Package 'fileplyr'

February 20, 2017

Version 0.2.0

Title Chunk Processing or Split-Apply-Combine on Delimited Files and Distributed Dataframes

Description Perform chunk processing or split-apply-combine on data in a delimited file (example: CSV) and Distributed Dataframes (DDF) across multiple cores of a single machine with low memory footprint. These functions are a convenient wrapper over the versatile package 'datadr'.

Imports datadr (>= 0.8.6), assertthat (>= 0.1), parallel

Depends R (>= 3.3.2), tibble (>= 1.2), utils

License GPL-3

URL https://github.com/talegari/fileplyr

BugReports https://github.com/talegari/fileplyr/issues

RoxygenNote 5.0.1

NeedsCompilation no

Author KS Srikanth [aut, cre]

Maintainer KS Srikanth <sri.teach@gmail.com>

Repository CRAN

Date/Publication 2017-02-20 08:11:20

R topics documented:

`fileplyr-package` *fileplyr*

Description

A package to perform chunk processing or split-apply-combine on data in a delimited file(example: CSV) across multiple cores of a single machine with low memory footprint. These functions are a convenient wrapper over the versatile package **datadr**.

Memory vs time tradeoff: Handling data that cannot be loaded into RAM involves a tradeoff with memory usage versus time required to complete the computation task. fileplyr lies at one end of the spectrum where it is assumed that the user has time at disposal as opposed to hardware facility. The size of the file to be processed has to be large enough to observe the advantage of the package.

Audience: Analysts with laptop/desktop with multiple cores and limited RAM(and lots of time) might find the package helpful for prototyping tasks. A proper estimation of memory usage will let other applications to function comfortably along with a R process running a fileplyr job.

`ddfply` *ddfply*

Description

performs chunk processing or split-apply-combine on the data in a distributed data frame(ddf)

Usage

```
ddfply(ddfdir, groupby, fun = identity, collect = "none",
  temploc = getwd(), nbins = 10, chunk = 50000, spill = 1e+06,
  cores = 1, buffer = 1e+09, ...)
```

Arguments

`ddfdir`	(string) path of ddf directory
`groupby`	(character vector) Columns names to used to split the data(if missing, `fun` is applied on each chunk)
`fun`	(object of class *function*) function to apply on each subset after the split
`collect`	(string) Collect the result as `list` or `dataframe` or `none`. `none` keeps the resulting ddo on disk.
`temploc`	(string) Path where intermediary files are kept
`nbins`	(positive integer) Number of directories into which the distributed dataframe (ddf) or distributed data object (ddo) is distributed
`chunk`	(positive integer) Number of rows of the file to be read at a time
`spill`	(positive integer) Maximum number of rows of any subset resulting from split

cores	(positive integer) Number of cores to be used in parallel
buffer	(positive integer) Size of batches of key-value pairs to be passed to the map OR Size of the batches of key-value pairs to flush to intermediate storage from the map output OR Size of the batches of key-value pairs to send to the reduce
...	Arguments to be passed to `data.table` function asis.

Details

see `fileply`

Value

list or a dataframe or a TRUE(when collect is 'none').

Examples

```
write.table(mtcars, "mtcars.csv", row.names = FALSE, sep = ",")
# create a ddf by keeping `keepddf = TRUE`
co <- capture.output(temp <- fileply("mtcars.csv"
                                     , groupby = c("carb", "gear")
                                     , fun     = identity
                                     , collect = "list"
                                     , sep     =  ","
                                     , header  = TRUE
                                     , keepddf = TRUE)
                     , file = NULL
                     , type = "message"
                     )
# use the ddf instead of reading the CSV again
temp2 <- ddfply(file.path(strsplit(co[6], ": ")[[1]][2], "data")
                , groupby = c("gear")
                , fun     = identity
                , collect = "list"
                , sep     =  ","
                , header  = TRUE
                )
temp2
unlink("mtcars.csv")
unlink(strsplit(co[6], ": ")[[1]][2], recursive = TRUE)
```

`fileply` *fileply*

Description

performs chunk processing or split-apply-combine on the data in a delimited file (example: CSV).

Usage

```
fileply(file, groupby, fun = identity, collect = "none",
  temploc = getwd(), nbins = 10, chunk = 50000, spill = 1e+06,
  cores = 1, buffer = 1e+09, keepddf = FALSE, ...)
```

Arguments

file	(string) path to input delimited file
groupby	(character vector) Columns names to used to split the data(if missing, fun is applied on each chunk)
fun	(object of class *function*) function to apply on each subset after the split
collect	(string) Collect the result as list or dataframe or none. none keeps the resulting ddo on disk.
temploc	(string) Path where intermediary files are kept
nbins	(positive integer) Number of directories into which the distributed dataframe (ddf) or distributed data object (ddo) is distributed
chunk	(positive integer) Number of rows of the file to be read at a time
spill	(positive integer) Maximum number of rows of any subset resulting from split
cores	(positive integer) Number of cores to be used in parallel
buffer	(positive integer) Size of batches of key-value pairs to be passed to the map OR Size of the batches of key-value pairs to flush to intermediate storage from the map output OR Size of the batches of key-value pairs to send to the reduce
keepddf	(flag) whether to save the distributed dataframe (on the disk)
...	Arguments to be passed to data.table function asis.

Details

- **Reading Stage:** The delimited file (example: CSV) is read in smaller chunks to create a distributed dataframe or **ddf** on disk. The number of lines read at once is specified by chunk argument and nbins specify the number of sub-directories the data is distributed. The ... are additional inputs to data.table function for reading the delimited file.
- **Split and Apply Stage:** The variables in groupby are used to split the data and load only the subset(possibly many if multiple cores are in action) into the memory. If groupby is missing, chunkwise processing is performed on each subset of the distributed dataframe. A user defined fun is applied and results are written to a distributed object(list or a KV pairs) on disk.
- **Combine Stage:** The distributed data object(**ddo**) is read into memory depending on collect argument. The default is set to 'none' which would not the data back into memory.

Memory usage: While processing heavy files(many times the RAM size), each core might hold a maximum of 800 MB to 1GB of memory overhead without accounting for the memory used by the user defind function. Memory usage depending on size of the subset, how many times it is copied by the user function, how frequently is gc called. Using appropriate number of cores keeps memory utilization in check. Setting a smaller buffer value keeps memory usage low, see localDiskControl, but makes the execution slower.

Value

list or a dataframe or a TRUE(when collect is 'none').

Examples

```
# split-apply-combine
write.table(mtcars, "mtcars.csv", row.names = FALSE, sep = ",")
temp <- fileply(file     = "mtcars.csv"
             , groupby = c("carb", "gear")
             , fun     = identity
             , collect = "list"
             , sep     =  ","
             , header  = TRUE
             )
temp
unlink("mtcars.csv")

# chunkwise processing
write.table(mtcars, "mtcars.csv", row.names = FALSE, sep = ",")
temp <- fileply(file     = "mtcars.csv"
             , chunk   = 10
             , fun     = function(x){list(nrow(x))}
             , collect = "dataframe"
             , sep     =  ","
             , header  = TRUE
             )
temp
unlink("mtcars.csv")

# example for collect='none'
write.table(mtcars, "mtcars.csv", row.names = FALSE, sep = ",")
outdir <- utils::capture.output(temp <- fileply(file      = "mtcars.csv"
                                             , groupby = c("carb", "gear")
                                             , fun     = identity
                                             , sep     =  ","
                                             , header  = TRUE
                                             )
                                , file = NULL
                                , type = "message"
                                )
outdir <- gsub("Output Directory: ", "", outdir[5])
diskKV <- datadr::ddo(datadr::localDiskConn(outdir))
diskKV
diskKV[[1]]
unlink(outdir, recursive = TRUE)
unlink("mtcars.csv")
```

---

to_tibble *to_tibble*

Description

convert list output of fileply to an object of class tibble

Usage

```
to_tibble(object, valuename = "value")
```

Arguments

object	list output of fileply
valuename	name of the value list-column

Value

an object of class tibble

Examples

```
# split-apply-combine
write.table(mtcars, "mtcars.csv", row.names = FALSE, sep = ",")
temp <- fileply(file     = "mtcars.csv"
             , groupby = c("carb", "gear")
             , fun     = identity
             , collect = "list"
             , sep     =  ","
             , header  = TRUE
             )
temp
to_tibble(temp)
unlink("mtcars.csv")
```

Index