Package 'RStorm'

July 2, 2014

Type Package

Title Simulate and Develop Streaming Processing in [R]

Version 0.902

Date 2013-07-26

Author Maurits Kaptein

Maintainer Maurits Kaptein <maurits@mauritskaptein.com>

Description While streaming processing provides opportunities to deal with extremely large and ever growing data sets in (near) real time, the development of streaming algorithms for complex models is often cumbersome: the software packages that facilitate streaming processing in production environments do not provide statisticians with the simulation, estimation, and plotting tools they are used to. Developers of streaming algorithms would thus benefit from the flexibility of [R] to create, plot and compute data while developing streaming algorithms. Package RStorm implements a streaming architecture modeled on Storm for easy development and testing of streaming algorithms in [R]. RStorm is not intended as a production package, but rather a development tool for streaming algorithms.

License GPL-2

Depends plyr

NeedsCompilation no

Repository CRAN

Date/Publication 2013-08-28 12:25:31

R topics documented:

`RStorm-package` *Simulate a Streaming Process in [R]*

Description

While streaming processing provides opportunities to deal with extremely large and ever growing data sets in (near) real time, the development of streaming algorithms for complex models is often cumbersome: the software packages that facilitate streaming processing in production environments do not provide statisticians with the simulation, estimation, and plotting tools they are used to. Developers of streaming algorithms would thus benefit from the flexibility of [R] to create, plot and compute data while developing streaming algorithms. RStorm implements a streaming architecture modeled on Storm for easy development and testing of streaming algorithms in [R]. Package RStorm is not intended as a production package, but rather a development tool for streaming algorithms. See the below examples for some of the usages of RStorm for the development and comparison of streaming algorithms.

Details of the package, examples of streaming algorithms, and examples of the use of RStorm can be found at http://software.mauritskaptein.com/RStorm

Details

Package:	RStorm
Type:	Package
Version:	0.9
Date:	2013-07-26
License:	GPL-2

Author(s)

Maurits Kaptein

Maintainer: Maurits Kaptein <maurits@mauritskaptein.com>

See Also

ddply RStorm Topology

Examples

```
##############################
# a simple stream to compute a sum:
##############################

# create some data:
x <- seq(1, 1000)

# start a topology
topology <- Topology(data.frame(x=x))

# define a bolt and add it to the topology
computeSum <- function(x, ...){
sum <- GetHash("sum")
if(is.data.frame(sum)){
x <- sum + (x[1])
}
SetHash("sum", x)
}
topology <- AddBolt(topology, Bolt(computeSum))

# Run the stream:
result <- RStorm(topology)

# Inspect the result
print(GetHash("sum", result))

#plot(topology)


##############################
# Example of a stream to compare two
# methods of streaming variance computation:
##############################

# Generate some data
set.seed(10)
t <- 100
x <- rnorm(t,0,1)
# Look at the variance as computed by var():
var(x)
```

```
# Start a topology
topology <- Topology(data.frame(x=x))

# Bolt for "Sum of Squares Method" with tracking over time
var.SS <- function(x, ...){
params <- GetHash("params1")
if(!is.data.frame(params)){
params <- list()
params$n <- params$sum <- params$sum2 <- 0
}
n <- params$n + 1
sum <- params$sum + as.numeric(x[1])
sum2 <- params$sum2 + as.numeric(x[1]^2)
if(n>1){
var <- 1/(n*(n-1)) * (n*sum2 - sum^2)
} else {
var <- 0
}
SetHash("params1", data.frame(n=n, sum=sum, sum2=sum2, var=var))
TrackRow("var.SS", data.frame(var=var))
}


## Bolt for "Welford's" Method:

var.Welford <- function(x, ...){
x <- as.numeric(x[1])
params <- GetHash("params2")
if(!is.data.frame(params)){
params <- list()
params$M <- x
params$S <- params$n <- 0
}
n <- params$n + 1
M <- params$M + ( x - params$M) / n
S <- params$S + (x - params$M)*(x-M)

if(n>1){
var <- S / (n-1)
} else {
var <- 0
}
SetHash("params2", data.frame(n=n, M=M, S=S, var=var))
TrackRow("var.Welford", data.frame(var=var))
}

# Add both topologies to a Stream:
topology <- AddBolt(topology, Bolt(var.SS))
topology <- AddBolt(topology, Bolt(var.Welford))
result <- RStorm(topology)

# Plot the results over the stream
plot(c(1:t), GetTrack("var.Welford", result)$var, type="l")
```

```
lines(c(1:t), GetTrack("var.SS", result)$var, col="red")


### Similar, but with a dataset
###  in which the mean is very large compared to the variance:
x2 <- rnorm(t,10^8,1)
topology2 <- Topology(data.frame(x=x2))
topology2 <- AddBolt(topology2, Bolt(var.SS))
topology2 <- AddBolt(topology2, Bolt(var.Welford))
result2 <- RStorm(topology2)

# This time the standard SS methods screws up (mind the different y scale):
# (And mind the fact that the SS method gives NEGATIVE variance)
plot(c(1:t), GetTrack("var.Welford", result2)$var, type="l", ylim=c(-10, 11))
lines(c(1:t), GetTrack("var.SS", result2)$var, col="red")
```

AddBolt — *Function to add a* Bolt *to a* Topology *object to specify a stream.*

Description

AddBolt is an auxiliary function for building up a RStorm topology. After initializing a Topology object the AddBolt function can be used to build the topology and specify the order of the Bolts. A Bolt receives as its first argument the Tuple emitted by the previous element in the Stream.

Usage

```
AddBolt(topology, bolt, .verbose = TRUE)
```

Arguments

topology	a Topology object to add the bolt to.
bolt	a Bolt to add to the topology. These are created using the Bolt() function.
.verbose	a logical indicator to state whether or not verbose output should be printed. Default TRUE.

Value

An object of type Topology which is a list containing the following elements:

spout	the data.frame passed as a spout
bolts	a list of bolts, see Bolt
finailze	the finalize function to be used for the stream

The specified Bolt has now been added to the list of bolts. See Topology for more info.

Warning

Functions which get added to a Topology using the AddBolt functionality should always use the ... argument. This argument is used to facilitate the processing of the stream. See example below for a minimal functional example of a correctly specified bolt.

Author(s)

Maurits Kaptein

See Also

See Also: `Topology`, `Bolt`, `RStorm`

Examples

```
# Create a topology and add a bolt to it.
bolt1 <- function(x, ...){print(x)}
topology <- Topology(data.frame(x=c(1:10)))
topology <- AddBolt(topology, Bolt(bolt1, listen=0))
topology
```

`AddFinalize` *Function to add a finalize function to a* `Topology`

Description

`AddFinalize` is an auxiliary function for building up a `RStorm` topology. After initializing a `Topology` object the `AddFinalize` function can be used to add a final function, which receives as its parameter a list of all hashMaps stored during the stream. After running the stream `GetHashList` can be used to receive an object that is the same as the object received by the finalize function.

Arguments

`topology`	a `Topology` object to add the bolt to.
`bolt`	a `Bolt` to add to as the finalize function.
`.verbose`	a logical indicator to state whether or not verbose output should be printed.

Value

An object of type `Topology` which is a list containing the following elements:

`spout`	the data.frame passed as a spout
`bolts`	a list of bolts, see `Bolt`
`finailze`	the finalize function to be used for the stream

The specified `Bolt` has now been added to the finalize function. See `Topology` for more info.

Author(s)

Maurits Kaptein

See Also

See Also: `Topology`, `Bolt`, `RStorm`

Examples

```
bolt1 <- function(x, ...){print(x)}
topology <- Topology(data.frame(x=c(1:10)))
topology <- AddFinalize(topology, Bolt(bolt1))
topology
```

`Bolt` *Function to create a* `Bolt` *object to add to a stream*

Description

Function to create a `Bolt` object. A `Bolt` object consists of a function which receives as its first argument the `Tuple` emitted by the element the `Bolt` listens to.

Usage

```
Bolt(FUNC, listen = 0, boltID = 0)
```

Arguments

`FUNC`	a `[R]` function with a first argument which receive a `Tuple` and using the ,... specification to receive arguments auxiliary to the functioning of the stream.
`listen`	a number indicating which element in the topology to listen too. $0$ indicates the Spout itself, while other integers refer to the Tuples emitted by other Bolts along the Stream. Default is $0$. Printing the Topology allows one to see the position number of each Bolt.
`boltID`	(optional) the ID of this bolt. A given name – mostly a number – to pass the name of the bolt to the bolt itself which can be used to create (e.g.) a hashMap that is distinct for the current bolt.

Value

An object of type `Bolt` which is a list containing the following elements:

`name`	the name of the function that was passed when initializing the Bolt
`func`	the function itself
`listen`	the identifier of the element in the stream from which the Bolt receives its Tuples
`id`	the id of the current Bolt

Warning

Functions used as bolt in a stream should always use the dots argument (...) to facilitate the internal working of `RStorm`.

Additional Info

The `is.Bolt` function checks whether an object is of Type Bolt and is used internally.

Author(s)

Maurits Kaptein

See Also

See Also: `Topology`, `AddBolt`, `RStorm`

Examples

```
# Create a Bolt:
bolt1 <- function(x, ...){print(x)}
Bolt(bolt1, listen=0, boltID=12)
```

`ChangeSpout` *Function to change the Spout of a Topology*

Description

The `ChangeSpout` function is used to change the spout of an topology that was already defined. It can be used (e.g) for the simulation of a data stream process on multiple data-sets.

Usage

```
ChangeSpout(topology, spout)
```

Arguments

`topology`	a RStorm Topology object.
`spout`	a new spout. E.g., a new codedata.frame.

Value

An object of class `Topology` which is a list containing the following elements:

`spout`	the data.frame passed as a spout
`bolts`	a list of bolts, see `Bolt`
`finailze`	the finalize function to be used for the stream

Warning

Functions used as bolt in a stream should always use the dots argument (...) to facilitate the internal working of `RStorm`.

Additional Info

The `is.Bolt` function checks whether an object is of Type Bolt and is used internally.

Author(s)

Maurits Kaptein

See Also

See Also: `Topology`, `AddBolt`, `RStorm`

Examples

```
# create a data set.
x <- seq(1, 100)
topology <- Topology(data.frame(x=x))

# Setup a simple topology to compute a sum
computeSum <- function(x, ...){
sum <- GetHash("sum")
if(is.data.frame(sum)){
x <- sum + (x[1])
}
SetHash("sum", x)
}

# Run the stream
topology <- AddBolt(topology, Bolt(computeSum))
result <- RStorm(topology)
print(GetHash("sum", result))

# Create an alternative dataset
x2 <- seq(2, 100)

# Change the dataset in the existing topology
topology <- ChangeSpout(topology, data.frame(x=x2))

# Run the new dataset
result <- RStorm(topology)
print(GetHash("sum", result))
```

Emit — *Function to emit a* `Tuple` *along the stream. The emitted data* `x` *should be a single row of a* `data.frame`.

Description

Function to emit a `Tuple` along the stream. The emitted data `x` should be a single row of a `data.frame`. Tuples are the main data format passed around in an `RStorm` stream, and each emitted object is checked by the `Emit` function to be of class `Tuple`.

Usage

```
Emit(x, .name = NULL, ...)
```

Arguments

`x`	a `Tuple`. The only arguments that needs to be provided by the user.
`.name`	(internal) the name of the emitter. Used internally.
`...`	Additional arguments.

Value

TRUE. The `Emit` function does not return anything but rather adds the emitted Tuple to the internal list of emitted objects to be used by Spouts listening to the `Spout` or `Bolt` from which the data is emitted. The ... argument always needs to be passed in a call to `Emit()` since it facilitates the internal working of the RStorm.

Author(s)

Maurits Kaptein

See Also

See Also: `Topology`, `AddBolt`, `RStorm`

Examples

```
# This example can only be run within a Stream.
# If run outside the Steam the Emit function will issue an error.
## Not run:
x <- data.frame(var1 = c("test", "test2"), var2 = c(2,5))
Emit(Tuple(x[1,]), ...)

## End(Not run)
```

GetHash — *Function to retrieve objects stored locally during the running of the stream.*

Description

Within bolts in used in a RStorm the `GetHash` and `SetHash` functions can be used to access a local store (or hashmap) during the stream. This corresponds to the ability of tracking parameters over the stream using a hashmap or database system as implemented in production streaming software. The function is overloaded to retrieve the state of the hashmap at the end of a stream from an RStorm result object. See the examples for the two usages.

Usage

```
GetHash(name, object = NULL)
```

Arguments

`name`	a string containing the name of the hashmap that is accessed from within the Stream.
`object`	(optional) the RStorm result object. If used outside of a bolt in a stream the result object needs to be passed in which the end-state of the hashmaps created in the stream are stored.

Value

a dataframe containing whatever was set using the SetHash function.

Author(s)

Maurits Kaptein

See Also

See Also: `SetHash`, `GetHashList`, `GetHashNames`

Examples

```
# Create a topology with a spout:
topology <- Topology(data.frame(x=rnorm(100,0,1)))

# declare a bolt and add it to the topology
computeSum <- function(x, ...){
sum <- GetHash("sum")  # get from local store
if(is.data.frame(sum)){
x <- sum + (x[1])
}
SetHash("sum", x)  # add to local store
}
topology <- AddBolt(topology, Bolt(computeSum))

# run the stream
result <- RStorm(topology)

# access the local store
print(GetHash("sum", result))
```

GetHashList — *Function to retrieve a list of locally stored object resulting from the stream.*

Description

Function retrieves from an RStorm result object (after running a stream) all the items stored in during the stream (using `SetHash`) as a list.

Usage

```
GetHashList(object = NULL)
```

Arguments

`object`	a `RStorm` result object

Value

a list containing all objects stored using `SetHash` during a stream.

Author(s)

Maurits Kaptein

See Also

See Also: `SetHash`, `GetHash`, `GetHashNames`

Examples

```
# Create a topology
topology <- Topology(data.frame(x=rnorm(100,0,1)))

# Create two bolts and add them to the topology
computeSum <- function(x, ...){
sum <- GetHash("sum")
if(is.data.frame(sum)){
```

```
x <- sum + (x[1])
}
SetHash("sum", x)
}
computeSumSquared <- function(x, ...){
sum2 <- GetHash("sum2")
if(is.data.frame(sum2)){
x <- sum2 + (x[1]^2)
}
SetHash("sum2", x)
}
topology <- AddBolt(topology, Bolt(computeSum))
topology <- AddBolt(topology, Bolt(computeSumSquared))

# Run the stream
result <- RStorm(topology)

# Get the names of all the stored objects during the stream
names(GetHashList(result))
```

GetHashNames *Function to retrieve the names of locally stored objects in the stream.*

Description

Function retrieves from an RStorm result object (after running a stream) all the names of all items stored in during the stream (using `SetHash`) as a list.

Usage

```
GetHashNames(object)
```

Arguments

`object`	a `RStorm` result object

Value

a list containing all names stored using `SetHash` during a stream.

Author(s)

Maurits Kaptein

See Also

See Also: `SetHash`, `GetHash`, `GetHashNames`

Examples

```
# Create a topology
topology <- Topology(data.frame(x=rnorm(100,0,1)))

# Create two bolts and add them to the topology
computeSum <- function(x, ...){
sum <- GetHash("sum")
if(is.data.frame(sum)){
x <- sum + (x[1])
}
SetHash("sum", x)
}
computeSumSquared <- function(x, ...){
sum2 <- GetHash("sum2")
if(is.data.frame(sum2)){
x <- sum2 + (x[1]^2)
}
SetHash("sum2", x)
}
topology <- AddBolt(topology, Bolt(computeSum))
topology <- AddBolt(topology, Bolt(computeSumSquared))

# Run the stream
result <- RStorm(topology)

# Get the names of all the stored objects during the stream
print(GetHashNames(result))
```

GetTrack — *Function to retrieve objects stored using the* `SetTrack` *functionality during a stream.*

Description

Within bolts in a RStorm stream the `TrackRow` function can be used to store the state of variables at that point during the stream. The `TrackRow` function will store values incrementally during the stream. Thus, `TrackRow` enables one to store a set of parameter at each event in a Bolt. The current `GetTrack` function allows for inspection of these stored values after running the Stream by passing the RStorm result object.

Usage

```
GetTrack(name, x)
```

Arguments

`name`	a string with the name of the tracked parameter values. Name corresponds to the name used in the call to the `TrackRow` function during the stream.
`x`	a RStorm result object.

Value

a data.frame containing the parameters that are tracked at each iteration of a bolt.

Author(s)

Maurits Kaptein

See Also

See Also: `TrackRow`, `SetHash`, `GetHash`, `GetTrackNames`

Examples

```
# Create a topology with a spout
topology <- Topology(data.frame(x=c(1:10)))

# Add a bolt to the topology
computeSum <- function(x, ...){
sum <- GetHash("sum")
if(is.data.frame(sum)){
x <- sum + (x[1])
}
SetHash("sum", x)
# Track the current state during the stream:
TrackRow("sum", data.frame(x=x))
}
topology <- AddBolt(topology, Bolt(computeSum))

# Run the stream
result <- RStorm(topology)

# Inspect the sums during the stream
GetTrack("sum", result)
```

`GetTrackNames` *Function to retrieve the names of all tracked objects using* `SetTrack`

Description

Within bolts in a RStorm stream the `TrackRow` function can be used to store the state of variables at that point during the stream. The `TrackRow` function will store values incrementally during the stream. Thus, `TrackRow` enables one to store a set of parameter at each event in a Bolt. The current `GetTrackNames` function allows to inspect all the tracked objects of a stream by passing the RStorm result object.

Usage

```
GetTrackNames(x)
```

Arguments

`x`	a RStorm result object.

Value

A list of names of the tracked objects during the stream.

Author(s)

Maurits Kaptein

See Also

See Also: `TrackRow`, `SetHash`, `GetHash`, `GetTrack`

Examples

```
# Create a topology with a spout
topology <- Topology(data.frame(x=c(1:10)))

# Add a bolt to the topology
computeSum <- function(x, ...){
sum <- GetHash("sum")
if(is.data.frame(sum)){
x <- sum + (x[1])
}
SetHash("sum", x)
# Track the current state during the stream:
TrackRow("sum", data.frame(x=x))
}
topology <- AddBolt(topology, Bolt(computeSum))

# Run the stream
result <- RStorm(topology)

# Inspect the sums during the stream
GetTrackNames(result)
```

`RStorm` *Main function to run a stream.*

Description

`RStorm` provides the main functionality of the RStorm package. The `RStorm` function is used to run a stream defined using a Topology. The Topology defines the spout (the data-source for the stream) and the order of processing units (bolts). See example below and in the main package description for examples of the usage of `RStorm`.

More details of the package, examples of streaming algorithms, and examples of the use of RStorm can be found at http://software.mauritskaptein.com/RStorm

Usage

```
RStorm(topology, .verbose = TRUE, .debug = FALSE, .batches = 100, ...)
```

Arguments

`topology`	a topology object specified using `Topology`. The topology contains all the necessary information (the definition of the spouts, the bolts, and the order of processing) for the stream to run in full.
`.verbose`	a logical indicator for verbose output. Default is TRUE.
`.debug`	a logical indicator for debug mode. If in debug mode all objects stored during the running of the stream will persist in memory and can be accessed using standard calls to `ls()`. Default is FALSE.
`.batches`	a number. Determines the size of batches processed by a stream. While RStorm simulates streaming processing, in actuality the rows of the data.frame defined in the spout are iterated through in batches to prevent memory overflow when the spout contains a large number of rows. This argument sets the size of these batches and with it limits the size of memory allocated to emitted data during the stream. Default batch size is 100.
`...`	additional arguments to pass to (e.g.) bolts or plotting functions.

Value

An object of type `RStorm` which is a list containing the following elements:

`data`	a list containing all the hashmaps stored during the stream using `SetHash`. Can be accessed by passing the result object to `GetHash`.
`track`	a list containing all the data.frames stored using the `TrackRow` functionality. Can be accessed by passing the result object of an RStorm to `GetTrack`.
`finalize`	the result of the finalize function. If a finalize function is added to the Topology this field will contain whatever was returned by the finalize function and can be accessed directly using `ShowFinalize`. If no finalize function was added to the topology or the finalize function does not return anything the value of finalize will be false

Additional Info

The is.RStorm function checks whether an object is of Type RStorm and is used internally.

Author(s)

Maurits Kaptein

References

http://software.mauritskaptein.com/RStorm

See Also

See Also: Topology, Bolt, Tuple, Emit, TrackRow, SetHash, GetHash, GetTrack

Examples

```
# Run a simple RStorm. First, create some data:
x <- seq(1, 1000)

# Second, we start defining the topology
topology <- Topology(data.frame(x=x))

# Third, we define a bolt.
# This bolt computes the sum of a number stored
#   in a local Hashmap and the Tuple (x) that is received
computeSum <- function(x, boltID, ...){
sum <- GetHash(paste("sum", boltID))
if(is.data.frame(sum)){
x <- sum + (x[1])
}
SetHash(paste("sum", boltID), x)
Emit(Tuple(x=x), ...)
}

# Add the bolts to the topology.
# Here the first bolt computes the sum of the sequence
#   and the second bolt computes the sum of summed elements
topology <- AddBolt(topology, Bolt(computeSum, listen=0, boltID=1))
topology <- AddBolt(topology, Bolt(computeSum, listen=1, boltID=2))
result <- RStorm(topology)
print(GetHash("sum 1", result))
print(GetHash("sum 2", result))
```

`RStorm.env` *Environment used by RStorm for internal storage of objects.*

Description

Environment used by RStorm for internal storage of objects. These objects facilitate the functioning of the Stream and store the emitted Tuples in between Bolts.

Author(s)

Maurits Kaptein

See Also

See Also: `RStorm`

`sentences` *Sentences of the first section of the paper by Student Introducing the T-Test.*

Description

This dataset gives a number of sentences – the first sentences from the article "The probable error of a mean" by Student.

Usage

```
rivers
```

Format

A data.frame containing two columns, one with an ID (number) for each sentence, and one with the sentences itself.

Source

The probable error of a mean, Biometrica, 1908

References

Student (1908) *The probable error of a mean*. Biometrica, 6, 1, 1–25.

SetHash *Function to store a* `data.frame` *during a stream.*

Description

Within bolts in used in a RStorm the `GetHash` and `SetHash` functions can be used to access a local store (or hashmap) during the stream. This corresponds to the ability of tracking parameters over the stream using a hashmap or database system as implemented in production streaming software.

Usage

```
SetHash(name, data, ...)
```

Arguments

`name`	a string containing the name of the stored object
`data`	a data.frame (or scalar) to be stored
`...`	

Value

If storing the value is successful returns TRUE.

Author(s)

Maurits Kaptein

See Also

See Also: `TrackRow`, `SetHash`, `GetHash`, `GetTrack`

Examples

```
topology <- Topology(data.frame(x=rnorm(100,0,1)))
computeSum <- function(x, ...){
sum <- GetHash("sum")
if(is.data.frame(sum)){
x <- sum + (x[1])
}
SetHash("sum", x)
}
topology <- AddBolt(topology, Bolt(computeSum))
result <- RStorm(topology)
print(GetHash("sum", result))
```

`ShowFinalize` *Function to display the name of the finalize function.*

Description

Utility function to display the finalize function of a RStorm topology object or display the result of a finalize function of an RStorm result object.

Usage

```
ShowFinalize(x)
```

Arguments

`x`	a topology created using `Topology`

Value

prints the finalize function of a Topology object.

Author(s)

Maurits Kaptein

See Also

See Also: `Topology`, `RStorm`, `GetHash`,

Examples

```
# Simple display of the finalize function itself
topology <- Topology(data.frame(x=c(1:10), y=rep(1,10)))

bolt.1 <- function(x, ...){ SetHash("finalize", data.frame(x=99)) }
topology <- AddBolt(topology, Bolt(bolt.1))

comp.av <- function(object, ...){
return( rep(object$finalize$x, 10)) }

topology <- AddFinalize(topology, Bolt(comp.av))
ShowFinalize(topology)

# and in the result object:
result <- RStorm(topology)
ShowFinalize(result)
```

Topology *Function to create a topology*

Description

By passing a spout (dataframe) to this function and storing its return object you can start building a topology for a RStorm stream. See codeRStorm for more detailed examples of the use of Topology. The Topology is the most important concept when defining a RStorm stream.

Usage

```
Topology(spout, name = NULL, .verbose = TRUE)
```

Arguments

`spout`	a data.frame containing multiple rows of data which are to be iterated through in the stream.
`name`	an optional name of this topology.
`.verbose`	an optional boolean to indicate whether you want verbose output or not. Default is TRUE

Value

An object of class `Topology` which is a list containing the following elements:

`spout`	the data.frame passed as a spout
`bolts`	a list of bolts, see `Bolt`
`finailze`	the finalize function to be used for the stream

Additional Info

The `is.Topology` function checks whether an object is of Type Topology and is used internally.

Note

For examples see www.mauritskaptein.com/software/RStorm

Author(s)

Maurits Kaptein

See Also

`Bolt`, `Tuple`, `RStorm`

Examples

```
##############################
# Example of a stream to compare two methods of streaming variance computation:
##############################

# Generate some data
set.seed(10)
t <- 100
x <- rnorm(t,0,1)
# Look at the variance as computed by var():
var(x)

# Start a topology
topology <- Topology(data.frame(x=x))

# Bolt for "Sum of Squares Method" with tracking over time
var.SS <- function(x, ...){
params <- GetHash("params1")
if(!is.data.frame(params)){
params <- list()
params$n <- params$sum <- params$sum2 <- 0
}
n <- params$n + 1
sum <- params$sum + as.numeric(x[1])
sum2 <- params$sum2 + as.numeric(x[1]^2)
if(n>1){
var <- 1/(n*(n-1)) * (n*sum2 - sum^2)
} else {
var <- 0
}
SetHash("params1", data.frame(n=n, sum=sum, sum2=sum2, var=var))
TrackRow("var.SS", data.frame(var=var))
}


## Bolt for "Welford's" Method:

var.Welford <- function(x, ...){
x <- as.numeric(x[1])
params <- GetHash("params2")
if(!is.data.frame(params)){
params <- list()
params$M <- x
params$S <- params$n <- 0
}
n <- params$n + 1
M <- params$M + ( x - params$M) / n
S <- params$S + (x - params$M)*(x-M)

if(n>1){
```

```
var <- S / (n-1)
} else {
var <- 0
}
SetHash("params2", data.frame(n=n, M=M, S=S, var=var))
TrackRow("var.Welford", data.frame(var=var))
}

# Add both topologies to a Stream:
topology <- AddBolt(topology, Bolt(var.SS))
topology <- AddBolt(topology, Bolt(var.Welford))
result <- RStorm(topology)

# Plot the results over the stream
plot(c(1:t), GetTrack("var.Welford", result)$var, type="l")
lines(c(1:t), GetTrack("var.SS", result)$var, col="red")


# Now the same variance calculation,
#    but with a dataset in which the mean is
#    very large compared to the variance:
x2 <- rnorm(t,10^8,1)
topology2 <- Topology(data.frame(x=x2))
topology2 <- AddBolt(topology2, Bolt(var.SS))
topology2 <- AddBolt(topology2, Bolt(var.Welford))
result2 <- RStorm(topology2)

# This time the standard SS methods screws up (mind the different y scale):
# (And mind the fact that the SS method gives NEGATIVE variance)
plot(c(1:t), GetTrack("var.Welford", result2)$var, type="l", ylim=c(-10, 11))
lines(c(1:t), GetTrack("var.SS", result2)$var, col="red")
```

`TrackRow` *Function to store the value of some object in the stream over time.*

Description

Within bolts in a RStorm stream the `TrackRow` function can be used to store the state of variables at that point during the stream. The `TrackRow` function will store values incrementally during the stream. Thus, `TrackRow` enables one to store a set of parameter at each event in a Bolt.

Usage

```
TrackRow(name, row)
```

Arguments

`name`	a string with the name of the object that is stored.
`row`	a single row data.frame containing the parameters that are supposed to be tracked over time. See example.

Value

TRUE if the row is correctly stored.

Author(s)

Maurits Kaptein

See Also

`Topology`, `GetTrack`, `GetTrackNames`

Examples

```
# Create a topology with a simple spout
topology <- Topology(data.frame(x=c(1:10)))

# Define the bolt and add it
computeSum <- function(x, ...){
sum <- GetHash("sum")
if(is.data.frame(sum)){
x <- sum + (x[1])
}
SetHash("sum", x)
TrackRow("sum", data.frame(x=x))
}
topology <- AddBolt(topology, Bolt(computeSum))

# Run the stream
result <- RStorm(topology)

# Insepct the result over the timepoints in the stream
GetTrack("sum", result)
```

`Tuple` *Function to create an object of type* `Tuple` *to emit down a stream.*

Description

A `Tuple` is the main data object that is passed around in a stream. The spout emits Tuples to the different Bolts, and Bolts can Emit Tuples to one another depending on the Topology. The `Emit` function checks whether indeed objects of type `Tuple` are emitted. A `Tuple` object is a single row data.frame.

Usage

```
Tuple(x, ...)
```

Arguments

`x`	a single row data.frame
`...`	any other argument

Value

an object of type `Tuple` to be used by an RStorm emitter function

Additional Info

The `is.Tuple` function checks whether an object is of Type Tuple and is used internally.

Author(s)

Maurits Kaptein

See Also

`Emit`, `Topology`, `RStorm`

Examples

```
# Example of a simple object emitted down a stream
spout <- data.frame(x=seq(1,4))
topology <- Topology(spout)

# The Emit function will check if the emitted object is indeed a Tuple
bolt.1 <- function(x, ...){
Emit(Tuple(x), ...)
}

bolt.2 <- function(x, ...){
x <- as.numeric(x[1])
print(x^2)
}

topology <- AddBolt(topology, Bolt(bolt.1, listen=0))
topology <- AddBolt(topology, Bolt(bolt.1, listen=1))
topology <- AddBolt(topology, Bolt(bolt.2, listen=2))

result <- RStorm(topology)
#plot(topology)
```

Index