Parallel Computing with R

Paul E. Johnson[1] [2]

[1]Department of Political Science

[2]Center for Research Methods and Data Analysis, University of Kansas

2017

Outline

Outline

crmda.ku.edu/computing

- That's the official reference for HOW TO use the cluster and its systems.
- If you want to know how to do something, please refer to that material first before asking Google how its done at the University of Alabama
- When uncertain, please review examples in hpcexamples.

Cluster management now in CRC

- Center for Research Computing
- Jargon changes. Login name changes. Procedural changes
  `https://crc.ku.edu/hpc/how-to`
- I don't understand everything there, our blog has updates:
  `https://www.crmda.dept.ku.edu/timeline`

Outline

What is a "cluster"?

Cluster: lots of separate workstations ("nodes") that are networked together.

Racks and racks of compute nodes

rack 0	rack 1	rack 2
compute-0-0	compute-1-0	compute-2-0
compute-0-1	compute-1-1	compute-2-1
compute-0-2	compute-1-2	compute-2-2
⋮	⋮	⋮
compute-0-19	compute-1-19	compute-2-19

The "compute nodes" are separate Linux systems where the computations take place.

The login node

hpc.crc.ku.edu: where you log in, in order to gain access.

Will redirect you to either

1. submit1.hpc.crc.ku.edu
2. submit2.hpc.crc.ku.edu
   Can login directly to either one.

The login node

RULE 1. Do not do "computing" on login nodes

- Software is missing there
  - admins will kill intensive jobs
  - programs are installed in the compute nodes

Login nodes on ACF: is a place to

- begin "interactive" sessions (the "interactive" que).
- submit and monitor longer running jobs (the "default" que).

node and processor

Users ask for resources from the cluster, referring to nodes and processor per node

node: one of those separate computer systems

ppn: processor per node (2017: most systems have 20 CPUs per node, some have more

Expect RAM 2GB per processor.

If you ask for nodes=8:ppn=1 the scheduler may assign your 8 jobs to ONE compute node.

If you ask for nodes=1:ppn=8, you are safer

Interactive que

- This is just like “logging into” a remote workstation, such as compute-1-1. It is used for
  - developing code
  - testing programs
- HOWTO? From submit2.hpc.crc.ku.edu, run this command:

```
$ msub -I -q interactive -l nodes=1:ppn=1
```

  - The “shortcut” we created for that is **qlogin**
  - That asks for permission to have a “login” session with one compute node with 1 processor in that node.

Interactive que ...

- To request an interactive session with more nodes, adjust the values accordingly.
- This does NOT ask for X11 “graphic forwarding”, will be a strictly TEXT session (CLI)

X-server Allowed

- If you have an X-server running, or are in NoMachine, add a -X in the command:

```
$ msub -I -X -q interactive -l nodes=1:ppn=1
```

- qxlogin does similar

```
$ cat /usr/local/bin/qxlogin
#!/bin/sh
ARGS=$@
/opt/moab/bin/msub -X -I -lnodes=1:ppn=1 $ARGS
```

- If your PC has an X11 server running, graphics display as sent out from compute node, through the login node, into your PC

Shortcuts to Interactive Sessions

Repeat: Shortcut programs to launch interactive sessions

```
$ qlogin
```

or, for X11 enabled sessions

```
$ qxlogin
```

- If more nodes or PPN are required, options are allowed:

```
$ msub -I -X -q interactive -l nodes=2:ppn=2
```

Would request 2 nodes with 2 ppn.

- While testing some cluster programs, I DO run interactive multi-node sessions.

The Queues

- Two queues to consider
  - “crmda”
  - “sixhour”
- “crmda”
  - jobs stay within nodes we own
  - can run indefinitely
- “sixhour”
  - time limit
  - access to all available nodes in community cluster

To run a long running job

- User must prepare 2 things.
  - A “submission script”
  - A program
- Run the “msub” program on the SCRIPT!

Submission Script

Most submission scripts have "boiler plate" at the top

```
1  #!/bin/sh
2  #
3  #
4  #MSUB -M your-name_here@ku.edu
5  #MSUB -N RParallelHelloWorld
6  #MSUB -q sixhour
7  #MSUB -l nodes=1:ppn=11:ib
8  #MSUB -l walltime=00:05:00
9  #MSUB -m bea
10
11 cd $PBS_O_WORKDIR
12 mpiexec -n 1 R --vanilla -f parallel-hello.R
```

Boiler Plate Submission Script

-N	the name of the job–you choose that as a reminder for yourself
-l nodes=	resource request. CAN be more elaborate, specifying memory and other restrictions
-l walltime	maximum time job can run before it is killed by OS
-M	YOUR email address. If you see "pauljohn@ku.edu", please change it
-m	bea send email to YOUR email address when job begins, ends, or crashes.
-q	The queue in which your job will run. There is no "default" queue anymore!

qstat

- Review the jobs that are running
  $ qstat
  - function from old queue that still works

```
$ showq
```

- grep through output to find your name
  $ qstat | grep your−name−here
- ACF also has a Web accessible program “ganglia” that can be used to survey running programs.
  `http://ganglia.acf.ku.edu`

Network File System

- We CAN write on /tmp inside each node's hard disk
- /home/username and $WORK {/panfs/pfs.local/work/crmda/your_name_here} are Network File System shares, they are links that point to a network storage device.
- Warning: reading and writing lots of small files in the NFS is SLOW, and if your project requires lots and lots of writing, it is worthwhile to learn how to read & write in /tmp and copy results into /home when needed.

Complication: Infiniband vs Ethernet

- If the compute cluster has some nodes witht "infiniband" and some without, we have a problem
  - Infiniband: super-fast communication system called Infiniband
  - Ethernet: boring ordinary 1GB ethernet cables

Infiniband Workaround

- "ib" option in submission scripts.
- Example:

```
#PBS -l nodes=11:ppn=1:ib
```

will confine the nodes to be selected from the infiniband systems.

- This is only necessary with programs that use the MPI, the Message Passing Interface.
- It is not necessary for running jobs that simply launch one instance or Mplus, SAS, or R.

Outline

Please Walk Before You Run (lots of jobs)

- Before you get fancy and try to run a job on 20 nodes at once
- Make sure it runs well on one node!
- This is necessary!
  - check your memory usage.
  - make sure your program is efficient
  - possible to profile 1 program on 1 node (not possible to profile a parallel run)

hpcexample Ex50: R example

- Some Windows or Mac users have never "run" a program, so try that now.
- Get your R program that works, say myCoolProgram.R. Be sure it writes output, such as a pdf, and saves some data output or a fitted model in a file.
  1. Copy it to hpc.crc.ku.edu.
  2. Put it in a subdirectory.
  3. Log into an interactive session.
  4. Change into the subdirectory where you wrote the R program.
  5. Run the program.

How to run R from the command line?

- There are many ways to run an R program, such as the traditional method:

```
R < myCoolProgram.R &
```

- but generally I like this style better (does same):

```
R -f myCoolProgram.R &
```

- At one time, I was very eager to use Rscript instead, but I don't see much difference now. Warning: some people are emphatic about Rscript or littler.

Monitor a long running job

- Start a few xterms from the same compute node.
- On one terminal, monitor CPU and memory

```
$ top
```

- In another terminal, run the R program.
- On another terminal, run something like this to get a different view

```
$ ps -aux | grep R
```

Prepare for Parallel: Be Cautious about R's environment

- Ever notice R reads your "saved session" and always wants to save a workspace when you close?
- Some people like that, but in the cluster YOU MUST NOT DEPEND on saved session files (.RData) to make your program work.
- So add command line options like "–no-restore-data" and "–no-save", or get all of the options with –vanilla. So test your program again like this:

```
$ R --vanilla  -f myCoolProgram.R &
```

—vanilla combines –no-save, –no-restore, –no-site-file, –no-init-file and –no-environ

How to schedule same R program on the default que

- Running interactively is FINE, it is just same as submitting a job on the que so that it can be sent to a compute node
- But, if it is a long-running job, it is necessary for you to “keep the terminal open” for a long time.
- Instead, schedule the job in the cluster, let the job go into the batch que.

Submission Script in Ex50 from hpcexamples

The submission script was shown above. See the file "sub-serial.sh"

```
1  #!/bin/sh
2  #
3  #MSUB -M your-name-here@ku.edu
4  #MSUB -N Rsimple
5  #MSUB -l nodes=1:ppn=1:ib
6  #MSUB -l walltime=00:10:00
7  #MSUB -m bea
8  #MSUB -q crmda
9
10 ## Your queue need not be "crmda". Could be
11 ## "sixhour" instead.
```

Submission Script in Ex50 from hpcexamples ...

```
12
13  cd $PBS_O_WORKDIR
14
15  ## Please check your ~/.bash_profile to make
        sure
16  ## the correct modules will be loaded with new
        shells.
17  ## See discussion:
18  ## http://www.crmda.dept.ku.edu/timeline/
        archives/184
19  ## It is not helpful or necessary to insert
        module
20  ## commands here (contrary to previous advice)
        .
21
```

Submission Script in Ex50 from hpcexamples ...

```
22 R --vanilla -f r-serial.R
23
24 ## Note previous does not use mpiexec, orterun
       , or mpirun, because
25 ## it is not using the message passing
       interface. It is one simple
26 ## R program, in isolation.
```

Run Ex50. r-serial.R is a very small example program

- Go to hpc.crc.ku.edu
- Go to compute node (interactive login)
- Make sure your GIT copy of hpcexample is up to date
- cd to Ex50, revise sub-serial.sh to fit your needs
- May need to exit back to login node to submit (maybe not!)
- Submit the job

```
$ msub sub-serial.sh
```

Your result files

- The R program generates several output files.
  - This job creates a PDF image of a graph
- The cluster also creates 2 files, an “e” file and an “o” file
  - “o” standard output, the messages that would have gone to terminal if you ran R from CLI
  - “e” standard error, which also would have gone to terminal.
- The “e” and “o” files will have the name you declared in your submission script (-N) and the job number in the cluster.
- For testing, I ran that job 2012-10-22.

Your result files ...

- Rsimple.o517451 looks just like an ordinary R output in the terminal:

```
R version 3.3.2 (2016-10-31) -- "Sincere
    Pumpkin Patch"
Copyright (C) 2016 The R Foundation for
    Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY
    NO WARRANTY.
You are welcome to redistribute it under
    certain conditions.
```

Your result files ...

```
Type 'license()' or 'licence()' for
    distribution details.

R is a collaborative project with many
    contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in
     publications.

Type 'demo()' for some demos, 'help()' for
    on-line help, or
'help.start()' for an HTML browser interface
    to help.
Type 'q()' to quit R.
```

Your result files ...

```
> ## Paul Johnson
> ## The second stupidest R program ever.
>
> x ← rnorm(1000, mean = 17, sd = 43)
>
> mean(x)
[1] 14.26859
>
> pdf("testGraph.pdf", height = 6, width = 6,
    onefile = F, family = "Times")
> hist(x, main = "One Stupid Histogram")
> dev.off()
null device
          1
```

Your result files ...

- And the “e” file Rsimple.e517451 is a big empty NOTHING, meaning no problem at all!

```
.
```

- Remember, the “e” file is reporting the standard error output from the SHELL, not R error messages. R errors still show up in Rsimple.o517451.

View System Usage in a Longer-Running Program

- HPC will email you when the job starts if you had “b” in “bea”.
  - it will tell you which node is running your job (ex, n499).
- If the job is running for a while, you can go track its memory usage
  - SSH to login node,
  - request an interactive session in any node
  - Then “ssh n499”.
- HPC DOES NOT allow ssh direct to n499.
- Also possible to access via the interactive que:

```
$ msub -I -l nodes=compute-0-0
```

- In the shell, run programs like “top” and “ps” to view memory usage of your program.

Outline

"Poor Person's" Parallel Computing

- Need to run a program 1000 times, with different
  - random number seeds, or
  - parameters
- Strategy. Write one program that accepts command-line arguments
- Write 1000 submission scripts,
  - adjust the submision scripts to provide slightly different arguments for each run
- hpcexamples Ex51-ManySerialJobs demonstrate this.

Command Line Arguments

- All programs worth using allow command line arguments. (no need to recompile or edit program, just run with different arguments!)
- Suppose, for example, we want to re-set the run number and the random seed.
- Re-write the R code to scan the command line arguments and do the right thing, so we can run:

```
R --vanilla -f r-commandline.R --args run="010"
    seed="12345"
```

- The R function commandArgs() can be used to receive command line arguments.

Ex51: command line arguments

- User has responsibility to "parse" those arguments and put them to use.
- The example program r-commandline.R demonstrates. Please read r-commandline.R.
- Examples:

```
$ R -f r-commandline.R --args myMeanN=88.1231 runI
    =11 whateverC="hell" nofcasesI=2323
$ R -f r-commandline.R --args myMeanN=77.1231 runI
    =12 whateverC="hell" nofcasesI=1121
```

- That will generate pdf and R data files in the current working directory (because outdirC was not specified).

Ex51 README

Its a very thorough README file

This assumes program accepts command line arguments

- Some programs are stupid, do not have way to receive command line options inside a program.
- In these cases, such as with Mplus, it is necessary to revise our plan. The script that creates the submission scripts must also write 1000 mplus programs.
- For an example of that, see hpcexamples Ex08-MplusRunall-1, Ex09-MplusRunall-2

Shortcomings

- Does not guarantee that random number streams do not "overlap"
- Makes integration of results across all simulations more difficult

But the benefit of this approach is plain. If you can write one R program, you can easily script it to run over and over again.

Job Array more Rigorous Version of Same

- hpcexamples 52
  Ex52-R-JobArray
- A Job Array is a way to submit a lot of separate R jobs without frustrating yourself too much.
- README and example was thoroughly re-worked in 2017.

Outline

Multicore

- "Shared memory" computing: the RAM in a machine is shared among several cores
- Special math and computation algorithms may be able to divide work among cores to speed things up.
- R parallel package (introduced with R-2.14 formalized the plan for multi-core support in the "parallel" package).
  - See the function "mclapply" to automatically divide work on list elements among cores
- I used recently in speeding up report generation in one project

Be Cautious

- Not available on Windows
- It CONFLICTS with the MPI framework that I have used more often.
- Will go BERSERK if you run it with browser() in your R code, will not allow debug.
- If you use this, try to learn about the ways that we can tell R how many cores to use.
- multicore work does not always go faster, because work of dividing up a job causes “overhead”.

Outline

Practice on your own computer

- Don't use "multicore"
- Do use your cores as if they were separate computers
- R parallel packages allow you to create a "sock" cluster using parallel idioms that are same as will be used in cluster
- See hpcexample Ex67-SOCK-Cluster

Create a Sock Cluster

This creates a cluster of 4 “fake nodes” on your personal computer

```
library(parallel)
cl <- makeCluster(4, type = "SOCK")
```

After that, you practice with the parallel programming ideas, using the functions that talk to the cluster

Which are topic of next section

Outline

Rmpi: R on top of OpenMPI

- C library "OpenMPI" implements the "message passing interface"
- R package Rmpi is a layer between OpenMPI and R
- Use Rmpi to "spawn a cluster" of worker nodes, send work to them, collect it up
- Other parallel packages sit on top of Rmpi
  - SNOW (Ex60-HelloWorldSnow), SNOWFT (Ex61-HelloWorldSnowFT)
  - doMPI, foreach, (and many other packages)
  - R's own "parallel" sits "on top" of SNOW and other cluster-level packages.

Write programs with the Rmpi package, or with SNOW?

Why use Rmpi?

- Plenty of features (was very "bare boned" in early days)
- When errors occur, source may be more obvious (not obscured by addon-components)

Why use SNOW or others that sit "on top" of Rmpi?

- SNOW may handle details more correctly than you can!
- R parallel attempts to standardize terminology, handling details for us (not perfect yet, but nothing is).

Submission script changes: orterun.

See: Ex53-HelloWorldRmpi

```
1  #!/bin/sh
2  #
3  #MSUB -N RmpiHelloWorld-1
4  #MSUB -l nodes=2:ppn=19:ib
5  #MSUB -l walltime=00:60:00
6  #MSUB -M pauljohn@ku.edu
7  #MSUB -m bea
8  #MSUB -q crmda
9
10 cd $PBS_O_WORKDIR
11
12 ## Please check your ~/.bash_profile to make sure
13 ## the correct modules will be loaded with new
       shells.
```

Submission script changes: orterun. ...

```
14 ## See discussion:
15 ## http://www.crmda.dept.ku.edu/timeline/archives/
       184
16 ## Don't load modules here, they will not apply to
       all
17 ## separate worker sessions.
18
19 mpiexec -n 1 R --vanilla -f mpi-hello-1.R
```

Look at the last line

- mpiexec is a program from OpenMPI, it cooperates with cluster framework. Same effect as “orterun”, which you will see in some of my examples.
- -n 1
  - cluster “spawn” mode means “claim one node for the master node”
  - Since script asks for more nodes above, code inside R program can spawn the worker nodes.
- In old cluster, was necessary to provide a node list: -hostfile $PBS_NODEFILE
  means “select nodes from available list”. Not necessary now, as of July 2017

How Do I Know it is using "a cluster?"

This job does almost nothing, except it spawns a set of nodes as a cluster

and asks each one for a little bit of information

```
if (!is.loaded("mpi_intitialize")){
  library(Rmpi)
}

## see http://math.acadiau.ca/ACMMaC/Rmpi/
    sample.html
# Spawn as many slaves as possible

##mpi.spawn.Rslaves()

### Try to spawn worker processes
```

How Do I Know it is using "a cluster?" ...

```
mpi.spawn.Rslaves(nslaves=20)

# In case R exits unexpectedly, have it
    automatically clean up
# resources taken up by Rmpi (slaves, memory,
    etc...)
.Last ← function(){
  if (is.loaded("mpi_initialize")){
    if (mpi.comm.size(1) > 0){
      print("Please use mpi.close.Rslaves() to
          close slaves.")
      mpi.close.Rslaves()
      }
      print("Please use mpi.quit() to quit R")
      .Call("mpi_finalize")
    }
```

How Do I Know it is using "a cluster?" ...

```
}

mpi.remote.exec(paste("I am process ",mpi.comm.rank
    (),"/ ",mpi.comm.size()))
```

To find out more about functions available in Rmpi, read the documentation provided with the package or find a more detailed example in hpcexamples Ex57-MISimulation-RMPI or Ex80 (which, at least now, uses RMPI directly).

Rmpi function list

```
## output from library(help=Rmpi)
     Information on package 'Rmpi'

Description:

Package:                Rmpi
Version:                0.5-9
Date:                   20010-11-30
Title:                  Interface (Wrapper) to MPI (
    Message-Passing
                        Interface)
Author:                 Hao Yu
Maintainer:             Hao Yu <hyu@stats.uwo.ca>
Depends:                R (≥ 2.2.0)
Suggests:               rsprng, rlecuyer
```

Rmpi function list ...

```
Description:          Rmpi provides an interface (
    wrapper) to MPI APIs.
                      It also provides interactive R
                          slave environment.
License:              GPL (≥ 2)
URL:                  http://www.stats.uwo.ca/faculty
    /yu/Rmpi
Packaged:             2010-11-30 19:38:01 UTC; hyu
Repository:           CRAN
Date/Publication:     2010-11-30 19:55:14
Built:                R 2.15.0;
    x86_64-redhat-linux-gnu; 2012-05-22
                      20:35:48 UTC; unix

Index:
```

Rmpi function list ...

MPI APIs:

```
mpi.abort                    Abort (quit) all tasks
    associated with a comm
mpi.allgather                Gather data from each
    process to all process
mpi.allgatherv               Gather diff size data from
    each process to all process
mpi.allreduce                Reduce all process's
    vectors into one vector
mpi.barrier                  Block the caller until all
    group have called it
mpi.bcast                    Broadcast a vector (int,
    double,char) to every process
mpi.cancel                   Cancel a nonblocking send
    or recv
```

Rmpi function list ...

```
mpi.cart.coords          Translate a rank to the
    Cartesian topology coordinate
mpi.cart.create          Create a Cartesian
    structure of arbitrary dimension
mpi.cartdim.get          Get dim information about a
     Cartesian topology
mpi.cart.get             Provide the Cartesian
    topology associated with a comm
mpi.cart.rank            Translate a Cartesian
    topology coordinate to the rank
mpi.cart.shift           Shift Cartesian topology in
     displacement and direction
mpi.comm.disconnect      Disconeect and free a comm
mpi.comm.dup             Duplicate a comm to a new
    comm
mpi.comm.free            Free a comm
```

Rmpi function list ...

```
mpi.comm.get.parent      Get the parent intercomm
mpi.comm.rank            Find the rank (process id)
    of master and slaves
mpi.comm.remote.size     Find the size of a remote
    group from an intercomm
mpi.comm.size            Find the size (total # of
    master and slaves)
mpi.comm.set.errhandler Set comm to error return (
    no crash)
mpi.comm.spawn           Spawn slaves
mpi.comm.test.inter      Test if a comm is an
    intercomm
mpi.dims.create          Create a Cartesian dim used
     by mpi.cart.create
mpi.finalize             Exit MPI environment (call
    MPI_Finalize())
```

Rmpi function list ...

```
mpi.gather                  Gather data from each
    process to a root process
mpi.gatherv                 Gather diff data from each
    process to a root process
mpi.get.count               Get the length of a message
     for given status and type
mpi.get.processor.name  Get the process (host) name
mpi.info.create             Create an info object
mpi.info.free               Free an info object
mpi.info.get                Get the value from an info
    object and a key
mpi.info.set                Set a key/value pair of an
    info object
mpi.intercomm.merge     Merge a intercomm to a comm
mpi.iprobe                  Nonblocking use a source
    and a tag to set status
```

Rmpi function list ...

```
mpi.irecv                    Nonblocking receive a
    vector from a specific process
mpi.isend                    Nonblocking send a vector
    to a specific process
mpi.probe                    Use a source and a tag to
    set status
mpi.recv                     Receive a vector from a
    specific process
mpi.reduce                   Reduce all processes's
    vectors into one (one process)
mpi.scatter                  Opposite of mpi.gather
mpi.scatterv                 Opposite of mpi.gatherv (
    diff size data)
mpi.send                     Send a vector to a specific
     process
```

Rmpi function list ...

```
mpi.sendrecv                Send & receive different
    vectors in one call
mpi.sendrecv.replace        Send & replace a vector in
    one call
mpi.test                    Test if a nonblocking send/
    recv request is complete
mpi.testall                 Test if all nonblocking
    send/recv requests are complete
mpi.testany                 Test if any nonblocking
    send/recv requests are complete
mpi.testsome                Test if some nonblocking
    send/recv requests are complete
mpi.test.cancelled          Test if a communication is
    cancelled by mpi.cancel
mpi.universe.size           Total number of CPUs
    available
```

Rmpi function list ...

```
mpi.wait                    Wait if a nonblocking send/
    recv request is complete
mpi.waitall                 Wait if all nonblocking
    send/recv requests are complete
mpi.waitany                 Wait if any nonblocking
    send/recv requests are complete
mpi.waitsome                Wait if some nonblocking
    send/recv requests are complete
*************************************************
    ************************
MPI Extensions in R Environment:
lamhosts                    Hosts id and machine host
    name mapping
mpi.allgather.Robj          Gather any type of objects
    to every number
```

Rmpi function list ...

```
mpi.any.source              A constant for receiving a
    message from any source
mpi.any.tag                 A constant for receiving a
    message from any tag
mpi.bcast.Robj              Broadcast an R object to
    every process
mpi.comm.maxsize            Find the length of comm
    array
mpi.exit                    Call MPI_Finalize and
    detach Rmpi library
mpi.gather.Robj             Gather any type of object
    to  a root process
mpi.get.sourcetag           Get the source and tag for
    a given status
mpi.hostinfo                Get the host information
    that the process is running
```

Rmpi function list ...

```
mpi.init.sprng              MPI wrapper to initialize
    SPRNG in rsprng library
mpi.is.master               TRUE if it is a master
    otherwise FALSE (slave)
mpi.isend.Robj              Nonblocking send an R
    object to a specific process
mpi.proc.null               Dummy source and
    destination
mpi.quit                    Call MPI_Finalize and quit
    R
mpi.recv.Robj               Receive an R object from a
    process (by mpi.send.Robj)
mpi.realloc.comm            Increase comm array to a
    new size
mpi.realloc.request         Increase request array to a
     new size
```

Rmpi function list ...

```
mpi.realloc.status          Increase status array to a
    new size
mpi.request.maxsize         Find the length of request
    array
mpi.scatter.Robj            Scatter an list to every
    number
mpi.send.Robj               Send an R object to a
    specific process
mpi.spawn.Rslaves           Spawn R slaves. The default
     R script is slavedaemon.R
mpi.status.maxsize          Find the length of status
    array
mpichosts                   finds host names from
    master Windows registery database
*************************************************
    ************************
```

Rmpi function list ...

```
MPI Extensions specifically to slavedaemon.R Script
    (interactive R slaves).

mpi.apply                   Scatter an array to slaves
    and then apply a fun
mpi.applyLB                 Load balancing version of
    mpi.apply
mpi.bcast.Robj2slave        Master sends an Robj to all
     slaves
mpi.bcast.cmd               Broadcast a commond to
    every process
mpi.close.Rslaves           Close all slaves launched
    by mpi.spawn.Rslaves()
mpi.parApply                (Load balancing) parallel
    apply
```

Rmpi function list ...

```
mpi.parCapply              (Load balancing) parallel
    column apply
mpi.parLapply              (Load balancing) parallel
    lapply
mpi.parRapply              (Load balancing) parallel
    row apply
mpi.parReplicate           A wrapper to mpi.parSapply
    for repeated eval of an expr
mpi.parSapply              (Load balancing) parallel
    sapply
mpi.parSim                 (Load balancing) parallel
    Monte Carlo simulation
mpi.remote.exec            Run a command remotely on
    slaves and return
                           results back to the master
```

Rmpi function list ...

```
mpi.setup.rngstream       Setup RNDstream (package
    rlecuyer) on all slaves
mpi.setup.sprng           Setup SPRNG (package rsprng
    ) on all slaves
slave.hostinfo            Show all slave rank, comm,
    host information
tailslave.log             Tail (view) last lines of
    slave log files
**************************************************
    **********************
Some Internal Functions used by Other MPI Functions

bin.nchar                 Find the length of a binary
     string
mpi.comm.is.null          Test if a comm is NULL (no
    members)
```

Rmpi function list ...

```
string                    Create a string (empty
    space character) buffer
*************************************************
    **********************
Deprecated functions

mpi.parallel.sim          It is renamed to
    mpi.parSim. Still available.
```

The Basic Parallel R Problem

- Each node needs to have EVERYTHING it needs to do its job.
- It is NOT sufficient to simply tell a node to run a function
- Before that, node must be
  - given required functions and data objects
  - told to do preparatory calculations
- If you forget to export a function to the worker nodes, a parallel R program will hang indefinitely, without generating an error or output. (Hence: be careful)

Rmpi Programming: sending everything out

Excerpt from Ex80-PrevSci2007, version-3.R

```
## OMIT code to create functions
## calculateRegressions, createOneTable,
    createTSPlot, drawSample
## imposeMissings plotRows, seedlist,
    conductSimulation

## Create a cluster
library(Rmpi)
mpi.spawn.Rslaves(nslaves=6)
## Send functions to each worker node:
mpi.bcast.Robj2slave(calculateRegressions)
mpi.bcast.Robj2slave(createOneTable)
mpi.bcast.Robj2slave(createTSPlot)
mpi.bcast.Robj2slave(drawSample)
```

Rmpi Programming: sending everything out ...

```
mpi.bcast.Robj2slave(imposeMissings)
mpi.bcast.Robj2slave(imputeDF)
mpi.bcast.Robj2slave(missingMask)
mpi.bcast.Robj2slave(plotRows)
mpi.bcast.Robj2slave(seedlist)
mpi.bcast.Robj2slave(tablelist)
mpi.bcast.Robj2slave(conductSimulation)

mpi.bcast.Robj2slave(dirname)
mpi.remote.exec(setwd(dirname))
##forces a random draw that initializes the random
    generators
mpi.remote.exec(rnorm(1))
```

Rmpi Programming: sending everything out ...

```
##results <- mpi.parLapply(1:nruns,
    conductSimulation, ssizes=ssizes,  b0=b0, b1=b1
    , pm=pm, m=m, setseed=NULL)
results <- mpi.applyLB(1:nruns, conductSimulation,
    ssizes=ssizes,  b0=b0, b1=b1, pm=pm, m=m,
    setseed=NULL)

save(results, file=paste("results", jobname,".Rda",
     sep=""))

## lets just check again, see if they are
    listening.
mpi.remote.exec(rnorm(1))

mpi.close.Rslaves()
mpi.quit()
```

SNOW: Simple Network of Workstations

- Pioneered by Luke Tierney, famous statistician U. Iowa (my Alma Mater!)
- SNOW package interface emphasizes CLARITY, attempting to handle fine-grained details behind the scenes
- Highlights:
  - Can spawn clusters of different types, including RMPI (which we use on ACF)
  - Parallel versions of apply and lapply

Remember R basics: apply, lapply

- R allows for loops, but discourages their usage for style & efficiency
- Instead, we use

  apply to apply a function to "rows" or "columns" from a matrix

  lapply to apply a function to each separate object in a list

  - practice with those functions! Read my lecture functions-1.
- SNOW offers several cluster-capable functions to do the same kind of work.
- R parallel package incorporates many SNOW functions and idioms

Snow Documents: ?parLapply

```
snow-parallel                          package:snow
                        R Documentation

Higher Level SNOW Functions

Description:

      Parallel versions of 'apply' and related
        functions.

Usage:

      parLapply(cl, x, fun, ...)
      parSapply(cl, X, FUN, ..., simplify = TRUE,
          USE.NAMES = TRUE)
      parApply(cl, X, MARGIN, FUN, ...)
```

Snow Documents: ?parLapply ...

```
     parRapply(cl, x, fun, ...)
     parCapply(cl, x, fun, ...)
     parMM(cl, A, B)

Arguments:

       cl: cluster object

      fun: function or character string naming a
           function

        X: array to be used

        x: matrix to be used
```

Snow Documents: ?parLapply ...

```
      FUN: function or character string naming a
           function

   MARGIN: vector specifying the dimensions to use.

 simplify: logical; see 'sapply'

USE.NAMES: logical; see 'sapply'

      ...: additional arguments to pass to standard
           function

        A: matrix

        B: matrix
```

Snow Documents: ?parLapply ...

```
Details:

     'parLapply', 'parSapply', and 'parApply' are
        parallel versions of
     'lapply', 'sapply', and 'apply'.

     'parRapply' and 'parCapply' are parallel row
        and column 'apply'
     functions for a matrix 'x'; they may be
        slightly more efficient
     than 'parApply'.

     'parMM' is a very simple(minded) parallel
        matrix multiply; it is
     intended as an illustration.
```

Snow Documents: ?parLapply ...

```
      For more details see <URL:
      http://www.stat.uiowa.edu/~luke/R/cluster/
          cluster.html>.

Examples:

      ## Not run:

      cl <- makeSOCKcluster(c("localhost","localhost
          "))
      parSapply(cl, 1:20, get("+"), 3)
      ## End(Not run)
```

Ex65-R parallel

- This is the “test case” we use, the template for many projects
- Shows how to create MPI cluster,.

```
## Paul Johnson
## 2012-01-05
## 2015-11-16
## 2016-10-24

## Use Rmpi via SNOW via R's parallel
## "Message Passing Interface" (OpenMPI
    implementation)

library(parallel)
```

Ex65-R parallel ...

```
## This is a nice boilerplate that is not necessary
    , but
## may help for graceful shutdown.  When R tries to
     quit,
## it tries to run the function .Last. If user
    forgets
## to closer cluster, this may relinquish resources
     more gracefully.
.Last ← function(){
    if (is.loaded("mpi_initialize")){
        if (Rmpi::mpi.comm.size(1) > 0){
            print("Close nodes.")
            snow::stopCluster(cl)
        }
        print("Please use mpi.quit() to quit R")
        Rmpi::mpi.quit()
```

Ex65-R parallel ...

```
    }
}

## The main functions we need to demonstrate are
## makeCluster: creates worker nodes
## clusterCall: calls function with same arg on
    each node
## clusterEvalQ: Sends a string to each node in
    cluster.
## clusterExport: sends object to each node
##
## and then a family of "apply-like" functions
## parLapply, parApply.
##
## clusterApply is unfamiliar to me, it appears to
    be
```

Ex65-R parallel ...

```
## the same as parLapply



## Causes an MPI cluster to exist. Same as snow
    package,
## uses Rmpi functionality
NCORES ← 10 ## Number of processors -1
cl ← makeCluster(NCORES, type = "MPI")


## clusterCall runs a given command on each node.
    Can be an
## individual command, such as loading a package
```

Ex65-R parallel ...

```
## clusterEvalq runs a command on each node.
    Output is written on the
## session unless we prevent that by assigning the
    result.  Note the
## return, node by node, is verbose
clusterEvalQ(cl, library(mvtnorm))

## Ask each system for information
clusterEvalQ(cl, Sys.info())


## If we wanted to, we could put each node to use
## the same random number stream.

## first, lets set the seed on the manager node
set.seed(123123)
```

Ex65-R parallel ...

```
(p ← rnorm(1, m = 33))
##See, nodes same as p!
clusterEvalQ(cl, {
    set.seed(123123)
    rnorm(1, m = 33)
})


## cluster Call is similar, but it is for running a
    function with the
## same arguments on each node.  Note how the
    syntax separates the
## name of the function from the arguments
clusterCall(cl, set.seed, 123123)
clusterCall(cl, rnorm, 1, m = 33)
## Note they stay in sync if you do it again
```

Ex65-R parallel ...

```
clusterCall(cl, rnorm, 1, m = 33)


### sends request to each node in system
clusterCall( cl, function() rnorm(1, 33,1 ) )



## It is not necessary to have arguments.  Here is
    a function with no
## arguments.

getNodeInfo ← function()  Sys.info()[c("nodename",
    "machine")]
clusterCall(cl, getNodeInfo)
```

Ex65-R parallel ...

```
## This does same, but improvises the function
## in the middle of the syntax.  Sometimes, this
    keeps a script
## cleaner because there are not so many functions
    floating about.
clusterCall( cl, function() Sys.info()[c("nodename"
    ,"machine")])


## Make the return invisible.  Catch the
## return value, which is a list
res1 <- clusterCall(cl, function() {
    a <- Sys.info()
    b <- date()
    invisible(list(a, b))
})
```

Ex65-R parallel ...

```
res1[[1]]
res1[[2]]

## Why do I keep asking the nodes who they are?
## Sometimes we have errors or slowness that is due
     to
## a flaw in MPI or the cluster, and it really
    helps to know
## which nodes are being contacted, and when.

## The key quality of clusterCall is that a
    function, and
## maybe the Arguments,
```

Ex65-R parallel ...

```
## parApply example.  Lets get the mean, and sum of
     each column.

myNorms ← matrix(rnorm(10000), ncol = 100)

myFn ← function(v){
     s ←Sys.info()[c("nodename")]
     ms ← sum(v)
     list(s, mysum = sum(v), mymean = mean(v))
}

system.time(
mypapply ← parApply(cl, myNorms, 2,  myFn)
)

## A chore like that might be faster if we run on
```

Ex65-R parallel ...

```
## just one node

system.time(
myonenode ← list(Sys.info(),
                 ms = colSums(myNorms),
                 mymean = apply(myNorms, 2, mean)
                 )
)


mypapply[[55]]
## Output is organized differently
myonenode$mymean[55]
```

Ex65-R parallel ...

```
## The output from parSapply is simplified in an
    understandable
## way
system.time(
    mysapply <- parSapply(cl, myNorms, myFn)
)



## Here is another example of parSapply copied from
    the vignette that
## was distributed with the original parallel
    package in R.  I do not
## understand what this is doing, maybe one of you
    can explain it to
## me.
```

Ex65-R parallel ...

```
library(spatial)
R ← 1000

tget ← function(z, r = 3.5) sum(dist(cbind(z$x, z$
    y)) < r)

x ← seq(0, 1, 0.1)
## pj: Don't know why following is able to find "
    towns.dat"
towns ← ppinit("towns.dat")
## The Strauss function can be executed for various
     values of x
tget(Strauss(69, c = x[1], r = 3.5))
tget(Strauss(69, c = x[4], r = 3.5))

## Will repeat that calculation 100 times,
```

Ex65-R parallel ...

```
## for a given value of the parameter c in tget.
run3 ← function(x) {
    library(spatial)
    towns ← ppinit("towns.dat") # has side effects
    mean(replicate(R, tget(Strauss(69, c = x, r = 3
        .5))))
}

clusterExport(cl, c("R", "towns", "tget"))
res ← c(0, parSapply(cl, x[-1], run3)) # 10 tasks
res

## Don't need to do this because I had the .last
    function
## above
## stopCluster(cl)
```

Ex65-R parallel ...

```
## Rmpi::mpi.quit()

## Sometimes, we want to stop the cluster and
    continue
## with calculations on one noode. We'd get rid of
    the
## .last function above, then we'd do this.

stopCluster(cl)

print(paste("I'm not dead yet, says your R session"
    ))

print(sum(rnorm(1000)))
```

Ex65-R parallel ...

```
## the next does kill the session entirely,
    however.
Rmpi::mpi.quit()
```

R parallel package

R parallel incorporates SNOW concepts and functions

Because parallel is in the R distribution, it should be considered a top-priority learning objective

Will Worry about Replication of Random Streams In Next Lecture

Each of the parallel packages has its own idioms to set the random number seed on the separate nodes.

So far, these packages aim at replication of a complete batch of simulations across a lot of compute nodes.

I will need to explain L'Ecuyer "many separate streams" and the SPRNG approaches, along with ways to replicate them.

My suggestion is "Portable Parallel Seeds" in Ex81.