Exercises Peregrine tutorial

Use the course slides and the Peregrine documentation on
https://redmine.hpc.rug.nl/redmine/projects/peregrine/wiki
to find out how to perform the exercises.

The hostname of the Peregrine cluster is:
peregrine.hpc.rug.nl

Before you begin

All exercises require some input files, e.g. scripts, parameter files, or input data. These can all be found in:
/data/pg-course/bio

Start by creating a directory named, for instance, *jobs* in your home directory. For each exercise that you are doing, make a subdirectory in *jobs* (with some suitable name) that will hold all the files for that particular exercise.

Exercise 1 – Using R within a job

a. Go into the *jobs* directory, make and enter a subdirectory for this job, and copy all input files from /data/pg-course/bio/mandelbrot into this directory.
b. Create a new and empty file that will be the job script for this exercise. Choose any filename you want.
c. Open this job script with an editor and make sure it will do the following:
    i. Define the following requirements / properties for the job:
        - Maximum wall clock time of 20 minutes.
        - 1 core on a single node.
        - Memory limit of 1 gigabyte.
        - Nice name.
    ii. Perform the following actions:
        - Load the R module for version *R/3.3.1-foss-2016a*
        - Start R using the command: *Rscript mandelbrot.R*
    iii. Did you forget to include the Shebang! line at the top of the file? Better do it now, then.
d. Submit the job. The job ID will be displayed here.
e. Find the job with *squeue* and check its status. If it is not there, your job might have finished already.
f. Study the resulting output file to see if everything went smoothly.
g. Use the *jobinfo* command to find the maximum memory usage of your job.
h. Copy the resulting gif file to your Windows desktop for inspection.
i. Produce the same gif file again, but this time with a resolution of 2000x2000. You can do this by editing the *mandelbrot.R* file and changing the *dx* and *dy* value. If you do not want to overwrite the existing gif file, you could also change the filename of the gif file in the last line of the script. Finally, resubmit the job.
j. Study the output file again and fix any issues that you encounter.

Exercise 2 - Using Python within a job

a. Go into the *jobs* directory, make and enter a subdirectory for this job, and copy all input files from /data/pg-course/bio/climate into this directory.
b. Did you note that this took a while? The *climate.csv* is more than 500MB, and it's better to store such a data file on the /data file system. So, go ahead and move it (only this file!) from your current working directory to /data/$USER.
c. Create a new and empty file that will be the job script for this exercise. Choose any filename you want.
d. Open this job script with an editor and make sure it will do the following:
    i. Define the following requirements / properties for the job:
        - Maximum wall clock time of 10 minutes.
        - 1 core on 1 node.
        - Memory limit of 2 gigabyte.
        - Nice name.
    ii. Performs the following actions:
        - Load the modules:
            - *Python/3.5.2-foss-2016a*
            - *matplotlib/1.5.3-foss-2016a-Python-3.5.2*
        - Run: *python script.py PATH_TO_CSV_FILE*
          Replace *PATH_TO_CSV_FILE* by the full path to where you stored the CSV file in the first part of this exercise.
    iii. You forgot the Shebang! again, didn't you? Back to top…
e. The script.py will make a plot of the average temperature in Groningen. If you want, edit the file and change this to a different (large) city.
f. Submit the job.
g. Use *squeue* or *jobinfo* to find the job status and wait until it completes.
h. Study the SLURM output file and solve any errors, if necessary.
i. Copy the resulting png file to your desktop for inspection.

Exercise 3 - Parallel job using MrBayes

MrBayes is a program for Bayesian inference and model choice across a wide range of phylogenetic and evolutionary models. MrBayes uses Markov chain Monte Carlo (MCMC) methods to estimate the posterior distribution of model parameters.
MrBayes can make use of the MPI library to be able to run on multiple cores and nodes, which we will use in this exercise to test a topological hypothesis that humans are more closely related to chimps than to other primates. Details can be found in section 4.4 of the MrBayes manual: http://mrbayes.sourceforge.net/mb3.2_manual.pdf

a. Go into the *jobs* directory, make and enter a subdirectory for this job, and copy all input files from /data/pg-course/bio/mrbayes into this directory.
b. Construct a job script with the following job specifications:
    i. Maximum wall clock time of 5 minutes
    ii. One node, with 4 tasks per node in order to start four MPI tasks
    iii. Use the "short" partition
c. Furthermore, the job script needs to perform the following actions:
    i. Load the MrBayes module. Use the module command to find the name of the module. Note that loading this module will automatically load some other modules for dependencies of MrBayes, including OpenMPI.

   ii. If you did not store your job script in the same directory as the input files, make sure to change to that directory in your job script. Otherwise MrBayes does not know where to find them.
   iii. Start MrBayes on our input file using the following command:
   srun mb primates.mb
   The srun command will use information from the scheduler to decide where the parallel MPI tasks for MrBayes should be started.

d. Submit the job. While waiting for the job to finish, you could have a look at the input file primates.mb to find out which commands are being executed by MrBayes.
e. Study the output of the job. If there are errors, try to fix them. Can you verify using the wall clock time and CPU time that the job really used four cores?
f. Submit the job again, but this time request two nodes with four cores per node. *Remark: in general it is better to run MPI tasks on as few nodes as possible (in this case: 1 node, 8 tasks per node), but we now do use two for demonstration purposes.*
g. Again, check the output file and verify that eight cores were used. Also compare the wall clock times of both jobs. Did you expect to find this difference?
h. Read the first paragraph of page 54 of the manual to find out how to draw a conclusion about our hypothesis. Is there strong evidence for humans and chimps to be each other's closest relatives?
i. Bonus: what happens if you try to run the job on even more cores, e.g. 1 node with 16 tasks or 2 nodes with 8 tasks per node? Can you explain what happens?

Exercise 4 – Compiling and using your own C++ application

a. Go into the *jobs* directory, make and enter a subdirectory for this job, and copy all input files from /data/pg-course/bio/sudoku into this directory.
b. Compile the C++ source code using the g++ compiler.
   i. Load the foss/2016a module to get a more recent compiler.
   ii. The name of the resulting executable should be "sudoku_solver".
   iii. Use the optimization flag "-O3" to improve the performance of the application.
   iv. E.g: g++ -O3 sudoku_solver.cpp -o sudoku_solver
c. Construct a job script with the following requirements and properties for the job:
   i. Maximum wall clock time of 5 minutes;
   ii. Run on 1 core on a single node;
   iii. Assign a nice name to the job;
   iv. Use the "short" partition.
d. The job should perform the following actions:
   i. Load the foss/2016a module
   ii. Solve the Sudoku in the input file by passing its filename as an argument to the executable.
e. Submit the job and wait for the job to complete.
f. Study the error and output file of the job to see if everything ran fine.
g. The Sudoku solver also generates an output file containing the solution. Check if this file indeed exists and if it contains the solution for the Sudoku.
h. You can check the Makefile and use that to compile the program:
   i. To remove previous files, use:
   make clean
   ii. Issue the command make to compile the program.