Hyak R programming

Hyak fully supports R, anything that you can do with R on your desktop computer can be done on hyak but on a much larger scale. However, since hyak is a large scale shared resource there are a few steps that you have to take before running R.

Hyak is a Linux supercomputer. If you have not used the linux or UNIX or MacOS X command line before, it is a good idea to get familiar with the command line before running on hyak. Google "linux command line" for useful links. Of course, you should also have experience with R programming and R scripts on your desktop computer before you try running R on a supercomputer like hyak.

Usually, you will be using sbatch to submit a slurm script to the hyak scheduler. The slurm script will contain instructions for running your R script. The hyak scheduler takes care of details like finding a suitable node to run your R script etc.

Interactive Session

However, first let us use an interactive session to get familiar with running R on hyak. Issue the command below at the hyak login node, to get an interactive hyak session.

```
Modules
```

The build node can connect to outside mox.hyak or ikt.hyak. It is useful for using git, transferring files to outside mox or getting files from outside hyak, installing packages in R or Python etc.

To get an interactive build node for 2 hours:

```
srun -p build --time=2:00:00 --mem=10G --pty /bin/bash
```

This opens an interactive session on a hyak node.

If we type R at the command line and press enter then we get an error “R: command not found”. What’s wrong? Unlike your desktop computer, hyak supports hundreds of users with different requirements and different versions of the same software. Hence, unlike your desktop computer, not all software executables are put in everyone's PATH environment variable.

Those fluent with the Linux will want to put the location of R in their own PATH. However, this should be done using module command instead. It essentially performs similar tasks by modifying the PATH, LD_LIBRARY_PATH and other necessary parts of your environment.

The below command shows you the available modules:

```
module avail
```

The list is long and right now we are only interested in the latest version R (3.6.0 as of September 2019). Hence we issue the command:

```
module load r_3.6.0
```

Now the command R works.

After working for sometime, you may want to know what modules are still loaded in your environment. You can use below command to list the modules:

```
module list
```

After you have completed your work, you can remove the module from your environment by using below command.

```
module unload r_3.6.0
```

Installing CRAN packages on hyak

As hyak is a shared resource, CRAN packages cannot be installed in their default locations. Below are the steps to install CRAN packages to a location specified by you. Here UVW is the package name, XYZ is your hyak group name and abc is your hyak userid.

make a directory where you will install the R packages. For example:

```
mkdir /gscratch/XYZ/abc/rpackages
```

(If your group uses the same packages you could use a path like /gscratch/XYZ/rpackages . You can also use a path like /sw/contrib/XYZ/rpackages.)

If you have not done this earlier then see above for the commands to get a build node.

On the build node, issue below command to load the R module

```
module load r_3.6.0
```

You may have to do extra steps for some packages. For later in this page for CRAN packages which require some extra steps before installing.
Choose one of the following steps (a) or (b):

(a) If you do not want to specify the location of the library then start the R command line and issue below command. Here replace UVW by the name of the package that you want to install.

\[
\text{install.packages("UVW")}
\]

The command will ask you to choose a mirror. Choose a nearby mirror (e.g OR or CA) and press enter. R will ask you if you want to install the packages at below default location in your home directory. (There is 10GB quota on your home directory. Hence, you may want to specify a different location as shown in the next paragraph)

\[
~/R/x86_64-unknown-linux-gnu-library/{R\_VERSION}
\]

Enter "y" and your package will be installed at above location.
Now in your R scripts and at the R command line you only need to use below command

\[
\text{library(UVW)}
\]

(b) If you want to specify the location of the library then start the R command line and issue below command.

\[
\text{install.packages("UVW", lib="/gscratch/XYZ/abc/packages")}
\]

The command will ask you to choose a mirror. Choose a nearby mirror and press enter. The package UVW will be built and installed at the location /gscratch/XYZ/abc/packages

Once the above is done, whenever you want to use the package UVW, just load it in R by

\[
\text{library(UVW, lib.loc="/gscratch/XYZ/abc/packages")}
\]

You may not want to give the lib.loc parameter every time. In that case, you can put the below in your .bashrc file

\[
\text{export R\_LIBS="/gscratch/XYZ/abc/packages"}
\]

Now you can load the library by just issuing below command

\[
\text{library(UVW)}
\]

### Updating CRAN packages on hyak

For updating R packages use the below command at the R prompt:

\[
\text{update.packages()}
\]

This will update the packages that you installed. It will also update the recommended packages. It will not update the base packages. For the difference between the base and recommended packages see Stackoverflow

### Running R using slurm scripts and sbatch

Elsewhere on this wiki (Mox scheduler) you will find details about submitting slurm scripts via sbatch. Here we will focus on the R specific part. In your slurm script you should have lines like below. Note that if your scripts produce graphs then you should save the graphs using the usual R commands.

\[
\text{module load r\_3.6.0}
\text{Rscript >output.txt 2>&1 /gscratch/XYZ/abc/myscript.R}
\]

### Installing R from source code

You may want to install the latest version of R.

Run below command to find if that you have anconda Python in your path:

\[
\text{which python}
\]

You should get /usr/bin/python.

If you get anconda python because you put it in your PATH in your .bashrc then remove it from your PATH, logout of mox.hyak and then login and get a build node.
If you get anaconda python because you loaded an anaconda module then unload the anaconda module.

Now the command "which python" should give /usr/bin/python.

Note that you must remove anaconda from your PATH because it has old versions of libcurl and R install needs newer versions of libcurl (which already exist on mox).

Below xyz is your group name and abc is your userid.

Below are steps to install R from source:

1. srun -p build --time=4:00:00 --mem=10G --pty /bin/bash

2. mkdir /gscratch/xyz/abc/Rstuff

3. mkdir /gscratch/xyz/abc/Rinstall

4. cd /gscratch/xyz/abc/Rstuff

5. Below wget command will get source code for R 3.6.1. If you want the latest version of R, goto https://cran.cnr.berkeley.edu/ and right click on the link "latest release" and copy link location to choose a the latest version of R.

   wget https://cran.cnr.berkeley.edu/src/base/R-3/R-3.6.1.tar.gz

6. tar -xvf R-3.6.1.tar.gz

7. cd R-3.6.1

8. Optional: Use below steps to get a newer version of gcc. This step may be useful because some R packages expect a newer version of the C compiler. If you do this then remember to do this every time before using R

   cd /opt/rh/devtoolset-6

   source enable

9. ./configure --prefix=/gscratch/xyz/abc/Rinstall

10. make

11. make install

12. After this you can put /gscratch/xyz/abc/Rinstall/bin in your PATH environment variable:

   export PATH=/gscratch/xyz/abc/Rinstall/bin:$PATH

13. Run below command to verify that you have got R 3.6.1:

   R --version

**Installing Rglpk**

When installing the CRAN package Rglpk then issue below commands before you start R

export PKG_CFLAGS="-I/usr/include/glpk"

export PKG_CPPFLAGS="-I/usr/include/glpk"

**Installing processx and sys**

1. Install your own copy of R as shown in the above section "Installing R from source code"

2. srun -p build --time=4:00:00 --mem=10G --pty /bin/bash

3. The processx C code uses C99 and the default on newer versions of gcc is C99 or later.
Use below steps to get a newer version of gcc:

cd /opt/rh/devtoolset-6
source enable

(4) Use below step to get access to libimf.so (the intel math library)
module load icc_18

(5) R

(6) install.packages("processx")

(7) install.packages("sys")

---

Installing leiden and igraph

(1) if you (or the anaconda installer) have put anaconda python in your PATH in your .bashrc or .bash_profile then remove it
(2) if you are already logged in to mox then logout.
(3) ssh abc@mox.hyak.uw.edu

(4) srun -p build --time=4:00:00 --mem=10G --pty /bin/bash

(5) module load r_3.6.0
(6) R
(7) install.packages("leiden")
(8) install.packages("igraph")

---

Installing Rtsne

(1) if you (or the anaconda installer) have put anaconda python in your PATH in your .bashrc or .bash_profile then remove it
(2) if you are already logged in to mox then logout.
(3) ssh abc@mox.hyak.uw.edu

(4) srun -p build --time=4:00:00 --mem=10G --pty /bin/bash

(5) cd
(6) cd .R
(7) edit Makevars and put below line in it
CXXFLAGS += -wd308
(8) module load r_3.6.0
(9) R
(10) install.packages("Rtsne")

---

List of Installed Packages

Below command shows you the list of packages which come with R.
i <- installed.packages()
i[i[,"Priority"] %in% c("base","recommended"). c("Package", "Priority")]

---

Single Node Parallel R

R allows parallel processing via the package parallel which comes with R.

On a single node (where all processor cores share the same memory) it is the most efficient to use mclapply or fork clusters:

library(parallel)
cl <- makeForkCluster(detectCores())
result <- parLapply(cl, i, myfunc)
stopCluster(cl)

This code creates a fork-cluster on a single node by "forking" the running R process. All the environment is shared, no data copying or initiating new instances of R is necessary. detectCores() detects the number of CPU cores. Don't forget to stop the cluster afterwards.
If your program is using the R parallel package then use the options `--nodes=1 --ntasks-per-node=28` with your srun command or sbatch script to ensure that your program can use 28 cores on the node. Note that the R parallel package only supports single node parallelism.

**Multi-node Parallel R using MPI**

This section is for experts only. It contains information for using R with multiple nodes via MPI.

Installing Rmpi can be complicated. Hence, on mox.hyak use the below command to access the R version which has the Rmpi package pre-installed.

```bash
module load r_3.6.0+Rmpi-impi_2019
```

For an example of a slurm sbatch script, see the section on "Batch usage Single Node" in the below link. Replace `--nodes=1` by `--ntasks=M`. Here M is the number of cores that you want the Rmpi program to use. The slurm scheduler will use the `--ntasks` value to calculate how many nodes should be assigned to your job.

**Mox Scheduler**

In your slurm sbatch script, use below commands to run your Rmpi program. Replace mpi_example.R by the name of your Rmpi program.

```bash
# Load the r_3.6.0 module with Rmpi support
module purge
module load r_3.6.0+Rmpi-impi_2019

# You must use "mpirun -n 1"; do not use srun
mpirun -n 1 Rscript mpi_example.R
```

See the mox directory `/sw/contrib/rmpi` for an example sbatch script.