Package ‘pbapply’

July 4, 2017

Type Package
Title Adding Progress Bar to ‘*apply’ Functions
Version 1.3-3
Date 2017-07-03
Author Peter Solymos [aut, cre], Zygmunt Zawadzki [aut]
Maintainer Peter Solymos <solymos@ualberta.ca>
Description A lightweight package that adds progress bar to vectorized R functions (‘*apply’). The implementation can easily be added to functions where showing the progress is useful (e.g. bootstrap). The type and style of the progress bar (with percentages or remaining time) can be set through options.
Supports several parallel processing backends.

Depends R (>= 3.2.0)
Imports parallel
License GPL-2

URL https://github.com/psolymos/pbapply
BugReports https://github.com/psolymos/pbapply/issues

NeedsCompilation no
Repository CRAN
Date/Publication 2017-07-04 06:48:29 UTC

R topics documented:

 pbapply ................................................................. 2
 pboptions ............................................................... 6
 splitpb ................................................................. 9
 timerProgressBar ..................................................... 10

Index 13
Adding Progress Bar to '*apply' Functions

Description

Adding progress bar to *apply functions, possibly leveraging parallel processing.

Usage

```r
pbapply(x, FUN, ..., cl = NULL)
pbapply(x, MARGIN, FUN, ..., cl = NULL)
pbsapply(x, FUN, ..., simplify = TRUE, USE.NAMES = TRUE, cl = NULL)
pbreplicate(n, expr, simplify = "array", cl = NULL)
```

Arguments

- **X**: For `pbsapply` and `pbapply`, a vector (atomic or list) or an expressions vector (other objects including classed objects will be coerced by `as.list`). For `pbapply` an array, including a matrix.
- **MARGIN**: A vector giving the subscripts which the function will be applied over. 1 indicates rows, 2 indicates columns, c(1,2) indicates rows and columns.
- **FUN**: The function to be applied to each element of `X`: see `apply`, `sapply`, and `lapply`.
- **...**: Optional arguments to `FUN`.
- **simplify**: Logical; should the result be simplified to a vector or matrix if possible?
- **USE.NAMES**: Logical; if `TRUE` and if `X` is character, use `X` as names for the result unless it had names already.
- **n**: Number of replications.
- **expr**: Expression (language object, usually a call) to evaluate repeatedly.
- **cl**: A cluster object created by `makeCluster`, or an integer to indicate number of child-processes (integer values are ignored on Windows) for parallel evaluations.

Details

The behaviour of the progress bar is controlled by the option `type` in `pbapply`, it can take values c("txt", "win", "tk", "none", ) on Windows, and c("txt", "tk", "none", ) on Unix systems.

Other options have elements that are arguments used in the functions `timerProgressBar`, `txtProgressBar`, and `tkProgressBar`. See `pboptions` for how to conveniently set these.

Parallel processing can be enabled through the `cl` argument. `parLapply` is called when `cl` is a 'cluster' object, `mclapply` is called when `cl` is an integer. Showing the progress bar increases the communication overhead between the main process and nodes / child processes compared to the parallel equivalents of the functions without the progress bar. The functions fall back to their original equivalents when the progress bar is disabled (i.e. `getOption("pbapply")$type == "none"`).
or dopb() is FALSE). This is the default when interactive() if FALSE (i.e. called from command line R script).

When doing parallel processing, other objects might need to pushed to the workers, and random numbers must be handled with care (see Examples).

**Value**

Similar to the value returned by the standard *apply functions.

A progress bar is showed as a side effect.

**Note**

Progress bar can add an overhead to the computation.

**Author(s)**

Peter Solymos <solymos@ualberta.ca>

**See Also**

Progress bars used in the functions: `txtProgressBar`, `tkProgressBar`, `timerProgressBar`

Sequential *apply functions: `apply`, `sapply`, `lapply`, `replicate`

Parallel *apply functions from package 'parallel': `parLapply`, `mclapply`.

Setting the options: `pbOptions`

Conveniently add progress bar to for-like loops: `startpb`, `setpb`, `getpb`, `closepb`

**Examples**

```r
## --- simple linear model simulation ---
set.seed(1234)
n <- 200
x <- rnorm(n)
y <- rnorm(n, crossprod(t(model.matrix(~ x)), c(0, 1)), sd = 0.5)
d <- data.frame(y, x)
## model fitting and bootstrap
mod <- lm(y ~ x, d)
ndat <- model.frame(mod)
B <- 100
bid <- sapply(1:B, function(i) sample(nrow(ndat), nrow(ndat), TRUE))
fun <- function(z) {
  if (missing(z))
    z <- sample(nrow(ndat), nrow(ndat), TRUE)
  coef(lm(mod$call$formula, data=ndat[z,]))
}
## standard '*apply' functions
system.time(res1 <- lapply(1:B, function(i) fun(bid[,i])))
system.time(res2 <- sapply(1:B, function(i) fun(bid[,i])))
system.time(res3 <- apply(bid, 2, fun))
```
system.time(res4 <- replicate(B, fun()))

## 'pbapply' functions
## try different settings:
## "none", "txt", "tk", "win", "timer"
op <- pboptions(type = "timer") # default
system.time(res1pb <- pblapply(1:B, function(i) fun(bid[,i])))
pboptions(op)

pboptions(type = "txt")
system.time(res2pb <- pbsapply(1:B, function(i) fun(bid[,i])))
pboptions(op)

pboptions(type = "txt", style = 1, char = "=")
system.time(res3pb <- pbapply(bid, 2, fun))
pboptions(op)

pboptions(type = "txt", char = ":")
system.time(res4pb <- pbreplicate(B, fun()))
pboptions(op)

## Not run:
## parallel evaluation using the parallel package
## (n = 2000 and B = 1000 will give visible timing differences)

library(parallel)
c1 <- makeCluster(2L)
clusterExport(c1, c("fun", "mod", "ndat", "bid"))

## parallel with no progress bar: snow type cluster
## (RNG is set in the main process to define the object bid)
system.time(res1c1 <- parLapply(c1 = c1, 1:B, function(i) fun(bid[,i])))
system.time(res2c1 <- parSapply(c1 = c1, 1:B, function(i) fun(bid[,i])))
system.time(res3c1 <- parApply(c1, bid, 2, fun))

## parallel with progress bar: snow type cluster
## (RNG is set in the main process to define the object bid)
system.time(res1pc1 <- pbapply(1:B, function(i) fun(bid[,i]), c1 = c1))
system.time(res2pc1 <- pbsapply(1:B, function(i) fun(bid[,i]), c1 = c1))
## (RNG needs to be set when not using bid)
parallel::clusterSetRNGStream(c1, iseed = 42L)
system.time(res3pc1 <- pbreplicate(B, fun(), c1 = c1))
system.time(res3pbc1 <- pbapply(bid, 2, fun, c1 = c1))
stopCluster(c1)

if (.Platform$OS.type != "windows") {
  ## parallel with no progress bar: multicore type forking
  ## (mc.set.seed = TRUE in parallel::mclapply by default)
  system.time(res2mc <- mclapply(1:B, function(i) fun(bid[,i]), mc.cores = 2L))
  ## parallel with progress bar: multicore type forking
  ## (mc.set.seed = TRUE in parallel::mclapply by default)
  system.time(res1pbc1 <- pbapply(1:B, function(i) fun(bid[,i]), cl = 2L))
system.time(res3pbmc <- pbapply(1:8, function(i) fun(bid[i]), cl = 2L))

system.time(res4pbmc <- pbreplicate(8, fun(), cl = 2L))
}

## End(Not run)

## --- Examples taken from standard 'apply' functions ---

## --- sapply, lapply, and replicate ---

require(stats); require(graphics)

x <- list(a = 1:10, beta = exp(-3:3), logic = c(TRUE,FALSE,FALSE,TRUE))
# compute the list mean for each list element
pbapply(x, mean)
# median and quartiles for each list element
pbapply(x, quantile, probs = 1:3/4)
pbsapply(x, quantile)
i39 <- sapply(3:9, seq) # list of vectors
pbsapply(i39, fivenum)

## sapply(x, "array") -- artificial example
(v <- structure(10*(5:8), names = LETTERS[1:4]))
f2 <- function(x, y) outer(rep(x, length.out = 3), y)
(a2 <- pbapply(v, f2, y = 2*(1:5), simplify = "array"))

hist(pbreplicate(100, mean(rexp(10))))

## use of replicate() with parameters:
foo <- function(x = 1, y = 2) c(x, y)
# does not work: bar <- function(n, ...) replicate(n, foo(...))
bar <- function(n, x) pbreplicate(n, foo(x = x))
bar(5, x = 3)

## --- apply ---

## Compute row and column sums for a matrix:
 x <- cbind(x1 = 3, x2 = c(4:1, 2:5))
dimnames(x)[[1]] <- letters[1:8]
pbapply(x, 2, mean, trim = .2)
col.sums <- pbapply(x, 2, sum)
row.sums <- pbapply(x, 1, sum)
rbind(cbind(x, Rtot = row.sums), Ctot = c(col.sums, sum(col.sums)))

stopifnot( pbapply(x, 2, is.vector))

## Sort the columns of a matrix
pbapply(x, 2, sort)

## keeping named dimnames
names(dimnames(x)) <- c("row", "col")
x3 <- array(x, dim = c(dim(x), 3),
           dimnames = c(dimnames(x), list(C = paste0("cop.",1:3))))
pboptions

Creating Progress Bar and Setting Options

Description

Creating progress bar and setting options.

Usage

pboptions(...)
startpb(min = 0, max = 1)
setpb(pb, value)
getpb(pb)
closepb(pb)
dopb()

Arguments

... Arguments in tag = value form, or a list of tagged values. The tags must come from the parameters described below.

pb A progress bar object created by startpb.

min, max Finite numeric values for the extremes of the progress bar. Must have min < max.

type, value New value for the progress bar.
Details

pboptions is a convenient way of handling options related to progress bar.
Other functions can be used for conveniently adding progress bar to for-like loops (see Examples).

Value

When parameters are set by pboptions, their former values are returned in an invisible named list. Such a list can be passed as an argument to pboptions to restore the parameter values. Tags are the following:

type Type of the progress bar: timer ("timer"), text ("txt"), Windows ("win"), TclTk ("tk"), or none ("none"). Default value is "timer" progress bar with estimated remaining time when in interactive mode, and "none" otherwise.
char The character (or character string) to form the progress bar. Default value is "+".
txt.width The width of the text based progress bar, as a multiple of the width of char. If NA, the number of characters is that which fits into getOption("width"). Default value is 50.
gui.width The width of the GUI based progress bar in pixels: the dialogue box will be 40 pixels wider (plus frame). Default value is 300.
style The style of the bar, see txtProgressBar and timerProgressBar. Default value is 3.
initial Initial value for the progress bar. Default value is 0.
title Character string giving the window title on the GUI dialogue box. Default value is "R progress bar".
label Character string giving the window label on the GUI dialogue box. Default value is "."
nout Integer, the maximum number of times the progress bar is updated. The default value is 100. Smaller value minimizes the running time overhead related to updating the progress bar. This can be especially important for forking type parallel runs.
min_time Minimum time in seconds. timerProgressBar output is printed only if estimated completion time is higher than this value. The default value is 0.

For startpb a progress bar object.
For getpb and setpb, a length-one numeric vector giving the previous value (invisibly for setpb). The return value is NULL if the progress bar is turned off by getOption("pboptions")$type ("none" or NULL value).
dopb returns a logical value if progress bar is to be shown based on the option getOption("pboptions")$type. It is FALSE if the type of progress bar is "none" or NULL.
For closepb closes the connection for the progress bar.

Author(s)

Peter Solymos <solymos@ualberta.ca>
See Also
Progress bars used in the functions: timerProgressBar, txtProgressBar, tkProgressBar

Examples

```r
## for loop
fun1 <- function() {
  pb <- startpb(0, 10)
  on.exit(closepb(pb))
  for (i in 1:10) {
    Sys.sleep(0.15)
    setpb(pb, i)
  }
  invisible(NULL)
}

## while loop
fun2 <- function() {
  pb <- startpb(0, 10-1)
  on.exit(closepb(pb))
  i <- 1
  while (i < 10) {
    Sys.sleep(0.15)
    setpb(pb, i)
    i <- i + 1
  }
  invisible(NULL)
}

## using original settings
fun1()

## resetting pboptions
opb <- pboptions(style = 1, char = ">")

## check new settings
getOption("pboptions")

## running again with new settings
fun2()

## resetting original
pboptions(opb)

## check reset
getOption("pboptions")
fun1()

## dealing with nested progress bars
## when only one the 1st one is needed
f <- function(x) Sys.sleep(0.01)
g <- function(x) pbapply(1:10, f)
tmp <- lapply(1:10, g) # undesirable

## here is the desirable solution
h <- function(x) {
  opb <- pboptions(type="none")
  on.exit(pboptions(opb))
  pbapply(1:10, f)
}
```
**splitpb**

```r
tmp <- pblapply(1:10, h)
```

---

**splitpb**  
*Divide Tasks for Progress-bar Friendly Distribution in a Cluster*

### Description
Divides up 1:nx into approximately equal sizes (ncl) as a way to allocate tasks to nodes in a cluster repeatedly while updating a progress bar.

### Usage
```r
splitpb(nx, ncl, nout = NULL)
```

### Arguments
- `nx` Number of tasks.
- `ncl` Number of cluster nodes.
- `nout` Integer, maximum number of partitions in the output (must be > 0).

### Value
A list of length min(nout, ceiling(nx / ncl)), each element being an integer vector of length `ncl * k` or less, where `k` is a tuning parameter constrained by the other arguments (`k = max(1, ceiling(ceiling(nx / ncl) / nout))` and `k = 1` if `nout = NULL`).

### Author(s)
Peter Solymos <solymos@ualberta.ca>

### See Also
Parallel usage of *pbapply* and related functions.

### Examples
```r
## define 1 job / worker at a time and repeat
splitpb(10, 4)
## compare this to the no-progress-bar split
## that defines all the jobs / worker up front
parallel::splitIndices(10, 4)

## cap the length of the output
splitpb(20, 2, nout = NULL)
splitpb(20, 2, nout = 5)
```
timerProgressbar

Timer Progress Bar

Description

Text progress bar with timer in the R console.

Usage

```r
timerProgressBar(min = 0, max = 1, initial = 0, char = "=",
width = NA, title, label, style = 1, file = "", min_time = 0)
getTimerProgressBar(pb)
setTimerProgressBar(pb, value, title = NULL, label = NULL)
```

Arguments

- `min, max` (finite) numeric values for the extremes of the progress bar. Must have `min < max`.
- `initial, value` initial or new value for the progress bar. See Details for what happens with invalid values.
- `char` the character (or character string) to form the progress bar. If number of characters is >1, it is silently stripped to length 1 unless `style` is 5 or 6 (see Details).
- `width` the width of the progress bar, as a multiple of the width of `char`. If `NA`, the default, the number of characters is that which fits into `getOption("width")`.
- `style` the style taking values between 1 and 6. 1: progress bar with elapsed and remaining time, remaining percentage is indicated by spaces between pipes (default for this function), 2: throbber with elapsed and remaining time, 3: progress bar with remaining time printing elapsed time at the end, remaining percentage is indicated by spaces between pipes (default for `style` option in `pboptions`), 4: throbber with remaining time printing elapsed time at the end, 5: progress bar with elapsed and remaining time with more flexible styling (see Details and Examples), 6: progress bar with remaining time printing elapsed time at the end with more flexible styling (see Details and Examples).
- `file` an open connection object or "" which indicates the console.
- `min_time` numeric, minimum processing time (in seconds) required to show a progress bar.
- `pb` an object of class "timerProgressBar".
- `title, label` ignored, for compatibility with other progress bars.

Details

timerProgressbar will display a progress bar on the R console (or a connection) via a text representation.
setTimerProgressBar will update the value. Missing (NA) and out-of-range values of value will be (silently) ignored. (Such values of initial cause the progress bar not to be displayed until a valid value is set.)

The progress bar should be closed when finished with: this outputs the final newline character (see closepb).

If style is 5 or 6, it is possible to define up to 4 characters for the char argument (as a single string) for the left end, elapsed portion, remaining portion, and right end of the progress bar (= | by default). Remaining portion cannot be the same as the elapsed portion (space is used for remaining in such cases). If 1 character is defined, it is taken for the elapsed portion. If 2-4 characters are defined, those are interpreted in sequence (left and right end being the same when 2-3 characters defined), see Examples.

Value

For timerProgressBar an object of class "timerProgressBar" inheriting from "txtProgressBar".
For getTimerProgressBar and setTimerProgressBar, a length-one numeric vector giving the previous value (invisibly for setTimerProgressBar).

Author(s)

Zygmunt Zawadzki <zawadzki.zygmunt@gmail.com>
Peter Solymos <solymos@ualberta.ca>

See Also

The timerProgressBar implementation follows closely the code of txtProgressBar.

Examples

```r
## increase sluggishness to admire the progress bar longer
sluggishness <- 0.02

test_fun <- function(...) {
  pb <- timerProgressBar(...)
  on.exit(close(pb))
  for(i in seq(0, 1, 0.05)) {
    Sys.sleep(sluggishness)
    setTimerProgressBar(pb, i)
  }
  invisible(NULL)
}

## check the different styles
test_fun(width = 35, char = "*", style = 1)
test_fun(style = 2)
test_fun(width = 50, char = ".", style = 3)
test_fun(style = 4)
test_fun(width = 35, char = "[=-]", style = 5)
test_fun(width = 50, char = "{.*}\", style = 6)
```
## no bar only percent and elapsed

test_fun(width = 0, char = " ", style = 6)

## this should produce a progress bar based on min_time
(elapsed <- system.time(test_fun(width = 35, min_time = 0))[["elapsed"]))

## this should not produce a progress bar based on min_time
system.time(test_fun(min_time = 2 * elapsed))[["elapsed"]]}
Index

*Topic IO
  pboptions, 6

*Topic manip
  pbapply, 2

*Topic utilities
  pbapply, 2
  pboptions, 6
  splitpb, 9
  timerProgressBar, 10

apply, 2, 3
as.list, 2

closepb, 3, 11
closepb(pboptions), 6

dopb(pboptions), 6

getpb, 3
getpb(pboptions), 6
getTimerProgressBar(timerProgressBar), 10

lapply, 2, 3

makeCluster, 2
mclapply, 2, 3

parLapply, 2, 3
pbapply, 2, 9
pblapply(pbapply), 2
pboptions, 2, 3, 6, 10
pbreplicate(pbapply), 2
pbsapply(pbapply), 2

replicate, 3

sapply, 2, 3
setpb, 3
setpb(pboptions), 6
setTimerProgressBar(timerProgressBar), 10

splitpb, 9
startpb, 3
startpb(pboptions), 6
timerProgressBar, 2, 3, 7, 8, 10
tkProgressBar, 2, 3, 8
txtProgressBar, 2, 3, 7, 8, 11