Clemson's Palmetto guide

Murwan Siddig

April 1, 2019

1 Getting started

Most material discussed below can be found in more thorough details inPalmetto documentation, which is very well written. It also provides a basic linux review, which if you are not fimiliar with using Linux system, I suggest you check it out

Step (0)- Making an account

You need an account to access the cluster. To request one, click: here.

Step (1) - Logging in

Before you do this, you need to make sure your device is Two-Factor Authentication (2FA) registered device. If not, go to: https://2fa.clemson.edu/

- **Max OS ir Linux** Type the following command on your terminal:

```
\$ ssh username@login.palmetto.clemson.edu
```

- **Windows** You need to have a SSH (Secure Shell) – Palmetto recommends to download MobaXterm. You can also use a remote network, to do this, you need to request a VM here which will give you a graphical desktop access to the Palmetto cluster using virtualization technology.

Step (2) - Setting up "your" Julia environment in Palmetto

- A rule of thumb: you are NOT allowed to perform your software processor (Julia, Python, C++, etc) on the login node. You need to request an interactive node: to do this type the following command:

```
qsub -I
```

- Next, go to this page here and copy the link for most recent Julia binaries under "Generic Linus Binaries for x86", and type:

```
wget https://julialang-s3.julialang.org/bin/linux/x64/0.6/julia-0.6.2-
    ↪ linux-x86_64.tar.gz
```

Next, untar your Julia downloads.

```
tar -xvzf julia-0.6.2-linux-x86_64.tar.gz
```

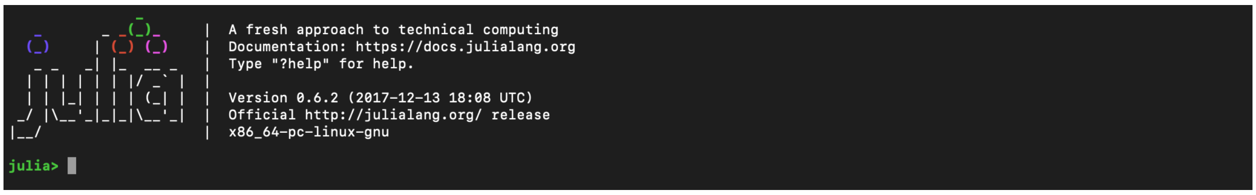
now change the name with Julia binary added to your system path

```
\mv julia-d386e40c17 julia
echo 'export PATH=/home/sitew/julia/bin:$PATH' >> ~/.bashrc
source ~/.bashrc
```

Now you should be able to call Julia directly (say your username is palmetto-user)

```
[palmetto-user@node0021 mediumttest]$ julia
```

you just need to type <julia> btw.
You should expect to see something like



```
               _
   _       _ _(_)_     |  A fresh approach to technical computing
  (_)     | (_) (_)    |  Documentation: https://docs.julialang.org
   _ _   _| |_  __ _   |  Type "?help" for help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 0.6.2 (2017-12-13 18:08 UTC)
 _/ |\__'_|_|_|\__'_|  |  Official http://julialang.org/ release
|__/                   |  x86_64-pc-linux-gnu

julia>
```

Step(3) - Installing some basic Julia basic packages

After you do this, you might wanna start installing your "favorite" packages: you can do this the same way you would normally do when using Julia – probably you want to start by installing JuMP – see below as an example

```
Pkg.add("JuMP")
Pkg.add("Distributions")
Pkg.add("LightGraphs")
```

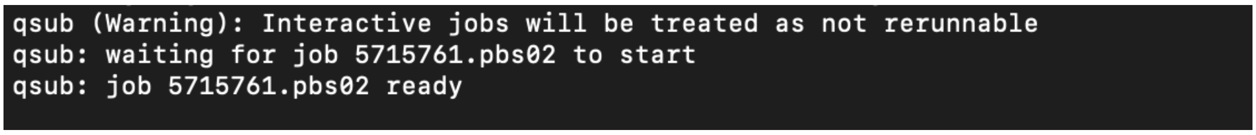
Note that this will be typed in Julia command line.

Optimization solver

- **Gurobi:** Clemson have license for version 6.5.0, 7.0.1 and 7.0.2 (as of April 1, 2019) which means you don't have to install Gurobi yourself (of course, you can always get the most recent academic license and include the binaries of your own download into your path).
  Adding Gurobi is very straight forward (assuming you want to use the one Palmetto already has). You can do this by typing this in the interactive node (NOT the login node):
  First request a computation node – I'll explain this step in more details later, but for now, type this:

```
[palmetto-user@login001 ~]$ qsub -I -l select=1:ncpus=2:mem=4gb,walltime
    ↪ =4:00:00
```

You should see something like



```
qsub (Warning): Interactive jobs will be treated as not rerunnable
qsub: waiting for job 5715761.pbs02 to start
qsub: job 5715761.pbs02 ready
```

Now, type this and call julia

```
[palmetto-user@node0180 ~]$ module add julia
[palmetto-user@node0180 ~]$ module add gurobi
[palmetto-user@node0180 ~]$ julia
```

Once Julia prompt shows up type

```
julia> Pkg.add("Gurobi")
```

- **Limitation of Palmetto license** Later I will explain how to send a huge number of jobs at the same time – to which using Palmetto Gurobi's license my not be the best course of action. Plametto Gurobi's license have limitation – so when the number of jobs in the queue reaches that limit (yours + others) some of your jobs might be killed. if you think you'll be sending large number of jobs at the same time I suggest installing Gurobi yourself. You can do this as follows: To install Gurobi.jl, you need to make sure that environment variable GUROBI_HOME is defined and can lead you to the Gurobi installation. In you command line write:

```
$ echo 'export GUROBI_HOME=/software/gurobi/7.0.2/gurobi702/linux64'
    ↪ >> ~/.bashrc
$ source ~/.bashrc
$ module add gurobi/7.0.2
$ julia -e 'Pkg.add("Gurobi")'
$ julia -e 'PKg.test("Gurobi")'
```

- **CPLEX:** To the best of my knowledge, Palmetto doesn't have license for CPLEX. However, you can always install the free academic license CPLEX provides. To do this go here. Once installed (say you saved it in /home/palmetto-user/cplex128) type the following:

```
$ chmod +x cplex_studio128.linux-x86-64.bin
$ ./cplex_studio128.linux-x86-64.bin
$ export 'CPLEXPATH=/home/palmetto-user/cplex128' >> ~/.bashrc
$ export 'PATH=$CPLEXPATH/cplex/bin/x86-64_linux:$PATH' >> ~/.bashrc
$ export 'PATH=$CPLEXPATH/cplex/include:$PATH' >> ~/.bashrc
$ export 'PATH=$CPLEXPATH/cplex/lib:$PATH' >> ~/.bashrc
$ export 'PATH=$CPLEXPATH/include/ilcplex:$PATH' >> ~/.bashrc
$ export 'PATH=$CPLEXPATH/cplex:$PATH' >> ~/.bashrc
$ export 'LD_LIBRARY_PATH=$CPLEXPATH/cplex/bin/x86-64_linux:
    ↪ $LD_LIBRARY_PATH' >> ~/.bashrc
$ julia -e 'Pkg.add("CPLEX")'
$ julia -e 'Pkg.test("CPLEX")'
```

2 Starting an interactive job

This is the simplest version of submitting a job to the cluster. Interactive jobs require you to be logged-in while your tasks are running (which could be inconvenient). Before you run the job you need all your code and its related files to be in the cluster. You can find more on how to move date in and out of the cluster here. I use JupyterHub to do this. JupyterHub allows you to interact with the Palmetto cluster from their web browsers using the Jupyter Notebook interface. To do this:

1. Start by visiting https://www.palmetto.clemson.edu/jupyterhub.
2. Log in with your Palmetto user ID and password
3. Once you are logged in, click on "Start my server" to start a notebook server.
4. Select the resources (CPU cores, memory, walltime, etc.,) required for your session.
5. If the resources you request are available, a notebook server will be started for you. This will open up the Notebook dashboard, where you will see the files and directories in your "home" directory on the Palmetto cluster.

Once your files in the cluster do the following

1. Login to the terminal

```
$ ssh palmetto-user@login.palmetto.clemson.edu
```

2. request an interactive job and specify your usage. You need to specify the following (in this example):

Command	Spesification
select=1	number of hardware "chunk" you want to use
ncpus=2	number of CPU cores per "chunk"
mem=4gb	how much memory you want to use
walltime=4:00:00	wall-time – which is how long you want the interactive node for

   all together the command is:

```
[palmetto-user@login001 ~]$ qsub -I -l select=1:ncpus=2:mem=4gb,walltime
   ↪ =4:00:00
```

   Once these resources are available, you willl receive a Job ID e.g <8730.pbs02>, and a command-line session running on e.g <node0180 >.

3. add Julia and gurobi by doing the following:

```
[palmetto-user@node0180 ~]$ module add julia
[palmetto-user@node0180 ~]$ module add gurobi
[almetto-user@node0180 ~]$ julia
```

4. login to jupyterhub and request your server (you DON'T need to do this if you already know the path of your code in cluster.). You can also do this using linux command (which is what most people do) and it's even easier.

5. copy the path of your code and do the following command (in this example I have the file saved in my home directory – by default you are in your home directory, to change the directory use the linux command)

```
[palmetto-user@node0180 ~]$ cd name_of_the_directory
```

   you need to do this before calling julia). Once you copied the path do:

```
julia> include("./Palmetto_test_code.jl")
```

   BTW, this is just one way to do this – you can do this in so many different ways too.

3 Submit a batch job

This is the one which you are most likely to use along with the next one.
Here, unlike the interactive job, you may logout after submitting a batch job, and examine the results at a later time. However, to do this you must prepare a batch script (you can do this using an editor like vim or nano) see the following example for batch script

```
#PBS -N My_name_is_test_code
#PBS -l select=1:ncpus=8:chip_type=l5420:mem=16gb,walltime=00:05:00
#PBS -M msiddig@clemson.edu
#PBS -o Palmetto_test_code_o.log
#PBS -e Palmetto_test_code_e.log
module add julia
module add gurobi
julia /home/msiddig/test/Palmetto_test_code.jl > /home/msiddig/test/
    ↪ Palmetto_test_code.out
```

here in this example we used the following

Command	Spesification
#PBS -N <name>	the name of the job – this could be anything you want it to be named
#PBS -l	I requested the computation node with the respective specification. Note that here I choose a specific type (Intel Xeon L5420), if you want you can simply skip <chip_type=l5420>. In fact, not specifying is what palmetto have in their example
#PBS -M palmetto-user@clemson.edu	an email will be sent to this address with the status of the job.
#PBS -o Palmetto_test_code_o.log	The standard output goes to this file
#PBS -e Palmetto_test_code_e.log	The error output goes to this file
module add julia	adds julia to the node
module add gurobi	adds gurobi to the node
julia path/code	runs the code

you can also specify the node you want to send it to. To do this simply replace

```
#PBS -l select=1:ncpus=8:chip_type=l5420:mem=16gb,walltime=00:05:00
```

with

```
#PBS -q <node_name>
#PBS -l select=1:ncpus=8:mem=16gb,walltime=00:05:00
```

BTW we have our own computation node in IE!. However, you need to be added to the "ieng" queue to be able to submit jobs to it. You can email Martin Clark and he will help you with this. It is worth mentioning the specification of the IE node

Table 1: Industrial Engineering node specification

Type	phase 0 which is a big-memory node.
Number of cores	the maximum number of cores is 40.
Walltime	The maximum walltime is 336:00:00 hours
Memory	The maximum memory is 1003gb

One final remark is, in this example we only submitted a single job. Clearly you might need to submit multiple jobs at the same time, here, multiple does NOT mean they are all going to be executed at the same time but rather one after the other. To do this you can simply write a simple for-loop in your bash file which create the jobs by itself. For sending multiple jobs which are executed simultaneously (which I personally recommend), see the next section.

4 Parallel Job submission

In my opinion this is the most useful feature of using the computer cluster. I should preface this by saying, in fact, writing a for-loop in your bash which creates a number of jobs in the queue is NOT recommended by palmetto, and if each job could take very long time to finish, this could take forever. Hence, parallel job submission is your best bet in this case.
To do this you need to use "gnu-parallel" which is a feature that allows you to execute multiple jobs independently at the same time. If you are not going to use the ieng queue (industrial engineering queue), the best queue in the cluster to excute this is the big-memory node "bigmem". To do this you need to do the following:

1. **Create a batch script:** Similary, you can do this using an editor like vim or nano. In this file you need to have the following:

   ```
   #/bin/bash

   #PBS -N <parallel_excution_example>
   #PBS -l select=1:ncpus=24:mem=494gb,walltime=72:00:00
   #PBS -q bigmem

   module add gnu-parallel
   module add julia
   module add gurobi

   cd $PBS_O_WORKDIR

   #cat $PBS_NODEFILE > test_nodes.txt

   parallel < commands.txt

   #rm ./test_nodes.txt
   ```

   Note that I didn't request the the status of the job to be sent to an email. To the best of my knowledge there's a bug regarding this, so even if you included <#PBS -M palmetto-user@clemson.edu> you will NOT receive an email when the submission is done.

2. **make the commands list:** Typically when you are doing this, you are most likely running different instances in the same code, or trying different parameters. You need to write a code which executes the batch script (let's call it "jobSubmit.sh" for the different instances). You will of course have different parameters in your code which gives different output according to the input. For example say you have 3 different instance (1, 2 and 3) and you want to test them for 3 different parameters (A, B and C) – in total you will have 9 jobs submitted to the queue (1-A, 1-B, 1-C . . . 3-C). This file will create another file which will send these parameters to your code (in which you need to specify how to read them) and it will submit a corresponding job (if this doesn't make sense to you, think of this code as someone who will submit all of these jobs for you manually!) Here's what you need to write in this file:

   ```
   #!/bin/bash

   file="./paramsList.txt"
   ofile="./commands.txt"
   runLog="./run.log"
   ```

```
rm -f $ofile

cat $file | while read line || [ -n "$line" ]; do
   fname="runLog.$(echo $line | tr -d [:space:]).txt"
   echo "echo $(date "+%Y-%m-%d %H:%M:%S") > $fname; echo Params: $line
       ↪ >> $fname; julia Palmetto_test_code.jl $line >> $fname" >>
       ↪ $ofile
done

qsub ./jobSubmit.sh
```

3. **Parameter List** As you can see the first command in the previous file is parameter "./paramsList.txt". This is a simple text file where you need to store all the parameters that you want to test, e.g, according to the previous example, this file will contain:

```
1 A
1 B
1 C
2 A
2 B
2 C
3 A
3 B
3 C
```

4. **parameters parse** As you could imagine, you need to have a julia file that will parse this text file into the parameters –this jl fie will have:

```
println("Argument one is:", ARGS[1]);
println("Argument two is:", ARGS[2]);
println("Just to test numbers: ", ARGS[1],"+",ARGS[2],"=",parse(Int32,
    ↪ ARGS[1])+parse(Int32, ARGS[2]))
```

In this example, the parameters will be parsed into your code in an array called ARGS. To use them, in your julia code, you need to have something like:

```
#to change the instance
if ARGS[1] == 1
      do instance 1
elseif ARGS[1] == 2
      do instance 2
elseif ARGS[1] == 3
      do instance 3
end
#to change the parameter
f ARGS[2] == "A"
      do parameter "A"
elseif ARGS[2] == "B"
      do parameter "B"
elseif ARGS[2] == "C"
      do parameter "C"
end
```

To recap, for parallel job submission you need:

- The job submission batch script (.sh file)
- The parameter list (.txt file)
- Parameter parsing code (.jl file)
- And of course you need your code and and all its dependencies to be in the same folder, and don't forget to write a script which translate the parameters to in your original code.

Finally, here are some commands that I find to be useful:

Table 2: Some useful commands

Command	Description
qstat <job id>	Check the status of the job with given job ID – the ID usually looks something like <5716605.pbs02>
qstat -u <username>	Check the status of all jobs submitted by given username – you may also check the jobs someone else is running if you know their name (this is by no-means a way to spy on other people but, I find this useful when I know that one of my colleagues is occupying the ieng node and I am waiting for them finish!)
qstat -xf <job id>	Check detailed information for job with given job ID
qsub -q <queuename> xyz.pbs	Submit to queue queuename
qdel <job id>	Delete the job (queued or running) with given job ID
qpeek <job id>	"Peek" at the standard output from a running job
qdel -Wforce <job id>	Use when job not responding to just qdel
whatsfree	gives information about how many nodes from each phase are currently in use, free, or offlined for maintenance.
module avail	List all packages available (on current system)
module add package/version	Add a package to your current shell environment
module list	List packages you have loaded
module rm package/version	Remove a currently loaded package
module purge	Remove all currently loaded packages
mkdir <directory name>	create a new directory with the given name
pwd	show current directory
cd <directory name>	change to the directory of the given name
cd	change to your /home directory
ls	list all the files in the files in the current directory
cp file1 file2	copy file1 to file1
cp -r dir1 dir2	copy dir1 to dir2
mv file1 file2	rename or move file1 to file2
chmod +x <filen-ame>	premit the file of the give name to be excuted.

If you have any question, you can always email @ msiddig@g.clemson.edu, and I'll try my best to help if I can.