Package 'DSL'

July 21, 2025

Version 0.1-7

Date 2020-01-12

Title Distributed Storage and List

Description An abstract DList class helps storing large list-type objects in a distributed manner. Corresponding high-level functions and methods for handling distributed storage (DStorage) and lists allows for processing such DLists on distributed systems efficiently. In doing so it uses a well defined storage backend implemented based on the DStorage class.

License GPL-3

Imports methods, utils

Suggests hive (>= 0.2-2), parallel

NeedsCompilation yes

Author Ingo Feinerer [aut],
Stefan Theussl [aut, cre],
Christian Buchta [ctb]

Maintainer Stefan Theussl <Stefan.Theussl@R-project.org>

Repository CRAN

Date/Publication 2020-01-15 06:40:02 UTC

Contents

DGather *Gather Distributed Data*

Description

Retrieves "DList" data distributed as chunks.

Usage

```
DGather( x, keys = FALSE, n = -1L, names = TRUE )
```

Arguments

x	a "DList" object.
keys	logical; should only keys be retrieved from chunks? Default: FALSE, i.e., only values are retrieved.
n	an integer specifying the number of chunks to be read.
names	logical; should the return value be a named list? Default: TRUE.

Details

DGather() is similar to an MPI_GATHER (see http://www.mpi-forum.org/docs/mpi-3.1/mpi31-report/node103.htm#Node103) where: "[...] each process (root process included) sends the contents of its send buffer to the root process. The root process receives the messages and stores them in rank order." For "DList" objects DGather() will gather data contained in chunks possibly distributed on a cluster of workstations and store it in a (possibly named) list. Note that depending of the size of the data, the resulting list may not fit into memory.

Value

A (named) list.

Examples

```
dl <- DList( line1 = "This is the first line.",
             line2 = "Now, the second line." )
DGather( dl )
## retrieve keys
unlist(DGather( dl, keys = TRUE, names = FALSE ))
## remove DList and garbage collect it
rm( dl )
gc()
```

DList *Distributed List Class*

Description

Functions to construct, coerce, check for, and interact with storage of objects of class "DList".

Usage

```
DList( ... )
as.DList( x, DStorage = NULL, ... )
is.DList( x )
DL_storage( x )
`DL_storage<-`( x, value )
```

Arguments

...	objects, possibly named.
x	an object.
DStorage	an object representing the virtual (distributed) storage for storing data. See class "DStorage" for details.
value	the new storage of class DStorage attached to the "DList".

Value

An object of class "DList" or, in case of DL_storage(), an object of class "DStorage".

Examples

```
## coerce to 'DList' object using a default virtual storage
l <- list( cow = "girl", bull = "boy" )
dl <- as.DList( l )
is.DList( dl )
DL_storage(dl)
## remove DList and garbage collect it
rm(dl)
gc()
```

DStorage *Virtual Distributed Storage Class*

Description

When using class DList the underlying 'virtual' storage plays an important role. It defines how to use the given storage (read/write methods, etc.), where the data is to be stored (i.e., the base directory on the file system), and how DMap as well as DReduce have to be applied.

Usage

```
DStorage(type = c("LFS", "HDFS"), base_dir, chunksize = 1024^2)
is.DStorage( ds )
```

Arguments

type	the type of the storage to be created. Currently only "LFS" and "HDFS" storage types are supported.
base_dir	specifies the base directory where data is to be stored.
chunksize	defines the size of each chunk written to the virtual storage.
ds	a virtual possibly distributed storage.

Value

An object which inherits from class DStorage, or, in case of is.DStorage() a logical indicating whether it inherits from "DStorage" or not.

Examples

```
## creating a new virtual storage using 50MB chunks
ds <- DStorage(type = "LFS", base_dir = tempdir(),
chunksize = 50 * 1024^2)
is.DStorage( ds )
```

KeyValue *Key/Value Pairs*

Description

Key/value pairs in "DList" objects.

Usage

```
DKeys( x )
```

Arguments

x	a "DList" object.

Value

A character vector representing all keys of the key/value pairs stored in chunks by "DList" objects.

Examples

```
## create a 2 elements DList
dl <- DList( line1 = "This is the first line.",
             line2 = "Now, the second line." )
## retrieve keys
DKeys( dl )
## remove DList and garbage collect it
rm( dl )
gc()
```

MapReduce *MapReduce for "DList" Objects*

Description

Interface to apply functions on elements of "DList" objects.

Usage

```
DLapply( x, FUN, parallel, ..., keep = FALSE )
DMap( x, MAP, parallel, keep = FALSE )
DReduce( x, REDUCE, parallel, ... )
```

Arguments

x	a "DList" object. Other objects (e.g., lists) will be coerced by as.DList.
FUN	the function to be applied to each element (i.e., the values) of x.
MAP	the function to be applied to each key/value pair in x.
REDUCE	the function to be applied to each key/value pair in x.
...	optional arguments to FUN or REDUCE.
parallel	logical; should the provided functions applied in parallel? Default: FALSE.
keep	logical; should the current data be kept as a separate revision for further processing later? Default: FALSE.

Details

The MapReduce programming model as defined by Dean and Ghemawat (2008) is as follows: the computation takes a set of input key/value pairs, and produces a set of output key/value pairs. The user expresses the computation as two functions: Map and Reduce. The Map function takes an input pair and produces a set of intermediate key/value pairs. The Reduce function accepts an intermediate key and a set of values for that key (possibly grouped by the MapReduce library). It merges these values together to form a possibly smaller set of values. Typically, just zero or one output value is produced per reduce invocation. Furthermore, data is usually stored on a (distributed) file system which is recognized by the MapReduce library. This allows such a framework to handle lists of values (here objects of class "DList") that are too large to fit in main memory (i.e., RAM).

Value

A "DList".

References

J. Dean and S. Ghemawat (2008). MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, **51**, 107–113.

Examples

```
dl <- DList( line1 = "This is the first line.",
             line2 = "Now, the second line." )
res <- DLapply( dl, function(x) unlist(strsplit(x, " ")) )
as.list( res )

foo <- function( keypair )
    list( key = paste("next_", keypair$key, sep = ""), value =
    gsub("first", "mapped", keypair$value) )

dlm <- DMap( x = dl, MAP = foo)
## retrieve keys
unlist(DGather(dlm, keys = TRUE, names = FALSE))
## retrieve values
as.list( dlm )
## simple wordcount based on two files:
dir(system.file("examples", package = "DSL"))
## first force 1 chunk per file (set max chunk size to 1 byte):
ds <- DStorage("LFS", tempdir(), chunksize = 1L)
## make "DList" from files, i.e., read contents and store in chunks
dl <- as.DList(system.file("examples", package = "DSL"), DStorage = ds)
## read files
dl <- DMap(dl, function( keypair ){
    list( key = keypair$key, value = tryCatch(readLines(keypair$value),
error = function(x) NA) )
})
## split into terms
splitwords <- function( keypair ){
    keys <- unlist(strsplit(keypair$value, " "))
   mapply( function(key, value) list( key = key, value = value), keys, rep(1L, length(keys)),
```

```
            SIMPLIFY = FALSE, USE.NAMES = FALSE )
}
res <- DMap( dl, splitwords )
as.list(res)
## now aggregate by term
res <- DReduce( res, sum )
as.list( res )
```

Index