Yeti - Job Examples

- General Information for Examples
- Hello World
- Perl or Python
- C/C++/Fortran
- C/C++/Fortran MPI
- C OpenMP
- GPU (CUDA C/C++)
- IDL
- R
- Matlab
- Spark
- Stata
- Schrodinger
- Knitro

General Information for Examples

In order for the scripts in these examples to work, you will need to make three changes.

1. Replace `<UNI>` with your Columbia UNI.
2. Replace `<GROUP>` with your cluster submit group.
3. Replace `<GROUP_DIR>` with your cluster submit group, minus the initial "hpc". For example, if your submit group is "hpcastro", use "astro". Special case for SSCC users: if you are in the hpcsscc group, use "sscc/work".

Hello World

This script will print "Hello World", sleep for 10 seconds, and then print the time and date. The output will be placed in a standard batch output file.

```bash
#!/bin/sh

# Directives
#PBS --N HelloWorld
#PBS --W group_list=<GROUP>
#PBS -l nodes=1,walltime=00:00:30,mem=20mb
#PBS -M <UNI>@columbia.edu
#PBS -m abe
#PBS -V

# Set output and error directories
#PBS -o localhost:/vega/<GROUP_DIR>/users/<UNI>/
#PBS -e localhost:/vega/<GROUP_DIR>/users/<UNI>/

# Print "Hello World"
echo "Hello World"

# Sleep for 10 seconds
sleep 10

# Print date and time
date

# End of script
```

Perl or Python

Perl and Python programs may be submitted to Yeti. It is important to remember that the first line of your program should specify the location of Perl or Python and to make sure that the file is executable. For example:

```bash
$ which python 
/usr/bin/python
```
$ which perl
/usr/bin/perl

For Perl programs, make sure the first line of your program reads:

#!/usr/bin/perl

For Python programs, make sure the first line of your program reads:

#!/usr/bin/python

Make sure to remember to change permissions so that the file is executable:

$ chmod +x example.pl
$ chmod +x example.py

The submit file will be similar to the Hello World example above. For example:

#!/bin/sh

# Directives
#PBS -N PythExample
#PBS -W group_list=<GROUP>
#PBS -l nodes=1,walltime=01:15:00
#PBS -M <UNI>@columbia.edu
#PBS -m abe
#PBS -V

# Set output and error directories
#PBS -o localhost:/vega/<GROUP_DIR>/users/<UNI>/
#PBS -e localhost:/vega/<GROUP_DIR>/users/<UNI>/

# Command to execute Python program
python example.py

# End of script

### Batch queue submission

$ qsub python.sh

### Interactive queue submission

$ qsub -q interactive -I -W group_list=<GROUP>

Then when your jobs starts:

$ example.py

C/C++/Fortran
To submit a precompiled binary to run on Yeti, the script will look just as it does in the Hello World example. The difference is that you will call your executable file instead of the shell commands "echo", "sleep", and "date".

### C/C++/Fortran MPI

Yeti supports OpenMPI. The MPI provided by the Intel compiler that we also support (see below) is derived from MPICH2.

**Note Regarding MPI on Yeti:** There are two ways to use MPI on the cluster: 1) with Ethernet transport between any nodes and 2) on Infiniband nodes only. We avail ourselves of environment modules to specify which option is used, and to set the appropriate environment in each case. The module command is to be invoked from within the submit script, as illustrated in the example script below.

For Ethernet transport between any node:

```bash
module load openmpi/1.6.5-no-ib
mpirun myprogram
```

For use of mpi on Infiniband nodes only:

```bash
module load openmpi/1.6.5
mpirun myprogram
```

To use MPI, your program must be compiled on the cluster. You can use the the module command as explained above to set your path so the mpicc will be found. Note that you may have to set additional environment variables in order to successfully compile your program.

```bash
$ module load openmpi/1.6.5-no-ib
$ which mpicc
/usr/local/openmpi-1.6.5/bin/mpicc
```

Compile your program using mpicc. For programs written in C:

```bash
$ mpicc -o <MPI_OUTFILE> <MPI_INFILE.c>
```

**Note Regarding compilers on Yeti:** If no special steps are taken, Yeti's compilers are from the GNU family. However, the more efficient Intel compilers are also available - in order to avail yourself of them, you just need to invoke the appropriate module first:

```bash
$ module load intel-parallel-studio/2015
(...)
$ which mpicc
/vega/opt/parallel_studio_xe_2015_update1/impi/5.0.2.044/intel64/bin/mpicc
$ which ifort
/vega/opt/parallel_studio_xe_2015_update1/composer_xe_2015.1.133/bin/intel64/ifort
```

The submit script below assumes that you have compiled the following simple MPI program used to compute pi, `mpi_test.c`, and created a binary called `pi_mpi`:
#!/bin/sh
# pi_mpi.sh
# Torque script to submit MPI program to compute pi.

# Torque directives
#PBS -N Pi_MPI
#PBS -W group_list=<GROUP>
#PBS -l nodes=2:ppn=1,walltime=00:05:00,mem=4000mb
#PBS -M <UNI>@columbia.edu
#PBS -m abe
#PBS -V

#set output and error directories
#PBS -o localhost:/vega/<GROUP_DIR>/users/<UNI>/
#PBS -e localhost:/vega/<GROUP_DIR>/users/<UNI>/

# Set MPI environment appropriate type of MPI usage
module load openmpi/1.6.5

# call MPI
mpirun pi_mpi

#End of script

As can be seen from the parameter to the PBS -l flag, this example requests one core (ppn=1) on each of two nodes (nodes=2). In practice users should refrain from requesting multiple nodes unless they require more than 16 cores. This ensures faster run times, since your application will be contained on a single node, as well as more efficient scheduling.

**Job Submission**

```
$ qsub pi_mpi.sh
```

**Infiniband**

Yeti has now 48 nodes equipped with Infiniband high-performance network connections. Many MPI applications run significantly faster on Infiniband systems. 16 of the nodes are equipped with an older generation Infiniband, and we do not allow jobs to straddle nodes with different generations of the fabric.

You need to make three changes to your submit script to request Infiniband nodes.

1. Add "#PBS -q infiniband" to request the infiniband queue.
2. Remove the core count, "ppn", from your "#PBS -l" directive. The infiniband queue requires that a job use an entire 16-core node. The inclusion of a ppn directive will probably prevent your job from starting.
3. Remove the memory requirement, "mem", from your "#PBS -l" directive. This is not a mandatory change but since entire nodes are being used there is usually little need to specify it. All Infiniband nodes have at least 64 GB of memory.

The following example requests 4 nodes, implying a total of 4 x 16 = 64 cores.

```
#PBS -q infiniband
#PBS -l nodes=4,walltime=6:00:00
```

Naturally, all jobs in the infiniband queue will only run on Infiniband nodes. To request that your job only run on the 2nd generation nodes with faster CPU, add the "ib2" flag to your request:

```
#PBS -q infiniband
#PBS -l nodes=4:ib2,walltime=6:00:00
```

**C OpenMP**

The cluster's gcc compilers support OpenMP API for shared memory parallel processing. It is not a separate package.

In order to activate the openmp library, apply the -fopenmp flag while compiling:
A trivial example:

```c
#include <stdio.h>

int main(void)
{
    #pragma omp parallel
    printf("Hello, world.\n");
    return 0;
}
```

GPU (CUDA C/C++)

The cluster includes nine GPU servers, four of which has two Nvidia (Tesla) K20 GPU modules, and the remaining five are of a newer generation of GPU modules, namely Nvidia K40.

Some applications, such as MATLAB, have built-in GPU support. To use a GPU server, you must submit the job to the gpu queue by specifying `-q gpu` and also use the `-l nodes=1:gpus=1` clause in your job submit script or qsub command, optionally along with the generation of the GPU modules you require. See the end of this section for more details.

In order to compile your CUDA C/C++ code and run it on the GPU modules in the cluster, you first have to set your paths so that the Nvidia compiler can be found. Please note you must be logged into a GPU node to access these commands. Load the cuda environment module which will set your PATH and LD_LIBRARY_PATH.

```bash
$ module load cuda
```

You then have to compile your program using `nvcc`:

```bash
$ nvcc -o <EXECUTABLE_NAME> <FILE_NAME.cu>
```

You can try and compile this sample code. You can compile it by running the following command:

```bash
$ nvcc -o hello_world hello_world.cu
```

For non-trivial code samples, refer to Nvidia’s CUDA Toolkit Documentation.

In order to submit the job, you can use the following template submit script:
It is important to request the correct number of GPU modules for your job in order to prevent multiple jobs from attempting to use a module at the same time. Unfortunately, at this time this mechanism works only if everybody adheres to this rule (this will be corrected in the near future).

It is also a good idea to specify the generation of the modules to be used, which will also determine which node(s) the job will run on. Hence, if you will be using one GPU K20 module:

```
#PBS -l nodes=1:gpus=1:k20
```

If you will be using two K40 GPU modules:

```
#PBS -l nodes=1:gpus=2:k40
```

The gpus=X syntax is replacing the older, now obsolete "other=gpu" specification, and we hope that along with the new version of the resource manager used on Yeti it will avoid resource locking and other problems that were more likely on the older versions. This is still under investigation so any feedback from users is welcome.

Please keep in mind that the older ("other=gpu") mechanism limited the number of threads specified in the "ppn=" clause in the submit script to maximum of 2, probably because there was an implicit correspondence between the threads and the GPU modules. We have not yet tested whether the same applies to the "gpus=X" mechanism. Incidentally, the above examples imply one core having been requested, i.e. when no "ppn=" is present, the PBS directives act as if the "ppn=1" specification were included.

There's a mode associated with a GPU module. The current setting for it is EXCLUSIVE_THREAD. Some application require it to be set to EXCLUSIVE_PROCESS. This can be accomplished on a job by job basis while submitting the job, like this:

```
qsub -l nodes=1:gpus=2:exclusive_process
```

**IDL**

To run IDL programs, we need to call IDL directly at the end of the Torque script and include the IDL pro file as an argument. The script below runs code in a file called wakecontourlg.pro (the file name is an example only and the file is not available for download). This program also uses several input files that are not included in this documentation. The files should all be in the same directory upon running. We first must create a wrapper IDL file called "do.pro" that serves to call the real program file "wakecontourlg.pro". The "do.pro" file will only contain one line with the name of the program we are actually interested in running. We do this to avoid adding $& to the end of each line in our IDL program file. For more information on IDL, please consult your local IDL expert. First, our do.pro file looks like:

```
;do.pro
;This file should contain the programs you would like to
;run in an IDL session - one on each line

wakecontourlg
```

An example script to run is:
#!/bin/sh
#idlex.sh
#Torque script to run IDL example

# Directives
#PBS -N IDLExample
#PBS -W group_list=<GROUP>
#PBS -l nodes=1,walltime=00:15:00,mem=1000mb
#PBS -M <UNI>@columbia.edu
#PBS -m abe
#PBS -V

#set output and error directories
#PBS -o localhost:/vega/<GROUP_DIR>/users/<UNI>/
#PBS -e localhost:/vega/<GROUP_DIR>/users/<UNI>/

#Command to execute IDL program
idl do.pro > idloutput

#End of script

Batch queue submission

$ qsub idlex.sh

Interactive queue submission

To use IDL interactively:

$ qsub -q interactive -I -W group_list=<GROUP>

When your interactive job starts:

$ idl

R

See the R examples page.

Matlab

See the Matlab examples page.

Spark

See the Spark usage page.

Stata

See the Stata examples page.

Schrodinger

See the Schrodinger page.

Knitro
Load the knitro module.

```
module load knitro/9.0.1
```

To use Knitro under Matlab:

```
[x fval] = ktrlink(@(x)cos(x),1)
```