

Package ‘lazyarray’

July 18, 2020

Type Package

Title Persistent Large Data Array with Lazy-Loading on Demand

Version 1.1.0

Language en-US

License AGPL-3

Encoding UTF-8

SystemRequirements C++11 little-endian platform

RoxygenNote 7.1.1

URL <https://github.com/dipterix/lazyarray>

BugReports <https://github.com/dipterix/lazyarray/issues>

Description Multi-threaded serialization of compressed array that fully utilizes modern solid state drives. It allows to store and load extremely large data on demand within seconds without occupying too much memories. With data stored on hard drive, a lazy-array data can be loaded, shared across multiple R sessions. For arrays with partition mode on, multiple R sessions can write to a same array simultaneously along the last dimension (partition). The internal storage format is provided by 'fstcore' package geared by 'LZ4' and 'ZSTD' compressors.

Imports Rcpp (>= 1.0.4), R6, fstcore, yaml

LinkingTo Rcpp, fstcore

Suggests testthat, knitr, fst, rmarkdown, dipsaus (>= 0.0.8)

VignetteBuilder knitr

NeedsCompilation yes

Author Zhengjia Wang [aut, cre, cph],
Mark Klik [ctb, cph] (Copyright holder of fstcore package)

Maintainer Zhengjia Wang <dipterix.wang@gmail.com>

Repository CRAN

Date/Publication 2020-07-18 06:10:02 UTC

R topics documented:

auto_clear_lazyarray	2
ClassLazyArray	3
create_lazyarray	7
lazyarray	8
load_lazyarray	11
set_lazy_threads	13

Index	15
--------------	-----------

auto_clear_lazyarray *Automatically remove array data*

Description

Remove the files containing array data once no 'lazyarray' instance is using the folder. Require installation of dipsaus package (at least version 0.0.8).

Usage

```
auto_clear_lazyarray(x, onexit = FALSE)
```

Arguments

x	'lazyarray' instance
onexit	passed to <code>reg.finalizer</code>

Details

auto_clear_lazyarray attempts to remove the entire folder containing array data. However, if some files are not created by the array, only partition data and meta file will be removed, all the artifacts will remain and warning will be displayed. One exception is if all files left in the array directory are *.meta files, all these meta files will be removed along with the folder.

Author(s)

Zhengjia Wang

Examples

```
path <- tempfile()
arr_dbl <- lazyarray(path, storage_format = 'double',
                    dim = 2:4, meta_name = 'meta-dbl.meta')
arr_dbl[] <- 1:24
auto_clear_lazyarray(arr_dbl)

arr_chr <- lazyarray(path, storage_format = 'character',
```

```
        dim = 2:4, meta_name = 'meta-chr.meta',
        quiet = TRUE)
auto_clear_lazyarray(arr_chr)

# remove either one, the directory still exists
rm(arr_dbl); invisible(gc(verbose = FALSE))

arr_chr[1,1,1]

# Remove the other one, and path will be removed
rm(arr_chr); invisible(gc(verbose = FALSE))

dir.exists(path)
arr_check <- lazyarray(path, storage_format = 'character',
                      dim = 2:4, meta_name = 'meta-chr',
                      quiet = TRUE)

# data is removed, so there should be no data (NAs)
arr_check[]
```

ClassLazyArray

Internal Class definition for LazyArray

Description

Internal class definition of lazy array objects

Active bindings

meta_name file name to store meta information

min_version minimal version supported, for backward compatibility concerns

version current version of lazy data instance

dim dimension of the data

dimnames dimension names of the data

ndim length of dimensions

can_write is array read-only or writable

storage_path directory where the data is stored at

Methods

Public methods:

- [ClassLazyArray#print\(\)](#)
- [ClassLazyArray\\$new\(\)](#)
- [ClassLazyArray\\$flag_auto_clean\(\)](#)
- [ClassLazyArray\\$finalize\(\)](#)

- `ClassLazyArray$remove_data()`
- `ClassLazyArray$make_writable()`
- `ClassLazyArray$make_readonly()`
- `ClassLazyArray$set_dim()`
- `ClassLazyArray$get_file_format()`
- `ClassLazyArray$get_storage_format()`
- `ClassLazyArray$is_multi_part()`
- `ClassLazyArray$partition_dim()`
- `ClassLazyArray$get_partition_fpath()`
- `ClassLazyArray$@set_data()`
- `ClassLazyArray$set_compress_level()`
- `ClassLazyArray$get_compress_level()`
- `ClassLazyArray$@get_data()`
- `ClassLazyArray$@sample_data()`
- `ClassLazyArray$clone()`

Method print(): Override print method

Usage:

```
ClassLazyArray#print(...)
```

Arguments:

... ignored

Returns: self instance

Method new(): Constructor

Usage:

```
ClassLazyArray$new(path, read_only = TRUE, meta_name = "lazyarray.meta")
```

Arguments:

path directory to store data into

read_only whether modification is allowed

meta_name meta file to store the data into

Method flag_auto_clean(): Set auto clean flag

Usage:

```
ClassLazyArray$flag_auto_clean(auto)
```

Arguments:

auto logical whether the data on hard disk will be automatically cleaned

Method finalize(): Override finalize method

Usage:

```
ClassLazyArray$finalize()
```

Method remove_data(): Remove data on hard disk

Usage:

ClassLazyArray\$remove_data(force = FALSE, warn = TRUE)

Arguments:

force whether to force remove the data

warn whether to show warning if not fully cleaned

Method make_writable(): Make instance writable

Usage:

ClassLazyArray\$make_writable()

Method make_readonly(): Make instance read-only

Usage:

ClassLazyArray\$make_readonly()

Method set_dim(): Set [dim](#) and [dimnames](#) of the array

Usage:

ClassLazyArray\$set_dim(dim, dimnames)

Arguments:

dim integer vector of the array dimension; see [dim](#)

dimnames named list of dimension names; see [dimnames](#)

Method get_file_format(): Partition format, currently only 'fst' is supported

Usage:

ClassLazyArray\$get_file_format()

Method get_storage_format(): Data storage format, expected to be one of the followings: 'double', 'integer', 'character', or 'complex'

Usage:

ClassLazyArray\$get_storage_format()

Method is_multi_part(): Whether partitioned based on the last dimension

Usage:

ClassLazyArray\$is_multi_part()

Method partition_dim(): Returns dimension of each partition

Usage:

ClassLazyArray\$partition_dim()

Method get_partition_fpath(): Get partition path

Usage:

ClassLazyArray\$get_partition_fpath(part, full_path = TRUE)

Arguments:

part integer representing the partition

full_path whether return the full system path

Returns: Character file name or full path

Method @set_data(): Internal method to set data

Usage:

ClassLazyArray@set_data(value, ...)

Arguments:

value vector of data to be set

... index set

Method set_compress_level(): Set compression level

Usage:

ClassLazyArray\$set_compress_level(level)

Arguments:

level from 0 to 100. 0 means no compression, 100 means max compression

Method get_compress_level(): Get compression level

Usage:

ClassLazyArray\$get_compress_level()

Method @get_data(): Internal method to read data

Usage:

ClassLazyArray@get_data(..., drop = TRUE)

Arguments:

... index set

drop whether to drop dimension after subset, default is true

Method @sample_data(): Internal method to obtain a sample data to be used to determine storage mode

Usage:

ClassLazyArray@sample_data()

Method clone(): The objects of this class are cloneable with this method.

Usage:

ClassLazyArray\$clone(deep = FALSE)

Arguments:

deep Whether to make a deep clone.

Author(s)

Zhengjia Wang

create_lazyarray *Create a lazy-array with given format and dimension*

Description

Create a directory to store lazy-array. The path must be missing. See [load_lazyarray](#) for more details

Usage

```
create_lazyarray(
  path,
  storage_format,
  dim,
  dimnames = NULL,
  compress_level = 50L,
  prefix = "",
  multipart = TRUE,
  multipart_mode = 1,
  file_names = NULL,
  meta_name = "lazyarray.meta"
)
```

Arguments

path	path to a local drive to store array data
storage_format	data type, choices are "double", "integer", "character", and "complex"
dim	integer vector, dimension of array, see dim
dimnames	list of vectors, names of each dimension, see dimnames
compress_level	0 to 100, level of compression. 0 means no compression, 100 means maximum compression. For persistent data, it's recommended to set 100. Default is 50.
prefix	character prefix of array partition
multipart	whether to split array into multiple partitions, default is true
multipart_mode	1, or 2, mode of partition, see details.
file_names	data file names without prefix/extensions; see details.
meta_name	header file name, default is "lazyarray.meta"

Details

Lazy array stores array into hard drive, and load them on demand. It differs from other packages such as "bigmemory" that the internal reading uses multi-thread, which gains significant speed boost on solid state drives.

One lazy array contains two parts: data file(s) and a meta file. The data files can be stored in two ways: non-partitioned and partitioned.

For non-partitioned data array, the dimension is set at the creation of the array and cannot be mutable once created

For partitioned data array, there are also two partition modes, defined by ``multipart_mode``. For mode 1, each partition has the same dimension size as the array. The last dimension is 1. For example, a data with dimension `c(2,3,5)` partitioned with mode 1 will have each partition dimension stored with `c(2,3,1)`. For mode 2, the last dimension will be dropped when storing each partitions.

`file_names` is used when irregular partition names should be used. If `multipart=FALSE`, the whole array is stored in a single file under path. The file name is `<prefix><file_name>.fst`. For example, by default `prefix=""`, and `file_name=""`, then `path/.fst` stores the array data. If `multipart=TRUE`, then `file_names` should be a character vector of length equal to array's last dimension. A `3x4x5` array has 5 partitions, each partition name follows `<prefix><file_name>.fst` convention, and one can always use `arr$get_partition_fpath()` to find location of partition files. For examples, see [lazyarray](#).

Value

A `ClassLazyArray` instance

Author(s)

Zhengjia Wang

lazyarray

Create or load 'lazyarray' instance

Description

If path is missing, create a new array. If path exists and meta file is complete, load existing file, otherwise create new meta file and import from existing data.

Usage

```
lazyarray(
  path,
  storage_format,
  dim,
  dimnames = NULL,
  multipart = TRUE,
  prefix = "",
  multipart_mode = 1,
  compress_level = 50L,
  file_names = list("", seq_len(dim[[length(dim)]])[[multipart + 1]]),
  meta_name = "lazyarray.meta",
  read_only = FALSE,
  quiet = FALSE,
  ...
)
```


Arguments

path	path to a local drive where array data is stored
storage_format	data type, choices are "double", "integer", "character", and "complex"; see details
dim	integer vector, dimension of array, see dim
dimnames	list of vectors, names of each dimension, see dimnames
multipart	whether to split array into multiple partitions, default is true
prefix	character prefix of array partition
multipart_mode	1, or 2, mode of partition, see create_lazyarray
compress_level	0 to 100, level of compression. 0 means no compression, 100 means maximum compression. For persistent data, it's recommended to set 100. Default is 50.
file_names	partition names without prefix nor extension; see details
meta_name	header file name, default is "lazyarray.meta"
read_only	whether created array is read-only
quiet	whether to suppress messages, default is false
...	ignored

Details

There are three cases and lazyarray behaves differently under each cases. Case 1: if path is missing, then the function calls [create_lazyarray](#) to create a blank array instance. Case 2: if path exists and it contains meta_name, then load existing instance with given read/write access. In this case, parameters other than read_only, path, meta_name will be ignored. Case 3: if meta_name is missing and path is missing, then lazyarray will try to create arrays from existing data files.

If lazyarray enters case 3, then file_names will be used to locate partition files. Under multi-part mode (multipart=TRUE), file_names is default to 1, 2, ..., dim[length(dim)]. These correspond to '1.fst', '2.fst', etc. under path folder. You may specify your own file_names if irregular names are used. and file format for each partition will be <prefix><file_name>.fst. For example, a file name file_names=c('A', 'B') and prefix="file-" means the first partition will be stored as "file-A.fst", and "file-B.fst". It's fine if some files are missing, the corresponding partition will be filled with NA when trying to obtain values from those partition. However, length of file_names must equals to the last dimension when multipart=TRUE. If multipart=FALSE, file_names should have length 1 and the corresponding file is the data file.

It's worth note to import from existing partition files generated by other packages such as 'fst', the partition files must be homogeneous, meaning the stored data length, dimension, and storage type must be the same. Because 'fstcore' package stores data in data frame internally, the column name must be 'V1', 'V2', etc. for non-complex elements or 'V1R', 'V1I', ... for complex numbers (real and imaginary data are stored in different columns).

Author(s)

Zhengjia Wang

See Also

[create_lazyarray](#), [load_lazyarray](#)

Examples

```

path <- tempfile()

# ----- case 1: Create new array -----
arr <- lazyarray(path, storage_format = 'double', dim = c(2,3,4),
                meta_name = 'lazyarray.meta')
arr[] <- 1:24

# Subset and get the first partition
arr[,,1]

# Partition file path (total 4 partitions)
arr$get_partition_fpath()

# Removing array doesn't clear the data
rm(arr); gc()

# ----- Case 2: Load from existing directory -----
## Important!!! Run case 1 first
# Load from existing path, no need to specify other params
arr <- lazyarray(path, meta_name = 'lazyarray.meta', read_only = TRUE)

arr[,,1]

# ----- Case 3: Import from existing data -----
## Important!!! Run case 1 first

# path exists, but meta is missing, all other params are required
# Notice the partition count increased from 4 to 5, and storage type converts
# from double to character
arr <- lazyarray(path = path, meta_name = 'lazyarray-character.meta',
                file_names = c(1,2,3,4,'additional'),
                storage_format = 'character', dim = c(2,3,5),
                quiet = TRUE, read_only = FALSE)

# partition names
arr$get_partition_fpath(1:4, full_path = FALSE)
arr$get_partition_fpath(5, full_path = FALSE)

# The first dimension still exist and valid
arr[,,1]

# The additional partition is all NA
arr[,,5]

# Set data to 5th partition
arr[,,5] <- rep(0, 6)

```

```

# ----- Advanced usage: create fst data and import manually -----

# Clear existing files
path <- tempfile()
unlink(path, recursive = TRUE)
dir.create(path, recursive = TRUE)

# Create array of dimension 2x3x4, but 3rd partition is missing
# without using lazyarray package

# Column names must be V1 or V1R, V1I (complex)
fst::write_fst(data.frame(V1 = 1:6), path = file.path(path, 'part-1.fst'))
fst::write_fst(data.frame(V1 = 7:12), path = file.path(path, 'part-B.fst'))
fst::write_fst(data.frame(V1 = 19:24), path = file.path(path, 'part-d.fst'))

# Import via lazyarray
arr <- lazyarray(path, meta_name = 'test-int.meta',
                 storage_format = 'integer',
                 dim = c(2,3,4), prefix = 'part-',
                 file_names = c('1', 'B', 'C', 'd'),
                 quiet = TRUE)

arr[]

# Complex case
fst::write_fst(data.frame(V1R = 1:6, V1I = 1:6),
              path = file.path(path, 'cplx-1.fst'))
fst::write_fst(data.frame(V1R = 7:12, V1I = 100:105),
              path = file.path(path, 'cplx-2.fst'))
fst::write_fst(data.frame(V1R = 19:24, V1I = rep(0,6)),
              path = file.path(path, 'cplx-4.fst'))
arr <- lazyarray(path, meta_name = 'test-cplx.meta',
                 storage_format = 'complex',
                 dim = c(2,3,4), prefix = 'cplx-',
                 file_names = 1:4, quiet = TRUE)

arr[]

```

load_lazyarray

Load Lazy Array from Given Path

Description

Load Lazy Array from Given Path

Usage

```
load_lazyarray(path, read_only = TRUE, meta_name = "lazyarray.meta")
```

Arguments

path	character, path of the array
read_only	whether setting data is allowed
meta_name	header file name, default is "lazyarray.meta"

Value

A ClassLazyArray instance

Author(s)

Zhengjia Wang

Examples

```

path <- tempfile()
create_lazyarray(path, 'double', dim = c(3,4,5), multipart = TRUE)

x <- load_lazyarray(path, read_only = FALSE)
x[2,3:4, 2:1] <- 1:4
x[, , 2]

# Expend dimension for multiple partition data only
dim(x) <- c(3,4,6)
dimnames(x) <- list(dim1 = as.character(1:3),
                    dim2 = letters[1:4],
                    dim3 = LETTERS[1:6])
x[, , 'B', drop = FALSE]

# Non-standard subset methods
names(dimnames(x))
subset(x, dim1 ~ dim1 == '2', dim2 ~ dim2 %in% c('a', 'c'), drop = TRUE)

# Free up space
x$remove_data()

# This example needs at least 4 GB hard disk space and it takes
# time to run for performance profile

# Speed test
path <- tempfile()
x <- create_lazyarray(path, 'complex', dim = c(100,200,300,20),
                      multipart = TRUE, multipart_mode = 1)

# automatically call x$remove_data() upon garbage collection
x$flag_auto_clean(TRUE)

```

```
# set data (4 GB data) using 4 cores, compression level 50
# data creation ~10 s, disk IO ~15-20 seconds, ~250MB/s

system.time({
  lapply(1:20, function(ii){
    # Generating partition data (~10 sec)
    tmp <- rnorm(100*200*300) * (1+2i)

    # Write to disk (~16 sec)
    x[,,ii] <- tmp
    NULL
  })
})

# Reading 64 MB data using 4 cores
# ~0.25 seconds

system.time({
  x[1:100, sample(200, 200), 100:1, 2:4]
})

# This call requires 4GB of RAM
# Reading all 4GB data using 4 cores
# ~4 seconds (1 GB/s)

system.time({
  x[]
})
```

set_lazy_threads *Set Number of Threads for Lazy Arrays*

Description

A ported function from [threads_fstlib](#).

Usage

```
set_lazy_threads(nr_of_threads = NULL, reset_after_fork = NULL)
```

Arguments

nr_of_threads number of CPU cores to use, or NULL to use all cores
reset_after_fork whether to reset after forked process

Value

Number of cores currently used.

See Also

[threads_fstlib](#)

Index

`auto_clear_lazyarray`, [2](#)

`ClassLazyArray`, [3](#)

`create_lazyarray`, [7](#), [9](#), [10](#)

`dim`, [5](#), [7](#), [9](#)

`dimnames`, [5](#), [7](#), [9](#)

`lazyarray`, [8](#), [8](#)

`load_lazyarray`, [7](#), [10](#), [11](#)

`reg.finalizer`, [2](#)

`set_lazy_threads`, [13](#)

`threads_fstlib`, [13](#), [14](#)