servers
- Use "qstat –j <*job_id*>" for details on queued and running jobs
- Use "qacct –j <*job_id*> [-t *task*]" for details of completed jobs

Overview of this talk…

- Where to find the applications
- Different types of "application"
- Relevant environment variables
- Setting up your environment
- The challenges of modular applications
- Diagnosing failures

Where To Find The Applications

/apps/well/

- possibly all some of you need to know…
- 253 different applications/libraries

/apps/strubi

- don't ask me why relion is under /apps/well

/apps/htseq

- applications used by the Sequencing Core pipelines

Why can't I just type, e.g., `tophat2`?

- the application version matters
- you can, but you'll need to do a bit of work…

Where To Find The Applications

- The `locate` command on rescomp1/rescomp2 can be used to find a file

For an exact match use, eg, `locate –b '\tophat2'`

jon@rescomp1:~

```
[jon@rescomp1 ~]$ locate -b '\tophat2'
/gpfs0/apps/well/irap/git-gcc4.7.2/bin/tophat2
/gpfs0/apps/well/irap/git-gcc4.7.2/bin/tophat2/bin/tophat2
/gpfs0/apps/well/irap/git-gcc4.7.2/tmp/tophat-2.0.10.Linux_x86_64/tophat2
/gpfs0/apps/well/tophat/2.0.11/bin/tophat2
/gpfs0/apps/well/tophat/2.0.14/bin/tophat2
[jon@rescomp1 ~]$
```

Different types of "application"

Binary

- produced by compilation of source code
- hard to understand and modify without source
- static
  - "no" dependencies – but may be making assumptions
- dynamic
  - depends on binary libraries that are not part of this binary

Script

- source code that is "interpreted" at runtime
- interpreter specified by first line starting #!
- easier to understand and to modify
- python, R, perl, shell (sh/bash/csh/tcsh/zsh) & env

The command `file` can tell the difference

Different types of "application"

- Marked by the 'execute' file permission bits
  - different permissions for owner, group and other
- Just because it is marked as executable doesn't mean it is
  - dynamic libraries tend to be marked executable
  - Windows uses the executable bit to mean something else

jon@rescomp1:~

```
[jon@rescomp1 ~]$ ls -l /apps/well/tophat/2.0.14/bin/tophat2
-rwxr-xr-x 1 root root 257 Apr 17  2015 /apps/well/tophat/2.0.14/bin/topha
t2
[jon@rescomp1 ~]$ file /apps/well/tophat/2.0.14/bin/tophat2
/apps/well/tophat/2.0.14/bin/tophat2: Bourne-Again shell script text execu
table
[jon@rescomp1 ~]$
```

Relevant Environment Variables

- Your ability to run an application is affected by a number of environment variables
  - where your shell looks for applications
  - where your applications look for libraries and packages
  - how many threads your application uses
  - …
- Some apply generally, some only to certain applications, some only if an application is configured in a certain way
  - going to look at the most common
  - by no means a complete list

PATH

Controls where your shell looks for binaries

- if a fully-specified path is not used

Colon-separated list of directory paths

- searched in turn for an executable file of the same name
- first match wins

The command `which` will show the full path to the first match

Some things to note:

- '.' (the current directory) is not on your path by default
- whilst there's essentially no limit to the length of PATH, some applications start to struggle if it gets too long

LD_LIBRARY_PATH

Controls where your dynamic binaries look for libraries

- sometimes…

Colon-separated list of directory paths

- searched in turn for a library with the same name
- first match wins

No `which` equivalent

If I ask you to set this it's because:

- of a build defect
  - possibly because I've been given a pre-built dynamic binary
- you want to try using a different but compatible library to the one the binary was built against

PYTHONPATH

Controls where python looks for its packages

- takes priority over some but not all python system paths

Colon-separated list of directory paths

- searched in turn for a package with the same name
- first match wins

`pywhich` is the PYTHONPATH equivalent to `which`

Set this to use personally/group-wide installed packages

- e.g., to use a newer version of a package than the one in the central install

R_LIBS / R_LIBS_USER

Controls where R looks for its packages

- by default:
  - R_LIBS is unset
  - R_LIBS_USER is set to directory '~/R/R.version$platform-library/x.y'

Colon-separated list of directory paths

- only directories that exist when R is started are searched
- R_LIBS processed before R_LIBS_USER
- searched in turn for a package with the same name
- first match wins

Set this to use personally/group-wide installed packages

- e.g., to use a newer version of a package than the one in the central install

PERL5LIB

Controls where perl looks for its packages

- takes priority over perl system paths

Colon-separated list of directory paths

- searched in turn for a package with the same name
- first match wins

Set this to use personally/group-wide installed packages

- e.g., to use a newer version of a package than the one in the central install

Unlike python and R, the way perl is installed means that you are likely to need to set PERL5LIB for normal use:

- . /apps/well/perl/5.10.1/apps-well-perl-5.10.1.sh
- source /apps/well/perl/5.10.1/apps-well-perl-5.10.1.csh

OTHERS

- There are lots of other environment variables that might affect your job. Some are application-specific, some are library-specific. A few I am aware of:
  - TMP
    - A directory that can/should be used for temporary files
  - JAVA_TOOL_OPTIONS
    - Default arguments passed to the Java JVM which can be overridden by command line arguments – see also _JAVA_OPTIONS
  - OMP_NUM_THREADS
    - Number of concurrent threads that should be used by an application linked against the OpenMP parallel library
  - SWEETPATH
    - Points to the top directory of the pb-jelly package install

Setting Up Your Environment

- Environment variables are variables that get passed through into the environment of the application
  - different from local variables, which don't get passed through
- How to set an environment variable depends on which shell you use
  - bash
    - export FOO=bar
  - tcsh
    - setenv FOO bar

Setting Up Your Environment

- Variables can also be passed into an application's environment by putting them at the start of the line that calls the application

```
FOO=bar /apps/well/tophat/2.0.14/bin/tophat2
```

  - note – no 'export'
  - affects only that run

Setting Up Your Environment

- Use `printenv` to check the value of an environment variable
  - `echo` won't differentiate between local and environment variables

jon@rescomp1:~

```
[jon@rescomp1 ~]$ echo $FOO

[jon@rescomp1 ~]$ printenv FOO
[jon@rescomp1 ~]$ FOO=bar
[jon@rescomp1 ~]$ echo $FOO
bar
[jon@rescomp1 ~]$ printenv FOO
[jon@rescomp1 ~]$ export FOO
[jon@rescomp1 ~]$ echo $FOO
bar
[jon@rescomp1 ~]$ printenv FOO
bar
[jon@rescomp1 ~]$
```

Setting Up Your Environment

So, back to why can't I just type, e.g., `tophat2`?

- find the version you want to use
- set your PATH environment variable appropriately

jon@rescomp1:~

```
[jon@rescomp1 ~]$ which tophat2
/apps/well/tophat/2.0.14/bin/tophat2
[jon@rescomp1 ~]$ locate -b '\tophat2'
/gpfs0/apps/htseq/tophat2
/gpfs0/apps/htseq/bin/tophat2
/gpfs0/apps/htseq/head-cluster2-usr-local-genetics/bin/tophat2
/gpfs0/apps/htseq/tophat-2.0.12.Linux_x86_64/tophat2
/gpfs0/apps/well/irap/git-gcc4.7.2/bin/tophat2
/gpfs0/apps/well/irap/git-gcc4.7.2/bin/tophat2/bin/tophat2
/gpfs0/apps/well/irap/git-gcc4.7.2/tmp/tophat-2.0.10.Linux_x86_64/tophat2
/gpfs0/apps/well/tophat/2.0.11/bin/tophat2
/gpfs0/apps/well/tophat/2.0.14/bin/tophat2
[jon@rescomp1 ~]$ export PATH=/apps/well/tophat/2.0.14/bin:"$PATH"
[jon@rescomp1 ~]$ which tophat2
/apps/well/tophat/2.0.14/bin/tophat2
[jon@rescomp1 ~]$
```

Setting Up Your Environment

- The `module` command simplifies setting the environment
  - can set several variables at once
  - can be aware of conflicts with another application, or a different version of the same application
  - useful subcommands are `avail`, `show`, `load` and `unload`
- You should probably use this mechanism for
  - python
  - R
- ShellShock bug fix has made it difficult to use with SGE
  - submit with the '-V' option to pass environment to job
  - or reactivate the module package within the script:

```
. /etc/profile.d/modules.sh
```

Setting Up Your Environment

jon@rescomp1:~

```
[jon@rescomp1 ~]$ module avail

---------------------------- /gpfs0/mgmt/modules/Modules/3.2.10/modulefiles ----------------------------
R/2.15                                          intel/compiler/1_ics_2013.0.028
R/2.15.3                                        intel/mpi/4.1.0.024
R/3.0                                           java/1.8.0_latest
R/3.0.3                                         matlab/mcr-2013b
R/3.1                                           matlab/mcr-2014a
R/3.1.0                                         matlab/mcr-2015a
R/3.1.2                                         mpich2/3.1/gcc/4.4.7
R/3.1.3                                         mvapich2/1.9/gcc/4.4.7
R/3.1.3-atlas-3.10.2-gcc4.7.2                   openmpi/1.6.5/gcc/4.4.7
R/3.1.3-gcc4.9.3                                python/2.7
R/3.1.3-openblas-0.2.14-omp-gcc4.7.2            python/2.7.10
R/3.2.0-openblas-0.2.14-omp-gcc4.7.2            python/2.7.10-gcc4.9.3
R/3.2.0-openblas-0.2.14-omp-gcc4.8.2            python/2.7.10-gcc5.4.0
R/3.2.2                                         python/2.7.11(default)
R/3.2.2-openblas-0.2.14-omp-gcc4.7.2            python/2.7.6
R/3.2.2-openblas-0.2.14-omp-gcc4.8.2            python/2.7.7-venv
R/3.2.5                                         python/2.7.7-venv-unicode
R/3.2.5-openblas-0.2.18-omp-gcc4.7.2            python/2.7.8
R/3.2.5-openblas-0.2.18-omp-gcc4.8.2(default)   python/3.4.3
R/3.3.0                                         python/3.5.2-gcc5.4.0
R/3.3.0-openblas-0.2.18-omp-gcc4.7.2            relion/1.2-gcc4.7.2
R/3.3.0-openblas-0.2.18-omp-gcc4.8.2            relion/1.3-3-gcc4.7.2
R/3.3.1                                         relion/1.3-beta3-gcc4.7.2
R/3.3.1-openblas-0.2.18-omp-gcc4.7.2            relion/1.3-compA-gcc4.7.2
R/3.3.1-openblas-0.2.18-omp-gcc4.8.2            relion/1.3-gcc4.7.2
R/3.3.1-openblas-0.2.18-omp-gcc4.9.3            relion/1.3-relax-1-gcc4.7.2
R/3.3.1-openblas-0.2.18-omp-gcc5.4.0            relion/1.3-relax-1.1-gcc4.7.2
R/default(default)                              relion/1.4-fftw-3.3.4-gcc4.7.2
amber/14-gcc4.9.1-mkl                           relion/1.4-gcc4.7.2
autoconf/2.69                                   relion/1.4-single-fftw-3.3.4-gcc4.7.2
cns/1.3                                         relion/1.4-single-gcc4.7.2
```

Setting Up Your Environment

jon@rescomp1:~

```
[jon@rescomp1 ~]$ which python
/usr/bin/python
[jon@rescomp1 ~]$ python --version
Python 2.6.6
[jon@rescomp1 ~]$ module show python/2.7
-------------------------------------------------------------------
/gpfs0/mgmt/modules/Modules/3.2.10/modulefiles/python/2.7:

module-whatis    Sets up your environment so calling python runs version 2
.7.6
conflict         python
prepend-path     PATH /apps/well/python/2.7.6/bin
prepend-path     PATH /apps/well/python/2.7/bin
prepend-path     LD_LIBRARY_PATH /apps/well/python/2.7.6/lib
prepend-path     LD_LIBRARY_PATH /apps/well/openblas/0.2.8-gcc4.7.2/lib
prepend-path     LD_LIBRARY_PATH /apps/well/fftw/3.3.3-gcc4.7.2/lib
prepend-path     PKG_CONFIG_PATH /apps/well/python/2.7.6/lib/pkgconfig
-------------------------------------------------------------------

[jon@rescomp1 ~]$ module load python/2.7
[jon@rescomp1 ~]$ python --version
Python 2.7.6
[jon@rescomp1 ~]$
```

The Challenges Of Modular Applications

- Most of the useful functionality of python, R and perl is delivered through add-on packages
  - there can be multiple versions of a package but only one is going to be used
  - packages tend to have dependencies on other packages
    - possibly a large number…
  - packages are not necessarily compatible with one another
    - tends to really mean they have conflicting requirements for the packages/versions on which they depend
- Once installed, it is hard for rescomp to update a package
  - may break running code
  - may break installed packages

The Challenges Of Modular Applications

Python tends to cause the most problems

- "accepted" solution is to use virtualenv, but
  - pushes the package installation problems down to users

Python package updates likely to be pushed out alongside python updates

- `module load python/2.7.8` has newer packages than 2.7.6, etc.
- 2.7.11 is latest available on rescomp
- will roll out 2.7.13 at some point with more updates

Diagnosing failures

Python / R / perl package missing

- try a newer version of python / R / perl
  - for python, use 2.7.10 or 2.7.11
  - module load python/2.7.11
- for R, try 3.2.5 or 3.3.1
  - module load R/3.3.1
- ask for the package to be installed
  - a link to the package's website is always useful

Diagnosing failures

- Application not found
  - use which and locate
  - set your PATH environment variable appropriately
  - use a fully-specified path
  - ask for the application to be installed
    - a link to the application's website is always useful

Diagnosing failures

GLIBC / GLIBCXX not found

- binary was built on a box with a newer glibc/kernel than the rescomp cluster has
- rebuild required on a rescomp box
  - or a box running RHEL6 / CentOS6, etc.

jon@rescomp1:~

```
$ cat runpleiov106.DIL.22.sh.e1904724
../../../NFBC66/ENGAGE1000G/pleiotropy/bin3/PLEIOTROPY: /lib64/libc.so.6:
version `GLIBC_2.14\' not found (required by
../../../NFBC66/ENGAGE1000G/pleiotropy/bin3/PLEIOTROPY)
../../../NFBC66/ENGAGE1000G/pleiotropy/bin3/PLEIOTROPY:
/usr/lib64/libstdc++.so.6: version `GLIBCXX_3.4.15\' not found (required b
y
../../../NFBC66/ENGAGE1000G/pleiotropy/bin3/PLEIOTROPY)
[jon@rescomp1 ~]$
```

Diagnosing failures

- Library not found ('libSomething.so: cannot open shared object file')
  - dynamic binary has failed to find a library on which it depends
  - install the library if it hasn't been installed
  - set LD_LIBRARY_PATH appropriately so the binary can find it
  - may be necessary to rebuild

jon@rescomp1:~

```
> EOT
/well/donnelly/PlatypusSeq/bin/rmblast-2.2.28/bin/rmblastn
/well/donnelly/PlatypusSeq/bin/rmblast-2.2.28/bin/rmblastn: error while lo
ading shared libraries: libbz2.so.1: cannot open shared object file: No su
ch file or directory



[jon@rescomp1 ~]$
```