Distributed R Manual

July 29, 2015

distributedR *Distributed R for Big Data*

Description

distributedR simplifies large-scale data analysis. It includes new language constructs to express distributed programs in R and an infrastructure to execute them. **distributedR** provides data-structures such as distributed array `darray` to partition and share data across multiple R instances. Users can express parallel execution using `foreach` loops.

Commands

distributedR contains the following commands. For more details use help function on each command.

Session manangement:

- `distributedR_start` - start session
- `distributedR_shutdown` - end session
- `distributedR_status` - obtain worker node information
- `distributedR_master_info` - obtain master host details

Distributed array, data frame and list:

- `darray` - create distributed array
- `dframe` - create distributed data frame
- `dlist` - create distributed list
- `as.darray` - create darray object from matrix object
- `is.darray` - check if object is distributed array
- `npartitions` - obtain total number of partitions
- `partitionsize` - obtain dimensions of partitions
- `getpartition` - fetch darray, dframe or dlist object
- `clone` - clone or deep copy of a darray

Distributed execution:

- `foreach` - execute function on cluster
- `splits` - pass partition to foreach loop
- `update` - make partition changes inside foreach loop globally visible

Author(s)

HP Vertica Development Team

References

- Venkataraman, S., Bodzsar, E., Roy, I., AuYoung, A., and Schreiber, R. (2013) Presto: Distributed Machine Learning and Graph Processing with Sparse Matrices. *EuroSys'13*, 197–210.
- Homepage: http://www.vertica.com/distributedr

Examples

```
## Not run:
  library(distributedR)
  distributedR_start()
  distributedR_status()
  distributedR_shutdown()

## End(Not run)
```

`distributedR_start` *distributedR_start*

Description

Starts distributedR in single-machine or cluster mode. By default, distributedR starts on the local machine with number of R instances equal to the number of CPU cores. For cluster mode, worker details should be present in cluster_conf file. After successful distributedR_start call, the master address and port number is displayed. This value is useful when a user wants to reference log files in workers.

Usage

```
distributedR_start (inst = 0, mem=0, cluster_conf="", log=2)
```

Arguments

`inst` number of R instances to launch at each worker. Default value is zero and will automatically start R instances equal to the number of CPU cores in each machine. This default value (zero) is ignored if Executors field is defined in cluster_conf file

`mem` allocated memory size of a worker node. The default value is zero and will use all the space present in /dev/shm on each worker. The default value (zero) is ignored if SharedMemory field is defined in cluster_conf file `cluster_conf` path to XML file describing configuration of master and workers. In the XML file, ServerInfo contains the IP address (hostname) and port number range of the master. StartPortRange and EndPortRange specifies the port number range that will be used by a master. The runtime will select a random open port within the range. In a master, the port number range has to contain at least two open ports.

Within the Workers field, information about multiple workers can be included. The worker field contains IP address (hostname) and port range. In a multi-worker cluster, loop-back address (127.0.0.1 or localhost) should not be used as it prohibits inter-node communication. StartPortRange and EndPortRange specifies the port number range that will be used by a worker. This range should contain at least 2*(number of workers)+1 open ports.

The Executor field sets the number of R instances. If the value is 0, the runtime will automatically start as many instances as the number of cores in each machine. SharedMemory determines the memory size assigned to each worker in MB. For the default value of 0, the runtime will automatically use the size of mounted shared memory region. Currently, memory constraints are not enforced.

An example configuration file is present in /opt/hp/distributedR/conf/cluster_conf.xml

`log` sets level of information generated in log files. The four severity levels are: 0 (ERROR), 1 (WARNING), 2 (INFOR) or 3 (DEBUG).

Severity level 0 (ERROR): only error messages are logged.

Severity level 1 (WARNING): error and warning messages are logged.

Severity level 2 (INFOR): additionally logs helpful messages. Set as default level.

Severity level 3 (DEBUG): verbose logging. Mainly applicable for debugging.

Details

distributedR execution generates three types of log files:

- Master log (`R_master_<username>_<master_addr>.<master_port>.log`) : contains Master level log messages on `foreach` functions received, task requests created and sent to Worker nodes for execution etc. It is created in the /tmp/ folder of the Master node.

- Worker log (`R_worker_<username>_<master_addr>.<master_port>.log`) : contains Worker level messages on requests received from Master node and other Worker nodes etc. It is created in /tmp/ folder of each Worker node.

- Executor log (`R_executor_<username>_<master_addr>.<master_port>_<executorID>.log`) : Each executor in each Worker node has its own log file. Normal execution log messages or Executor exceptions (depending on severity level chosen by user) are logged here. It is created in /tmp/ folder of each Worker node.

Review the Master and Executor Master logs for complete exception details if an Executor exception is encountered.

Author(s)

HP Vertica Development Team

References

- Venkataraman, S., Bodzsar, E., Roy, I., AuYoung, A., and Schreiber, R. (2013) Presto: Distributed Machine Learning and Graph Processing with Sparse Matrices. *EuroSys'13*, 197–210.
- Homepage: http://www.vertica.com/distributedr

See Also

`distributedR_shutdown`, `distributedR_status`, `distributedR_master_info`

Examples

```
## Not run:
library(distributedR)
##Start worker process
distributedR_start()
distributedR_status()
distributedR_master_info()
distributedR_shutdown()
## Cluster mode. Assumes location of configuration file
conf.dir = getwd()
distributedR_start(cluster_conf=paste(conf.dir,"/conf/cluster_conf.xml",sep=""))
distributedR_shutdown()

## End(Not run)
```

`distributedR_shutdown` *distributedR_shutdown*

Description

Shutdown session. Stops all workers, closes connections to them, and cleans resources. `distributedR_shutdown` is called automatically in the following cases:

- a worker or an R instance is killed
- user interrupts execution using CTRL-C and decides to shutdown the whole session

Usage

```
distributedR_shutdown()
```

Author(s)

HP Vertica Development Team

References

- Venkataraman, S., Bodzsar, E., Roy, I., AuYoung, A., and Schreiber, R. (2013) Presto: Distributed Machine Learning and Graph Processing with Sparse Matrices. *EuroSys'13*, 197–210.
- Homepage: http://www.vertica.com/distributedr

See Also

distributedR_start, distributedR_status

Examples

```
## Not run:
library(distributedR)
##Start worker process
distributedR_start()
distributedR_status()
distributedR_shutdown()

## End(Not run)
```

distributedR_status *distributedR_status*

Description

Show status of **distributedR** workers.

Usage

```
distributedR_status (help=FALSE)
```

Arguments

help	If true, describes each column

Value

Worker information is returned as a data.frame with the following columns:

Workers	IP and port of each worker.
Inst	number of R instances at each worker.
SysMem	total system memory at each worker.
MemUsed	used system memory at each worker.
DarrayQuota	total memory assgined for arrays. Not enforced by runtime.
DarrayUsed	memory used to store arrays.

Author(s)

HP Vertica Development Team

References

- Venkataraman, S., Bodzsar, E., Roy, I., AuYoung, A., and Schreiber, R. (2013) Presto: Distributed Machine Learning and Graph Processing with Sparse Matrices. *EuroSys'13*, 197–210.
- Homepage: http://www.vertica.com/distributedr

See Also

distributedR_start, distributedR_shutdown

Examples

```
## Not run:
library(distributedR)
##Start worker process
distributedR_start()
distributedR_status()
distributedR_shutdown()

## End(Not run)
```

darray *darray*

Description

Store in-memory, multi-dimensional data across several machines. Data can be partitioned into chunks of rows, columns, or blocks. Distributed arrays can store only numeric data.

Usage

```
darray (dim, blocks, sparse = FALSE, data = 0, empty=FALSE, distribution_policy)
darray (npartitions, sparse = FALSE, distribution_policy='roundrobin')
```

Arguments

`dim` the dim attribute for the array to be created. A vector specifying number of rows and columns.

`blocks` size of each partition as a vector specifying number of rows and columns.

`sparse` logical. Indicates if input array is a sparse.

`data` initial value of all elements in array. Default is 0.

`empty` if TRUE array is left unitialized, each partition is a zero matrix. Default is FALSE unless used with 'npartitions'.

`npartitions` vector specifying number of partitions.

`distribution_policy` defines policy to distribute array partitions across the workers. Default distribution policy is 'roundrobin'. Currently, this argument is only for internal use.

Details

By default, array partitions are internally stored as dense matrices. If an array is specified sparse, partitions are stored in the compressed sparse column format. Last set of partitions may have fewer rows or columns if array size is not an integer multiple of partition size. For example, the distributed array `darray(dim=c(5,5),blocks=c(2,5))` has three partitions. The first two partitions have two rows each but the last partition has only one row. All three partitions have five columns.

Distributed arrays can also be defined by specifying just the number of partitions, but not their sizes. This flexibility is useful when the size of an array is not known apriori. For example, `darray(npartitions=c(5,1))` is a dense array with five partitions. Each partition can contain any number of rows, though the number of columns should be same to conform to a well formed array.

Too many partitions increase the overheads of managing distributed objects. We recommend users to create objects with as many partitions as the total number of CPU cores in the system. We restrict users from creating objects when the number of partitions is more than 100,000 or more than no-of-cores*max(50, no-of-cores).

Distributed arrays can be read-shared by multiple concurrent tasks, but modified by only a single writer per partition. Programmers express parallelism by applying functions on array partitions in `foreach` loops. Loop body is executed at workers. Partitions can be passed as arguments using `splits`. Array modifications can be published globally using `update`.

Distributed arrays can be fetched at the master using `getpartition`. Number of partitions can be obtained by `npartitions`. Partitions are numbered from left to right, and then top to bottom, i.e., row major order. Dimension of each partition can be obtained using `partitionsize`.

Value

Returns a distributed array with the specified dimensions. Data may reside as partitions in remote nodes.

Author(s)

HP Vertica Development Team

References

- Venkataraman, S., Bodzsar, E., Roy, I., AuYoung, A., and Schreiber, R. (2013) Presto: Distributed Machine Learning and Graph Processing with Sparse Matrices. *EuroSys'13*, 197–210.
- Homepage: http://www.vertica.com/distributedr

See Also

`getpartition`, `npartitions`, `partitionsize`, `foreach`, `splits`, `update`, `dframe`, `dlist` `dimnames`

Examples

```
## Not run:
library(distributedR)
distributedR_start()
```

```
##Sparse array of size 10X10 with 10 partitions and each partition is of size 1X10
da<-darray(dim=c(10,10), blocks=c(1,10), sparse=TRUE)
getpartition(da)
cat("Input matrix dimension: ", da@dim, " block dimension: ", da@blocks,
" total number of partitions: ", npartitions(da),"\n")

##Dense array of size 9X9 with 3 partitions and each partition is of size 3X3
db<-darray(dim=c(9,9), blocks=c(3,3), sparse=FALSE, data=11)
cat("value of 3rd partition is: \n", getpartition(db,3),"\n")

##Flexible sized dense array. Five partitions, each with variable number of rows.
dc<-darray(npartitions=c(5,1))
foreach(i, 1:npartitions(dc), initArrays<-function(y=splits(dc,i), index=i) {
     y<-matrix(index, nrow=index,ncol=5)
     update(y)
})
cat("value of 2nd partition is: \n")
getpartition(dc,2)
getpartition(dc)
distributedR_shutdown()

## End(Not run)
```

dframe *dframe*

Description

Store in-memory, multi-dimensional data across several machines. Data can be partitioned into chunks of rows, columns, or blocks. Unlike distributed arrays, dframe can store both numeric and string data. However, dframe can be space-inefficient, and should be replaced by darray whereever possible.

Usage

```
dframe (dim, blocks, distribution_policy='roundrobin')
dframe (npartitions, distribution_policy='roundrobin')
```

Arguments

dim the dim attribute for the data frame to be created. A vector specifying number of rows and columns.

blocks size of each partition as a vector specifying number of rows and columns.

npartitions vector specifying number of partitions.

distribution_policy
defines policy to distribute data.frame partitions across the workers. The distribution policy used is 'roundrobin'. Currently, this argument is only for internal use.

Details

Distributed data frame partitions are internally stored as data.frame objects. Last set of partitions may have fewer rows or columns if data frame size is not an integer multiple of partition size. For example, the distributed data frame `dframe(dim=c(5,5),  blocks=c(2,5))` has three partitions. The first two partitions have two rows each but the last partition has only one row. All three partitions have five columns.

Distributed data frames can also be defined by specifying just the number of partitions, but not their sizes. This flexibility is useful when the size of an data frame is not known apriori. For example, `dframe(npartitions=c(5,1))` is a data frame with five partitions. Each partition can contain any number of rows, though the number of columns should be same to conform to a well formed data frame.

Too many partitions increase the overheads of managing distributed objects. We recommend users to create objects with as many partitions as the total number of CPU cores in the system. We restrict users from creating objects when the number of partitions is more than 100,000 or more than no-of-cores*max(50, no-of-cores).

Distributed data frames can be read-shared by multiple concurrent tasks, but modified by only a single writer per partition. Programmers express parallelism by applying functions on partitions in `foreach` loops. Loop body is executed at workers. Partitions can be passed as arguments using `splits`. Data frame modifications can be published globally using `update`.

Distributed data frames can be fetched at the master using `getpartition`. Number of partitions can be obtained by `npartitions`. Partitions are numbered from left to right, and then top to bottom.

Value

Returns a distributed data frame with the specified dimensions. Data may reside as partitions in remote nodes.

Author(s)

HP Vertica Development Team

References

- Venkataraman, S., Bodzsar, E., Roy, I., AuYoung, A., and Schreiber, R. (2013) Presto: Distributed Machine Learning and Graph Processing with Sparse Matrices. *EuroSys'13*, 197–210.
- Homepage: http://www.vertica.com/distributedr

See Also

`getpartition`, `npartitions`, `foreach`, `splits`, `update`, `darray`, `dimnames`

Examples

```
## Not run:
library(distributedR)
distributedR_start()
df <- dframe(c(20,4),c(10,2))
data_path<-system.file("extdata",package="distributedR")
```

```
file_path <- paste(data_path,"/df_data",sep="")
##Populate distributed data frame
foreach(i, 1:npartitions(df), function(sf=splits(df,i),ii=i,path=file_path){
  sf<-read.table(paste(path,ii,sep=""))
  update(sf)
})
getpartition(df)
##Rename columns
name_sample <- as.character(sample(1:4))
dimnames(df)[[2]] <- name_sample
getpartition(df)

##Flexible sized data frame. Five partitions, each with variable number of rows.
dc<-dframe(npartitions=c(5,1))
foreach(i, 1:npartitions(dc), initArrays<-function(y=splits(dc,i), index=i) {
     y<-data.frame(matrix(index, nrow=index,ncol=5))
     update(y)
})
cat("value of 2nd partition is: \n")
getpartition(dc,2)
getpartition(dc)

distributedR_shutdown()

## End(Not run)
```

dlist *dlist*

Description

Stores in-memory lists across several machines.

Just like R's list, dlist can store other R objects such as character, numeric and logical vectors, lists, matrices, and models. However, dlist can be space-inefficient, and should be replaced by darray whereever possible.

Usage

```
dlist (npartitions)
```

Arguments

npartitions   an integer specifying number of partitions of the list.

Details

Distributed lists are internally stored as list objects. Each partition of the list can have variable number of elements in it. For example, the distributed list dlist(npartitions=5) has five partitions. Each partition is an empty list list().

Too many partitions increase the overheads of managing distributed objects. We recommend users to create objects with as many partitions as the total number of CPU cores in the system. We restrict users from creating objects when the number of partitions is more than 100,000 or more than no-of-cores*max(50, no-of-cores).

Distributed lists can be read-shared by multiple concurrent tasks, but modified by only a single writer per partition. Programmers express parallelism by applying functions on dlist partitions in `foreach` loops. Loop body is executed at workers. Partitions can be passed as arguments using `splits`. List modifications can be published globally using `update`.

Distributed lists can be fetched at the master using `getpartition`. Number of partitions can be obtained by `npartitions`. Partitions are numbered from left to right

Value

Returns a distributed list with the specified number of partitions. Data may reside as partitions in remote nodes.

Author(s)

HP Vertica Development Team

References

- Venkataraman, S., Bodzsar, E., Roy, I., AuYoung, A., and Schreiber, R. (2013) Presto: Distributed Machine Learning and Graph Processing with Sparse Matrices. *EuroSys'13*, 197–210.
- Homepage: http://www.vertica.com/distributedr

See Also

`getpartition`, `npartitions`, `foreach`, `splits`, `update`, `darray`

Examples

```
## Not run:
library(distributedR)
distributedR_start()
dl <- dlist(5)
##Populate distributed list
foreach(i, 1:npartitions(dl), function(sf=splits(dl,i), idx=i){
  sf<-list(c("HP", idx))
  update(sf)
})
getpartition(dl)
distributedR_shutdown()

## End(Not run)
```

as.darray *as.darray*

Description

Convert input matrix into a distributed array.

Usage

```
as.darray(input, blocks)
```

Arguments

`input`	input matrix that will be converted to darray.
`blocks`	size of each partition as a vector specifying number of rows and columns.

Details

If partition size (blocks) is missing then the input matrix is row partitioned and striped across the cluster, i.e., the returned distributed array has approximately as many partitions as the number of R instances in the Distributed R session.

The last set of partitions may have fewer rows or columns if input matrix size is not an integer multiple of partition size. If `A` is a 5x5 matrix, then `as.darray(A, blocks=c(2,5))` is a distributed array with three partitions. The first two partitions have two rows each but the last partition has only one row. All three partitions have five columns.

To create a distributed array with just one partition, pass the dimension of the input array, i.e. `as.darray(A, blocks=dim(A))`

Value

Returns a distributed array with dimensions equal to that of the input matrix and partitioned according to argument 'blocks'. Data may reside as partitions on remote nodes.

Author(s)

HP Vertica Development Team

References

- Venkataraman, S., Bodzsar, E., Roy, I., AuYoung, A., and Schreiber, R. (2013) Presto: Distributed Machine Learning and Graph Processing with Sparse Matrices. *EuroSys'13*, 197–210.
- Homepage: http://www.vertica.com/distributedr

See Also

`darray`,`partitionsize`

Examples

```
## Not run:
library(distributedR)
distributedR_start()
##Create 4x4 matrix
mtx<-matrix(sample(0:1, 16, replace=T), nrow=4)
##Create distributed array spread across the cluster
da<-as.darray(mtx)
partitionsize(da)
##Create distributed array with single partition
db<-as.darray(mtx, blocks=dim(mtx))
partitionsize(db)
##Create distributed array with two partitions
dc<- as.darray(mtx, blocks=c(2,4))
partitionsize(dc)
##Fetch first partition
getpartition(dc,1)
distributedR_shutdown()

## End(Not run)
```

`as.dframe` *as.dframe*

Description

Convert input matrix/data.frame into a distributed frame.

Usage

```
as.dframe(input, blocks)
```

Arguments

`input` input matrix/data.frame that will be converted to dframe.

`blocks` size of each partition as a vector specifying number of rows and columns.

Details

If partition size (blocks) is missing then the input matrix/data.frame is row partitioned and striped across the cluster, i.e., the returned distributed frame has approximately as many partitions as the number of R instances in the Distributed R session.

The last set of partitions may have fewer rows or columns if input matrix size is not an integer multiple of partition size. If `A` is a 5x5 matrix, then `as.dframe(A, blocks=c(2,5))` is a distributed frame with three partitions. The first two partitions have two rows each but the last partition has only one row. All three partitions have five columns.

To create a distributed frame with just one partition, pass the dimension of the input frame, i.e. `as.dframe(A, blocks=dim(A))`

Value

Returns a distributed frame with dimensions equal to that of the input matrix/data.frame and partitioned according to argument 'blocks'. Data may reside as partitions on remote nodes.

Author(s)

HP Vertica Development Team

References

- Venkataraman, S., Bodzsar, E., Roy, I., AuYoung, A., and Schreiber, R. (2013) Presto: Distributed Machine Learning and Graph Processing with Sparse Matrices. *EuroSys'13*, 197–210.
- Homepage: http://www.vertica.com/distributedr

See Also

dframe,partitionsize

Examples

```
## Not run:
library(distributedR)
distributedR_start()
##Create 4x4 matrix
mtx<-matrix(sample(0:1, 16, replace=T), nrow=4)
##Create distributed frame spread across the cluster
df<-as.dframe(mtx)
partitionsize(df)
##Create distributed frame with single partition
db<-as.dframe(mtx, blocks=dim(mtx))
partitionsize(db)
##Create distributed frame with two partitions
dc<- as.dframe(mtx, blocks=c(2,4))
partitionsize(dc)
##Fetch first partition
getpartition(dc,1)
#creating of dframe with data.frame
dfa <- c(2,3,4)
dfb <- c("aa","bb","cc")
dfc <- c(TRUE,FALSE,TRUE)
df <- data.frame(dfa,dfb,dfc)
#creating dframe from data.frame with default block size
ddf <- as.dframe(df)
getpartition(ddf)
#creating dframe from data.frame with 1x1 block size
ddf <- as.dframe(df,blocks=c(1,1))
getpartition(ddf)
distributedR_shutdown()

## End(Not run)
```

is.darray *is.darray*

Description

Check if input object is darray.

Usage

```
is.darray(x)
```

Arguments

x	input object.

Value

Returns true if object is distributed array.

Author(s)

HP Vertica Development Team

References

- Venkataraman, S., Bodzsar, E., Roy, I., AuYoung, A., and Schreiber, R. (2013) Presto: Distributed Machine Learning and Graph Processing with Sparse Matrices. *EuroSys'13*, 197–210.
- Homepage: http://www.vertica.com/distributedr

See Also

darray

Examples

```
## Not run:
library(distributedR)
distributedR_start()
m<-matrix(sample(0:1, 16, replace=T), nrow=4)
is.darray(m)
dm<-darray(dim=c(5,5),blocks=c(1,5))
is.darray(dm)
distributedR_shutdown()

## End(Not run)
```

is.dframe *is.dframe*

Description

Check if input object is dframe.

Usage

```
is.dframe(x)
```

Arguments

x	input object.

Value

Returns true if object is distributed data.frame.

Author(s)

HP Vertica Development Team

References

- Venkataraman, S., Bodzsar, E., Roy, I., AuYoung, A., and Schreiber, R. (2013) Presto: Distributed Machine Learning and Graph Processing with Sparse Matrices. *EuroSys'13*, 197–210.
- Homepage: http://www.vertica.com/distributedr

See Also

dframe

Examples

```
## Not run:
library(distributedR)
distributedR_start()
df<-data.frame(x=rep(sample(0:1),4), y=rep(sample(5:6), 4), z=rep(sample(2:3), 4))
is.dframe(df)
ddf<-dframe(dim=c(5,5),blocks=c(1,5))
is.dframe(ddf)
distributedR_shutdown()

## End(Not run)
```

is.dlist *is.dlist*

Description

Check if input object is dlist.

Usage

```
is.dlist(x)
```

Arguments

`x`	input object.

Value

Returns true if object is distributed list.

Author(s)

HP Vertica Development Team

References

- Venkataraman, S., Bodzsar, E., Roy, I., AuYoung, A., and Schreiber, R. (2013) Presto: Distributed Machine Learning and Graph Processing with Sparse Matrices. *EuroSys'13*, 197–210.
- Homepage: http://www.vertica.com/distributedr

See Also

`dlist`

Examples

```
## Not run:
library(distributedR)
distributedR_start()
l <- list(x=sample(1:10, 5), y=sample(1:50, 5))
is.dlist(l)
dl<-dlist(naprtitions=4)
is.dlist(dl)
distributedR_shutdown()

## End(Not run)
```

as.factor.dframe *as.factor.dframe*

Description

Creates a clone of the input dframe with the specified categorical columns converted to factor.

Usage

```
as.factor.dframe(DF, colName, colID, trace=FALSE)
```

Arguments

`DF`	the input dframe. It must be partitioned row-wise.
`colName`	a vector of the name of the interested categorical columns.
`colID`	when colName is not available, column positions can be specified using a numerical vector.
`trace`	when it is FALSE (default) the progress of the foreach will be hidden.

Details

A factor on a column of a dframe is actually a set of ordinary factors on each data.frame of the dframe (each partition of a dframe is an ordinary data.frame). The important point is that all these distributed factors should have the same array of 'levels' to make sure that any particular type is mapped exactly to the same integer number all across the dframe.

The specified columns should be of type character, logical, or factor. When neither colName nor colID is specified, any column of these types will be selected. When both are specified, colID will be ignored.

Value

a clone of the input dframe with the specified categorical columns converted to factor.

Author(s)

HP Vertica Development Team

References

- Venkataraman, S., Bodzsar, E., Roy, I., AuYoung, A., and Schreiber, R. (2013) Presto: Distributed Machine Learning and Graph Processing with Sparse Matrices. *EuroSys'13*, 197–210.
- Homepage: http://www.vertica.com/distributedr

See Also

`dframe` `factor.dframe` `levels.dframe` `unfactor.dframe`

Examples

```
## Not run:
library(distributedR)
distributedR_start()
originalDF <- dframe(c(9,3),c(3,3))
foreach(i,1:npartitions(originalDF),function(dfi=splits(originalDF,i),idx=i){
    if(idx==1) {
        dfi[,1] <- 1:3
        dfi[,2] <- c('c1','c2','c3')
        dfi[,3] <- c('t1','t2','t3')
    } else if(idx==2) {
        dfi[,1] <- 2:4
        dfi[,2] <- c('c2','c3','c4')
        dfi[,3] <- c('t1','t2','t3')
    } else {
        dfi[,1] <- 11:13
        dfi[,2] <- c('c3','c4','c5')
        dfi[,3] <- c('t4','t5','t6')
    }
    update(dfi)
})
copyDF <- as.factor.dframe(originalDF, colID=c(2,3))
cp <- getpartition(copyDF,3)
cp[,3]
distributedR_shutdown()

## End(Not run)
```

---

factor.dframe *factor.dframe*

Description

This function converts several categorical columns to factor in-place; that is, the given input dframe will be modified.

Usage

```
factor.dframe(DF, colName, colID, trace=FALSE)
```

Arguments

DF the input dframe. It must be partitioned row-wise.

colName a vector of the name of the interested categorical columns.

colID when colName is not available, column positions can be specified using a numerical vector.

trace when it is FALSE (default) the progress of the foreach will be hidden.

Details

A factor on a column of a dframe is actually a set of ordinary factors on each data.frame of the dframe (each partition of a dframe is an ordinary data.frame). The important point is that all these distributed factors should have the same array of 'levels' to make sure that any particular type is mapped exactly to the same integer number all across the dframe.

The specified columns should be of type character, logical, or factor. When neither colName nor colID is specified, any column of these types will be selected. When both are specified, colID will be ignored.

Value

the input dframe is modified in-place and the function returns nothing.

Author(s)

HP Vertica Development Team

References

- Venkataraman, S., Bodzsar, E., Roy, I., AuYoung, A., and Schreiber, R. (2013) Presto: Distributed Machine Learning and Graph Processing with Sparse Matrices. *EuroSys'13*, 197–210.
- Homepage: http://www.vertica.com/distributedr

See Also

`dframe` `as.factor.dframe` `levels.dframe` `unfactor.dframe`

Examples

```
## Not run:
library(distributedR)
distributedR_start()
originalDF <- dframe(c(9,3),c(3,3))
foreach(i,1:npartitions(originalDF),function(dfi=splits(originalDF,i),idx=i){
    if(idx==1) {
        dfi[,1] <- 1:3
        dfi[,2] <- c('c1','c2','c3')
        dfi[,3] <- c('t1','t2','t3')
    } else if(idx==2) {
        dfi[,1] <- 2:4
        dfi[,2] <- c('c2','c3','c4')
        dfi[,3] <- c('t1','t2','t3')
    } else {
        dfi[,1] <- 11:13
        dfi[,2] <- c('c3','c4','c5')
        dfi[,3] <- c('t4','t5','t6')
    }
    update(dfi)
})
factor.dframe(originalDF, colID=c(2,3))
```

```
op <- getpartition(originalDF,3)
op[,3]
distributedR_shutdown()

## End(Not run)
```

unfactor.dframe *unfactor.dframe*

Description

Converts in-place the specified categorical columns of a dframe from factor to their labels of type character.

Usage

```
unfactor.dframe(DF, colName, colID, trace=FALSE)
```

Arguments

DF	the input dframe. It must be partitioned row-wise.
colName	a vector of the name of the interested categorical columns.
colID	when colName is not available, column positions can be specified using a numerical vector.
trace	when it is FALSE (default) the progress of the foreach will be hidden.

Details

The specified columns should be of type character, logical, or factor. When neither colName nor colID is specified, any column of these types will be selected. When both are specified, colID will be ignored.

Value

the input dframe is modified in-place and the function returns nothing.

Author(s)

HP Vertica Development Team

References

- Venkataraman, S., Bodzsar, E., Roy, I., AuYoung, A., and Schreiber, R. (2013) Presto: Distributed Machine Learning and Graph Processing with Sparse Matrices. *EuroSys'13*, 197–210.
- Homepage: http://www.vertica.com/distributedr

See Also

dframe factor.dframe as.factor.dframe levels.dframe

Examples

```
## Not run:
library(distributedR)
distributedR_start()
originalDF <- dframe(c(9,3),c(3,3))
foreach(i,1:npartitions(originalDF),function(dfi=splits(originalDF,i),idx=i){
    if(idx==1) {
        dfi[,1] <- 1:3
        dfi[,2] <- c('c1','c2','c3')
        dfi[,3] <- c('t1','t2','t3')
    } else if(idx==2) {
        dfi[,1] <- 2:4
        dfi[,2] <- c('c2','c3','c4')
        dfi[,3] <- c('t1','t2','t3')
    } else {
        dfi[,1] <- 11:13
        dfi[,2] <- c('c3','c4','c5')
        dfi[,3] <- c('t4','t5','t6')
    }
    update(dfi)
})
factor.dframe(originalDF, colID=c(2,3))
op <- getpartition(originalDF,3)
op[,3]
unfactor.dframe(originalDF, colID=c(2,3))
op <- getpartition(originalDF,3)
op[,3]
distributedR_shutdown()

## End(Not run)
```

---

levels.dframe *levels.dframe*

Description

Finds the list of the labels on the categorical columns of a dframe.

Usage

```
levels.dframe(DF, colName, colID, trace=FALSE)
```

Arguments

DF the input dframe. It must be partitioned row-wise.

colName a vector of the name of the interested categorical columns.

colID when colName is not available, column positions can be specified using a numerical vector.

trace when it is FALSE (default) the progress of the foreach will be hidden.

Details

The specified columns should be of type character, logical, or factor. When neither colName nor colID is specified, any column of these types will be selected. When both are specified, colID will be ignored.

Value

Levels the array of the labels on the categorical columns of a dframe.

columns the position of columns that their levels are respectively returned in Levels.

Author(s)

HP Vertica Development Team

References

- Venkataraman, S., Bodzsar, E., Roy, I., AuYoung, A., and Schreiber, R. (2013) Presto: Distributed Machine Learning and Graph Processing with Sparse Matrices. *EuroSys'13*, 197–210.
- Homepage: http://www.vertica.com/distributedr

See Also

dframe factor.dframe as.factor.dframe unfactor.dframe

Examples

```
## Not run:
library(distributedR)
distributedR_start()
originalDF <- dframe(c(9,3),c(3,3))
foreach(i,1:npartitions(originalDF),function(dfi=splits(originalDF,i),idx=i){
    if(idx==1) {
        dfi[,1] <- 1:3
        dfi[,2] <- c('c1','c2','c3')
        dfi[,3] <- c('t1','t2','t3')
    } else if(idx==2) {
        dfi[,1] <- 2:4
        dfi[,2] <- c('c2','c3','c4')
        dfi[,3] <- c('t1','t2','t3')
    } else {
        dfi[,1] <- 11:13
        dfi[,2] <- c('c3','c4','c5')
        dfi[,3] <- c('t4','t5','t6')
    }
    update(dfi)
})
levels.dframe(originalDF, colID=c(2,3))
distributedR_shutdown()

## End(Not run)
```

---

npartitions *npartitions*

Description

Return number of partitions in darray, dframe or dlist.

Usage

```
npartitions (x)
npartitions2D (x)
```

Arguments

x	input distributed array, distributed data frame or distributed list.

Details

npartitions returns the total number of partitions in the distributed object. Use npartitions2D to obtain the number of partitions along each direction.

In darray(dim=c(9,10), blocks=c(3,5)), the distributed array is partitioned blockwise. npartittions will return 6 (total number of partitions) while npartitions2D will return (3,2), i.e., 3 partitions along the row and 2 along the column axis.

Mathematically, npartitions(x)=npartitions2D(x)[1]*npartitions2D(x)[2]

Value

npartitions return an integer that denotes the number of partitions. npartitions2D return a vector that denotes the number of partitions in each direction.

Author(s)

HP Vertica Development Team

References

- Venkataraman, S., Bodzsar, E., Roy, I., AuYoung, A., and Schreiber, R. (2013) Presto: Distributed Machine Learning and Graph Processing with Sparse Matrices. *EuroSys'13*, 197–210.
- Homepage: http://www.vertica.com/distributedr

See Also

darray, dframe, getpartition, dlist

Examples

```
## Not run:
library(distributedR)
distributedR_start()
##Input array of size 5X5 with 4 partitions
da<-darray(dim=c(5,5), blocks=c(3,3), data=7)
npartitions(da)
npartitions2D(da)
distributedR_shutdown()

## End(Not run)
```

partitionsize *partitionsize*

Description

Return dimension of partitions in darray, dframe or dlist.

Usage

```
    partitionsize (x, index)
    partitionsize (x)
```

Arguments

x input distributed array, distributed data frame or distributed list.

index index of partition. If missing sizes of all partitions are returned.

Value

A matrix that denotes the number of rows and columns in the partition. Row i of the matrix corresponds or size of i'th partition.

Author(s)

HP Vertica Development Team

References

- Venkataraman, S., Bodzsar, E., Roy, I., AuYoung, A., and Schreiber, R. (2013) Presto: Distributed Machine Learning and Graph Processing with Sparse Matrices. *EuroSys'13*, 197–210.
- Homepage: http://www.vertica.com/distributedr

See Also

darray, dframe, getpartition, dlist

Examples

```
## Not run:
library(distributedR)
distributedR_start()
##Input array of size 5X5 with 4 partitions
da<-darray(dim=c(5,5), blocks=c(3,3), data=7)
partitionsize(da,1)
partitionsize(da,2)
partitionsize(da)
distributedR_shutdown()

## End(Not run)
```

getpartition *getpartition*

Description

Fetch partition(s) of darray, dframe or dlist from remote workers.

Usage

```
getpartition (x, y, z)
```

Arguments

x	input distributed array, distributed data frame or distributed list.
y	index of partition to fetch. In a 2-D partition this is the row-index of partition (number of partitions above).
z	column-index of the partition in a 2-D partitioning scheme (number of partitions to the left).

Details

If both y and z are missing then the full input darray, dframe or dlist is returned.

2-D partitioning is valid only for darray and dframe. Since dlist is paritioned length wise, only argument y is used to fetch a dlist partition. Argument z is undefined for dlist.

Partitions are numbered from left to right and then top to bottom, i.e., row-major order. Partition numbers start from 1. For row partitioning (each partition has all the columns) or column partitioning (each partition has all the rows) index argument z should not be used. For 2-D partitioning, both index argument y and z may be used.

For example, the array darray(dim=c(5,5),blocks=c(3,3)) has four partitions. To fetch the bottom left partition we can either only use argument y = 3 or 2-D indexing where y=2, z=1.

Value

An array, data.frame or list corresponding to the input darray, dframe or dlist partition(s).

Author(s)

HP Vertica Development Team

References

- Venkataraman, S., Bodzsar, E., Roy, I., AuYoung, A., and Schreiber, R. (2013) Presto: Distributed Machine Learning and Graph Processing with Sparse Matrices. *EuroSys'13*, 197–210.
- Homepage: http://www.vertica.com/distributedr

See Also

darray, dframe

Examples

```
## Not run:
library(distributedR)
distributedR_start()
##Input array of size 5X5 with 4 partitions
da<-darray(dim=c(5,5), blocks=c(3,3), data=7)
##Return full array
getpartition(da)
##Return third partition (bottom-left)
getpartition(da,3)
##Return fourth partition (bottom-right)
getpartition(da,2,2)
##Input list with 5 partitions
dl<- dlist(5)
##Return the third partition
getpartition(dl,3)
distributedR_shutdown()

## End(Not run)
```

clone *clone*

Description

Create a copy of input object. Can be used to clone the structure of the object, e.g., same number of partitions and each partition with the same dimension.

Usage

```
clone(input)
clone(input, nrow=NA, ncol=NA, data=0, sparse=NA)
```

Arguments

`input`	object to be cloned.
`nrow`	number of rows in the output. By default each partition in the output will have same number of rows as the input object's partitions.
`ncol`	number of columns in the output. By default each partition in the output will have same number of columns as the input object's partitions.
`data`	value of each element in the output object. Default is 0.
`sparse`	whether the output object should be a sparse array. By default the output object is dense (sparse) if the input objet is dense (sparse). Used only when input object is an array.

Details

Setting distributed data-structures such as a `darray` equal to another does not result in a copy. For example, after assignment `da = db`, the two distributed arrays `da` and `db` will refer to the same data. Operations on any of these arrays will manipulate the same single copy of data. To make a copy, a `darray` needs to be explicitly cloned using `clone`.

`clone` can also be used to copy just the structure of a distributed object, such as the number of partitions and the partition sizes. For example, if `da` is a Nx10 distributed dense array, `db<-clone(da, ncol=1, data=2)` will create a dense array with same number of rows as `da` but with only 1 column. All elements in the resulting darray will be 2. When copying the structure of a distributed object, only one of `nrow` or `ncol` can be used, ensuring that the system keeps one of the dimension same as the original data-structure. Note that if any argument, such as `nrow` or `ncol`, is used with `clone` then only the structure, and not the contents, of the input object is copied. The content of the output object is determined by the argument `data`.

Value

A `darray` with the dimension, block size, and values as the input distributed array unless `clone` is called with options.

Author(s)

HP Vertica Development Team

References

- Venkataraman, S., Bodzsar, E., Roy, I., AuYoung, A., and Schreiber, R. (2013) Presto: Distributed Machine Learning and Graph Processing with Sparse Matrices. *EuroSys'13*, 197–210.
- Homepage: http://www.vertica.com/distributedr

See Also

darray

Examples

```
## Not run:
library(distributedR)
distributedR_start()
mtx<-matrix(sample(0:1, 16, replace=T), nrow=4)
da<-as.darray(mtx)
db<-clone(da)
all(da==db)
dc<-clone(da, ncol=2, data=2)
getpartition(dc)
distributedR_shutdown()

## End(Not run)
```

---

foreach *foreach*

Description

Execute function in parallel as distributed tasks. Implicit barrier at the end of loop.

Usage

```
foreach(index, range, func, progress=TRUE, scheduler=0)
```

Arguments

`index`	loop index.
`range`	vector. Range of loop index.
`func`	function to execute in parallel.
`progress`	display progress bar if TRUE.
`scheduler`	choose task placement policy. Default policy minimizes data movement. Set to 1 if tasks should be placed on the worker where the first argument resides.

Details

`foreach` executes a function in parallel on worker nodes. Programmers can pass any R object as argument to the function. Distributed arrays, data frames or lists, and their partitions can be passed using `splits`.

The `foreach` loop or the function executed by it does not return any value. Instead, users can call `update` inside `func` to modify distributed arrays, data frames or lists and publish changes. Note that `update` is the only way to make side-effects globally visible.

Author(s)

HP Vertica Development Team

References

- Venkataraman, S., Bodzsar, E., Roy, I., AuYoung, A., and Schreiber, R. (2013) Presto: Distributed Machine Learning and Graph Processing with Sparse Matrices. *EuroSys'13*, 197–210.
- Homepage: http://www.vertica.com/distributedr

See Also

darray, dframe, dlist, splits, update, npartitions

Examples

```
## Not run:
library(distributedR)
distributedR_start()
da <- darray(dim=c(9,9), blocks=c(3,3), sparse=FALSE, data=10)
cat("Number of partitions of da are ", npartitions(da),"\n")
db <- darray(dim=c(9,9), blocks=c(3,3), sparse=FALSE, data=5)
result <- darray(dim=c(9,9), blocks=c(3,3))
##Add two matrices in parallel
foreach(i, 1:npartitions(da),
  add<-function(a = splits(da,i),
              b = splits(db,i),
              c = splits(result,i)){
    c <- a + b
    update(c)
  })
getpartition(result)
distributedR_shutdown()

## End(Not run)
## Not run:
library(distributedR)
distributedR_start()

da <- darray(dim=c(6,9),blocks=c(3,3),sparse=FALSE,data=0)

##First index gets c(1,2,3), second index gets c(4,5,6), etc.
get.three.indices <- function(index){
    start = 3*(index-1) + 1
    indices = c(start:(start+2))
    indices
}

##Update darray 3 splits at a time, with parallelism of 2
foreach(i,1:2,function(a=splits(da,get.three.indices(i)),data=get.three.indices(i)){
       a[[1]] = matrix(data[[1]],nrow(a[[1]]),ncol(a[[1]]))
       a[[2]] = matrix(data[[2]],nrow(a[[2]]),ncol(a[[2]]))
```

```
        a[[3]] = matrix(data[[3]],nrow(a[[3]]),ncol(a[[3]]))
        update(a)
    })

getpartition(da) ##counts from 1 to 6 in 3x3 blocks!
distributedR_shutdown()

## End(Not run)
```

---

splits *splits*

Description

Pass partition(s) of `darray`, `dframe` or `dlist` to function in `foreach`.

Usage

```
splits(x, y)
```

Arguments

`x`	input distributed array, distributed data frame or distributed list.
`y`	index or indices of partition to fetch; this can be an expression that evaluates to a scalar, vector, or list

Details

`splits` can be used only as an argument to the function in a `foreach` loop.

If y is missing then the full input `darray`, `dframe` or `dlist` is returned.

y can be either evaluate to a scalar value, a vector, or an R list object.

If y is a scalar, it represents one split. If y is a vector or a list, then it represents anything from one to multiple splits.

If y is defined to be a vector or a list (of length > 1), then `splits(x,y)` will represent multiple splits. For example, in the statement, `foreach(i,1:2,function(B=splits(A,1:3)){})`, B will contain splits 1-3 of dobject A. Inside the body of the foreach function statement, B will be a list – i.e., `B[[1]]` will contain the first split of A.

Similarly, one can may also write `B = splits(A,list(2,4,5))`, in which case `B[[1]]` would be equivalent to `splits(A,2)`.

In place of y may also be a function. If this function evaluates to a scalar,list,or vector, then it will be a valid splits statement.

Partitions are numbered from left to right and then top to bottom, i.e., row-major order. Partition numbers start from 1.

For example, the array `A=darray(dim=c(5,5),blocks=c(3,3))` has four partitions. To fetch the bottom left partition we can use `splits(A, 3)`.

Value

A reference to the darray, dframe or dlist partition(s).

Author(s)

HP Vertica Development Team

References

- Venkataraman, S., Bodzsar, E., Roy, I., AuYoung, A., and Schreiber, R. (2013) Presto: Distributed Machine Learning and Graph Processing with Sparse Matrices. *EuroSys'13*, 197–210.
- Homepage: http://www.vertica.com/distributedr

See Also

darray, dframe, dlist, update, foreach

Examples

```
## Not run:
library(distributedR)
distributedR_start()
da <- darray(dim=c(9,9), blocks=c(3,3), sparse=FALSE, data=10)
cat("Number of partitions of da are ", npartitions(da),"\n")
db <- darray(dim=c(9,9), blocks=c(3,3), sparse=FALSE, data=5)
result <- darray(dim=c(9,9), blocks=c(3,3))
##Add two matrices in parallel
foreach(i, 1:npartitions(da),
  add<-function(a = splits(da,i),
              b = splits(db,i),
              c = splits(result,i)){
    c <- a + b
    update(c)
  })
getpartition(result)
distributedR_shutdown()

## End(Not run)
## Not run:
library(distributedR)
distributedR_start()

da <- darray(dim=c(6,9),blocks=c(3,3),sparse=FALSE,data=0)

##First index gets c(1,2,3), second index gets c(4,5,6), etc.
get.three.indices <- function(index){
    start = 3*(index-1) + 1
    indices = c(start:(start+2))
    indices
}
```

```
##Update darray 3 splits at a time, with parallelism of 2
foreach(i,1:2,function(a=splits(da,get.three.indices(i)),data=get.three.indices(i)){
       a[[1]] = matrix(data[[1]],nrow(a[[1]]),ncol(a[[1]]))
       a[[2]] = matrix(data[[2]],nrow(a[[2]]),ncol(a[[2]]))
       a[[3]] = matrix(data[[3]],nrow(a[[3]]),ncol(a[[3]]))
       update(a)
    })

getpartition(da) ##counts from 1 to 6 in 3x3 blocks!
distributedR_shutdown()

## End(Not run)
```

update *update*

Description

Globally publish modifications done to a darray, dframe or dlist inside a foreach loop.

Usage

```
update(x)
```

Arguments

x	input array, data.frame, list, or list of multiple splits.

Details

update can be used only inside the foreach loop function.

The foreach loop or the function executed by it does not return any value. Instead, users can call update to modify distributed arrays, data frames or lists and publish changes. Note that update is the only way to make side-effects globally visible.

Also note that if x is a list of multiple splits, then all of the splits associated with the list will be updated. Updating a specific partition of a list of multiple splits is not currently supported.

Author(s)

HP Vertica Development Team

References

- Venkataraman, S., Bodzsar, E., Roy, I., AuYoung, A., and Schreiber, R. (2013) Presto: Distributed Machine Learning and Graph Processing with Sparse Matrices. *EuroSys'13*, 197–210.
- Homepage: http://www.vertica.com/distributedr

See Also

darray, dframe, dlist, update, foreach

Examples

```
## Not run:
library(distributedR)
distributedR_start()
da <- darray(dim=c(9,9), blocks=c(3,3), sparse=FALSE, data=10)
cat("Number of partitions of da are ", npartitions(da),"\n")
db <- darray(dim=c(9,9), blocks=c(3,3), sparse=FALSE, data=5)
result <- darray(dim=c(9,9), blocks=c(3,3))
##Add two matrices in parallel
foreach(i, 1:npartitions(da),
  add<-function(a = splits(da,i),
                b = splits(db,i),
                c = splits(result,i)){
    c <- a + b
    update(c)
  })
getpartition(result)
distributedR_shutdown()

## End(Not run)
## Not run:
library(distributedR)
distributedR_start()

da <- darray(dim=c(6,9),blocks=c(3,3),sparse=FALSE,data=0)

##First index gets c(1,2,3), second index gets c(4,5,6), etc.
get.three.indices <- function(index){
    start = 3*(index-1) + 1
    indices = c(start:(start+2))
    indices
}

##Update darray 3 splits at a time, with parallelism of 2
foreach(i,1:2,function(a=splits(da,get.three.indices(i)),data=get.three.indices(i)){
       a[[1]] = matrix(data[[1]],nrow(a[[1]]),ncol(a[[1]]))
       a[[2]] = matrix(data[[2]],nrow(a[[2]]),ncol(a[[2]]))
       a[[3]] = matrix(data[[3]],nrow(a[[3]]),ncol(a[[3]]))
       update(a)
    })

getpartition(da) ##counts from 1 to 6 in 3x3 blocks!
distributedR_shutdown()

## End(Not run)
```

Index