Distributed R Manual

February 6, 2014

`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

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
- `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.hpl.hp.com/research/distributedr.htm

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 one less than 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=3)
```

Arguments

`inst`	number of R instances to launch at each worker. Setting this to zero will automatically start R instances one less than the number of CPU cores in each machine. This value is ignored if Executors field is defined in cluster_conf file
`mem`	allocated memory size of a worker node. This value is ignored if SharedMemory field is defined in cluster_conf file
`cluster_conf`	path to XML file describing configuration of master and workers. File should contain hostname (or IP address) and port number of master and workers. In Workers field, Executors field determines the number of executors in a worker, and SharedMemory determines the size of shared memory. Executors and SharedMemory fileds are optional, and default value (0) will be used unless inst or mem are specified in the arguments. Example configuration file is in $distributedR_HOME /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 file (`R_master_<username>_<hostname>.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 file (`R_worker_<username>_<hostname>.<worker_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 file (`R_executor_<username>_<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.hpl.hp.com/research/distributedr.htm

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.hpl.hp.com/research/distributedr.htm

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.hpl.hp.com/research/distributedr.htm

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 = 1)
```

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 1.

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 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.

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.hpl.hp.com/research/distributedr.htm

See Also

getpartition, npartitions, 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")
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)
```

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.

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 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.hpl.hp.com/research/distributedr.htm

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)
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 not present then a distributed array with only a single partition is created. Last set of partitions may have fewer rows or columns if input matrix size is not an integer multiple of partition size. For example, the distributed array `as.darray(matrix(1,nrow=5,ncol=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.

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 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.hpl.hp.com/research/distributedr.htm

See Also

darray

Examples

```
## Not run:
library(distributedR)
distributedR_start()
##Create 4x4 matrix
mtx<-matrix(sample(0:1, 16, replace=T), nrow=4)
##Create distributed array with single partition
da<-as.darray(mtx)
da@dim
da@blocks
getpartition(da)
##Create distributed array with two partitions
db<- as.darray(mtx, blocks=c(2,4))
db@blocks
##Fetch first partition
getpartition(db,1)
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.hpl.hp.com/research/distributedr.htm

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)
```

npartitions *npartitions*

Description

Return number of partitions in darray, dframe or dlist.

Usage

```
npartitions (x)
```

Arguments

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

Value

An integer that denotes the number of partitions.

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.hpl.hp.com/research/distributedr.htm

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)
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.hpl.hp.com/research/distributedr.htm

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 darray.

Usage

```
clone(input)
```

Arguments

`input`	darray to be cloned.

Details

Setting 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`.

Value

A `darray` with the same dimension, block size and values as the input 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.hpl.hp.com/research/distributedr.htm

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)
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)
```

Arguments

index	loop index.
range	vector. Range of loop index.
func	function to execute in parallel.
progress	display progress bar if TRUE.

Details

foreach executes a function in parallel on worker nodes. Programmers can pass any R object as argument to the function. Distributed array, data frame 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.hpl.hp.com/research/distributedr.htm

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)
```

---

splits *splits*

Description

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

Usage

```
splits(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

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

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

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.hpl.hp.com/research/distributedr.htm

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)
```

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 or list.

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.

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.hpl.hp.com/research/distributedr.htm

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)
```

Index