Package 'pbdDEMO'

October 25, 2016

Type Package

Title Programming with Big Data -- Demonstrations and Examples Using 'pbdR' Packages

Version 0.3-1

Description A set of demos of 'pbdR' packages, together with a useful, unifying vignette.

License Mozilla Public License 2.0

Depends R (>= 3.0.0), methods, pbdMPI (>= 0.3-0), pbdBASE(>= 0.4-3), pbdDMAT(>= 0.4-0)

Enhances maps, RColorBrewer, pbdNCDF4 (>= 0.1-2), pmclust, MixSim, EMCluster, phyclust, MASS, rjags

SystemRequirements OpenMPI (>= 1.5.4) on Solaris, Linux, Mac, and FreeBSD. MS-MPI (Microsoft HPC Pack 2012 R2 MS-MPI Redistributable Package) on Windows.

LazyLoad yes

LazyData yes

ByteCompile yes

NeedsCompilation yes

URL http://r-pbd.org/

BugReports http://group.r-pbd.org/

MailingList Please send questions and comments regarding pbdR to RBigData@gmail.com

Maintainer Drew Schmidt <schmidt@math.utk.edu>

Author Drew Schmidt [aut, cre],
Wei-Chen Chen [aut],
George Ostrouchov [aut],
Pragneshkumar Patel [aut]

Repository CRAN

Date/Publication 2016-10-25 23:31:25

R topics documented:

pbdDEMO-package . . . 2
gbd_dmat . . . 2
load_balance . . . 4
mpi_example . . . 6
ncvar . . . 7
plot_dmat . . . 8
read.csv.ddmatrix . . . 9
Temperature at Reference Height . . . 10
timer . . . 12
verify . . . 12

Index **14**

`pbdDEMO-package` *Demonstrations and Examples for the pbd Project*

Description

A set of demos of pbdR packages, together with a useful, unifying vignette.

Details

Package:	pbdDEMO
Type:	Package
License:	MPL 2.0
LazyLoad:	yes

This package requires an MPI library (OpenMPI, MPICH2, or LAM/MPI).

Author(s)

Drew Schmidt <schmidt AT math.utk.edu>, Wei-Chen Chen, George Ostrouchov, and Pragneshkumar Patel.

References

Programming with Big Data in R Website: http://r-pbd.org/

`gbd_dmat` *GBD Matrix to Distributed Dense Matrix and vice versa*

Description

This function convert a GBD matrix and a distributed dense matrix.

Usage

```
gbd2dmat(X.gbd, skip.balance = FALSE, comm = .pbd_env$SPMD.CT$comm,
  gbd.major = .pbd_env$gbd.major, bldim = .pbd_env$BLDIM,
  ICTXT = .pbd_env$ICTXT)

dmat2gbd(X.dmat, bal.info = NULL, comm = .pbd_env$SPMD.CT$comm,
  gbd.major = .pbd_env$gbd.major)
```

Arguments

X.gbd	a GBD matrix.
skip.balance	if load.balance were skipped.
comm	a communicator number.
gbd.major	1 for row-major storage, 2 for column-major.
bldim	the blocking dimension for block-cyclically distributing the matrix across the process grid.
ICTXT	BLACS context number for return.
X.dmat	a ddmatrix matrix.
bal.info	a returned object from balance.info.

Details

X.gbd is a matrix with dimension N.gbd * p and exists on all processors. N.gbd may be vary across processors.

If skip.balance = TRUE, then load.balance will not be called and X.gbd is preassumed to be balanced.

For demonstration purpose, these objects should not contains weird values such as NA.

dmat2gbd is supposed returned a balanced gbd matrix if bal.info is not supplied.

Value

gbd2dmat returns a ddmatrix object. dmat2gbd returns a (balanced) gbd matrix.

Examples

```
## Not run:
### Under command mode, run the demo with 4 processors by
### (Use Rscript.exe for windows system)
mpiexec -np 4 Rscript -e "demo(gbd_dmat,'pbdDEMO',ask=F,echo=F)"

## End(Not run)
```

load_balance *Load Balancing a Dataset*

Description

These functions will rearrange data for all processors such that the data amount of each processor is nearly equal.

Usage

```
balance.info(X.gbd, comm = .pbd_env$SPMD.CT$comm,
  gbd.major = .pbd_env$gbd.major, method = .pbd_env$divide.method[1])

load.balance(X.gbd, bal.info = NULL, comm = .pbd_env$SPMD.CT$comm,
  gbd.major = .pbd_env$gbd.major)

unload.balance(new.X.gbd, bal.info, comm = .pbd_env$SPMD.CT$comm)
```

Arguments

X.gbd	a GBD data matrix (converted if not).
comm	a communicator number.
gbd.major	1 for row-major storage, 2 for column-major.
method	"block.cyclic" or "block0".
bal.info	a returned object from balance.info.
new.X.gbd	a GBD data matrix or vector

Details

X.gbd is the data matrix with dimension N.gbd * p and exists on all processors where N.gbd may be vary across processors. If X.gbd is a vector, then it is converted to a N.gbd * 1 matrix.

balance.info provides the information how to balance data set such that all processors own similar amount of data. This information may be also useful for tracking where the data go or from.

load.balance does the job to transfer data from one processor with more data to the other processors with less data based on the balance information balance.info.

unload.balance is the inversed function of load.balance, and it takes the same information bal.info to reverse the balanced result back to the original order. new.X.gbd is usually the output of load.balance{X.gbd} or other results of further computing of it. Again, if new.X.gbd is a vector, then it is converted to an one column matrix.

Value

`balance.info` returns a list contains two data frames and two vectors.

Two data frames are `send` and `recv` for sending and receiving data. Each data frame has two columns `org` and `belong` for where data original in and new belongs. Number of row of `send` should equal to the `N.gbd`, and number of row of `recv` should be nearly equal to `n = N / COMM.SIZE` where `N` is the total observations of all processors.

Two vectors are `N.allgbd` and `new.N.allgbd` which are all numbers of rows of `X.gbd` on all processes before and after load balance, correspondingly. Both have length equals to `comm.size(comm)`.

`load.balance` returns a matrix for each processor and the matrix has the dimension nearly equal to `n * p`.

`unload.balance` returns a matrix with the same length/rows as the original number of row of `X.gbd`.

Warning(s)

These function only support total object length is less than 2^32 - 1 for machines using 32-bit integer.

Examples

```
## Not run:
# Save code in a file "demo.r" and run in 4 processors by
# > mpiexec -np 4 Rscript demo.r

### Setup environment.
library(pbdDEMO, quiet = TRUE)

### Generate an example data.
N.gbd <- 5 * (comm.rank() * 2)
X.gbd <- rnorm(N.gbd * 3)
dim(X.gbd) <- c(N.gbd, 3)
comm.cat("X.gbd[1:5,]\n", quiet = TRUE)
comm.print(X.gbd[1:5,], rank.print = 1, quiet = TRUE)

bal.info <- balance.info(X.gbd)
new.X.gbd <- load.balance(X.gbd, bal.info)
org.X.gbd <- unload.balance(new.X.gbd, bal.info)

comm.cat("org.X.gbd[1:5,]\n", quiet = TRUE)
comm.print(org.X.gbd[1:5,], rank.print = 1, quiet = TRUE)
if(any(org.X.gbd - X.gbd != 0)){
  cat("Unbalance fails in the rank ", comm.rank(), "\n")
}

### Quit.
finalize()

## End(Not run)
```

mpi_example *MPI Demonstrations*

Description

These functions are examples of simple statistics via MPI calls.

Usage

```
mpi.stat(x.gbd)

mpi.bin(x.gbd, breaks = pi/3 * (-3:3))

mpi.quantile(x.gbd, prob = 0.5)

mpi.ols(y.gbd, X.gbd)
```

Arguments

x.gbd	gbd a GBD vector.
breaks	a set to break data in groups.
prob	a desired probability for quantile.
y.gbd	a GBD vector.
X.gbd	a GBD matrix.

Details

x.gbd and y.gbd are vectors with length N.gbd. X.gbd is a matrix with dimension N.gbd * p and exists on all processors. N.gbd may be vary across processors.

For demonstration purpose, these objects should not contains weird values such NA.

Value

mpi.stat returns sample mean and sample variance. mpi.bin returns binning counts for the given breaks. mpi.quantile returns a quantile. mpi.ols returns ordinary least square estimates (beta_hat).

Examples

```
## Not run:
### Under command mode, run the demo with 4 processors by
### (Use Rscript.exe for windows system)
mpiexec -np 4 Rscript -e "demo(sample_stat,'pbdDEMO',ask=F,echo=F)"
mpiexec -np 4 Rscript -e "demo(binning,'pbdDEMO',ask=F,echo=F)"
mpiexec -np 4 Rscript -e "demo(quantile,'pbdDEMO',ask=F,echo=F)"
mpiexec -np 4 Rscript -e "demo(ols,'pbdDEMO',ask=F,echo=F)"
mpiexec -np 4 Rscript -e "demo(gbd2dmat,'pbdDEMO',ask=F,echo=F)"
```

```
mpiexec -np 4 Rscript -e "demo(balance,'pbdDEMO',ask=F,echo=F)"

## End(Not run)
```

---

ncvar *Read and Write Parallel NetCDF4 Files in GBD and ddmatrix Format*

Description

These functions write and read NetCDF4 files in GBD and ddmatrix format.

Usage

```
demo.ncvar_put_dmat(nc, varid, vals, verbose = FALSE,
  comm = .pbd_env$SPMD.CT$comm)

demo.ncvar_put_gbd(nc, varid, vals, verbose = FALSE,
  comm = .pbd_env$SPMD.CT$comm, gbd.major = .pbd_env$gbd.major)

demo.ncvar_get_dmat(nc, varid, verbose = FALSE, signedbyte = TRUE,
  collapse_degen = TRUE, bldim = .pbd_env$bldim, ICTXT = .pbd_env$ictxt,
  comm = .pbd_env$SPMD.CT$comm)

demo.ncvar_get_gbd(nc, varid, verbose = FALSE, signedbyte = TRUE,
  collapse_degen = TRUE, comm = .pbd_env$SPMD.CT$comm,
  gbd.major = .pbd_env$gbd.major)
```

Arguments

nc	an object of class ncdf4 (as returned by either function nc_open_par or function nc_create_par), indicating what file to read from.
varid	See ncvar_get for details.
vals	See ncvar_put for details.
verbose	See ncvar_get for details.
comm	a communicator number.
gbd.major	a GBD major, either 1 for row-major or 2 for column-major.
signedbyte	See ncvar_get for details.
collapse_degen	See ncvar_get for details.
bldim	the blocking dimension for block-cyclically distributing the matrix across the process grid.
ICTXT	BLACS context number for return.

Details

demo.ncvar_get_* are similar to ncvar_get of **pbdNCDF4**, but focus on 2D arrays and return a ddmatrix or GBD matrix.

demo.ncvar_put_* are also similar to ncvar_put of **pbdNCDF4**, but only dump 2D arrays.

Value

demo.ncvar_get_dmat returns a ddmatrix, and demo.ncvar_get_gbd returns a GBD matrix in either row- or column major specified by gbd.major.

Examples

```
## Not run:
### Under command mode, run the demo with 4 processors by
### (Use Rscript.exe for windows system)
mpiexec -np 4 Rscript -e "demo(nc4_serial,'pbdDEMO',ask=F,echo=F)"
mpiexec -np 4 Rscript -e "demo(nc4_parallel,'pbdDEMO',ask=F,echo=F)"
mpiexec -np 4 Rscript -e "demo(nc4_dmat,'pbdDEMO',ask=F,echo=F)"
mpiexec -np 4 Rscript -e "demo(nc4_gbdc,'pbdDEMO',ask=F,echo=F)"

## End(Not run)
```

---

plot_dmat *Visualizing the DMAT Data Structure*

Description

Plot a (small) global matrix as though it had been chopped up into pieces in the block-cyclic fashion.

Usage

```
plot_dmat(nrow, ncol, nprow, npcol, bldim, ..., labeling = "blacs",
  col = "rainbow")
```

Arguments

nrow, ncol	Number of global rows/columns of the matrix.
nprow, npcol	Number of processor rows/columns in the BLACS grid.
bldim	The blocking factor for the data distribution.
...	Additional arguments
labeling	Character argument; should be "blacs" or "mpi". This determines how the processor labeling should be, either in the 2-d BLACS way, or in the 1-d MPI way.
col	R plots color argument

Details

This function helps the user visualize 2-d block-cyclic distributed data.

read.csv.ddmatrix *A Simple Parallel CSV Reader*

Description

Read in a table from a CSV file in parallel as a distributed matrix.

Usage

```
read.csv.ddmatrix(file, sep = ",", nrows, ncols, header = FALSE,
  bldim = 4, num.rdrs = 1, ICTXT = 0, exact.linecount = TRUE)
```

Arguments

file	csv file name.
sep	separator character.
nrows, ncols	dimensions of the csv file. Allowed to be missing in function call.
header	logical indicating presence/absence of character header for file.
bldim	the blocking dimension for block-cyclically distributing the matrix across the process grid
num.rdrs	numer of processes to be used to read in the table
ICTXT	BLACS context number for return
exact.linecount	linecount In the event that nrows is missing, this determines whether or not the exact number of rows should be determined (which requires a file read), or if an estimate should be used. Default is TRUE, meaning that the file will be scanned.

Details

The function reads in data from a csv file into a distributed matrix. This function sits somewhere between scan() and read.csv(), but for parallel reads into a distributed matrix.

The arguments nrow= and ncol= are optional. In the case that they are left blank, they will be determined. However, note that doing so is costly, so knowing the dimensions beforehand can greatly improve performance.

Although frankly, the performance-minded should not be using csv's in the first place. Consider using the pbdNCDF4 package for managing data.

Value

Returns a distributed matrix.

`Temperature at Reference Height`

Surface Air Temperature at Reference Height (TREFHT)

Description

This is a practical example in NetCDF4 format and for data reading, writing, and transforming. This dataset is a partial output of the Surface Air Temperature at Reference Height (TREFHT) which is monthly averaged of Jan. 2004 from a CAM5 simulation. This dataset only contains a tiny part of ultra-large simulations conducted by Mr Prabhat and Michael Wehner of Lawrence Berkeley National Laboratory.

Format

An R data file contains two lists: `def` for structure definition of "TREFHT" in `ncvar4` class (see **pbdNCDF4** package for details), and `data` for output values of simulation in a matrix where rows are for 1152 longitudes and columns are for 768 latitudes.

Details

Version 5.0 of the Community Atmosphere Model (CAM) is the latest in a series of global atmosphere models developed primarily at the National Center for Atmospheric Research (NCAR).

`TREFHT` contains two lists: `def` and `data`.

`def` is a list contains usual definitions of NetCDF4. In this case, they define the variable "TREFHT" including 2D dimensions 1152 longitudes and 768 latitudes, 1 time step, the unit in Kelvin, ... etc.

`data` contains values in matrix with dimension 1152×768. Note that this matrix stores data in C format (column major), so it needs a transpose to obtains the R/Fortran format (row major). Also, the longitude order is not the same as the **maps** package. Please see the example below for the adjustment or by calling `demo('trefht','pbdDEMO')` inside an R session.

Author(s)

Mr Prabhat and Michael Wehner.

References

More datasets are available on ESGF (`http://www.earthsystemgrid.org/`) through the C20C project (on the NERSC portal).

CAM5: `http://www.cesm.ucar.edu/models/cesm1.0/cam/`

Programming with Big Data in R Website: `http://r-pbd.org/`

See Also

`ncvar_put_2D` and `ncvar_get_2D`.

Examples

```
 ## Not run:  library(maps)
library(RColorBrewer)
library(pbdDEMO, quiet = TRUE)

lon <- TREFHT$def$dim[[1]]$vals              # longitude
lat <- TREFHT$def$dim[[2]]$vals              # latitude
da <- TREFHT$data                            # surface temperature

# Define Axes.
x <- c(lon[lon > 180] -360, lon[lon <= 195])  # adjustment for maps
y <- lat
z <- rbind(da[lon > 180,], da[lon <= 195,])   # adjustment for maps
xlim <- range(x)
ylim <- range(y)
zlim <- range(z)
col.z <- c(colorRampPalette(c("#0000FF", "#2BFCD3"))(100),
           colorRampPalette(c("#2BFCD3", "#5300AB"))(100),
           colorRampPalette(c("#5300AB", "#7CFA82"))(100),
           colorRampPalette(c("#7CFA82", "#A90055"))(100),
           colorRampPalette(c("#A90055", "#D6FC28"))(100),
           colorRampPalette(c("#D6FC28", "#FE0001"))(100))

# Plot
layout(matrix(c(1, 2), ncol = 1), heights = c(2, 1))
par(mar = c(4, 4, 4, 0))
plot(NULL, NULL, xlim = xlim, ylim = ylim, type = "n", axes = FALSE,
     xlab = "Longitude", ylab = "Latitude", main = "TREFHT (Jan. 2004)")
image(x, y, z, zlim = zlim, xlim = xlim, ylim = ylim,
      col = col.z, add = TRUE)

# Add Map.
map(add = TRUE)
abline(h = c(-23.5, 0, 23.5), v = 0, lty = 2)
xtickets <- seq(-180, 180, by = 30)
ytickets <- seq(-90, 90, by = 30)
box()
axis(1, at = xtickets, labels = xtickets)
axis(2, at = ytickets, labels = ytickets)

# Add Legend.
z.temp <- matrix(seq(zlim[1], zlim[2], length = 500), ncol = 1)
ztickets <- seq(230, 300, by = 10)
par(mar = c(4, 4, 0, 1))
plot(NULL, NULL, xlim = zlim, ylim = c(0, 1), type = "n", axes = FALSE,
     xlab = "TREFHT (Kelvin)", ylab = "")
image(z.temp, 0, z.temp, zlim = zlim, xlim = zlim, ylim = c(0, 1),
      col = col.z, add = TRUE)
axis(1, at = ztickets, labels = ztickets)

## End(Not run)
```

timer *A Timing Function for SPMD Routines*

Description

A timing function for use with parallel codes executed in the batch SPMD style.

Usage

```
timer(timed)
```

Arguments

timed	expression to be timed.

Details

Finds the min, mean, and max execution time across all independent processes executing the operation `timed`.

Value

A named vector containing the minimum, mean, and maximum time across all processors in the communicator. All values are global.

verify *Distributed Linear Algebra Verification*

Description

At-scale verification routines for distributed linear algebra.

Usage

```
verify.svd(nrows = 1000, ncols = 1000, mean = 0, sd = 1, bldim = 8,
  tol = 1e-07, ICTXT = .pbd_env$ictxt)

verify.chol(nrows = 1000, mean = 0, sd = 1, bldim = 8, tol = 1e-07,
  ICTXT = .pbd_env$ictxt)

verify.inverse(nrows = 1000, mean = 0, sd = 1, bldim = 8, tol = 1e-07,
  ICTXT = .pbd_env$ictxt)

verify.solve(nrows = 1000, mean = 0, sd = 1, const = 1, bldim = 8,
  tol = 1e-07, ICTXT = .pbd_env$ictxt)
```

Arguments

nrows, ncols	global dimension.
mean, sd	mean and standard deviation when sampling from a normal distribution.
bldim	blocking dimension.
tol	numeric tolerance for testing equality. Differences smaller than tol are considered equal.
ICTXT	BLACS context
const	numerical value for generating a constant ddmatrix.

Details

These routines numerically verify the accuracy of the given operation. Each operation generates only the local data that is needed, and one never needs to store the global problem on any one rank (unless bldim is set inappropriately).

For example, verify.solve() will generate the A matrix and "true solution" x to the problem Ax=b, each as distributed objects. Next, the "right hand side" b is generated by multiplying A and x together. Finally, the numericaly solution x is computed and compared against the known true value at the specified numerical tolerance.

Index

∗Topic **Data**
plot_dmat, 8
read.csv.ddmatrix, 9
∗Topic **Distributing**
plot_dmat, 8
read.csv.ddmatrix, 9
∗Topic **Package**
pbdDEMO-package, 2
∗Topic **Timing**
timer, 12
verify, 12
∗Topic **datasets**
Temperature at Reference Height, 10
∗Topic **programming**
gbd_dmat, 2
load_balance, 4
mpi_example, 6
ncvar, 7

balance.info *(*load_balance*)*, 4

demo.ncvar_get_dmat *(*ncvar*)*, 7
demo.ncvar_get_gbd *(*ncvar*)*, 7
demo.ncvar_put_dmat *(*ncvar*)*, 7
demo.ncvar_put_gbd *(*ncvar*)*, 7
dmat2gbd *(*gbd_dmat*)*, 2

Example *(*Temperature at Reference Height*)*, 10

gbd2dmat *(*gbd_dmat*)*, 2
gbd_dmat, 2

load.balance *(*load_balance*)*, 4
load_balance, 4

mpi.bin *(*mpi_example*)*, 6
mpi.ols *(*mpi_example*)*, 6
mpi.quantile *(*mpi_example*)*, 6
mpi.stat *(*mpi_example*)*, 6
mpi_example, 6

ncvar, 7

pbdDEMO-package, 2
plot_dmat, 8
Practical *(*Temperature at Reference Height*)*, 10

read.csv.ddmatrix, 9

Temperature at Reference Height, 10
timer, 12
TREFHT *(*Temperature at Reference Height*)*, 10

unload.balance *(*load_balance*)*, 4

verify, 12