

22S:166 Computing in Statistics

Parallel computing in R using `snow`

Lect 27
Nov.30, 2009

Kate Cowles
374 SH, 335-0727
kcowles@stat.uiowa.edu

Main classes of functions in `snow`

- `makeCluster` and `stopCluster`: start and stop a cluster of slave processes
- `clusterEvalQ`: evaluate a literal expression on each slave and return results
 - useful for having each node run functions using arguments in its own memory
- `clusterCall`: runs a specified function on all slave nodes with identical arguments sent from master to all nodes
- `clusterApply` and functions built on it: runs a specified function with arguments sent from master and split up among the nodes

Running parallel jobs in R in the Linux lab

- use `top` and `finger` to make sure that your target machines are not being used intensively by others
- use the `snow` (“Simple Network of Workstations”) package


```
> library(snow)
```
- master/slave design: master R process creates a cluster of slave processes that carries out computations and returns results to master

Starting a cluster

- `makeCluster` to start up a cluster of processes communicating with master
 - node you started R in will be master

```
# create an 8-process socket cluster
> cl <- makeSOCKcluster(c("localhost","localhost",
"localhost","localhost","1-lnx211.divms.uiowa.edu",
"1-lnx211.divms.uiowa.edu","1-lnx211.divms.uiowa.edu",
"1-lnx211.divms.uiowa.edu"))

# prompts for password for any machines I'm not already
# logged into
```

`clusterCall`: to get all slave processes to execute the same function with the same arguments

```
# call Sys.info function on all processes
> clusterCall(cl, function(cl) Sys.info() )

> do.call("rbind", clusterCall(cl,
  function(cl) Sys.info()["nodename"]))

      nodename
[1,] "1-lnx207.divms.uiowa.edu"
[2,] "1-lnx207.divms.uiowa.edu"
[3,] "1-lnx207.divms.uiowa.edu"
[4,] "1-lnx207.divms.uiowa.edu"
[5,] "1-lnx211.divms.uiowa.edu"
[6,] "1-lnx211.divms.uiowa.edu"
[7,] "1-lnx211.divms.uiowa.edu"
[8,] "1-lnx211.divms.uiowa.edu"
```

```
# calc summary statistics on each row of matrix using
# 8 processes on 2 machines

> system.time(parRapplyout1 <- parRapply(cl, mymat,summary))

      user  system elapsed
0.060   0.012   0.543

# calc summary statistics on each row of matrix using
# 4 processes on 1 machine

> system.time(parRapplyout2 <- parRapply(cl[1:4], mymat,summary)
      user  system elapsed
0.089   0.006   0.410
```

Various `parApply` functions to get processes to carry out same function on different arguments

```
> mymat <- matrix(rnorm(1000000), nrow=1000)

# calc summary statistics on each row of matrix using
# master process only

> system.time(applyout <- apply(mymat,1,summary))
      user  system elapsed
1.217   0.000   1.217

> applyout <- t(applyout)

> help(parRapply)
```

```
> is.matrix(parRapplyout1)
[1] FALSE

> is.vector(parRapplyout1)
[1] TRUE

> parCapplyoutM <- matrix( parCapplyout, ncol=6, byrow=T)

> parRapplyout1[1:12]
 [1] -3.189000 -0.617100  0.022220  0.012210  0.652900  3.00700
 [8] -0.615500 -0.006234  0.005136  0.660500  3.563000

> parRapplyout1M <- matrix(parRapplyout1, ncol=6, byrow=T )

> parRapplyout1M[1:5, ]

> applyout[1:5, ]

      Min. 1st Qu.  Median      Mean 3rd Qu.  Max.
[1,] -3.189 -0.6171  0.022220  0.012210  0.6529  3.007
[2,] -2.887 -0.6155 -0.006234  0.005136  0.6605  3.563
[3,] -2.833 -0.7610 -0.090580 -0.078740  0.5933  3.529
[4,] -3.102 -0.6295 -0.051070 -0.016630  0.6048  3.774
[5,] -3.535 -0.5880  0.099620  0.053010  0.7009  2.815
```

Parallel random number generation

- If you don't initialize a separate random number stream for each node, it is possible to get identical random number streams on all cluster nodes.
- two parallel random number generators are available in snow, based on two packages: `rsprng` and `rlecuyer`

```
# seed independent uniform random number streams for snow clust
# L'Ecuyer's generator
> clusterSetupRNGstream(c1, seed=rep(12345,6))

> do.call("rbind", clusterCall( c1, rnorm, 5 ))
      [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 0.68567690 -1.40176821 0.4644450 0.06166349 0.3081519
[2,] 1.59218721 0.98941360 -0.2260639 1.03891974 0.5725584
[3,] -0.16213049 0.08560169 1.9705866 -1.10948803 1.7657923
[4,] -0.57209565 1.79293554 0.9375416 -0.52962240 0.2293538
[5,] 0.91600693 -0.23380300 -0.1648971 0.09438836 -1.1816592
[6,] 0.05928533 0.01071665 -0.8184931 0.25428280 1.4123132
[7,] 2.19466922 0.48531364 0.5833869 -1.18760882 -0.3608804
[8,] -0.90408653 -0.32904130 0.5819193 0.19522605 1.1392602
```

11

```
> clusterCall( c1, getwd )
[[1]]
[1] "/mnt/nfs/netapp1/fs2/kcowles/166/parallel/dir1"

[[2]]
[1] "/mnt/nfs/netapp1/fs2/kcowles/166/parallel/dir2"

[[3]]
[1] "/mnt/nfs/netapp1/fs2/kcowles/166/parallel/dir3"

[[4]]
[1] "/mnt/nfs/netapp1/fs2/kcowles/166/parallel/dir4"

# clusterEvalQ evaluates a literal expression on each node
> clusterEvalQ( c1, load("y.Rdata") )
[[1]]
[1] "y"

[[2]]
[1] "y"

[[3]]
[1] "y"

[[4]]
[1] "y"
```

Getting individual processes to operate on different data

```
> stopCluster(c1)      # I want a smaller cluster for
                       # this example, so shut down 8-slave
                       # cluster and make a smaller one

> c1 <- makeSOCKcluster(
      c("localhost","localhost","localhost","localhost"))

> clusterSetupRNGstream(c1, seed=rep(12345,6))

# make the working directory different for each slave process
# set up a vector of arguments for clusterApply

> dirnames <- c("dir1","dir2","dir3","dir4")

> clusterApply(c1, dirnames, setwd )
[[1]]
[1] "/mnt/nfs/netapp1/fs2/kcowles/166/parallel"

[[2]]
[1] "/mnt/nfs/netapp1/fs2/kcowles/166/parallel"

[[3]]
[1] "/mnt/nfs/netapp1/fs2/kcowles/166/parallel"

[[4]]
[1] "/mnt/nfs/netapp1/fs2/kcowles/166/parallel"
```

12

```
> clusterEvalQ( c1, print(y) ) # wouldn't work with clusterCall
[[1]]
[1] -0.3337495 -0.1518703 0.6569270 -0.7677969 -1.3420129

[[2]]
[1] -0.5946356 1.0493249 2.1990838 -0.9631471 0.1113470 -1.1

[[3]]
[1] 0.1571528 0.8139019 0.9223264 0.2692795

[[4]]
[1] 0.86589506 1.27566060 -2.01340611 0.23612130 -0.5668075
[7] -1.13304140 -0.64429002 0.02611008 -0.75593859

> clusterEvalQ( c1, meany <- mean(y) )
[[1]]
[1] -0.3877005

[[2]]
[1] 0.1232192

[[3]]
[1] 0.5406652

[[4]]
[1] -0.2766687
```