Bioconductor

Lori Shepherd
Bioconductor Core Team
lori.shepherd@roswellpark.org

Website:
https://www.bioconductor.org

Help/Support Site:
https://support.bioconductor.org/

BiocFileCache

Local File Management

BiocParallel

Parallel Processing of Large Data

DelayedArray

Delayed Analysis of on-disk representation

BiocFileCache

Local File Management

Motivation:

It can be time consuming to download remote resource from the web. Let’s design a way to check a local resource to see if it needs to be updated or not.

Let’s also have a way to better organize local files

BiocFileCache( )

- creates a cache object
- sqlite database backend
- add ‘resources’ (files) to the cache object to track

Cache Info:

- bfccache ( )
- length ( )
- show ( )
- bfcinfo ( )

Adding Resources:

- bfcadd( )
- bfcnew ( )

Removing Resources:

- bfcremove ( )
- bfcsync ( )

Investigating Resources:

- bfcquerycols ( )
- bfcquery ( )
- bfccount ( )
- bfcrid ( )
- bfcpath ( )
- bfcrpath ( )
- [

Web Resources:

- bfcneedsupdate ( )
- bfcdownload ( )

Updating Resources:

- bfcupdate ( )
- [[

MetaData:

- bfcmetalist ( )
- bfcmeta ( )
- bfcmeta ( ) <-
- bfcmetaremove ( )

Export/Import Cache:

- importbfc ( )
- exportbfc ( )
- makeBiocFileCacheFromDataFrame( )

Clean/Remove Cache:

- cleanbfc ( )
- removebfc ( )

Example:

```
> BiocFileCache()
class: BiocFileCache
bfccache: /home/lori/.cache/BiocFileCache
bfccount: 0
For more information see: bfcinfo() or bfcquery()


> bfcinfo()
# A tibble: 0 x 10
# ... with 10 variables: rid <chr>, rname <chr>, create_time <dbl>,
#   access_time <dbl>, rpath <chr>, rtype <chr>, fpath <chr>,
#   last_modified_time <dbl>, etag <chr>, expires <dbl>
```

Example:

```
> bfcadd(rname="Wiki", fpath="https://en.wikipedia.org/wiki/Bioconductor")
  |======================================================================| 100%
                                                          BFC1
"/home/lori/.cache/BiocFileCache/282e8be47f6_Bioconductor"

> bfcinfo()
# A tibble: 1 x 10
  rid   rname create_time  access_time rpath rtype fpath last_modified_t… etag
  <chr> <chr> <chr>        <chr>       <chr> <chr> <chr> <chr>            <chr>
1 BFC1  Wiki  2018-07-12 … 2018-07-12… /hom… web   http… 2018-07-07 07:1… NA
# ... with 1 more variable: expires <chr>


> library(dplyr)
> bfcinfo() %>% select(last_modified_time, rpath)
# A tibble: 1 x 2
  last_modified_time  rpath
  <chr>               <chr>
1 2018-07-07 07:13:52 /home/lori/.cache/BiocFileCache/282e8be47f6_Bioconductor
```

Example:

```
> pathToSave = bfcnew(rname="My RDS File", ext=".rds")

> pathToSave
                                                                    BFC2
  "/home/lori/.cache/BiocFileCache/2feb30a96058_2feb30a96058.rds"

> bfcinfo()
# A tibble: 2 x 10
  rid   rname  create_time access_time rpath rtype fpath last_modified_t… etag
  <chr> <chr>  <chr>       <chr>       <chr> <chr> <chr> <chr>            <chr>
1 BFC1  Wiki   2018-07-12… 2018-07-12… /hom… web   http… 2018-07-07 07:1… NA
2 BFC2  My RD… 2018-07-12… 2018-07-12… /hom… rela… 388d… NA               NA
# ... with 1 more variable: expires <chr>

> saveRDS(myObj, file=pathToSave)
```

Example:

```
> bfcneedsupdate()
 BFC1
TRUE
```

Utilizes functions from httr to capture Expires, Last-modified time, and Etag

1. HEAD()
2. cache_info()

```
> library(httr)

> cache_info(HEAD("https://en.wikipedia.org/wiki/Bioconductor"))
<cache_info>  https://en.wikipedia.org/wiki/Bioconductor
  Cacheable:     TRUE
  Expires:       Thu, 12 Jul 2018 13:37:06 GMT <expired>
  Last-Modified: Sat, 07 Jul 2018 07:13:52 GMT
  Etag:


> cache_info(HEAD("https://bioconductor.org/packages/3.8/data/annotation/src/contrib/PANTHER.db_1.0.4.tar.gz"))
<cache_info>  https://bioconductor.org/packages/3.8/data/annotation/src/contrib/PANTHER.db_1.0.4.tar.gz
  Cacheable:     TRUE
  Last-Modified: Wed, 27 Sep 2017 17:09:56 GMT
  Etag:          "608b685-55a2edc70632a"
```

Example:

```
> bfcquery(query="RDS")
# A tibble: 1 x 10
  rid   rname  create_time access_time rpath rtype fpath last_modified_t… etag
  <chr> <chr>  <chr>       <chr>       <chr> <chr> <chr>            <dbl> <chr>
1 BFC2  My RD… 2018-07-12… 2018-07-12… /hom… rela… 388d…               NA NA
# ... with 1 more variable: expires <dbl>


> bfcrid(bfcquery(query="RDS"))
[1] "BFC2"

> bfcrpath(rids="BFC2")

                                                            BFC2
  "/home/lori/.cache/BiocFileCache/2feb30a96058_2feb30a96058.rds"


> readRDS(bfcrpath(rids="BFC2"))
```

Example:

```
# data.frame or tibble

> meta = data.frame(rid="BFC2", info="pipeLine project X", numSamples=2000)

> bfc = BiocFileCache()

> bfcmeta(bfc, name="pipeLineXmeta") <- meta
> bfcmetalist()
[1] "pipeLineXmeta"


> library(dplyr)
> bfcinfo(bfc) %>% select(rid, rname, info, numSamples)
# A tibble: 2 x 4
    rid          rname                 info numSamples
  <chr>          <chr>                <chr>      <dbl>
1  BFC1           Wiki                 <NA>         NA
2  BFC2 My RData File pipeLine project X       2000
```

Example:

```
> bfcquery(query="project X", field="info")
# A tibble: 1 x 12
  rid   rname  create_time access_time rpath rtype fpath last_modified_t… etag
  <chr> <chr>  <chr>       <chr>       <chr> <chr> <chr>            <dbl> <chr>
1 BFC2  My RD… 2018-07-12… 2018-07-12… /hom… rela… 388d…                NA NA
# ... with 3 more variables: expires <dbl>, info <chr>, numSamples <dbl>


> bfcquerycols()
 [1] "rid"                  "rname"                "create_time"
 [4] "access_time"          "rpath"                "rtype"
 [7] "fpath"                "last_modified_time" "etag"
[10] "expires"              "info"                 "numSamples"
```

BiocFileCache

- Easy handling of downloading remote data
- Easily retrieve a local file location within R

BiocParallel

Parallel Processing of Large Data

Why use BiocParallel over others?

- Unified interface that is cross platform compatible through the use of the BiocParallelParam instances. The BiocParallelParm object will:
    - Defines the method of parallelization
    - Defines the computing resources
- Built in parallelization functions:
    - bplapply / bpmapply
    - bpiterate
    - bpvec / bpvectorize
    - bpaggregate
- Cluster scheduling
    - BatchJobsParam (older implementation)
    - BatchtoolsParam (NEW!!)
- Foreach and iterator packages are fully supported
- Better handling of errors, logs, and debugging

BiocParallelParm Instances

- SerialParam
  - Parallel evaluation disabled
- MulticoreParam (Unix and Mac only)
  - Multiple cores on a single computer
- SnowParam
  - Evaluate across several distinct R instances, on one or several computers. Facilities implemented in the snow package. (different snow ‘back-ends’ are supported including socker and MPI)
- BatchJobsParam (old) / BatchtoolsParam (NEW!!)
  - Cluster scheduler (sge, slurm, lsf, torque, openlava)
- DoparParam
  - Parallel back-end supported by foreach package

Parallelized Functions

- bplapply(X, FUN, …, BPPARAM=bpparam( ))
- bpmapply(FUN, …, BPPARAM=bpparam( ))
- bpiterate(ITER, FUN, …, BPPARAM=bpparam( ))
- bpvec(X, FUN, …, BPPARAM=bpparam( ))
- bpvectorize(FUN, …, BPPARAM=bpparam( ))
- bpaggregate(x, data, FUN, …, BPPARAM=bpparam( ))

All of these functions take an optional BPPARAM argument that is one of the previously listed BiocParallelParam instances and determines the parallel back-end to use!

Don’t forget to register!

List of Registered BiocParallelParam instances representing user preferences

Behaves like a ‘stack’ where as a BiocParallelParm instance gets added to the registry, it automatically is added to the top of the list and is used as default.

```
> registered()
$MulticoreParam
class: MulticoreParam
  bpisup: FALSE; bpnworkers: 6; bptasks: 0; bpjobname: BPJOB
  bplog: FALSE; bpthreshold: INFO; bpstopOnError: TRUE
  bptimeout: 2592000; bpprogressbar: FALSE; bpexportglobals: TRUE
  bpRNGseed:
  bplogdir: NA
  bpresultdir: NA
  cluster type: FORK

$SnowParam
class: SnowParam
  bpisup: FALSE; bpnworkers: 6; bptasks: 0; bpjobname: BPJOB
  bplog: FALSE; bpthreshold: INFO; bpstopOnError: TRUE
  bptimeout: 2592000; bpprogressbar: FALSE; bpexportglobals: TRUE
  bpRNGseed:
  bplogdir: NA
  bpresultdir: NA
  cluster type: SOCK

$SerialParam
class: SerialParam
  bpisup: TRUE; bpnworkers: 1; bptasks: 0; bpjobname: BPJOB
  bplog: FALSE; bpthreshold: INFO; bpstopOnError: TRUE
  bptimeout: 2592000; bpprogressbar: FALSE; bpexportglobals: TRUE
  bplogdir: NA
```

```
> bpparam()

class: MulticoreParam
  bpisup: FALSE; bpnworkers: 6; bptasks: 0; bpjobname: BPJOB
  bplog: FALSE; bpthreshold: INFO; bpstopOnError: TRUE
  bptimeout: 2592000; bpprogressbar: FALSE; bpexportglobals: TRUE
  bpRNGseed:
  bplogdir: NA
  bpresultdir: NA
  cluster type: FORK
```

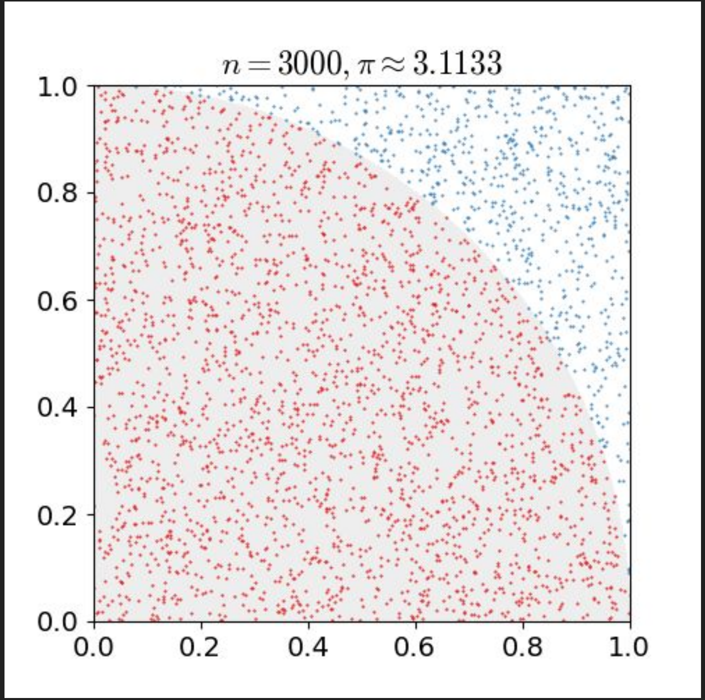
```
> register(BatchtoolsParam())
```

```
> names(registered())
[1] "BatchtoolsParam" "MulticoreParam"  "SnowParam"
"SerialParam"
```

```
> class(bpparam())
[1] "BatchtoolsParam"
attr(,"package")
[1] "BiocParallel"
```

Approximate value of π for demonstration

```
piApprox <- function(n) {
  nums = matrix(runif(2 * n), ncol = 2)
  # pythagoras' theorem
  d = sqrt(nums[, 1]^2 + nums[, 2]^2)
  # 4 quadrants
  4 * mean(d <= 1)
}
```



https://en.wikipedia.org/wiki/Approximations_of_%CF%80#Summing_a_circle's_area

bplapply( ) using different parallel backends

Using BiocParallel::SerialParam()

```
> bplapply(rep(100000,4), piApprox, BPPARAM = SerialParam())
```

Using BiocParallel::MulticoreParam()

```
> bplapply(rep(100000,4), piApprox, BPPARAM = MulticoreParam())
```

Using BiocParallel::BatchtoolsParam

```
> bplapply(rep(100000,4), piApprox, BPPARAM = BatchtoolsParam(cluster='sge'))
```

BatchtoolsParam as BPPARAM

BatchtoolsParam() can be passed as a value to BPPARAM, to evaluate your job on multiple types of clusters utilizing batchools package functionality.

```
BatchtoolsParam(cluster =
                c(‘socket’, ‘multicore’, ‘interactive’,
                  ‘sge’, ‘slurm’, ‘lsf’, ‘torque’,
                 ‘openlava’))
```

Customizing the BatchtoolsParam object

```
BatchtoolsParam(workers = batchtoolsWorkers(cluster),
                 cluster = batchtoolsCluster(),
                 registryargs = batchtoolsRegistryargs(),
                 template = batchtoolsTemplate(cluster),
                 stop.on.error = TRUE,
                 progressbar = FALSE,
                 RNGseed = NA_integer_,
                 timeout = 30L * 24L * 60L * 60L,
                 log = FALSE, logdir = NA_character_,
                 resultdir = NA_character_,
                 jobname = ‘BPJOB’)
```

The template designation is important! See batchtools package for available templates.

Errors, Logs, and Debugging in BiocParallel

There is a whole vignette describing enhancements and options

https://bioconductor.org/packages/devel/bioc/vignettes/BiocParallel/inst/doc/Errors_Logs_And_Debugging.pdf

Error Handling

The stop.on.error field controls if the job is terminated as soon as one task throws an error. By default is TRUE

Set either in the constructor of the BiocParallelParam instance

```
param <- MulticoreParam(stop.on.error = TRUE)
```

Or bpstopOnError accessor method

```
bpstopOnError(param) = FALSE
```

Error Handling

bptry( ) function is a convenient way of trying to evaluate a bpapply-like expression, returning the evaluated results without signalling an error.

bpok( ) function is a quick way to determine which (if any) tasks failed.

BPREDO (re-do) argument for recomputing only the tasks that failed

Example of Error Handling

```
> param <- MulticoreParam(workers=2, stop.on.error=FALSE)
> X <- list(1, "2", 3)
> result <- bptry(bplapply(X, sqrt, BPPARAM=param))

> bpok(result)
[1]  TRUE FALSE  TRUE
> result
[[1]]
[1] 1

[[2]]
<remote_error in FUN(...): non-numeric argument to mathematical function>
traceback() available as 'attr(x, "traceback")'

[[3]]
[1] 1.732051


> X.redo <- list(1, 2, 3)
> bplapply(X.redo, sqrt, BPREDO=result, BPPARAM=param)
resuming previous calculation ...
[[1]]
[1] 1

[[2]]
[1] 1.414214

[[3]]
[1] 1.732051
```

BiocParallel

- Cross compatible parallel processing through BiocParallelParm instances
- Handles a variety of back-ends including cluster management
- Built in parallel functions
- Useful debugging, logging, and error handling

DelayedArray / DelayedMatrixStats

Delayed Analysis of on-disk representation

Data is only getting larger!!!

- TENxBrainData package as an example
  - 1.3 million brain cell single-cell RNA-seq (scRNA-seq) data set generated by 10X Genomics
  - Dimensions of the matrix 27998 (genes) x 1306127 (samples)
  - In an ordinary array would be 136 Gb in memory

DelayedArray 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.

Pagès H (2018). DelayedArray: Delayed operations on array-like objects. R package version 0.6.1.

Implementation

- Every DelayedArray MUST have a seed!
  - Seed stores the actual data.
  - In memory, on-disk, or remote = Essentially DelayedArray can then be extended to any file format that can store array data by creating a seed class implementation.
    https://bioconductor.org/packages/release/bioc/vignettes/DelayedArray/inst/doc/02-Implementing_a_backend.html
    - dim
    - dimnames
    - Ability to realize a rectangular subset of the data (extract_array)

Example existing seeds

- In-memory
  - DelayedArray with seed as Matrix, data.frame, DataFrame, tibbles, … (DelayedArray package)
  - RleArray with Rle object as seed (DelayedArray package)
- On-disk
  - HDF5Array for data stored as HDF5 file (HDF5Array package)
  - GDSArray for data stored as GDS file (GDSArray package)
- Remote
  - H5S_Array for data on HDF Server (rhdf5client package)
  - restfulSE package for remote data, google genomics cloud data or hdf5 server data

So...

In-memory:

```
DelayedArray(seed = as.data.frame(matrix(1:1000, 1:1000, ncol=2)))
```

On-disk:

Let’s looks at the TENxBrainData

TENxBrainData

```
> library(TENxBrainData)

# load the experiment data from the Bioconductor experiment Hub
> tenx = TENxBrainData()
> tenx

# get the assay matrix of the single cell experiment object notice it is a DelayedMatrix with a hdf5array backend
> assay(tenx)
> assay(tenx)@seed

# super fast operations because its 'delayed'
> log(1 + assay(tenx))
> t(log(1 + assay(tenx)))[, 1:1000]

# look at the call stack for 'delayed' operations
> t(log(1 + assay(tenx))) @seed
> showtree((t(log(1 + assay(tenx))) * t(log(1 + assay(tenx)))))

# This takes awhile because involves a operation requiring the whole matrix so we won't run right now
> # hist(rowSums(t(log(1 + assay(tenx)))))

# using realize( ) function would also perform the
# 'delayed' operations and realize the data
```

Block processing, Parallal processing, DelayedMatrixStats and beachmat

- Block processing for faster in-memory processing to avoid full realization.
    - Each row is a block (perform something like rowSums)
    - Each column is a block (perform something like columnSums)
    - Variable chunks
    - Optimal chunks
    - Often used with blockApply( ) and blockReduce( )
- Parallel Option with blockApply( )
- DelayedMatrixStats package for matrixStat API with block processing for DelayedArray framework
- beachmat package for C++ class interface for variety of commonly used matrix types

DelayedArray

- Work with an array-like data object without having to load the data directly into memory (may be in-memory or on-disk or remote)
- Block processing (for in memory) and parallel processing (for speed) options
- Ability to be extended and built-upon for any class that holds array like data.

Bioconductor

Lori Shepherd
Bioconductor Core Team
lori.shepherd@roswellpark.org