Scaling R options

R on Comet strengths

- **Some examples what R is especially good for that you might want to do on Comet:**
- **Sampling/bootstrap methods,**
- **Data Wrangling,**
- **Particular Statistical procedures that you won't find implemented anywhere else, e.g.**
  - Multiple Imputation methods,
  - Instrument Variable (2 stage) Regression
  - Matching observational data subjects

R on Comet

At Unix prompt:

- *module load R*
- *R (for interactive R session)*

or

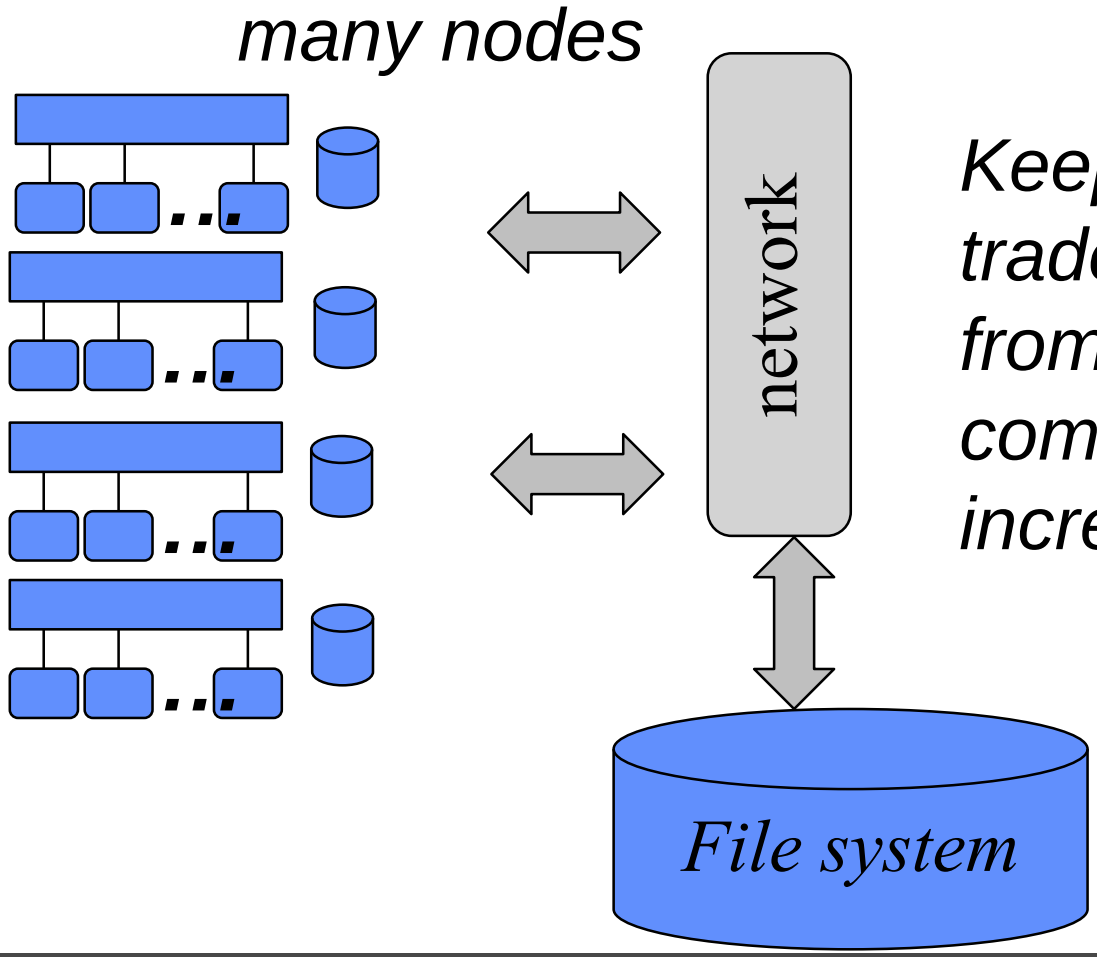
- *Rscript script-name (for batch)*

Scaling, practically

- **Scaling (with or without more data):**
  - more complex analysis (ie optimizations)
  - more sampling (ie more trees in Random Forest)

- **Sometimes easy to parallelize (like with sampling),**
- **Sometimes too much communication between parts (matrix inversion)**

On Distributed Computing



Keep in mind:
trade offs between gains from distributed computing vs loss from increased communication

Scaling In a nutshell

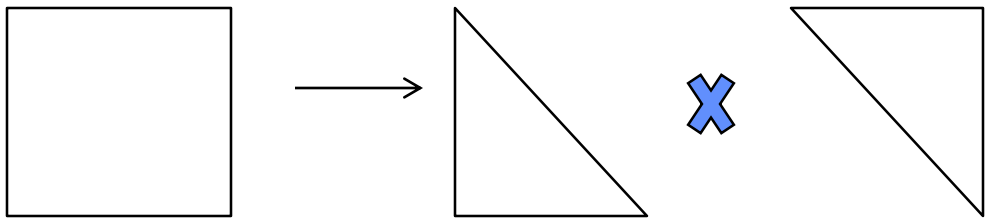
- **R takes advantage of math libraries for vector operations**
- **R packages provide multicore, multinode (snow), or map/reduce (RHadoop) options**
- **However, model implementations not necessarily built to use parallel backends**
  - Some models more amenable to parallel versions

Consider Regression Computations

- **Linear Model:** $Y = X * B$

  where Y=outcomes , X=data matrix

- **Algebraically, we could**:
  - take "inverse" $of\ X * Y = B$ (time consuming)
  - use derivatives to search for solutions (very general)
- **Or, better:**
  - QR decomposition of *X* into triangular matrices (easier to solve but more memory)



Consider Regression models in R

- **Related Models and Functions :**

  lm() #Linear Model

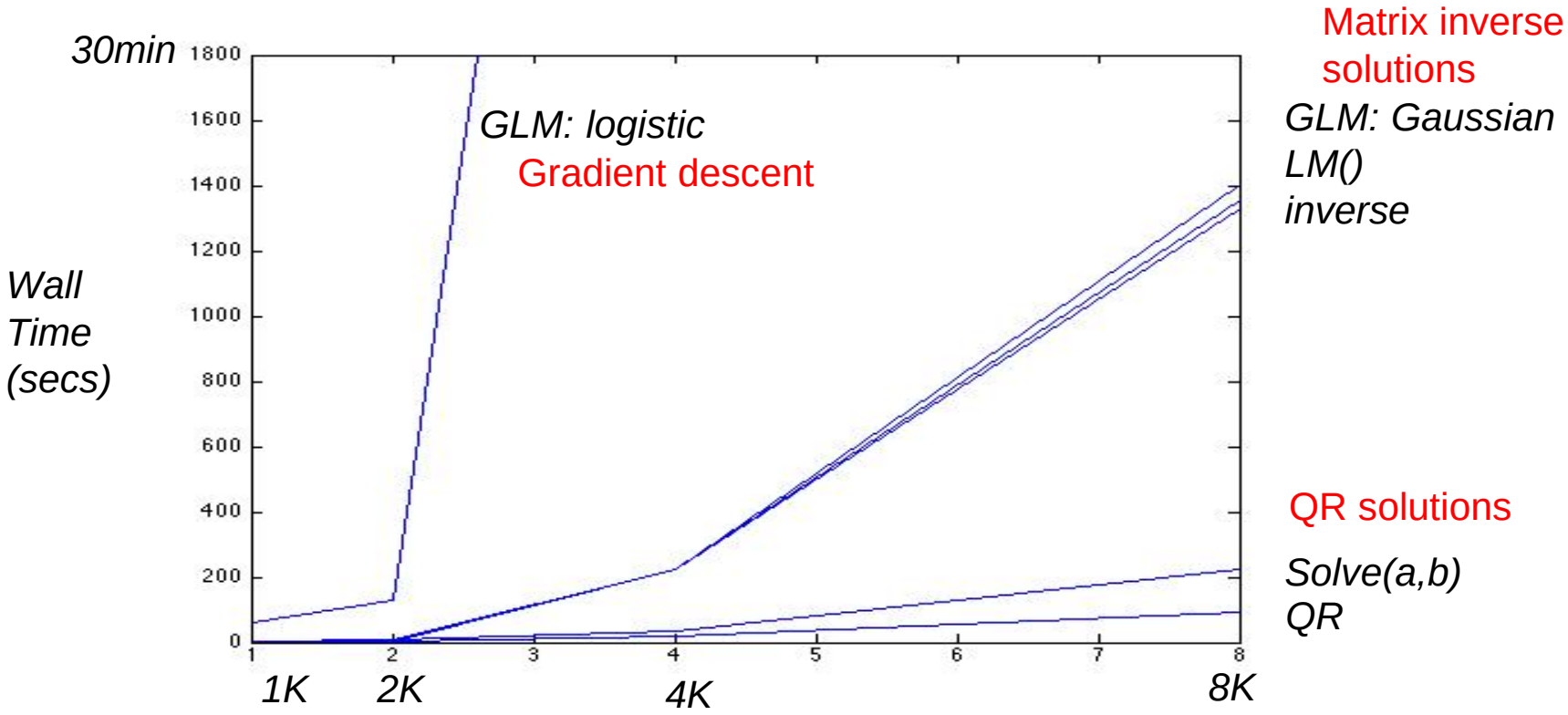
  glm() #Generalized Linear Model
  (logistic regression, etc)

  aov() #Analysis of Variance
  (returns ANOVA table of F-scores)

All these work on system of equations

Solving Linear Systems Performance with R, 1 compute node

R:
glm(Y~X,family=gaussian) #gaussn regrssn (like lm)
glm(Y~X,family=binomial) # logistic regrssn (Y=0 or 1)



R multicore

- **Intel Math Kernel Libraries provides fast operations for vector operations**
- **Uses threads across cpu cores to pass data & commands**

R multicore

- **Run loop iterations on separate cores**

%dopar% puts loops across cores, (loops are independent) %do% runs it serially

allocate workers

```
install.packages(doMC)
library(doMC)
registerDoMC(cores=24)
getDoParWorkers()

results = foreach(i=1:24,.combine=rbind) %dopar%
  {   … your code here

  return(  a variable or object )
})
```

returned items 'combined' into list by default

specify to combine results into array with row bind

R multinode: parallel backend

- **Run loop iterations on separate nodes**

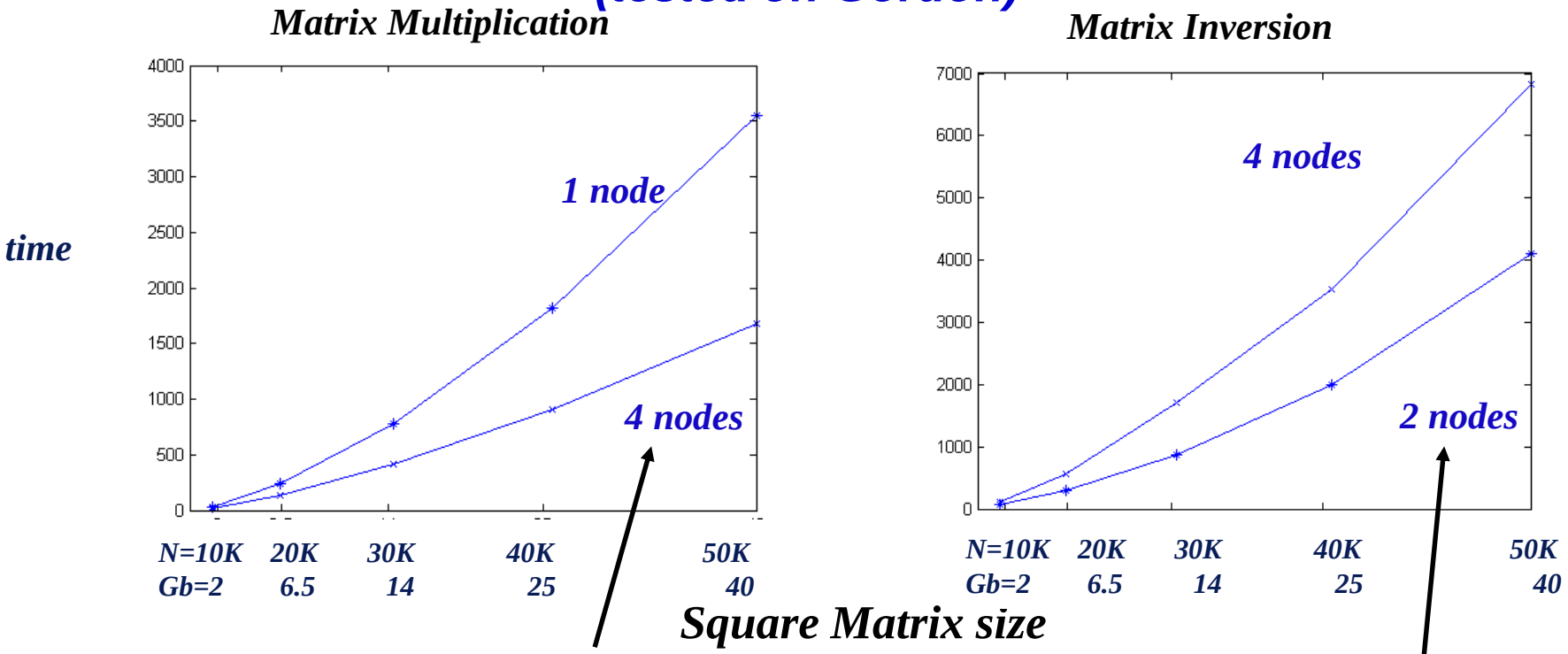
```
install.packages('doSNOW')
library('doSNOW')
…
cl <- makeCluster( mpi.universe.size()-1, type='MPI' )
clusterExport(cl,c('data'))
registerDoSNOW(cl)

results = foreach(i=1:47,.combine=rbind) %dopar%
  {   … your code here


      return(  a variable or object )
})
stopCluster(cl)
mpi.exit()
```

allocate cluster as parallel backend

%dopar% puts loops across cores and nodes

Multiple Compute Nodes not always help

(tested on Gordon)



multinodes: more nodes is less time for multiplication,

less nodes is better for inversion

Another Parallel option:

- Serially packing R jobs onto cores

1. batch job and calls MPI
2. MPI executes a Perl script on each core
3. Perl script gets cpu-id and passes it to R
4. R uses cpu-id to process some particular input

Normal batch job info

mpirun the 'bundler' perl script on 24 nodes

train100@comet-ln3:~/Rtrain/Rpacking_serial

```
[train100@comet-ln3 Rpacking_serial]$ more comet_sbatch_serial_packed
#!/bin/bash
# ----------------------------
# slurm script for a batch job on comet
# to run a task on individual cores
# ----------------------------
#SBATCH --job-name="serial-pack"
#SBATCH --output="serial-pack.%j.%N.out"
#SBATCH --partition=shared
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=4
#SBATCH --export=ALL
#SBATCH -t 00:30:00

bash

#Generate a hostfile from the slurm node list
export SLURM_NODEFILE=`generate_pbs_nodefile`

#Run job from working directory or do something like this:
#cd /oasis/scratch/comet/$USER/mydirectory/

module load R

#mpirun executes the bundler perl script on each core, the bundler will launch
#  R with arguments to indicate which file to process
# NOTE the argument to bundler is the number of R tasks to execute
#         it should be = ntask-per-node X nodes   (but >= would work)
mpirun_rsh  -hostfile $SLURM_NODEFILE -np 24 ./bundler.pl 48
[train100@comet-ln3 Rpacking_serial]$
```

the 'bundler' script

The argument 'was 48 tasks to do

Get current cpu id and number of processes

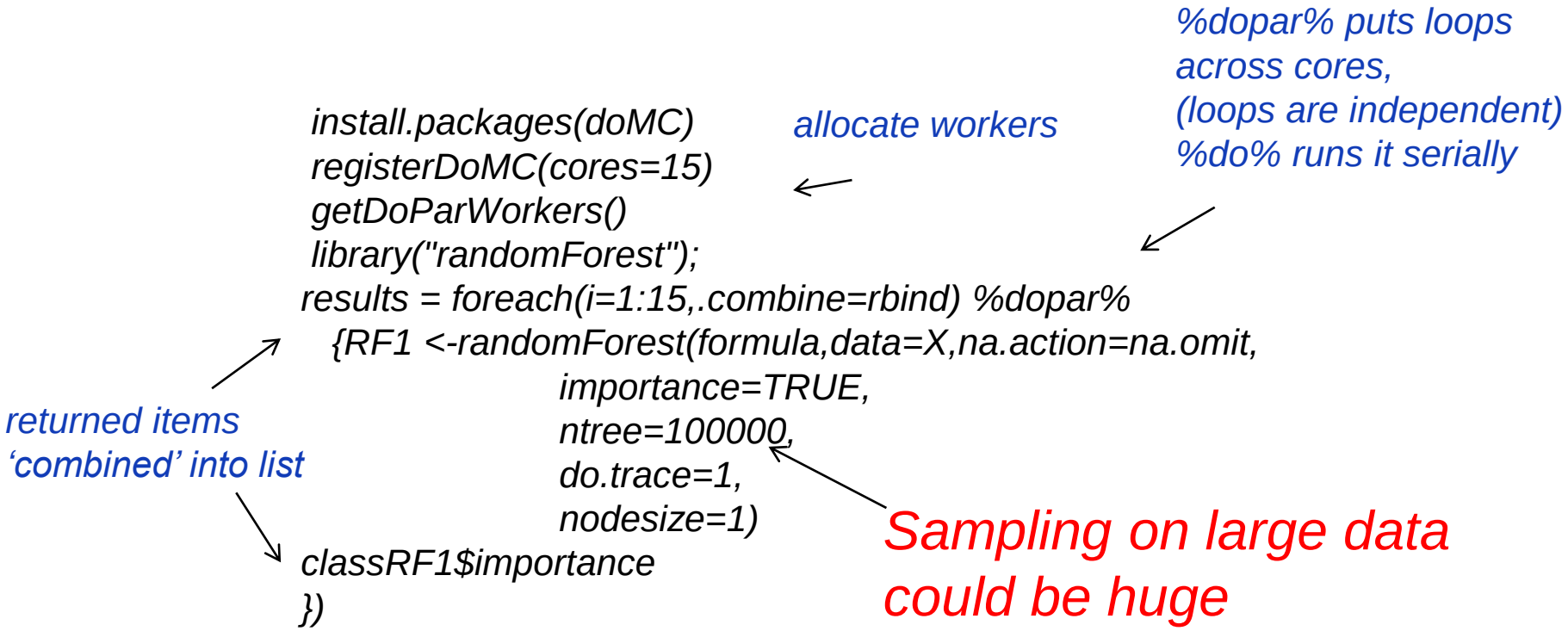
Depending on the cpu-id, execute R and pass an input file

train100@comet-06-55:Rpacking_serial

```
#!/usr/bin/perl
use strict;
use warnings;
# ------------------------------------------------
# Perl 'wrapper' script to help launch tasks (ie a program)
# This script is called by:
#      mpirun_rsh -np N ./bundler.pl num_tasks
#  which will put an instance of this script on each core
# ------------------------------------------------
my $ntasks            = $ARGV[0];
my ($myid, $numprocs) = split(/\s+/, `./getid`); #getid will return cpu id and t
otal processor in MPIjob

# ------------------------------------
# using getid, launch only a task in turn
#   pass use the task id as an argument
#  (note there can be more tasks than processors)
# ------------------------------------
for (my $i=0; $i<$ntasks; $i++) {
    if($myid == $i % $numprocs) {
        print "Perl Bundler: $myid -th Task\n";
        my $filei_2use=$i+1;
        system("/opt/R/bin/Rscript R_LinModel_randXY.R $filei_2use");
    }
}
~
~
                                                          22,2-9        All
```

Packing Serial with large Random Forest job

- **Option 1: Run separate trees on separate cores**



Packing Serial with large Random Forest job

- **Option 2: split tree sampling to make it embarrasingly parallel**

  ie run R script on separate cores and average results

  Can speed up processing without losing interesting variable combinations

R and Map/Reduce

- Map/Reduce streaming can pipe Input/Output through R
- R can make direct map/reduce calls with R-Hadoop interface (from Revolution Analytics)

Hadoop Map/Reduce Interfaces with R

(slides from G.Lockwood SDSC)

- **R Streaming (simplest) or Hadoop API**

- **E.g. streaming Hadoop is like piping input/output**

```
cat input | Rscript mapper.R | sort | Rscript reducer.R > output
```

You provide these two scripts; Hadoop does the rest

Paradigmatic Example: Word Counting

How would you count all the words in Moby Dick?

> *Call me Ishmael. Some years ago - never mind how long precisely - having little or no money in my purse, and nothing particular to interest me on shore, I thought I would sail about a little and see the watery part of the world. It is a way I have of driving off the spleen and regulating the circulation. ….*

How could you count all the words in all web pages? *(assume the data is spread out over many nodes)*

Use Map/Reduce, take computation to nodes

Wordcount: Hadoop streaming mapper

Emit key-value pairs (‘cat’ is ‘concatenate and print’)

Split line Into words

Use words as keys

```
emit.keyval <- function(key, value) {
    cat(key, '\t', value, '\n', sep='')
}

stdin <- file('stdin', open='r')
while ( length(line <- readLines(stdin, n=1)) > 0 ) {
    line <- gsub('(^\\s+|\\s+$)', '', line)
    keys <- unlist(strsplit(line, split='\\s+'))
    value <- 1
    lapply(keys, FUN=emit.keyval, value=value)
}
close(stdin)
```

Example from Glen Lockwood, SDSC

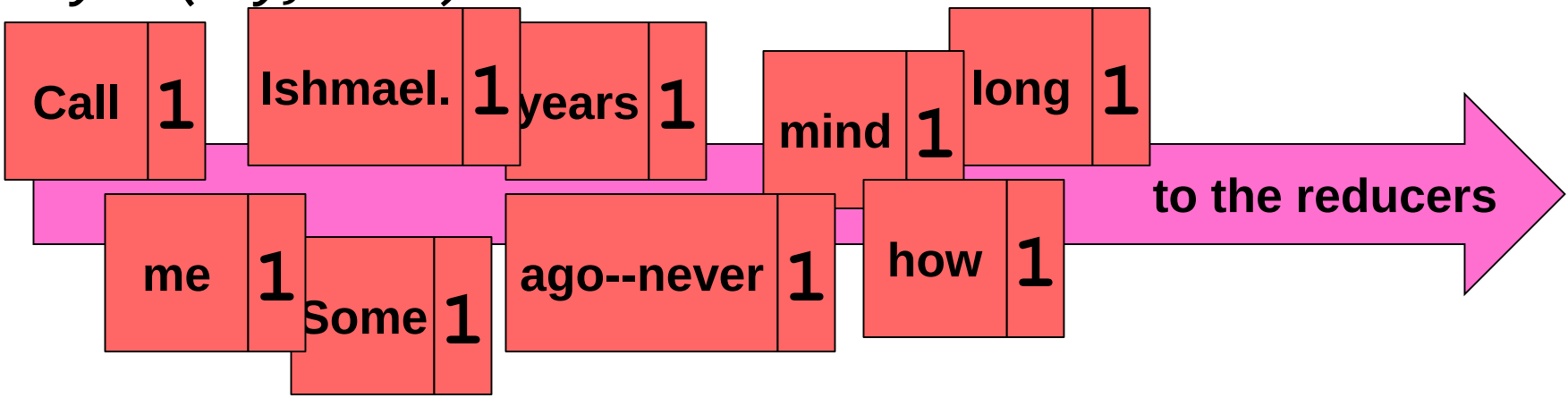
What One Mapper Does

line = Call me Ishmael. Some years ago—never mind how long

keys =

Call	me	Ishmael.	Some	years	ago--never	mind	how	long

emit.keyval(key,value) ...



Reducer Loop

- **If this key is the same as the previous key,**
  - add this key's value to our running total.
- **Otherwise,**
  - print out the previous key's name and the running total,
  - reset our running total to 0,
  - add this key's value to the running total, and
  - "this key" is now considered the "previous key"

Wordcount: Streaming Reducer (1/2)

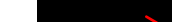
Get key, Value

Add up values

```
last_key <- ""
running_total <- 0

stdin <- file('stdin', open='r')
while ( length(line <- readLines(stdin,n=1)) > 0 ) {
    line <- gsub('(^\\s+)|(\\s+$)', '', line)
    keyvalue <- unlist(strsplit(line, split='\t', fixed=TRUE))
    this_key <- keyvalue[[1]]
    value <- as.numeric(keyvalue[[2]])

    if ( last_key == this_key ) {
        running_total <- running_total + value
    }
    else {
        (to be continued...)
```

Wordcount: Streaming Reducer (2/2)

For each new key, emit <key, sum>



```
    else {
        if ( last_key != "" ) {
            cat(
paste(last_key,'\t',running_total,'\n',sep='') )
        }
        running_total <- value
        last_key <- this_key
    }
}

if ( last_key == this_key ) {
    cat( paste(last_key,'\t',running_total,'\n',sep='') )
}

close(stdin)
```

Testing Mappers/Reducers

- Debugging Hadoop is not fun

```
$ head -n100 pg2701.txt |
  ./wordcount-streaming-mapper.R | sort |
  ./wordcount-streaming-reducer.R
...
with  5
word, 1
world.      1
www.gutenberg.org 1
you   3
You   1
```

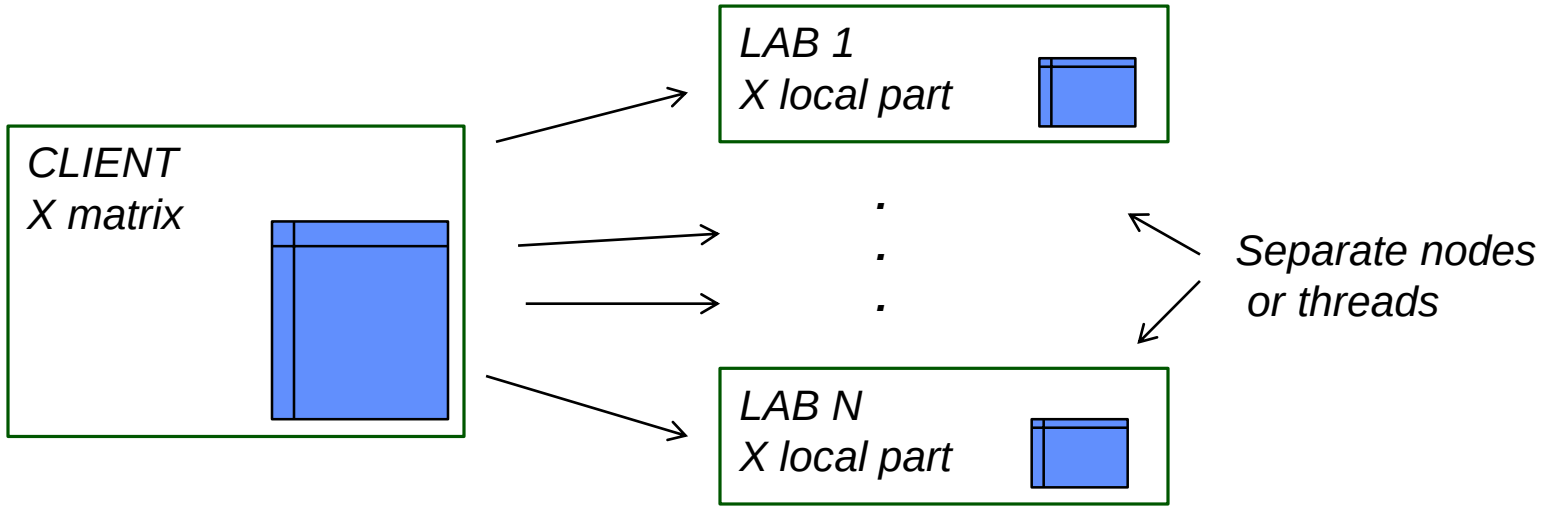
Launching Hadoop Streaming

```
$ hadoop dfs -copyFromLocal ./pg2701.txt mobydick.txt

$ hadoop jar \
 /opt/hadoop/contrib/streaming/hadoop-streaming-1.0.3.jar \
    -D mapred.reduce.tasks=2 \
    -mapper "Rscript $PWD/wordcount-streaming-mapper.R" \
    -reducer "Rscript $PWD/wordcount-streaming-reducer.R" \
    -input mobydick.txt \
    -output output

$ hadoop dfs -cat output/part-* > ./output.txt
```

Ask XSEDE support for latest Hadoop scripts

Matlab quickview

- **Distributed Toolbox:**
  - allocate distributed matrices using ‘spmd’ code
  - MPI or threads under the hood
  - You decide data/task set up



SPMD Matlab – general scheme

```
%Matlab code using SPMD toolbox, where X is a matrix
…
spmd
        fprintf('in 1st spmd %i \n',labindex);
        Xsd=codistributed(X, codistributor('1d', 1)); %distrib rows
        XsdLoc=getLocalPart(Xsd);
end
% …. Other code…
 spmd
        local-result ~ calculations on XsdLoc
 end;
gather local-result
% … rest of code….
```

Distribute Matrix X and process local part

It stays distributed in memory for further processing within SPMD block

- **Some future stuff**

Other R packages:

- Rspark - R interface to Spark
- pdbR - higher level over R-MPI, distributed matrix support and other
- HiPLAR - GPU and multicore for linear algebra
- Rgputools – GPU support
- R openMP , better data mgt than dopar, parallel (mclapply)
- Ff, bigmemory; Revolution Scale R (commercial) – map data to files

- **R install.packages on Comet**

- **Need to specify url and yes to personal library;**
- **If compiling is required you might get an error**

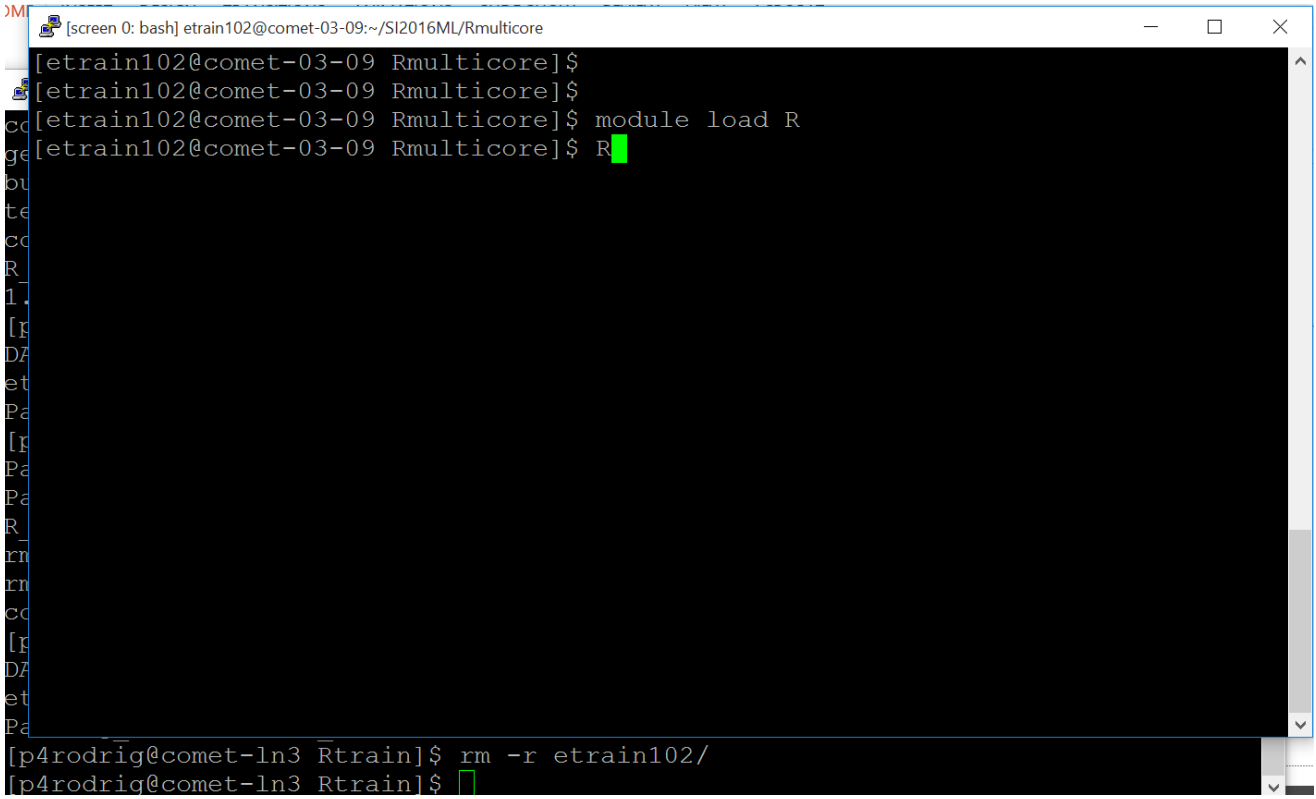
Exercise: R multicore

Login into Comet,

cd to Rtrain/Rmulticore

module load R

R



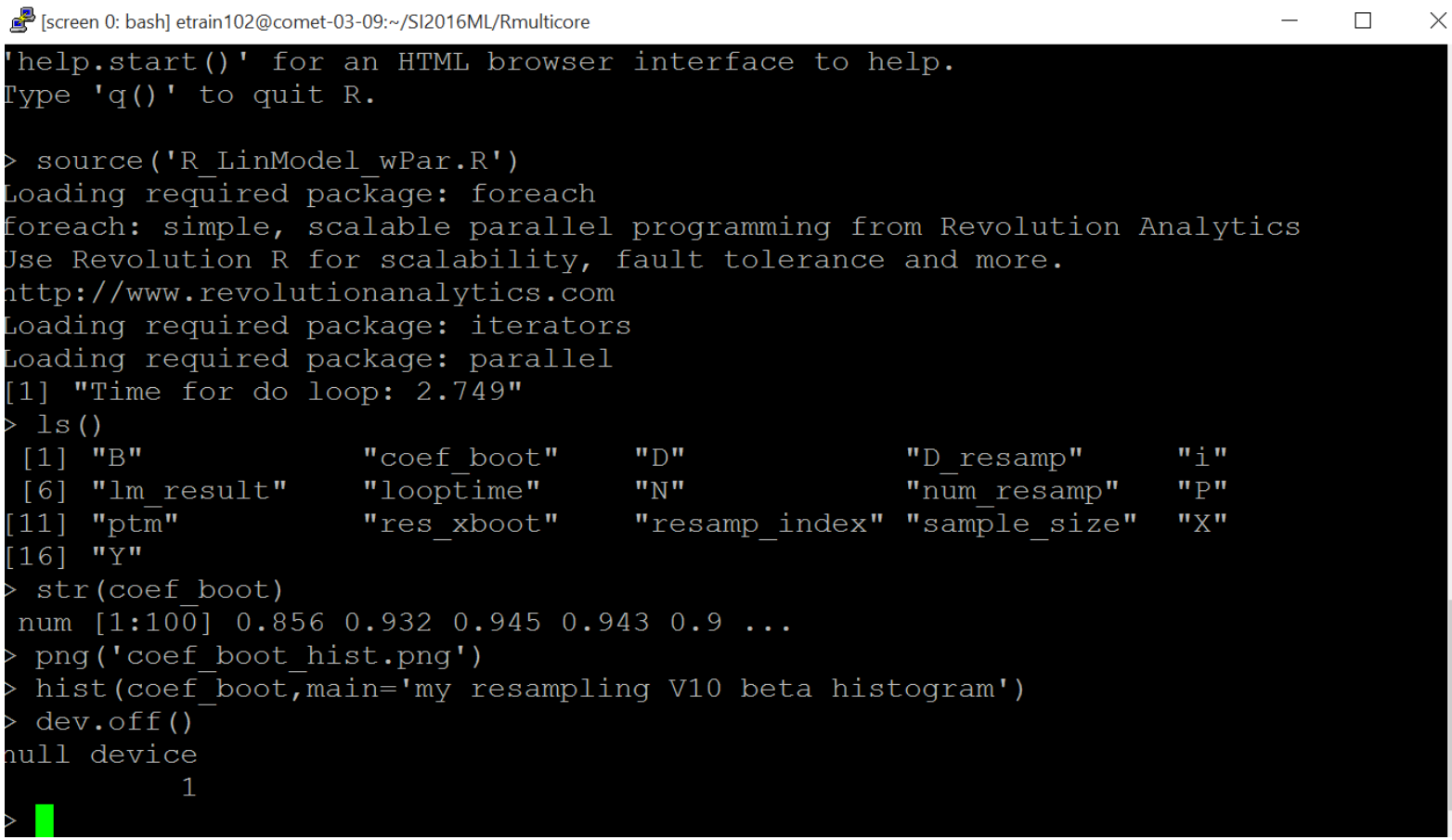
Exercise: R multicore

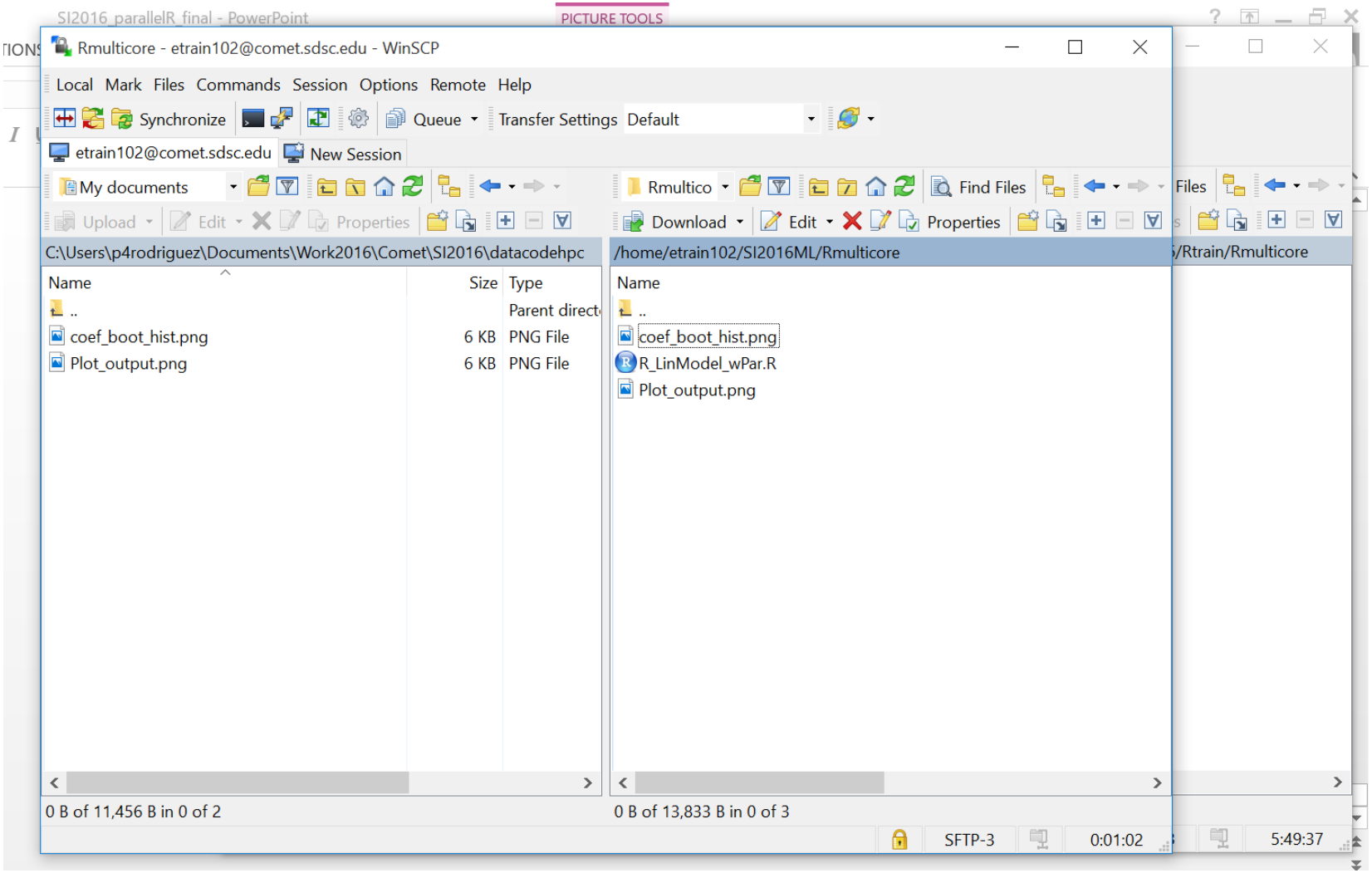
Login into Comet,

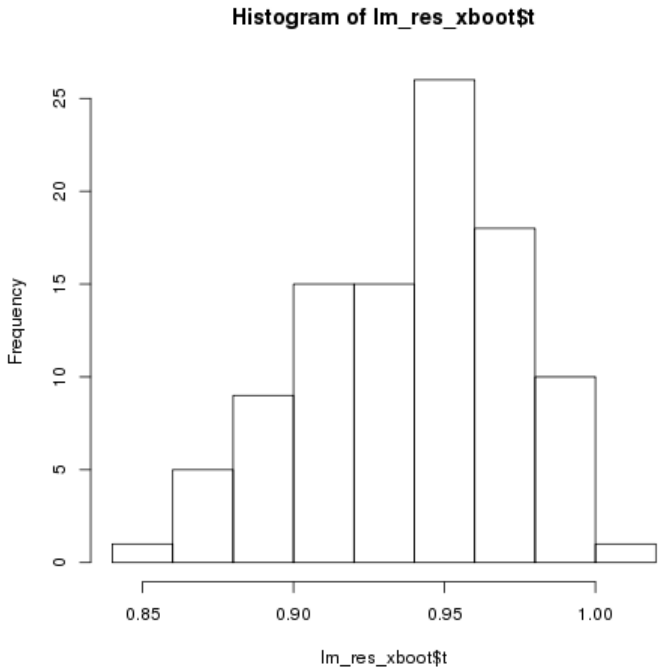
cd to Rtrain/Rmulticore

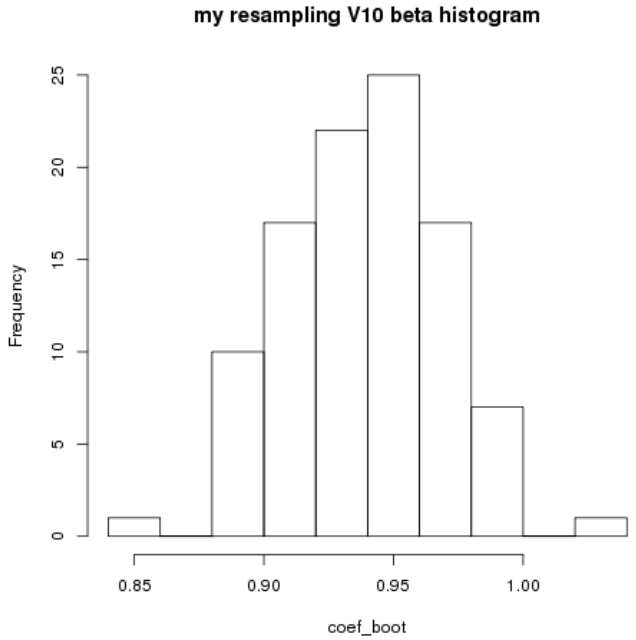
```
#Step 1    generate random data
#Step 2, setup sampling #Step 3 set up parallel environment

#Step 3  - -------------Parallel Options ----------

library('doMC');
registerDoMC(23)      # < ---- number of cores here
```

```
# Now run a bootstrap sample of regression

coef_boot=vector('numeric',num_resamp)  #an empty vector to save
results

# Use 'do' or 'dopar' to test single/multicore execution
ptm <- proc.time()
res_xboot    = foreach(i = 1:num_resamp) %do% {
        resamp_index   = sample(1:N, size=sample_size, replace=T)
        D_resamp       = D[resamp_index,]                #pick out rows
        lm_result      = lm(V1~.,data=D_resamp)          #rerun model
        coef_boot[i]   = lm_result$coefficients['V10']   #save this beta
}

looptime=proc.time() - ptm        #gather timing info
print(paste('Time for do loop:',signif(looptime[3],5)))
```


[screen 0: bash] etrain102@comet-03-09:~/SI2016ML/Rmulticore
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
> source('R_LinModel_wPar.R')
Loading required package: foreach
foreach: simple, scalable parallel programming from Revolution Analytics
Use Revolution R for scalability, fault tolerance and more.
http://www.revolutionanalytics.com
Loading required package: iterators
Loading required package: parallel
[1] "Time for do loop: 2.749"
> ls()
[1] "B" "coef_boot" "D" "D_resamp" "i"
[6] "lm_result" "looptime" "N" "num_resamp" "P"
[11] "ptm" "res_xboot" "resamp_index" "sample_size" "X"
[16] "Y"
> str(coef_boot)
num [1:100] 0.856 0.932 0.945 0.943 0.9 ...
> png('coef_boot_hist.png')
> hist(coef_boot,main='my resampling V10 beta histogram')
> dev.off()
null device
1
>


SI2016_parallelR_final - PowerPoint
PICTURE TOOLS
Rmulticore - etrain102@comet.sdsc.edu - WinSCP
Local Mark Files Commands Session Options Remote Help
Synchronize
Queue
Transfer Settings Default
etrain102@comet.sdsc.edu
New Session
My documents
Upload
Edit
Properties
Rmultico
Download
Edit
Properties
Find Files
C:\Users\p4rodriguez\Documents\Work2016\Comet\SI2016\datacodehpc
/home/etrain102/SI2016ML/Rmulticore
Name
Size
Type
..
Parent direct
coef_boot_hist.png
6 KB
PNG File
Plot_output.png
6 KB
PNG File
Name
..
coef_boot_hist.png
R_LinModel_wPar.R
Plot_output.png
0 B of 11,456 B in 0 of 2
0 B of 13,833 B in 0 of 3
SFTP-3
0:01:02
5:49:37


Histogram of lm_res_xboot$t
Frequency
0
5
10
15
20
25
0.85
0.90
0.95
1.00
lm_res_xboot$t


my resampling V10 beta histogram
Frequency
0
5
10
15
20
25
0.85
0.90
0.95
1.00
coef_boot

Set to 1

And you can rerun and compare times

/oasis/scratch/comet/p4rodrig/temp_project/SI2016/Rtrain/Rmulticore/R_LinModel_wPar.R - p4rodrig@comet.sdsc.edu - E...

Encoding Color

```
if (0) {   #if 0, then this section is blocked out,
# -----------------------
#Now try using the R Boot library
#you might need to install.packages('boot')
#  parallel = c("no", "multicore", "snow")  is the option we will play with
# -----------------------
library(boot)

#'boot' will call this and passes the data and indices
get_lm_res<-function(D_resamp,indices) {
       lm_resamp = lm(V1~.,data=D_resamp[indices,])       #rerun model
       return(lm_resamp$coefficient['V10'])
     }

# -----------------
#call 'boot' function, a list of results from get_lm_res is returned
#the returned object has lots of boot information, $t is returned items from funtction
# -----------------
ptm          =proc.time()
lm_res_xboot =boot(D, get_lm_res, R = num_resamp,parallel = "multicore")
looptime     =proc.time() - ptm        #gather timing info
print(paste('Time for Bootstrap loop:',signif(looptime[3],5)))


#A sample of how to plot output to a file
png('Plot_output.png')
hist(lm_res_xboot$t)
dev.off()
# -----------------------
```

Line: 74/76 Column: 767 Encoding: 1252 (ANSI - L