Package 'DelayedArray'

October 17, 2017

Title Delayed operations on array-like 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. Note that this also works on in-memory array-like objects like DataFrame objects (typically with Rle columns), Matrix objects, and ordinary arrays and data frames.

Version 0.2.7

Encoding UTF-8

Author Hervé Pagès

Maintainer Hervé Pagès <hpages@fredhutch.org>

biocViews Infrastructure, DataRepresentation, Annotation, GenomeAnnotation

Depends R (>= 3.4), methods, BiocGenerics, S4Vectors (>= 0.14.3), IRanges, matrixStats

Imports stats

Suggests Matrix, HDF5Array, genefilter, BiocStyle

License Artistic-2.0

Collate utils.R ArrayBlocks-class.R show-utils.R DelayedArray-class.R cbind-methods.R realize.R block_processing.R DelayedArray-utils.R DelayedMatrix-utils.R DelayedArray-stats.R DelayedMatrix-stats.R RleArray-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 in the **HDF5Array** package.
- array objects in base R.

Examples

```
## ---------------------------------------------------------------------
## rbind/cbind
## ---------------------------------------------------------------------
library(HDF5Array)
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(seed)  # constructor function
seed(x)             # seed getter
type(x)
```

Arguments

`seed`	An array-like object.
`x`	A DelayedArray object. (Can also be an ordinary array in case of `type`.)

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 `writeHDF5Array` on it (this function is defined in the **HDF5Array** package) or coercing it to an HDF5Array object with `as(x, "HDF5Array")`. 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 `?writeHDF5Array` in the **HDF5Array** package for more information about this.

Accessors

DelayedArray objects support the same set of getters as ordinary arrays i.e. `dim()`, `length()`, and `dimnames()`. In addition, they support `type()`, which is the DelayedArray equivalent of `typeof()` or `storage.mode()` for ordinary arrays. Note that, for convenience and consistency, `type()` also works on ordinary arrays.

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

Subsetting

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

- *Multi-dimensional single bracket subsetting* (i.e. subsetting of the form `x[i_1, i_2, ..., i_n]` with one (possibly missing) subscript per dimension) returns a DelayedArray object where the subsetting is actually delayed. So it's a very light operation.
- 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 as the original object. You need to call `drop()` on the subsetted object to actually drop its ineffective dimensions (i.e. the dimensions equal to 1). `drop()` is also a delayed operation so is very light.
- *Linear single bracket subsetting* (a.k.a. 1D-style subsetting, that is, subsetting of the form `x[i]`) only works if subscript `i` is a numeric vector at the moment. Furthermore, `i` cannot contain NAs and all the indices in it must be >= 1 and <= `length(x)` for now. It returns an atomic vector of the same length as `i`. This is NOT a delayed operation.

Subsetting with `[[` is supported but only the *linear* form of it at the moment i.e. the `x[[i]]` form where `i` is a *single* numeric value >= 1 and <= `length(x)`. It is equivalent to `x[i]`.

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

See Also

- `realize` for realizing a DelayedArray object in memory or on disk.
- DelayedArray-utils for common operations on DelayedArray objects.
- cbind in this package (**DelayedArray**) for binding DelayedArray objects along their rows or columns.
- RleArray objects.
- HDF5Array objects in the **HDF5Array** package.
- DataFrame objects in the **S4Vectors** package.
- array objects in base R.

Examples

```
## ---------------------------------------------------------------------
## A. WRAP AN ORDINARY ARRAY IN A DelayedArray OBJECT
## ---------------------------------------------------------------------
a <- array(runif(1500000), dim=c(10000, 30, 5))
A <- DelayedArray(a)
A
## The seed of A is treated as a "read-only" object so won't change when
## we start operating on A:
stopifnot(identical(a, seed(A)))
type(A)

## Multi-dimensional single bracket subsetting:
m <- a[11:20 , 5, ]        # a matrix
A[11:20 , 5, ]             # not a DelayedMatrix (still 3 dimensions)
M <- drop(A[11:20 , 5, ])  # a DelayedMatrix object
stopifnot(identical(m, as.array(M)))
stopifnot(identical(a, seed(M)))

## Linear single bracket subsetting:
```

```
A[11:20]
A[which(A <= 1e-5)]

## Other operations:
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))

## ---------------------------------------------------------------------
## B. 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
seed(A2)  # 'DF'

## Coercion of a DelayedMatrix object to DataFrame produces a DataFrame
## object with Rle columns:
as(A2, "DataFrame")
stopifnot(identical(DF, as(A2, "DataFrame")))

t(A2)  # transposition is delayed so is very fast and very memory
       # efficient
stopifnot(identical(DF, seed(t(A2))))  # the "seed" is still the same
colSums(A2)

## ---------------------------------------------------------------------
## C. A HDF5Array OBJECT IS A (PARTICULAR KIND OF) DelayedArray OBJECT
## ---------------------------------------------------------------------
library(HDF5Array)
A3 <- as(a, "HDF5Array")   # write 'a' to an HDF5 file
A3
is(A3, "DelayedArray")     # TRUE
seed(A3)                   # a HDF5ArraySeed object

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

B3                         # not a HDF5Array object because now it
                           # carries delayed operations
B3 <- drop(B3)

CS3 <- colSums(B3)
```

```
stopifnot(identical(cs, CS3))

## ---------------------------------------------------------------------
## D. PERFORM THE DELAYED OPERATIONS
## ---------------------------------------------------------------------
as(B3, "HDF5Array")        # "realize" 'B3' on disk

## If this is just an intermediate result, you can either keep going
## with B3 or replace it with its "realized" version:
B3 <- as(B3, "HDF5Array")  # no more delayed operations on new 'B3'
seed(B3)

## For convenience, realize() can be used instead of explicit coercion.
## The current "realization backend" controls where realization
## happens e.g. in memory if set to NULL or in an HDF5 file if set
## to "HDF5Array":
D <- cbind(B3, exp(B3))
D
setRealizationBackend("HDF5Array")
D <- realize(D)
D
## See '?realize' for more information about "realization backends".

## ---------------------------------------------------------------------
## E. WRAP A SPARSE MATRIX IN A DelayedArray OBJECT
## ---------------------------------------------------------------------
## Not run:
library(Matrix)
M <- 75000L
N <- 1800L
p <- sparseMatrix(sample(M, 9000000, replace=TRUE),
                  sample(N, 9000000, replace=TRUE),
                  x=runif(9000000), dims=c(M, N))
P <- DelayedArray(p)
P
p2 <- as(P, "sparseMatrix")
stopifnot(identical(p, p2))

## The following is based on the following post by Murat Tasan on the
## R-help mailing list:
##   https://stat.ethz.ch/pipermail/r-help/2017-May/446702.html

## As pointed out by Murat, the straight-forward row normalization
## directly on sparse matrix 'p' would consume too much memory:
row_normalized_p <- p / rowSums(p^2)  # consumes too much memory
## because the rowSums() result is being recycled (appropriately) into a
## *dense* matrix with dimensions equal to dim(p).

## Murat came up with the following solution that is very fast and memory
## efficient:
row_normalized_p1 <- Diagonal(x=1/sqrt(Matrix::rowSums(p^2)))

## With a DelayedArray object, the straight-forward approach uses a
## block processing strategy behind the scene so it doesn't consume
## too much memory.

## First, let's see the block processing in action:
```

```
DelayedArray:::set_verbose_block_processing(TRUE)
## and set block size to a bigger value than the default:
getOption("DelayedArray.block.size")
options(DelayedArray.block.size=80e6)

row_normalized_P <- P / sqrt(DelayedArray::rowSums(P^2))

## Increasing the block size increases the speed but also memory usage:
options(DelayedArray.block.size=200e6)
row_normalized_P2 <- P / sqrt(DelayedArray::rowSums(P^2))
stopifnot(all.equal(row_normalized_P, row_normalized_P2))

## Back to sparse representation:
DelayedArray:::set_verbose_block_processing(FALSE)
row_normalized_p2 <- as(row_normalized_P, "sparseMatrix")
stopifnot(all.equal(row_normalized_p1, row_normalized_p2))

options(DelayedArray.block.size=10e6)

## End(Not run)
```

`DelayedArray-utils` *Common operations on DelayedArray objects*

Description

Common operations on DelayedArray objects.

Details

The operations currently supported on 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`
- matrix multiplication (`%*%`) of an ordinary matrix by a DelayedMatrix object
- matrix row/col summarization [DelayedMatrix objects only]: `rowSums`, `colSums`, `rowMeans`, `colMeans`, `rowMaxs`, `colMaxs`, `rowMins`, `colMins`, `rowRanges`, and `colRanges`

See Also

- is.na, !, mean, apply, and %*% in the **base** package for the corresponding operations on ordinary arrays or matrices.
- rowSums in the **base** package and rowMaxs in the **matrixStats** package for row/col summarization of an ordinary matrix.
- setRealizationBackend for how to set a *realization backend*.
- writeHDF5Array in the **HDF5Array** package for writting an array-like object to an HDF5 file and other low-level utilities to control the location of automatically created HDF5 datasets.
- DelayedArray objects.
- HDF5Array objects in the **HDF5Array** package.
- S4groupGeneric in the **methods** package for the members of the Ops, Math, and Math2 groups.
- array objects in base R.

Examples

```
library(HDF5Array)
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)

M3 <- rbind(M1, t(M2))
M3

## ---------------------------------------------------------------------
## MATRIX MULTIPLICATION
## ---------------------------------------------------------------------

## Matrix multiplication is not delayed: the output matrix is realized
## block by block. The current "realization backend" controls where
## realization happens e.g. in memory if set to NULL or in an HDF5 file
## if set to "HDF5Array". See '?realize' for more information about
## "realization backends".
## The output matrix is returned as a DelayedMatrix object with no delayed
## operations on it. The exact class of the object depends on the backend
## e.g. it will be HDF5Matrix with "HDF5Array" backend.

m <- matrix(runif(50000), ncol=nrow(M1))

## Set backend to NULL for in-memory realization:
setRealizationBackend()
P1 <- m %*% M1
P1

## Set backend to HDF5Array for realization in HDF5 file:
setRealizationBackend("HDF5Array")
```

```
## With the HDF5Array backend, the output matrix will be written to an
## automatic location on disk:
getHDF5DumpFile()  # HDF5 file where the output matrix will be written
lsHDF5DumpFile()

P2 <- m %*% M1
P2

lsHDF5DumpFile()

## Use setHDF5DumpFile() and setHDF5DumpName() from the HDF5Array package
## to control the location of automatically created HDF5 datasets.

stopifnot(identical(as.array(P1), as.array(P2)))

## ---------------------------------------------------------------------
## MATRIX ROW/COL SUMMARIZATION
## ---------------------------------------------------------------------

rowSums(M1)
colSums(M1)

rowMeans(M1)
colMeans(M1)

rmaxs <- rowMaxs(M1)
cmaxs <- colMaxs(M1)

rmins <- rowMins(M1)
cmins <- colMins(M1)

rranges <- rowRanges(M1)
cranges <- colRanges(M1)

stopifnot(identical(cbind(rmins, rmaxs, deparse.level=0), rranges))
stopifnot(identical(cbind(cmins, cmaxs, deparse.level=0), cranges))
```

`realize` *Realize a DelayedArray object*

Description

Realize a DelayedArray object in memory or on disk. Get or set the *realization backend* for the current session with `getRealizationBackend` or `setRealizationBackend`.

Usage

```
supportedRealizationBackends()
getRealizationBackend()
setRealizationBackend(BACKEND=NULL)

realize(x, ...)
```

```
## S4 method for signature 'ANY'
realize(x, BACKEND=getRealizationBackend())
```

Arguments

x	The array-like object to realize.
...	Additional arguments passed to methods.
BACKEND	NULL (the default), or a single string specifying the name of the backend. When the backend is set to NULL, x is realized in memory as an ordinary array by just calling as.array on it.

Details

The *realization backend* controls where/how realization happens e.g. as an ordinary array if set to NULL, as an RleArray object if set to "RleArray", or in an HDF5 file if set to "HDF5Array".

Value

realize(x) returns a DelayedArray object. More precisely, it returns DelayedArray(as.array(x)) when the backend is set to NULL (the default). Otherwise it returns an instance of the class associated with the specified backend (which should extend DelayedArray).

See Also

- DelayedArray objects.
- RleArray objects.
- HDF5Array objects in the **HDF5Array** package.
- array objects in base R.

Examples

```
library(HDF5Array)
toy_h5 <- system.file("extdata", "toy.h5", package="HDF5Array")
h5ls(toy_h5)
M1 <- HDF5Array(toy_h5, "M1")
M2 <- HDF5Array(toy_h5, "M2")
M3 <- rbind(log(M1), t(M2))

supportedRealizationBackends()
getRealizationBackend()  # backend is set to NULL
realize(M3)  # realization as ordinary array

setRealizationBackend("RleArray")
getRealizationBackend()  # backend is set to "RleArray"
realize(M3)  # realization as RleArray object

setRealizationBackend("HDF5Array")
getRealizationBackend()  # backend is set to "HDF5Array"
realize(M3)  # realization in HDF5 file
```

RleArray-class *RleArray objects*

Description

The RleArray class is an array-like container where the values are stored in a run-length encoding format. RleArray objects support delayed operations and block processing.

Usage

```
RleArray(rle, dim, dimnames=NULL)  # constructor function
```

Arguments

rle	An Rle object.
dim	The dimensions of the object to be created, that is, an integer vector of length one or more giving the maximal indices in each dimension.
dimnames	Either NULL or the names for the dimensions. This must a list of length the number of dimensions. Each list element must be either NULL or a character vector along the corresponding dimension.

Details

RleArray extends DelayedArray. All the operations available on DelayedArray objects work on RleArray objects.

See Also

- Rle objects in the **S4Vectors** package.
- DelayedArray objects.
- DelayedArray-utils for common operations on DelayedArray objects.
- `realize` for realizing a DelayedArray object in memory or on disk.
- HDF5Array objects in the **HDF5Array** package.
- DataFrame objects in the **S4Vectors** package.
- array objects in base R.

Examples

```
rle <- Rle(sample(6L, 500000, replace=TRUE), 8)
a <- array(rle, dim=c(50, 20, 4000))  # array() expands the Rle object
                                      # internally with as.vector()

A <- RleArray(rle, dim=c(50, 20, 4000))  # Rle object is NOT expanded
A

object.size(a)
object.size(A)

stopifnot(identical(a, as.array(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

stopifnot(identical(b, as.array(B)))

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

## Coercion of a DelayedMatrix object to DataFrame produces a DataFrame
## object with Rle columns:
as(B, "DataFrame")
```

Index