Package 'partools'

July 23, 2025

Version 1.1.6

Author Norm Matloff <normmatloff@gmail.com> [cre,aut],
Clark Fitzgerald <clarkfitzg@gmail.com> [aut], with
contributions by Alex Rumbaugh <aprumbaugh@ucdavis.edu> and
Hadley Wickham <h.wickham@gmail.com>

Maintainer Norm Matloff <normmatloff@gmail.com>

Title Tools for the 'Parallel' Package

Description Miscellaneous utilities for parallelizing large
computations. Alternative to MapReduce.
File splitting and distributed operations such as sort and aggregate.
``Software Alchemy'' method for parallelizing most statistical methods,
presented in N. Matloff, Parallel Computation for Data Science,
Chapman and Hall, 2015. Includes a debugging aid.

Depends regtools,parallel,stats,utils,data.table,pdist,methods

Suggests rpart,e1071,testthat

ByteCompile yes

NeedsCompilation no

License GPL (>= 2)

URL https://github.com/matloff/partools

BugReports https://github.com/matloff/partools/issues

Repository CRAN

Date/Publication 2017-04-10 09:16:20 UTC

Contents

`ca,cabase,calm,caglm,caprcomp,cakm,cameans,caquantile,caagg,caknn`
Software Alchemy: Turning Complex Statistical Computations into Embarrassingly-Parallel Ones

---

Description

Easy parallelization of most statistical computations.

Usage

```
ca(cls,z,ovf,estf,estcovf=NULL,findmean=TRUE,scramble=FALSE)
cabase(cls,ovf,estf,estcovf=NULL,findmean=TRUE,cacall=FALSE,z=NULL,scramble=FALSE)
calm(cls,lmargs)
caglm(cls,glmargs)
caprcomp(cls,prcompargs, p)
cakm(cls,mtdf,ncenters,p)
cameans(cls,cols,na.rm=FALSE)
caquantile(cls,vec, probs = c(0.25, 0.5, 0.75),na.rm=FALSE)
caagg(cls,ynames,xnames,dataname,FUN)
caknn(cls, yname, k, xname='')
```

Arguments

`cls`	A cluster run under the **parallel** package.
`z`	A data frame, matrix or vector, one observation per row/element.
`ovf`	Overall statistical function, say `glm`.
`estf`	Function to extract the point estimate (typically vector-valued) from the output of `ovf`.
`estcovf`	If provided, function to extract the estimated covariance matrix of the output of `estf`
.	
`findmean`	If TRUE, output the average of the estimates from the chunks; otherwise, output only the estimates themselves.
`lmargs`	Quoted string representing arguments to `lm`, e.g. R formula, `data` specification.
`glmargs`	Quoted string representing arguments to `glm`, e.g. R formula, `data` specification, and `family` argument.
`prcompargs`	Quoted string representing arguments to `prcomp`.
`p`	Number of columns in data

na.rm	If TRUE, remove NA values from the analysis.
mtdf	Quoted name of a distributed matrix or data frame.
ncenters	Number of clusters to find.
cacall	If TRUE, indicates that cabase had been called by ca
scramble	If this and cacall are TRUE, randomize the data before distributing.
cols	A quoted string that evaluates to a data frame or matrix.
vec	A quoted string that evaluates to a vector.
yname	A quoted variable name, for the Y vector.
k	Number of nearest neighbors.
xname	A quoted variable name, for the X matrix/data frame. If empty, it is assumed that preprocessx has already been run on the nodes; if nonempty, that function is run on this X data.
ynames	A vector of quoted variable names.
xnames	A vector of quoted variable names.
dataname	Quoted name of a data frame or matrix.
probs	As in the argument with the same name in quantile. Should not be 0.00 or 1.00, as asymptotic normality doesn't hold.
FUN	Quoted name of a function.

Details

Implements the "Software Alchemy" (SA) method for parallelizing statistical computations (N. Matloff, *Parallel Computation for Data Science*, Chapman and Hall, 2015, with further details in N. Matloff, Software Alchemy: Turning Complex Statistical Computations into Embarrassingly-Parallel Ones, *Journal of Statistical Software*, 2016.) This can result in substantial speedups in computation, as well as address limits on physical memory.

The method involves breaking the data into chunks, and then applying the given estimator to each one. The results are averaged, and an estimated covariance matrix computed (optional).

Except for ca, it is assumed that the chunking has already been done, say via distribsplit or readnscramble.

Note that in cabase, the data object is not specified explicitly in the argument list. This is done through the function ovf.

Key point: *The SA estimator is statistically equivalent to the original, nonparallel one, in the sense that they have the SAME asymptotic statistical accuracy. Neither the non-SA nor the SA estimator is "better" than the other*, and usually they will be quite close to each other anyway. Since we would use SA only with large data sets anyway (otherwise, parallel computation would not be needed for speed), the asymptotic aspect should not be an issue. In other words, with SA we achieve the same statistical accuracy while possibly attaining much faster computation.

It is vital to keep in mind that *The memory space issue can be just as important as run time*. Even if the problem is run on many cores, if the total memory space needed exceeds that of the machine, the run may fail.

Wrapper functions, applying SA to the corresponding R function (or function elsewere in this package):

- calm: Wrapper for lm.
- caglm: Wrapper for glm.
- caprcomp: Wrapper for prcomp.
- cakm: Wrapper for kmeans.
- cameans: Wrapper for colMeans.
- caquantile: Wrapper for quantile.
- caagg: Like distribagg, but finds the average value of FUN across the cluster nodes.

A note on NA values: Some R functions such as lm, glm and prcomp have an na.action argument. The default is na.omit, which means that cases with at least one NA value will be discarded. (This is also settable via options().) However, na.omit seems to have no effect in prcomp unless that function's formula option is used. When in doubt, apply the function na.omit directly; e.g. na.omit(d) for a data frame d returns a data frame consisting of only the intact rows of d.

The method assumes that the base estimator is asymptotically normal, and assumes i.i.d. data. If your data set had been stored in some sorted order, it must be randomized first, say using the scramble option in distribsplit or by calling readnscramble, depending on whether your data is already in memory or still in a file.

Value

R list with these components:

- thts, the results of applying the requested estimator to the chunks; the estimator from chunk i is in row i
- tht, the chunk-averaged overall estimator, if requested
- thtcov, the estimated covariance matrix of tht, if available

The wrapper functions return the following list elements:

- calm, caglm: estimated regression coefficients and their estimated covariance matrix
- caprcomp: sdev (square roots of the eigenvalues) and rotation, as with prcomp; thts is returned as well.
- cakm: centers and size, as with kmeans; thts is returned as well.

The wrappers that return thts are useful for algorithms that may expose some instability in the original (i.e. non-SA) algorithm. With prcomp, for instance, the eigenvectors corresponding to the smaller eigenvalues may have high variances in the nonparallel version, which will be reflected in large differences from chunk to chunk in SA, visible in thts. Note that this reflects a fundamental problem with the algorithm on the given data set, not due to Software Alchemy; on the contrary, an important advantage of the SA approach is to expose such problems.

Author(s)

Norm Matloff

References

N. Matloff N (2016). "Software Alchemy: Turning Complex Statistical Computations into Embarrassingly-Parallel Ones." *Journal of Statistical Software*, **71(4)**, 1-15.

Examples

```
# set up 'parallel' cluster
cls <- makeCluster(2)
setclsinfo(cls)

# generate simulated test data, as distributed data frame
n <- 10000
p <- 2
tmp <- matrix(rnorm((p+1)*n),nrow=n)
u <- tmp[,1:p]  # "X" values
# add a "Y" col
u <- cbind(u,u %*% rep(1,p) + tmp[,p+1])
# now in u, cols 1,2 are the "X" variables, and col 3 is "Y",
# with regress coefs (0,1,1), with tmp[,p+1] being the error term
distribsplit(cls,"u")  # form distributed d.f.
# apply the function
#### calm(cls,"u[,3] ~ u[,1]+u[,2]")$tht
calm(cls,"V3 ~ .,data=u")$tht
# check; results should be approximately the same
lm(u[,3] ~ u[,1]+u[,2])
# without the wrapper
ovf <- function(dummy=NULL) lm(V3 ~ .,data=z168)
ca(cls,u,ovf,estf=coef,estcovf=vcov)$tht

## Not run:
# Census data on programmers and engineers; include a quadratic term for
# age, due to nonmonotone relation to income
data(prgeng)
distribsplit(cls,"prgeng")
caout <- calm(cls,"wageinc ~ age+I(age^2)+sex+wkswrkd,data=prgeng")
caout$tht
# compare to nonparallel
lm(wageinc ~ age+I(age^2)+sex+wkswrkd,data=prgeng)
# get standard errors of the beta-hats
sqrt(diag(caout$thtcov))

# find mean age for all combinations of the cit and sex variables
caagg(cls,"age",c("cit","sex"),"prgeng","mean")
# compare to nonparallel
aggregate(age ~ cit+sex,data=prgeng,mean)

data(newadult)
distribsplit(cls,"newadult")
caglm(cls," gt50 ~ ., family = binomial,data=newadult")$tht

caprcomp(cls,'newadult,scale=TRUE',5)$sdev
prcomp(newadult,scale=TRUE)$sdev
```

```
cameans(cls,"prgeng")
cameans(cls,"prgeng[,c('age','wageinc')]")
caquantile(cls,'prgeng$age')

pe <- prgeng[,c(1,3,8)]
distribsplit(cls,"pe")
z1 <- cakm(cls,'pe',3,3); z1$size; z1$centers
# check algorithm unstable
z1$tht  # looks unstable

pe <- prgeng
pe$ms <- as.integer(pe$educ == 14)
pe$phd <- as.integer(pe$educ == 16)
pe <- pe[,c(1,7,8,9,12,13)]
distribsplit(cls,'pe',scramble=TRUE)
kout <- caknn(cls,'pe[,3]',50,'pe[,-3]')

## End(Not run)

stopCluster(cls)
```

`caclassfit,caclasspred,vote,re_code`

Software Alchemy for Machine Learning

Description

Parallelization of machine learning algorithms.

Usage

```
caclassfit(cls,fitcmd)
caclasspred(fitobjs,newdata,yidx=NULL,...)
vote(preds)
re_code(x)
```

Arguments

`cls`	A cluster run under the **parallel** package.
`fitcmd`	A string containing a model-fitting command to be run on each cluster node. This will typically include specification of the distributed data set.
`fitobjs`	An R list of objects returned by the `fitcmd` calls.
`newdata`	Data to be predicted from the fit computed by `caclassfit`.
`yidx`	If provided, index of the true class values in `newdata`, typically in a cross-validation setting.

...	Arguments to be passed to the underlying prediction function for the given method, e.g. predict.rpart.
preds	A vector of predicted classes, from which the "winner" will be selected by voting.
x	A vector of integers, in this context class codes.

Details

This should work for almost any classification code that has a "fit" function and a predict method.

The method assumes i.i.d. data. If your data set had been stored in some sorted order, it must be randomized first, say using the scramble option in distribsplit or by calling readnscramble, depending on whether your data is already in memory or still in a file.

It is assumed that class labels are 1,2,... If not, use re_code.

Value

The caclassfit function returns an R list of objects as in fitobjs above.

The caclasspred function returns an R list with these components:

- predmat, a matrix of predicted classes for newdata, one row per cluster node
- preds, the final predicted classes, after using vote to resolve possible differences in predictions among nodes
- consensus, the proportion of cases for which all nodes gave the same predictions (higher values indicating more stability)
- acc, if yidx is non-NULL, the proportion of cases in which preds is correct
- confusion, if yidx is non-NULL, the confusion matrix

Author(s)

Norm Matloff

Examples

```
## Not run:
# set up 'parallel' cluster
cls <- makeCluster(2)
setclsinfo(cls)
# data prep
data(prgeng)
prgeng$occ <- re_code(prgeng$occ)
prgeng$bs <- as.integer(prgeng$educ == 13)
prgeng$ms <- as.integer(prgeng$educ == 14)
prgeng$phd <- as.integer(prgeng$educ == 15)
prgeng$sex <- prgeng$sex - 1
pe <- prgeng[,c(1,7,8,9,12,13,14,5)]
pe$occ <- as.factor(pe$occ)   # needed for rpart!
# go
distribsplit(cls,'pe')
```

```
library(rpart)
clusterEvalQ(cls,library(rpart))
fit <- caclassfit(cls,"rpart(occ ~ .,data=pe)")
predout <- caclasspred(fit,pe,8,type='class')
predout$acc  # 0.36

stopCluster(cls)

## End(Not run)
```

---

dbs *Debugging aid for* **parallel** *cluster code.*

Description

Aids in debugging of code written for the cluster operations in the **parallel** package.

Usage

```
dbs(nwrkrs,xterm=NULL,src=NULL,ftn=NULL)
writemgrscreen(cmd)
killdebug()
dbsmsgstart(cls)
dbsmsg(msg)
dbsdump()
```

Arguments

cls	A cluster for the **parallel** package.
nwrkrs	Number of workers, i.e. size of the cluster.
xterm	The string "xterm" or name of compatible terminal.
src	Name of the source file to be debugged.
ftn	Name of the function to be debugged.
cmd	R command to be executed in manager screen.
msg	A message to write to the debugging record file. Can be either a character string or any expression that is printable by cat.

Details

A major obstacle to debugging cluster-based **parallel** applications is the lack of a terminal, thus precluding direct use of debug and browser. This set of functions consists of two groups, one for "quick and dirty" debugging, that writes debugging information to disk files, and the other for more sophisticated work that deals with the terminal restriction. For both methods, make sure setclsinfo has been called.

For "quick and dirty" debugging, there is dbsmsg, which prints messages to files, invoked from within code running at the cluster nodes. There is one file for each member of the cluster, e.g.

dbs.001, dbs.002 and so on, and dbsmsg writes to the file associated with the worker invoking it. Initialize via dbsmsgstart.

Another quick approach is to call dbsdump, which will call R's dump.frames, making a separate output file for each cluster node. These can then be input to debugger to examine stack frames.

The more elaborate debugging tool, dbs, is the only one in this **partools** package requiring a Unix-family system (Linux, Mac). To discuss it, suppose you wish to debug the function f in the file x.R. Run, say, dbs(2,xterm="xterm",src="x.R",ftn="f"). Then three new terminal windows will be created, one for the cluster manager and two for the cluster workers. The cluster will be named cls. Automatically, the file x.R will be sourced by the worker windows, and debug(f) will be run in them.

Then you simply debug as usual. Go to the manager window, and run your **parallel** application launch call in the usual way, say clusterEvalQ(cls,f(5)). The function f will run in each worker window, with execution automatically entering browser mode. You are now ready to single-step through them, or execute any other browser operation.

If xterm is NULL, you will be prompted to create the terminal windows by hand (or use existing ones), and run screen there as instructed. Terminal works on Macs; label the windows by hand, by clicking "Shell" then "Edit".

When finished with the debugging session, run killdebug from the original window (the one from which you invoked dbs) to quit the various screen processes.

Author(s)

Norm Matloff

Examples

```
## Not run:
# quick-and-dirty method
cls <- makeCluster(2)
setclsinfo(cls)
# define 'buggy' function
g <- function(x,y) {u<-x+y; v<-x-y; dbsmsg(c(u,v)); u^2+v^2}
clusterExport(cls,"g")
# set x and y at cluster nodes
clusterEvalQ(cls,{x <- runif(1); y <- runif(1)})
# start debugging session
dbsmsgstart(cls)
# run
clusterEvalQ(cls,g(x,y))
# files dbs.1 and dbs.2 created, each reporting u,v values

# dbs() method
# make a test file
cat(c("f <- function(x) {","   x <- x + 1","   x^2","}"),file="x.R",sep="\n")
dbs(2,src="x.R",ftn="f")
# now type in manager window:
clusterEvalQ(cls,f(5))
# the 2 worker windows are now in the browser, ready for debugging
```

```
stopCluster(cls)

## End(Not run)
```

---

formrowchunks,addlists,matrixtolist,setclsinfo,getpte,distribsplit,distribcat,distribagg,distribrange

Utilities for **parallel** *cluster code.*

Description

Miscellaneous code snippets for use with the **parallel** package, including "Snowdoop."

Usage

```
formrowchunks(cls,m,mchunkname,scramble=FALSE)
matrixtolist(rc,m)
addlists(lst1,lst2,add)
setclsinfo(cls)
getpte()
exportlibpaths(cls)
distribsplit(cls,dfname,scramble=FALSE)
distribcat(cls,dfname)
distribagg(cls,ynames,xnames,dataname,FUN,FUNdim=1,FUN1=FUN)
distribrange(cls,vec,na.rm=FALSE)
distribcounts(cls,xnames,dataname)
distribmeans(cls,ynames,xnames,dataname,saveni=FALSE)
dwhich.min(cls,vecname)
dwhich.max(cls,vecname)
distribgetrows(cls,cmd)
distribisdt(cls,dataname)
docmd(toexec)
doclscmd(cls,toexec)
geteltis(lst,i)
ipstrcat(str1 = stop("str1 not supplied"), ..., outersep = "", innersep = "")
```

Arguments

cls	A cluster for the **parallel** package.
scramble	If TRUE, randomize the row order in the resulting data frame.
rc	Set to 1 for rows, other for columns.
m	A matrix or data frame.
mchunkname	Quoted name to be given to the created chunks.
lst1	An R list.
lst2	An R list.
add	"Addition" function, which could be summation, concatenation and so on.

dfname	Quoted name of a data frame, either centralized or distributed.
ynames	Vector of quoted names of variables on which FUN is to be applied.
vecname	Quoted name of a vector.
...	One of more vectors of character strings, where the vectors are typically of length 1.
xnames	Vector of quoted names of variables that define the grouping.
dataname	Quoted name of a distributed data frame or data.table.
saveni	If TRUE, save the chunk sizes.
FUN	Quoted name of a single-argument function to be used in aggregating within cluster nodes. If dataname is the name of a data.table, FUN must be a vector of function names, of length equal to that of ynames.
FUNdim	Number of elements in the return value of FUN. Must be 1 for data.tables.
FUN1	Quoted name of function to be used in aggregation between cluster nodes.
vec	Quoted expression that evaluates to a vector.
na.rm	Remove NA values.
cmd	An R command.
toexec	Quoted string containing command to be executed.
lst	An R list of vectors.
i	A column index
str1	A character string.
outersep	Separator, e.g. a comma, between strings specified in ...
innersep	Separator, e.g. a comma, within string vectors specified in ...

Details

The setclsinfo function does initialization needed for use of the tools in the package.

The function formrowchunks forms chunks of rows of m, corresponding to the number of worker nodes in the cluster m. For any given worker, the code places its chunk in mchunk in the global space of the worker.

A call to matrixtolist extracts the rows or columns of a matrix or data frame and forms an R list from them.

The function addlists does the following: Say we have two lists, with numeric values. We wish to form a new list, with all the keys (names) from the two input lists appearing in the new list. In the case of a key in common to the two lists, the value in the new list will be the sum of the two individual values for that key. (Here "sum" means the result of applying add.) For a key appearing in one list and not the other, the value in the new list will be the value in the input list.

The function exportlibpaths, invoked from the manager, exports the manager's R search path to the workers.

The function distribsplit splits a data frame dfname into approximately equal-sized chunks of rows, placing the chunks on the cluster nodes, as global variables of the same name. The opposite action is taken by distribcat, coalsecing variables of the given name in the cluster nodes into one grand data frame as the calling (i.e. manager) node.

The package's distribagg function is a distributed (and somewhat restricted) form of aggregate. The latter is called to each distributed chunk with the function FUN. The manager collects the results and calls FUN1.

The special cases of aggregating counts and means is handled by the wrappers distribcounts and distribmeans. In each case, cells are defined by xnames, and aggregation done first within workers and then across workers.

The distribrange function is a distributed form of range.

The dwhich.min and dwhich.max functions are distributed analogs of R's which.min and which.max.

The distribgetrows function is useful in a variety of situations. It can be used, for instance, as a distributed form of select. In the latter case, the specified rows will be selected at each cluster node, then rbind-ed together at the caller.

The docmd function executes the quoted command, useful for building up complex command for remote execution. The doclscmd function does that directly.

An R formula will be constructed from the arguments ynames and xnames, with the latter put on the left side of the ~ sign, with cbind for combining, and the latter put on the right side, with + signs as delimiters.

The geteltis function extracts from an R list of vectors element i from each.

Value

In the case of addlists, the return value is the new list.

The distribcat function returns the concatenated data frame; distribgetrows works similarly.

The distribagg function returns a data frame, the same as would a call to aggregate, though possibly in different row order; distribcounts works similarly.

The dwhich.min and dwhich.max functions each return a two-tuple, consisting of the node number and row number which node at which the min or max occurs.

Author(s)

Norm Matloff

Examples

```
# examples of addlists()
l1 <- list(a=2, b=5, c=1)
l2 <- list(a=8, c=12, d=28)
addlists(l1,l2,sum)  # list with a=10, b=5, c=13, d=28
z1 <- list(x = c(5,12,13), y = c(3,4,5))
z2 <- list(y = c(8,88))
addlists(z1,z2,c)  # list with x=(5,12,13), y=(3,4,5,8,88)

# need 'parallel' cluster for the remaining examples
cls <- makeCluster(2)
setclsinfo(cls)

# check it
clusterEvalQ(cls,partoolsenv$myid)  # returns 1, 2
```

```
clusterEvalQ(cls,partoolsenv$ncls)  # returns 2, 2

# formrowchunks example; see up a matrix to be distributed first
m <- rbind(1:2,3:4,5:6)
# apply the function
formrowchunks(cls,m,"mc")
# check results
clusterEvalQ(cls,mc)  # list of a 1x2 and a 2x2 matrix

matrixtolist(1,m)  # 3-component list, first is (1,2)

# test of of distribagg():
# form and distribute test data
x <- sample(1:3,10,replace=TRUE)
y <- sample(0:1,10,replace=TRUE)
u <- runif(10)
v <- runif(10)
d <- data.frame(x,y,u,v)
distribsplit(cls,"d")
# check that it's there at the cluster nodes, in distributed form
clusterEvalQ(cls,d)
d
# try the aggregation function
distribagg(cls,c("u","v"), c("x","y"),"d","max")
# check result
aggregate(cbind(u,v) ~ x+y,d,max)

# real data
mtc <- mtcars
distribsplit(cls,"mtc")

distribagg(cls,c("mpg","disp","hp"),c("cyl","gear"),"mtc","max")
# check
aggregate(cbind(mpg,disp,hp) ~ cyl+gear,data=mtcars,FUN=max)

distribcounts(cls,c("cyl","gear"),"mtc")
# check
table(mtc$cyl,mtc$gear)

# find mean mpg, hp for each cyl/gear combination
distribmeans(cls,c('mpg','hp'),c('cyl','gear'),'mtc')

# extract and collect all the mtc rows in which the number of cylinders is 8
distribgetrows(cls,'mtc[mtc$cyl == 8,]')
# check
mtc[mtc$cyl == 8,]

# same for data.tables
mtc <- as.data.table(mtc)
setkey(mtc,cyl)
distribsplit(cls,'mtc')
distribcounts(cls,c("cyl","gear"),"mtc")
distribmeans(cls,c('mpg','hp'),c('cyl','gear'),'mtc')
```

```
dwhich.min(cls,'mtc$mpg')  # smallest is at node 1, row 15
dwhich.max(cls,'mtc$mpg')  # largest is at node 2, row 4

stopCluster(cls)
```

`newadult` *UCI adult income data set, adapted*

Description

This data set is adapted from the Adult data from the UCI Machine Learning Repository, which was in turn adapted from Census data on adult incomes and other demographic variables. The UCI data is used here with permission from Ronny Kohavi.

The variables are:

- `gt50`, which converts the original `>50K` variable to an indicator variable; 1 for income greater than $50,000, else 0
- `edu`, which converts a set of education levels to approximate number of years of schooling
- `age`
- `gender`, 1 for male, 0 for female
- `mar`, 1 for married, 0 for single

Usage

```
data(newadult); newadult
```

`parpdist` *Partools Apps*

Description

General parallel applications.

Usage

```
parpdist(x,y,cls)
```

Arguments

`cls`	A cluster run under the **parallel** package.
`x`	A data matrix
`y`	A data matrix

Details

Parallel wrapper for `pdist` from package of the same name. Finds all the distances from rows in `x` to rows in `y`.

Value

Object of type `"pdist"`.

Author(s)

Norm Matloff

Examples

```
# set up 'parallel' cluster
cls <- makeCluster(2)
setclsinfo(cls)

x <- matrix(runif(20),nrow=5)
y <- matrix(runif(32),nrow=8)
# 2 calls should have identical resultsW
pdist(x,y,cls)@dist
parpdist(x,y,cls)@dist

stopCluster(cls)
```

---

prgeng *Silicon Valley programmers and engineers*

Description

This data set is adapted from the 2000 Census (5% sample, person records). It is restricted to programmers and engineers in the Silicon Valley area.

The variable codes, e.g. occupational codes, are available from the Census Bureau, at http://www.census.gov/prod/cen2000/doc/pums.pdf. (Short code lists are given in the record layout, but longer ones are in the appendix Code Lists.)

The variables are:

- `age`, with a U(0,1) variate added for jitter
- `cit`, citizenship; 1-4 code various categories of citizens; 5 means noncitizen (including permanent residents
- `educ`: 01-09 code no college; 10-12 means some college; 13 is a bachelor's degree, 14 a master's, 15 a professiona deal and 16 is a doctorate
- `occ`, occupation
- `birth`, place of birth

- wageinc, wage income
- wkswrkd, number of weeks worked
- yrentry, year of entry to the U.S. (0 for natives)
- powpuma, location of work
- gender, 1 for male, 2 for female

Usage

```
data(prgeng); prgeng
```

`snowdoop,filechunkname, etc...` *Snowdoop.*

Description

"Snowdoop": Utilities for distributed file storage, access and related operations.

Usage

```
filechunkname(basenm,ndigs,nodenum=NULL)
filesort(cls,infilenm,colnum,outdfnm,infiledst=FALSE,
   ndigs=0,nsamp=1000,header=FALSE,sep="",usefread=FALSE)
filesplit(nch,basenm,header=FALSE,seqnums=FALSE)
filesplitrand(cls,fname,newbasename,ndigs,header=FALSE,sep)
fileshuffle(inbasename, nout, outbasename, header = FALSE)
linecount(infile,header=FALSE,chunksize=100000)
filecat(cls, basenm, header = FALSE)
readnscramble(cls,basenm,header=FALSE,sep= " ")
filesave(cls,dname,newbasename,ndigs,sep)
fileread(cls,fname,dname,ndigs,header=FALSE,sep=" ",usefread=FALSE)
getnumdigs(nch)
fileagg(fnames,ynames,xnames,header=FALSE,sep= " ",FUN,FUN1=FUN)
dfileagg(cls,fnames,ynames,xnames,header=FALSE,sep=" ",FUN,FUN1=FUN)
filegetrows(fnames,tmpdataexpr,header=FALSE,sep=" ")
dfilegetrows(cls,fnames,tmpdataexpr,header=FALSE,sep=" ")
```

Arguments

cls	A cluster for the **parallel** package.
nch	Number of chunks for the file split.
basenm	A chunked file name, minus suffix.
infile	Name of a nonchunked file.
ndigs	Number of digits in the chunked file name suffix.

nodenum	If non-NULL, get the name of the file chunk of cluster node nodenum; otherwise, get the name for the chunk associated with this node.
infilenm	Name of input file (without suffix, if distributed).
outdfnm	Name of output file (without suffix).
infiledst	If TRUE, infilenm is distributed.
colnum	Column number on which the sort will be done. It is assumed that this data column is free of NAs.
usefread	If true, use fread instead of read.table; generally much faster; requires data.table package.
nsamp	Number of records to sample in each file chunk to determine bins for the bucket sort.
header	TRUE if the file chunks have headers.
seqnums	TRUE if the file chunks will have sequence numbers.
sep	Field delimiter used in read.table.
chunksize	Number of lines to read at a time, for efficient I/O.
dname	Quoted name of a distributed data frame or matrix. For filesave, the object must have column names.
fname	Quoted name of a distributed file.
fnames	Character vector of file names.
newbasename	Quoted name of the prefix of a distributed file, e.g. xyz for a distributed file xyz.01, xyz.02 etc.
ynames	Vector of quoted names of variables on which FUN is to be applied.
xnames	Vector of quoted names of variables to be used for cell definition.
tmpdataexpr	Expression involving a data frame tmpdataexpr. See below.
FUN	First-level aggregation function.
FUN1	Second-level aggregation function.
inbasename	basename of the input files, e.g. x for x.1, x.2, ...
outbasename	basename of the output files
nout	number of output files

Details

Use filesplit to convert a single file into distributed one, with nch chunks. The file header, if present, will be retained in the chunks. If seqnums is TRUE, each line in a chunk will be preceded by the line number it had in the original file.

The reverse operation to filesplit is performed by filecat, which converts a distributed file into a single one.

The fileagg function does an out-of-memory, multifile version of aggregate, reading the specified files one at a time, and returning a grand aggregation. The function dfileagg partitions the specified group of files to a partools cluster, has each call fileagg, and again aggregates the results.

The function `filegetrows` reads in the files in `fnames`, one at a time, naming the resulting in-memory data `tmpdata` each time. (It is assumed that the data fit in memory.) The function applies the user command `tmpdataexpr` to `tmpdata`, producing a subset of `tmpdata`. All of these subsets are combined using `rbind`, yielding the return value. The paired function `dfilegetrows` is a distributed wrapper for `filegetrows`, just as `dfileagg` is for `fileagg`.

Use `filesort` to do a file sort, with the input file being either distributed or ordinary, placing the result as a distributed data frame/matrix in the memories of the cluster nodes. The first `nsamp` records are read from the file, and are used to form one quantile range for each cluster node. Each node then reads the input file, retaining the records in its assigned range, and sorts them. This results in the input file being sorted, in memory, in a distributed manner across nodes, under the specifid name. At present, this utility is not very efficient.

Operations such as `ca` need i.i.d. data. If the original file storage was ordered on some variable, one needs to randomize the data first. There are several options:

- `readnscramble`: This produces a distributed data frame/matrix under the name `basenm`. Note that a record in chunk `i` of the distributed file will likely end up in chunk `j` in the distributed data frame/matrix, with `j` different from `i`.
- `filesplitrand`: Use this you wish to directly produce a randomized distributed file from a monolithic one. It will read the file into memory, chunk it at the cluster nodes, each of which will save its chunk to disk.
- `fileshuffle`: If you need to avoid reading big files into memory, use this. You must run `filesplit` first, and then run `fileshuffle` several times for a good shuffle.

  Note that this function is also useful if your cluster size changes. A distributed file of m chunks can now be converted to one with n chunks, either more or fewer than before.

If you wish to use this same randomized data in a future session, you can save it as a distributed file by calling `filesave`. Of course, this function is also useful if one wishes to save a distributed data frame or matrix that was created computationally rather than from read from a distributed file. To go the other direction, i.e. read a distributed file, use `fileread`.

Some of the functions here are useful mainly as intermediate operations for the others:

- The function `filechunkname` returns the name of the file chunk for the calling cluster node.
- The `linecount` function returns the number of lines in a text file.
- A call to `getnumdigs` returns the number of digits in a distributed file name suffix.

Author(s)

Norm Matloff

Examples

```
cls <- makeCluster(2)
setclsinfo(cls)

# example of filesplit()
# make test input file
m <- rbind(1:2,3:4,5:6)
write.table(m,"m",row.names=FALSE,col.names=FALSE)
```

```
# apply the function
filesplit(2,"m",seqnums=TRUE)
# file m.1 and m.2 created, with contents c(1,1,2) and
# rbind(c(2,3,4),c(3,5,6)), respectively
# check it
read.table("m.1",header=FALSE,row.names=1)
read.table("m.2",header=FALSE,row.names=1)
m

# example of filecat(); assumes filesplit() example above already done
# delete file m so we can make sure we are re-creating it
unlink("m")
filecat(cls,"m")
# check that file m is back
read.table("m",row.names=1)

# example of filesave(), fileread()
# make test distributed data frame
clusterEvalQ(cls,x <- data.frame(u = runif(5),v = runif(5)))
# apply filesave()
filesave(cls,'x','xfile',1,' ')
# check it
fileread(cls,'xfile','xx',1,header=TRUE,sep=' ')
clusterEvalQ(cls,xx)
clusterEvalQ(cls,x)


# example of filesort()
# make test distributed input file
m1 <- matrix(c(5,12,13,3,4,5,8,8,8,1,2,3,6,5,4),byrow=TRUE,ncol=3)
m2 <- matrix(c(0,22,88,44,5,5,2,6,10,7,7,7),byrow=TRUE,ncol=3)
write.table(m1,"m.1",row.names=FALSE)
write.table(m2,"m.2",row.names=FALSE)
# sort on column 2 and check result
filesort(cls,"m",2,"msort",infiledst=TRUE,ndigs=1,nsamp=3,header=TRUE)
clusterEvalQ(cls,msort)  # data should be sorted on V2
# check by comparing to input
m1
m2
m <- rbind(m1,m2)
write.table(m,"m",row.names=FALSE)
clusterEvalQ(cls,rm(msort))
filesort(cls,"m",2,"msort",infiledst=FALSE,nsamp=3,header=TRUE)
clusterEvalQ(cls,msort)  # data should be sorted on V2

# example of readnscramble()
co2 <- head(CO2,25)
write.table(co2,"co2",row.names=FALSE)  # creates file 'co2'
filesplit(2,"co2",header=TRUE)  # creates files 'co2.1', 'co2.2'
readnscramble(cls,"co2",header=TRUE)  # now have distrib. d.f.
# save the scrambled version to disk
filesave(cls,'co2','co2s',1,sep=',')
```

```
# example of fileshuffle()
# make test file, 'test'
cat('a','bc','def','i','j','k',file='test',sep='\n')
filesplit(2,'test')  # creates files 'test.1','test.2'
fileshuffle('test',2,'testa')  # creates shuffled files 'testa.1','testa.2'

# example of filechunkname()
clusterEvalQ(cls,filechunkname("x",3))  # returns "x.001", "x.002"

# example of getnumdigs()
getnumdigs(156)  # should be 3

# examples of filesave() and fileread()
mtc <- mtcars
distribsplit(cls,"mtc")
# save distributed data frame to distributed file
filesave(cls,'mtc','ctm',1,',')
# read it back in to a new distributed data frame
fileread(cls,'ctm','ctmnew',1,header=TRUE,sep=',')
# check it
clusterEvalQ(cls,ctmnew)
# try dfileagg() on it (not same as distribagg())
dfileagg(cls,c('ctm.1','ctm.2'),c("mpg","disp","hp"),c("cyl","gear"),header=TRUE,sep=",","max")
# check
aggregate(cbind(mpg,disp,hp) ~ cyl+gear,data=mtcars,FUN=max)
# extract the records with 4 cylinders and 4 gears (again, different
# from distribgetrows())
cmd <- 'tmpdata[tmpdata$cyl == 4 & tmpdata$gear == 4,]'
dfilegetrows(cls,c('ctm.1','ctm.2'),cmd,header=TRUE,sep=',')
# check
mtc[mtc$cyl == 4 & mtc$gear == 4,]

stopCluster(cls)
```

Index