Package ‘HDF5Array’

April 14, 2017

Title An array-like container for convenient access and manipulation of HDF5 datasets

Description This package implements the HDF5Array class for convenient access and manipulation of HDF5 datasets. In order to reduce memory usage and optimize performance, operations on an HDF5Array object are either delayed or executed using a block processing mechanism.
The delaying and block processing mechanisms are independent of the on-disk backend and implemented via the DelayedArray class. They even work on in-memory array-like objects like DataFrame objects (typically with Rle columns), Matrix objects, or ordinary arrays or data frames, where they can improve performance.

Version 1.2.1

Encoding UTF-8

Author Hervé Pagès

Maintainer Hervé Pagès <hpages@fredhutch.org>

biocViews Infrastructure, DataRepresentation, Sequencing, Annotation, Coverage, GenomeAnnotation

Depends R (>= 3.2), methods, BiocGenerics (>= 0.15.3), S4Vectors (>= 0.9.43), IRanges (>= 2.7.6)

Imports stats, rhdf5

Suggests Matrix, h5vcData, SummarizedExperiment, GenomicRanges, genefilter, BiocStyle, knitr, rmarkdown

VignetteBuilder knitr

License Artistic-2.0

Collate utils.R block_processing.R setHDF5DumpFile.R show-utils.R DelayedArray-class.R DelayedArray-utils.R DelayedMatrix-utils.R cbind-methods.R DelayedArray-stats.R HDF5Array-class.R zzz.R

NeedsCompilation no

R topics documented:

`cbind-methods` *Bind DelayedArray objects along their rows or columns*

Description

Methods for binding DelayedArray objects along their rows or columns.

Details

`rbind`, `cbind`, `arbind`, `acbind` methods are defined for DelayedArray objects. They perform delayed binding along the rows (`rbind` and `arbind`) or columns (`cbind` and `acbind`) of the objects passed to them.

See Also

- `cbind` in the **base** package for rbind/cbind'ing ordinary arrays.
- `acbind` in the **IRanges** package for arbind/acbind'ing ordinary arrays.
- DelayedArray-utils for common operations on DelayedArray objects.
- DelayedArray objects.
- HDF5Array objects.
- array objects in base R.

Examples

```
## ---------------------------------------------------------------------
## rbind/cbind
## ---------------------------------------------------------------------
library(rhdf5)
toy_h5 <- system.file("extdata", "toy.h5", package="HDF5Array")
h5ls(toy_h5)

M1 <- HDF5Array(toy_h5, "M1")
M2 <- HDF5Array(toy_h5, "M2")

M <- rbind(M1, t(M2))
M
colMeans(M)

## ---------------------------------------------------------------------
## arbind/acbind
## ---------------------------------------------------------------------
a1 <- array(1:60, c(3, 5, 4),
            dimnames=list(NULL, paste0("M1y", 1:5), NULL))
a2 <- array(101:240, c(7, 5, 4),
            dimnames=list(paste0("M2x", 1:7), paste0("M2y", 1:5), NULL))
a3 <- array(10001:10100, c(5, 5, 4),
            dimnames=list(paste0("M3x", 1:5), NULL, paste0("M3z", 1:4)))

A1 <- DelayedArray(a1)
A2 <- DelayedArray(a2)
A3 <- DelayedArray(a3)
```

```
A <- arbind(A1, A2, A3)
A

## Sanity check:
stopifnot(identical(arbind(a1, a2, a3), as.array(A)))
```

`DelayedArray-class` *DelayedArray objects*

Description

Wrapping an array-like object (typically an on-disk object) in a DelayedArray object allows one to perform common array operations on it without loading the object in memory. In order to reduce memory usage and optimize performance, operations on the object are either delayed or executed using a block processing mechanism.

Usage

```
DelayedArray(x)  # constructor function
```

Arguments

`x`	An array-like object.

In-memory versus on-disk realization

To *realize* a DelayedArray object (i.e. to trigger execution of the delayed operations carried by the object and return the result as an ordinary array), call `as.array` on it. However this realizes the full object at once *in memory* which could require too much memory if the object is big. A big DelayedArray object is preferrably realized *on disk* e.g. by calling the `writeHDF5Dataset` function or the `HDF5Dataset` constructor on it (other on-disk backends can be supported). This uses a block-processing strategy so that the full object is not realized at once in memory. Instead the object is processed block by block i.e. the blocks are realized in memory and written to disk one at a time. See ?`writeHDF5Dataset` for more information about this.

Accessors

DelayedArray objects support the same set of getters as ordinary arrays i.e. `dim()`, `length()`, and `dimnames()`.

Only `dimnames()` is supported as a setter.

Subsetting

A DelayedArray object can be subsetted like an ordinary object but with the following differences:

- The `drop` argument of the `[` operator is ignored i.e. subsetting a DelayedArray object always returns a DelayedArray object with the same number of dimensions. You need to call `drop()` on the subsetted object to actually drop its ineffective dimensions (i.e. the dimensions equal to 1).
- Linear subsetting (a.k.a. 1D-style subsetting, that is, subsetting with a single subscript `i`) is not supported.

Subsetting with `[[` is supported but only the linear form of it.

DelayedArray objects don't support subassignment (`[<-` or `[[<-`).

See Also

- DelayedArray-utils for common operations on DelayedArray objects.
- cbind in this package (**HDF5Array**) for binding DelayedArray objects along their rows or columns.
- setHDF5DumpFile to control the location of automatically created HDF5 datasets.
- HDF5Array objects.
- array objects in base R.

Examples

```
## ---------------------------------------------------------------------
## WRAP AN ORDINARY ARRAY IN A DelayedArray OBJECT
## ---------------------------------------------------------------------
a <- array(runif(1500000), c(10000, 30, 5))
A <- DelayedArray(a)
A

toto <- function(x) (5 * x[ , , 1] ^ 3 + 1L) * log(x[, , 2])
b <- toto(a)
head(b)

B <- toto(A)  # very fast! (operations are delayed)
B             # still 3 dimensions (subsetting a DelayedArray object
              # never drops dimensions)
B <- drop(B)
B

cs <- colSums(b)
CS <- colSums(B)
stopifnot(identical(cs, CS))

## ---------------------------------------------------------------------
## WRAP A DataFrame OBJECT IN A DelayedArray OBJECT
## ---------------------------------------------------------------------

## Generate random coverage and score along an imaginary chromosome:
cov <- Rle(sample(20, 5000, replace=TRUE), sample(6, 5000, replace=TRUE))
score <- Rle(sample(100, nrun(cov), replace=TRUE), runLength(cov))

DF <- DataFrame(cov, score)
A2 <- DelayedArray(DF)
A2

t(A2)  # delayed transpose is very fast and very memory efficient because
       # the matrix data is not copied
colSums(A2)

## ---------------------------------------------------------------------
## WRAP A HDF5Dataset OBJECT IN A DelayedArray OBJECT
## ---------------------------------------------------------------------
h5a <- HDF5Dataset(a)    # create the dataset
h5a

A3 <- DelayedArray(h5a)  # wrap the dataset in a DelayedArray object
A3
```

```
B3 <- toto(A3)  # very fast! (operations are delayed)
B3 <- drop(B3)

CS3 <- colSums(B3)
stopifnot(identical(cs, CS3))

## ---------------------------------------------------------------------
## STORE THE RESULT IN A NEW HDF5Dataset OBJECT
## ---------------------------------------------------------------------
b3 <- HDF5Dataset(B3)  # "realize" B3 on disk (as an HDF5 dataset)

## If this is just an intermediate result, you can either keep going
## with B3 or replace it with b3 wrapped in a DelayedArray object etc...
B3 <- DelayedArray(b3)  # semantically equivalent to the previous B3
```

DelayedArray-utils   *Common operations on DelayedArray objects*

Description

Common operations on DelayedArray objects.

Details

The operations currently supported by DelayedArray objects are:

Delayed operations:

- all the members of the `Ops`, `Math`, and `Math2` groups
- `!`
- `is.na`, `is.finite`, `is.infinite`, `is.nan`
- `nchar`, `tolower`, `toupper`
- `pmax2` and `pmin2`
- `rbind` and `cbind` (documented in cbind)

Block-processed operations:

- `anyNA`, `which`
- all the members of the `Summary` group
- `mean`
- `apply`
- `rowSums`, `colSums`, `rowMeans`, and `colMeans` [DelayedMatrix objects only]
- matrix multiplication (%*%) of an ordinary matrix by a DelayedMatrix object

See Also

- is.na, !, mean, apply, colSums, %*% in the **base** package for the corresponding operations.
- S4groupGeneric in the **methods** package for the members of the Ops, Math, and Math2 groups.
- cbind in this package (**HDF5Array**) for binding DelayedArray objects along their rows or columns.
- DelayedArray objects.
- setHDF5DumpFile to control the location of automatically created HDF5 datasets.
- HDF5Array objects.
- array objects in base R.

Examples

```
library(rhdf5)
toy_h5 <- system.file("extdata", "toy.h5", package="HDF5Array")
h5ls(toy_h5)

M1 <- HDF5Array(toy_h5, "M1")
range(M1)
M1 >= 0.5 & M1 < 0.75
log(M1)

M2 <- HDF5Array(toy_h5, "M2")
pmax2(M2, 0)

M <- rbind(M1, t(M2))
M
colMeans(M)

## Matrix multiplication writes a new HDF5 dataset to disk and returns
## an HDF5Matrix object that points to this new dataset.
m <- matrix(runif(60), ncol=12)
M <- DelayedArray(matrix(runif(240), nrow=12))

getHDF5DumpFile()  # HDF5 file where the new dataset will be written
lsHDF5DumpFile()
P <- m %*% M
P
getHDF5DumpFile()
lsHDF5DumpFile()
```

HDF5Array-class *HDF5 datasets as array-like objects*

Description

We provide 2 classes for representing an (on-disk) HDF5 dataset as an array-like object in R:

- HDF5Array: A high-level class HDF5Array that extends DelayedArray. All the operations available on DelayedArray objects work on HDF5Array objects.

- HDF5Dataset: A low-level class for pointing to an HDF5 dataset. No operation can be performed directly on an HDF5Dataset object. It needs to be wrapped in a DelayedArray or HDF5Array object first. An HDF5Array object is just an HDF5Dataset object wrapped in a DelayedArray object.

Usage

```
## Constructor functions
HDF5Array(file, name, type=NA)
HDF5Dataset(file, name, type=NA)

## Write an array-like object to an HDF5 file
writeHDF5Dataset(x, file, name)
```

Arguments

`file`	For `HDF5Array` and `HDF5Dataset`: The path (as a single character string) to the HDF5 file where the dataset is located. Alternatively `file` can be a DelayedArray object or an ordinary array, in which case the object is written to disk as a new HDF5 dataset by calling `writeHDF5Dataset` internally. Note that, when given a DelayedArray object, `writeHDF5Dataset` *realizes* it on disk, that is, all the delayed operations carried by the object are executed while the object is written to disk. See "On-disk realization of a DelayedArray object as an HDF5 dataset" section below for more information. For `writeHDF5Dataset`: The path (as a single character string) to the (new or existing) HDF5 file where to write the dataset.
`name`	For `HDF5Array` and `HDF5Dataset`: The name of the dataset in the HDF5 file. For `writeHDF5Dataset`: The name of the HDF5 dataset to write.
`type`	`NA` or the *R atomic type* (specified as a single string) corresponding to the type of the HDF5 dataset.
`x`	The array-like object to write to an HDF5 file. Can be an ordinary array, or a DelayedArray or HDF5Dataset object at the moment. If `x` is a DelayedArray object, `writeHDF5Dataset` *realizes* it on disk, that is, all the delayed operations carried by the object are executed while the object is written to disk. See "On-disk realization of a DelayedArray object as an HDF5 dataset" section below for more information. If `x` is an HDF5Dataset object, `writeHDF5Dataset` first wraps it in a DelayedArray object in order to trigger the block-processing strategy used by the on-disk realization mechanism.

Details

The `HDF5Array` and `HDF5Dataset` constructor functions can be used either to point to an existing HDF5 dataset or to create a new one (see description of the `file` argument above).

When used to create a new HDF5 dataset, the location where to write the dataset can be controlled with the `setHDF5DumpFile` and `setHDF5DumpName` utility functions.

Value

An HDF5Array object for `HDF5Array()`.

An HDF5Dataset object for `HDF5Dataset()`.

An invisible HDF5Dataset object for `writeHDF5Dataset()`.

On-disk realization of a DelayedArray object as an HDF5 dataset

When passed an DelayedArray object, `writeHDF5Dataset` *realizes* it on disk, that is, all the delayed operations carried by the object are executed on-the-fly while the object is written to disk. This uses a block-processing strategy so that the full object is not realized at once in memory. Instead the object is processed block by block i.e. the blocks are realized in memory and written to disk one at a time.

In other words, `writeHDF5Dataset(x, ...)` is semantically equivalent to `writeHDF5Dataset(as.array(x), ...)`, except that `as.array(x)` is not called because this would realize the full object at once in memory.

See `?DelayedArray` for general information about DelayedArray objects.

See Also

- DelayedArray objects.
- DelayedArray-utils for common operations on DelayedArray objects.
- setHDF5DumpFile to control the location of the new HDF5 datasets created by `HDF5Array` and `HDF5Dataset`.
- `h5ls` in the **rhdf5** package.
- The **rhdf5** package on top of which HDF5Array objects are implemented.
- array objects in base R.

Examples

```
## ---------------------------------------------------------------------
## CONSTRUCTION
## ---------------------------------------------------------------------
library(rhdf5)
library(h5vcData)

tally_file <- system.file("extdata", "example.tally.hfs5",
                          package="h5vcData")
h5ls(tally_file)

## Pick up "Coverages" dataset for Human chromosome 16:
cov0 <- HDF5Array(tally_file, "/ExampleStudy/16/Coverages")
cov0

## ---------------------------------------------------------------------
## dim/dimnames
## ---------------------------------------------------------------------
dim(cov0)

dimnames(cov0)
dimnames(cov0) <- list(paste0("s", 1:6), c("+", "-"))
dimnames(cov0)

## ---------------------------------------------------------------------
## SLICING (A.K.A. SUBSETTING)
## ---------------------------------------------------------------------
cov1 <- drop(cov0[ , , 29000001:29000007])
cov1

dim(cov1)
as.array(cov1)
```

```
stopifnot(identical(dim(as.array(cov1)), dim(cov1)))
stopifnot(identical(dimnames(as.array(cov1)), dimnames(cov1)))

cov2 <- drop(cov0[ , "+", 29000001:29000007])
cov2
as.matrix(cov2)

## ---------------------------------------------------------------------
## DelayedMatrix OBJECTS AS ASSAYS OF A SummarizedExperiment OBJECT
## ---------------------------------------------------------------------
library(SummarizedExperiment)

pcov <- drop(cov0[ , 1, ])  # coverage on plus strand
mcov <- drop(cov0[ , 2, ])  # coverage on minus strand

nrow(pcov)  # nb of samples
ncol(pcov)  # length of Human chromosome 16

## The convention for a SummarizedExperiment object is to have 1 column
## per sample so first we need to transpose 'pcov' and 'mcov':
pcov <- t(pcov)
mcov <- t(mcov)
se <- SummarizedExperiment(list(pcov=pcov, mcov=mcov))
se
stopifnot(validObject(se, complete=TRUE))

## A GPos object can be used to represent the genomic positions along
## the dataset:
gpos <- GPos(GRanges("16", IRanges(1, nrow(se))))
gpos
rowRanges(se) <- gpos
se
stopifnot(validObject(se))

## ---------------------------------------------------------------------
## writeHDF5Dataset()
## ---------------------------------------------------------------------
out_file <- tempfile()
writeHDF5Dataset(cov1, out_file, "cov1")
h5ls(out_file)
```

`setHDF5DumpFile` *Control the location of automatically created HDF5 datasets*

Description

Utility functions to control the location of automatically created HDF5 datasets.

Usage

```
setHDF5DumpFile(file=paste0(tempfile(), ".h5"))
getHDF5DumpFile()
lsHDF5DumpFile()
```

```
setHDF5DumpName(name)
getHDF5DumpName()
```

Arguments

`file`	The path to the *current output HDF5 file*, that is, to the HDF5 file where all newly created datasets shall be written.
`name`	The name of the *next* dataset to be written to the current output HDF5 file. This is for a one-time use only.

Note

`lsHDF5DumpFile()` is a just convenience wrapper for `rhdf5::h5ls(getHDF5DumpFile())`.

See Also

- DelayedArray objects.
- DelayedArray-utils for common operations on DelayedArray objects.
- HDF5Array objects.
- The `h5ls` function in the **rhdf5** package, on which `lsHDF5DumpFile` is based.

Examples

```
getHDF5DumpFile()

## Use setHDF5DumpFile() to change the current output HDF5 file.
## If the specified file exists, then it must be in HDF5 format or
## an error will be raised. If it doesn't exist, then it will be
## created.
#setHDF5DumpFile("path/to/some/HDF5/file")

lsHDF5DumpFile()

a <- array(1:600, c(150, 4))
h5a <- HDF5Dataset(a)
lsHDF5DumpFile()
A <- HDF5Array(h5a)   # DelayedArray object
A

b <- array(runif(6000), c(4, 2, 150))
h5b <- HDF5Dataset(b)
lsHDF5DumpFile()
B <- HDF5Array(h5b)   # DelayedArray object
B

C <- (log(2 * A + 0.88) - 5)^3 * t(drop(B[ , 1, ]))
C
HDF5Dataset(C)        # realize C on disk
lsHDF5DumpFile()

## Matrix multiplication is not delayed:
P <- C %*% matrix(runif(20), nrow=4)
lsHDF5DumpFile()
```

Index