Package ‘snow’

February 20, 2015

Title  Simple Network of Workstations
Version  0.3-13
Author  Luke Tierney, A. J. Rossini, Na Li, H. Sevcikova
Description  Support for simple parallel computing in R.
Maintainer  Luke Tierney <luke@stat.uiowa.edu>
Suggests  Rmpi,rpvm,rlecuyer,rsprng,nws
License  GPL
Depends  R (>= 2.12.1), utils
NeedsCompilation  no
Repository  CRAN
Date/Publication  2013-09-27 15:50:12

R topics documented:

snow-cluster .................................................................. 1
snow-parallel ................................................................. 3
snow-rand ........................................................................ 4
snow-startstop ................................................................. 5
snow-timing ................................................................. 7

Index

snow-cluster  Cluster-Level SNOW Functions

Description

Functions for computing on a SNOW cluster.
Usage

- `clusterSplit(cl, seq)`
- `clusterCall(cl, fun, ...)`
- `clusterApply(cl, x, fun, ...)`
- `clusterApplyLB(cl, x, fun, ...)`
- `clusterEvalQ(cl, expr)`
- `clusterExport(cl, list, envir = .GlobalEnv)`
- `clusterMap(cl, fun, ..., MoreArgs = NULL, RECYLE = TRUE)`

Arguments

- `cl` (cluster object)
- `fun` (function or character string naming a function)
- `expr` (expression to evaluate)
- `seq` (vector to split)
- `list` (character vector of variables to export)
- `envir` (environment from which to export variables)
- `x` (array)
- `...` (additional arguments to pass to standard function)
- `MoreArgs` (additional argument for `fun`)
- `RECYLE` (logical; if true, shorter arguments are recycled)

Details

These are the basic functions for computing on a cluster. All evaluations on the slave nodes are done using tryCatch. Currently an error is signaled on the master if any one of the nodes produces an error. More sophisticated approaches will be considered in the future.

- `clusterCall` calls a function `fun` with identical arguments ... on each node in the cluster `cl` and returns a list of the results.

- `clusterEvalQ` evaluates a literal expression on each cluster node. It is a cluster version of evalq, and is a convenience function defined in terms of `clusterCall`.

- `clusterApply` calls `fun` on the first cluster node with arguments `seq[[1]]` and ..., on the second node with `seq[[2]]` and ..., and so on. If the length of `seq` is greater than the number of nodes in the cluster, then cluster nodes are recycled. A list of the results is returned; the length of the result list will equal the length of `seq`.

- `clusterApplyLB` is a load balancing version of `clusterApply`. If the length `p` of `seq` is greater than the number of cluster nodes `n`, then the first `n` jobs are placed in order on the `n` nodes. When the first job completes, the next job is placed on the available node; this continues until all jobs are complete. Using `clusterApplyLB` can result in better cluster utilization than using `clusterApply`. However, increased communication can reduce performance. Furthermore, the node that executes a particular job is nondeterministic, which can complicate ensuring reproducibility in simulations.

- `clusterMap` is a multi-argument version of `clusterApply`, analogous to `mapply`. If `RECYLE` is true, shorter arguments are recycled; otherwise, the result length is the length of the shortest argument. Cluster nodes are recycled if the length of the result is greater than the number of nodes.
clusterExport assigns the values on the master of the variables named in list to variables of the same names in the global environments of each node. The environment on the master from which variables are exported defaults to the global environment.

clusterSplit splits seq into one consecutive piece for each cluster and returns the result as a list with length equal to the number of cluster nodes. Currently the pieces are chosen to be close to equal in length. Future releases may attempt to use relative performance information about nodes to choose split proportional to performance.

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

Examples

```r
## Not run:
cl <- makeSOCKcluster(c("localhost","localhost"))
clusterApply(cl, 1:2, get("+"), 3)
clusterEvalQ(cl, library(boot))
x<-1
clusterExport(cl, "x")
clusterCall(cl, function(y) x + y, 2)
```

## End(Not run)

snow-parallel  Higher Level SNOW Functions

Description

Parallel versions of apply and related functions.

Usage

```r
parLapply(cl, x, fun, ...)
parSapply(cl, X, FUN, ..., simplify = TRUE, USE.NAMES = TRUE)
parApply(cl, X, MARGIN, FUN, ...)
parRapply(cl, x, fun, ...)
parCapply(cl, x, fun, ...)
parMM(cl, A, B)
```

Arguments

```r
cl    cluster object
fun   function or character string naming a function
X     array to be used
x     matrix to be used
```
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

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

Examples

## Not run:
c1 <- makeSOCKcluster(c("localhost","localhost"))
parSapply(c1, 1:20, get("+"), 3)

## End(Not run)
Arguments

cl  Cluster object.
type type="RNGstream" (default) initializes the L’Ecuyer’s RNG. type="SPRNG" initializes the SPRNG generator.
...  Arguments passed to the underlying function (see details below).
seed  Integer value (SPRNG) or a vector of six integer values (RNGstream) used as seed for the RNG.
prngkind  Character string naming generator type used with SPRNG.
para  Additional parameters for the generator.

Details

clusterSetupRNG calls (subject to its argument values) one of the other functions, passing arguments (cl, ...). If the "SPRNG" type is used, then the function clusterSetupSPRNG is called. If the "RNGstream" type is used, then the function clusterSetupRNGstream is called.

clusterSetupSPRNG loads the rsprng package and initializes separate streams on each node. For further details see the documentation of initNsprng. The generator on the master is not affected.

clusterSetupRNGstream loads the rlecuyer package, creates one stream per node and distributes the stream states to the nodes.

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

Examples

```r
# Not run:
clusterSetupSPRNG(cl)
clusterSetupSPRNG(cl, seed=1234)
clusterSetupRNG(cl, seed=rep(1,6))

# End(Not run)
```

Description

Functions to start and stop a SNOW cluster and to set default cluster options.

Usage

```r
makeCluster(spec, type = getClusterOption("type"), ...)
stopCluster(cl)

setDefaultClusterOptions(...)

makeSOCKcluster(names, ..., options = defaultClusterOptions)
```
makePVMcluster(count, ..., options = defaultClusterOptions)
makeMPIcluster(count, ..., options = defaultClusterOptions)
makeNWScluster(names, ..., options = defaultClusterOptions)
getMPIcluster()

Arguments

spec cluster specification
count number of nodes to create
names character vector of node names
options cluster options object
c1 cluster object
... cluster option specifications
type character; specifies cluster type.

Details

makeCluster starts a cluster of the specified or default type and returns a reference to the cluster. Supported cluster types are "SOCK", "PVM", "MPI", and "NWS". For "PVM" and "MPI" clusters the spec argument should be an integer specifying the number of slave nodes to create. For "SOCK" and "NWS" clusters spec should be a character vector naming the hosts on which slave nodes should be started; one node is started for each element in the vector. For "SOCK" and "NWS" clusters spec can also be an integer specifying the number of slaves nodes to create on the local machine.

For SOCK and NWS clusters the spec can also be a list of machine specifications, each a list of named option values. Such a list must include a character value named host specifying the name or address of the host to use. Any other option can be specified as well. For SOCK and NWS clusters this may be a more convenient alternative than inhomogeneous cluster startup procedure. The options rscript and snowlib are often useful; see the examples below.

stopCluster should be called to properly shut down the cluster before exiting R. If it is not called it may be necessary to use external means to ensure that all slave processes are shut down.

setDefaultClusterOptions can be used to specify alternate values for default cluster options. There are many options. The most useful ones are type and homogeneous. The default value of the type option is currently set to "MPI" if Rmpi is on the search path. Otherwise it is set to "PVM" if the rpvm package is available, to "MPI" if Rmpi is available but rpvm is not, and to "SOCK" if neither of these packages is found.

The homogeneous option should be set to FALSE to specify that the startup procedure for inhomogeneous clusters is to be used; this requires some additional configuration. The default setting is TRUE unless the environment variable R_SNOW_LIB is defined on the master host with a non-empty value.

The option outfile can be used to specify the file to which slave node output is to be directed. The default is /dev/null; during debugging of an installation it can be useful to set this to a proper file. On some systems setting outfile to "" or to /dev/tty will result in worker output being sent to the terminal running the master process.

The functions makeSOCKcluster, makePVMcluster, makeMPIcluster, and makeNWScluster can be used to start a cluster of the corresponding type.
In MPI configurations where process spawning is not available and something like mpirun is used to start a master and a set of slaves the corresponding cluster will have been pre-constructed and can be obtained with getMPIcluster. It is also possible to obtain a reference to the running cluster using makecluster or makeMPIcluster. In this case the count argument can be omitted; if it is supplied, it must equal the number of nodes in the cluster. This interface is still experimental and subject to change.

For SOCK and NWS clusters the option manual = TRUE forces a manual startup mode in which the master prints the command to be run manually to start a worker process. Together with setting the outfile option this can be useful for debugging cluster startup.

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

Examples

```r
## Not run:
## Two workers run on the local machine as a SOCK cluster.
cl <- makeCluster(c("localhost","localhost"), type = "SOCK")
clusterApply(cl, 1:2, get("+"), 3)
stopCluster(cl)

## Another approach to running on the local machine as a SOCK cluster.
cl <- makeCluster(2, type = "SOCK")
clusterApply(cl, 1:2, get("+"), 3)
stopCluster(cl)

## A SOCK cluster with two workers on Mac OS X, two on Linux, and two
## on Windows:
macOptions <-
  list(host = "owasso",
       rscript = "/Library/Frameworks/R.framework/Resources/bin/Rscript",
       snowlib = "/Library/Frameworks/R.framework/Resources/library")

lnxOptions <-
  list(host = "itasca",
       rscript = "/usr/lib64/R/bin/Rscript",
       snowlib = "/home/luke/tmp/lib")

winOptions <-
  list(host="192.168.1.168",
       rscript="C:/Program Files/R/R-2.7.1/bin/Rscript.exe",
       snowlib="C:/Rlibs")

cl <- makeCluster(c(rep(list(macOptions), 2), rep(list(lnxOptions), 2),
                   rep(list(winOptions), 2)), type = "SOCK")
clusterApply(cl, 1:6, get("+"), 3)
stopCluster(cl)

## End(Not run)
```

Description

Experimental functions to collect and display timing data for cluster computations.
Usage

snow.time(expr)
## S3 method for class 'snowTimingData'

print(x, ...)
## S3 method for class 'snowTimingData'

plot(x, xlab = "Elapsed Time", ylab = "Node",
title = "Cluster Usage", ...)

Arguments

expr expression to evaluate
x timing data object to plot or print
xlab x axis label
ylab y axis label
title plot main title
... additional arguments

Details

snow.time collects and returns timing information for cluster usage in evaluating expr. The return value is an object of class snowTimingData; details of the return value are subject to change. The print method for snowTimingData objects shows the total elapsed time, the total communication time between master and worker nodes, and the compute time on each worker node. The plot, motivated by the display produced by xpvm, produces a Gantt chart of the computation, with green rectangles representing active computation, blue horizontal lines representing a worker waiting to return a result, and red lines representing master/worker communications.

Examples

## Not run:
c1 <- makeCluster(2, type="SOCK")
x <- rnorm(1000000)
tm <- snow.time(clusterCall(c1, function(x) for (i in 1:100) sum(x), x))
print(tm)
plot(tm)
stopCluster(c1)

## End(Not run)
Index

*Topic programming

snow-cluster, 1
snow-parallel, 3
snow-rand, 4
snow-startstop, 5
snow-timing, 7

clusterApply (snow-cluster), 1
clusterApplyLB (snow-cluster), 1
clusterCall (snow-cluster), 1
clusterEvalQ (snow-cluster), 1
clusterExport (snow-cluster), 1
clusterMap (snow-cluster), 1
clusterSetupRNG (snow-rand), 4
clusterSetupRNGstream (snow-rand), 4
clusterSetupSPRNG (snow-rand), 4
clusterSplit (snow-cluster), 1

getMPICluster (snow-startstop), 5
makeCluster (snow-startstop), 5
makeMPICluster (snow-startstop), 5
makeNWSCluster (snow-startstop), 5
makePVMCluster (snow-startstop), 5
makeSOCKcluster (snow-startstop), 5

parApply (snow-parallel), 3
parCapply (snow-parallel), 3
parLapply (snow-parallel), 3
parMM (snow-parallel), 3
parRapply (snow-parallel), 3
parSapply (snow-parallel), 3
plot.snowTimingData (snow-timing), 7
print.snowTimingData (snow-timing), 7

setDefaultClusterOptions
(snow-startstop), 5

snow-cluster, 1
snow-parallel, 3
snow-rand, 4
snow-startstop, 5