Package ‘ParallelLogger’

August 22, 2023

Type Package

Title Support for Parallel Computation, Logging, and Function Automation

Version 3.3.0

Date 2023-08-22

Maintainer Martijn Schuemie <schuemie@ohdsi.org>

Description Support for parallel computation with progress bar, and option to stop or proceed on errors. Also provides logging to console and disk, and the logging persists in the parallel threads. Additional functions support function call automation with delayed execution (e.g. for executing functions in parallel).

License Apache License 2.0

VignetteBuilder knitr

Depends R (>= 4.0.0)

Imports snow, xml2, jsonlite, methods, utils

Suggests sendmailR, testthat, shiny, DT, knitr, rmarkdown, tibble

URL https://ohdsi.github.io/ParallelLogger/, https://github.com/OHDSI/ParallelLogger

BugReports https://github.com/OHDSI/ParallelLogger/issues

NeedsCompilation no

RoxygenNote 7.2.3

Encoding UTF-8

Author Martijn Schuemie [aut, cre],
Marc Suchard [aut],
Observational Health Data Science and Informatics [cph]

Repository CRAN

Date/Publication 2023-08-22 14:50:02 UTC
topics documented:

- addDefaultConsoleLogger ........................................... 3
- addDefaultEmailLogger ............................................. 3
- addDefaultErrorReportLogger ...................................... 5
- addDefaultFileLogger ............................................... 5
- clearLoggers ......................................................... 6
- clusterApply ......................................................... 6
- clusterRequire ....................................................... 7
- convertJsonToSettings ............................................. 7
- convertSettingsToJson ............................................. 8
- createArgFunction .................................................. 8
- createConsoleAppender ............................................ 9
- createEmailAppender ............................................... 10
- createFileAppender ............................................... 11
- createLogger ......................................................... 12
- excludeFromList .................................................... 13
- getLoggers .......................................................... 13
- launchLogViewer .................................................... 14
- layoutEmail .......................................................... 14
- layoutErrorReport .................................................. 15
- layoutParallel ....................................................... 15
- layoutSimple ......................................................... 16
- layoutStackTrace .................................................... 16
- layoutTimestamp ..................................................... 17
- loadSettingsFromJson ............................................. 17
- logDebug ............................................................. 18
- logError .............................................................. 18
- logFatal .............................................................. 19
- logInfo ............................................................... 19
- logTrace .............................................................. 20
- logWarn .............................................................. 21
- makeCluster ........................................................ 21
- matchInList ........................................................ 22
- registerLogger ....................................................... 23
- saveSettingsToJson ................................................. 24
- selectFromList ...................................................... 24
- stopCluster ........................................................ 25
- unregisterLogger ................................................... 26

Index ................................................................. 27
addDefaultConsoleLogger

Add the default console logger

Description
Add the default console logger

Usage
addDefaultConsoleLogger(name = "DEFAULT_CONSOLE_LOGGER")

Arguments
name A name for the logger.

Details
Creates a logger that writes to the console using the "INFO" threshold and the layoutSimple layout.

Examples
logger <- addDefaultConsoleLogger()
logTrace("This event is below the threshold (INFO)")
logInfo("Hello world")
unregisterLogger(logger)

addDefaultEmailLogger Add the default e-mail logger

Description
Add the default e-mail logger

Usage
addDefaultEmailLogger(
    mailSettings,
    label = Sys.info()["nodename"],
    name = "DEFAULT_EMAIL_LOGGER",
    test = FALSE
)
**addDefaultEmailLogger**

**Arguments**

- `mailSettings`: Arguments to be passed to the sendmail function in the sendmailR package (except subject and msg).
- `label`: A label to be used in the e-mail subject to identify a run. By default the name of the computer is used.
- `name`: A name for the logger.
- `test`: If TRUE, a message will be displayed on the console instead of sending an e-mail.

**Details**

Creates a logger that writes to e-mail using the "FATAL" threshold and the `layoutEmail` layout. This function uses the sendmailR package. Please make sure your e-mail settings are correct by using the sendmailR package before using those settings here. ParallelLogger will not display any messages if something goes wrong when sending the e-mail.

**Using GMail**

To use a GMail account, make sure to enable 2-step verification on your Google account (see 'Security'). Click on 2-Step Verification, and scroll down to 'App passwords'. Here, you can create an app-specific password to be used with ParallelLogger. You can set `host.name = "smtp.gmail.com:587"`, and be sure to use `engine = "curl"`.

**Examples**

```r
mailSettings <- list(
  from = "someone@gmail.com",
  to = "someone_else@gmail.com",
  engine = "curl",
  engineopts = list(
    username = "someone@gmail.com",
    password = "Secret!"
  ),
  control = list(
    host.name = "smtp.gmail.com:587"
  )
)

# Setting test to TRUE in this example so we don't really send an e-mail:
addDefaultEmailLogger(mailSettings, "My R session", test = TRUE)
logFatal("Something bad")

unregisterLogger("DEFAULT_EMAIL_LOGGER")
```
addDefaultErrorReportLogger

Add the default error report logger

Description
Add the default error report logger

Usage
addDefaultErrorReportLogger(
    fileName = file.path(getwd(), "errorReportR.txt"),
    name = "DEFAULT_ERRORREPORT_LOGGER"
)

Arguments
- fileName: The name of the file to write to.
- name: A name for the logger.

Details
Creates a logger that writes to a file using the "FATAL" threshold and the layoutErrorReport layout. The file will be overwritten if it is older than 60 seconds. The user will be notified that the error report has been created, and where to find it.

addDefaultFileLogger
Add the default file logger

Description
Add the default file logger

Usage
addDefaultFileLogger(fileName, name = "DEFAULT_FILE_LOGGER")

Arguments
- fileName: The name of the file to write to.
- name: A name for the logger.

Details
Creates a logger that writes to a file using the "TRACE" threshold and the layoutParallel layout. The output can be viewed with the built-in log viewer that can be started using launchLogViewer.
clearLoggers  
Remove all registered loggers

Description
Remove all registered loggers

Usage
clearLoggers()

clusterApply  
Apply a function to a list using the cluster

Description
Apply a function to a list using the cluster

Usage
clusterApply(cluster, x, fun, ..., stopOnError = FALSE, progressBar = TRUE)

Arguments
- cluster: The cluster of threads to run the function.
- x: The list on which the function will be applied.
- fun: The function to apply. Note that the context in which the function is specified matters (see details).
- ...: Additional parameters for the function.
- stopOnError: Stop when one of the threads reports an error? If FALSE, all errors will be reported at the end.
- progressBar: Show a progress bar?

Details
The function will be executed on each element of x in the threads of the cluster. If there are more elements than threads, the elements will be queued. The progress bar will show the number of elements that have been completed. It can sometimes be important to realize that the context in which a function is created is also transmitted to the worker node. If a function is defined inside another function, and that outer function is called with a large argument, that argument will be transmitted to the worker node each time the function is executed. It can therefore make sense to define the function to be called at the package level rather than inside a function, to save overhead.
Value

A list with the result of the function on each item in x.

Examples

```r
fun <- function(x) {
  return (x^2)
}

cluster <- makeCluster(numberOfThreads = 3)
clusterApply(cluster, 1:10, fun)
stopCluster(cluster)
```

clusterRequire

Require a package in the cluster

Description

Calls the `require` function in each node of the cluster.

Usage

`clusterRequire(cluster, package)`

Arguments

- `cluster` The cluster object.
- `package` The name of the package to load in all nodes.

convertJsonToSettings

Converts a JSON string to a settings object

Description

Converts a JSON string to a settings object

Usage

`convertJsonToSettings(json)`

Arguments

- `json` A JSON string.
createArgFunction

Details

Converts a JSON string generated using the convertSettingsToJson function to a settings object, restoring object classes and attributes.

Value

An R object as specified by the JSON.

convertSettingsToJson

Convert a settings object to a JSON string

Description

Convert a settings object to a JSON string

Usage

convertSettingsToJson(object)

Arguments

object R object to be converted.

Details

Convert a settings object to a JSON string, using pretty formatting and preserving object classes and attributes.

Value

A JSON string representing the R object.

createArgFunction

Create an argument function

Description

Create an argument function
createConsoleAppender

Usage

createConsoleAppender(
  layout = layoutSimple
)

Description

Create console appender

Usage

createConsoleAppender(layout = layoutSimple)
**createEmailAppender**

**Arguments**

- **layout**: The layout to be used by the appender.

**Details**

Creates an appender that will write to the console.

**Examples**

```r
appender <- createConsoleAppender(layout = layoutTimestamp)

logger <- createLogger(name = "SIMPLE",
                        threshold = "INFO",
                        appenders = list(appender))

registerLogger(logger)
logTrace("This event is below the threshold (INFO)"
logInfo("Hello world")
unregisterLogger("SIMPLE")
```

---

**createEmailAppender**  
*Create e-mail appender*

**Description**

Create e-mail appender

**Usage**

```r
createEmailAppender(
  layout = layoutEmail,
  mailSettings,
  label = Sys.info()[['nodename']],
  test = FALSE
)
```

**Arguments**

- **layout**: The layout to be used by the appender.
- **mailSettings**: Arguments to be passed to the sendmail function in the sendmailR package (except subject and msg).
- **label**: A label to be used in the e-mail subject to identify a run. By default the name of the computer is used.
- **test**: If TRUE, a message will be displayed on the console instead of sending an e-mail.
createFileAppender

Details

Creates an appender that will send log events to an e-mail address using the sendmailR package. Please make sure your settings are correct by using the sendmailR package before using those settings here. ParallelLogger will not display any messages if something goes wrong when sending the e-mail.

Using GMail

To use a GMail account, make sure to enable 2-step verification on your Google account (see 'Security'). Click on 2-Step Verification, and scroll down to 'App passwords'. Here, you can create an app-specific password to be used with ParallelLogger. You can set host.name = "smtp.gmail.com:587", and be sure to use engine = "curl".

Examples

```r
mailSettings <- list(
  from = "someone@gmail.com",
  to = "someone_else@gmail.com",
  engine = "curl",
  engineopts = list(
    username = "someone@gmail.com",
    password = "Secret!"
  ),
  control = list(
    host.name = "smtp.gmail.com:587"
  )
)
# Setting test to TRUE in this example so we don't really send an e-mail:
appender <- createEmailAppender(
  layout = layoutEmail,
  mailSettings = mailSettings,
  label = "My R session",
  test = TRUE
)

logger <- createLogger(name = "EMAIL", threshold = "FATAL", appenders = list(appender))
registerLogger(logger)

logFatal("Something bad")

unregisterLogger("EMAIL")
```

createFileAppender Create file appender

Description

Create file appender
createLogger

Usage

createFileAppender(
    layout = layoutParallel,
    fileName,
    overwrite = FALSE,
    expirationTime = 60
)

Arguments

layout The layout to be used by the appender.
fileName The name of the file to write to.
overwrite Overwrite the file if it is older than the expiration time?
expirationTime Expiration time in seconds

Details

Creates an appender that will write to a file.

createLogger Create a logger

Description

Create a logger

Usage

createLogger(
    name = "SIMPLE",
    threshold = "INFO",
    appenders = list(createConsoleAppender())
)

Arguments

name A name for the logger.
threshold The threshold to be used for reporting.
appenders A list of one or more appenders as created for example using the createConsoleAppender or createFileAppender function.

Details

Creates a logger that will log messages to its appenders. The logger will only log messages at a level equal to or higher than its threshold. For example, if the threshold is "INFO" then messages marked "INFO" will be logged, but messages marked "TRACE" will not. The order of levels is "TRACE", "DEBUG", "INFO", "WARN", "ERROR", "and FATAL".
Value

An object of type Logger, to be used with the registerLogger function.

Examples

appender <- createConsoleAppender(layout = layoutTimestamp)

logger <- createLogger(name = "SIMPLE",
                        threshold = "INFO",
                        appenders = list(appender))

registerLogger(logger)
logTrace("This event is below the threshold (INFO)")
logInfo("Hello world")
unregisterLogger("SIMPLE")

excludeFromList

Exclude variables from a list of objects of the same type

Description

Exclude variables from a list of objects of the same type

Usage

excludeFromList(x, exclude)

Arguments

x A list of objects of the same type.
exclude A character vector of names of variables to exclude.

getLoggers

Get all registered loggers

Description

Get all registered loggers

Usage

getLoggers()

Value

Returns all registered loggers.
launchLogViewer  

*Launch the log viewer Shiny app*

**Description**
Launch the log viewer Shiny app

**Usage**

```r
launchLogViewer(logFileName)
```

**Arguments**

- `logFileName`  
  Name of the log file to view.

**Details**
Launches a Shiny app that allows the user to view a log file created using the default file logger. Use `addDefaultFileLogger` to start the default file logger.

**Examples**

```r
# Create a log file:
logFile <- file.path(tempdir(), "log.txt")
addDefaultFileLogger(logFile)
logInfo("Hello world")

# Launch the log file viewer (only if in interactive mode):
if (interactive()) {
  launchLogViewer(logFile)
}

# Delete the log file:
unlink(logFile)
```

layoutEmail  

*Logging layout for e-mail*

**Description**
A layout function to be used with an e-mail appender. This layout creates a short summary e-mail message on the event, including stack trace.

**Usage**

```r
layoutEmail(level, message)
```
**layoutErrorReport**

**Arguments**

- **level**
  - The level of the message (e.g. "INFO")
- **message**
  - The message to layout.

---

**Description**

A layout function to be used with an appender. This layout creates a more elaborate error message, for sharing with the developer. If an error occurs in the main thread a summary of the system info will be included.

**Usage**

```
layoutErrorReport(level, message)
```

**Arguments**

- **level**
  - The level of the message (e.g. "INFO")
- **message**
  - The message to layout.

---

**layoutParallel**

**Logging layout for parallel computing**

---

**Description**

A layout function to be used with an appender. This layout adds the time, thread, level, package name, and function name to the message.

**Usage**

```
layoutParallel(level, message)
```

**Arguments**

- **level**
  - The level of the message (e.g. "INFO")
- **message**
  - The message to layout.
layoutSimple  Simple logging layout

Description

A layout function to be used with an appender. This layout simply includes the message itself.

Usage

layoutSimple(level, message)

Arguments

level  The level of the message (e.g. "INFO")
message  The message to layout.

layoutStackTrace  Logging layout with stack trace

Description

A layout function to be used with an appender. This layout adds the stack trace to the message.

Usage

layoutStackTrace(level, message)

Arguments

level  The level of the message (e.g. "INFO")
message  The message to layout.
**layoutTimestamp**

*Logging layout with timestamp*

**Description**

A layout function to be used with an appender. This layout adds the time to the message.

**Usage**

```
layoutTimestamp(level, message)
```

**Arguments**

- **level**
  - The level of the message (e.g. "INFO")
- **message**
  - The message to layout.

**Examples**

```
appender <- createConsoleAppender(layout = layoutTimestamp)
logger <- createLogger(name = "SIMPLE",
                        threshold = "INFO",
                        appenders = list(appender))
registerLogger(logger)
logTrace("This event is below the threshold (INFO)")
logInfo("Hello world")
unregisterLogger("SIMPLE")
```

**loadSettingsFromJson**

*Load a settings object from a JSON file*

**Description**

Load a settings object from a JSON file

**Usage**

```
loadSettingsFromJson(fileName)
```

**Arguments**

- **fileName**
  - Name of the JSON file to load.

**Details**

Load a settings object from a JSON file, restoring object classes and attributes.
Value
An R object as specified by the JSON.

---

`logDebug` *Log a message at the DEBUG level*

**Description**
Log a message at the DEBUG level

**Usage**
`logDebug(...)`

**Arguments**

```
...
```
Zero or more objects which can be coerced to character (and which are pasted together with no separator).

**Details**
Log a message at the specified level. The message will be sent to all the registered loggers.

---

`logError` *Log a message at the ERROR level*

**Description**
Log a message at the ERROR level

**Usage**
`logError(...)`

**Arguments**

```
...
```
Zero or more objects which can be coerced to character (and which are pasted together with no separator).

**Details**
Log a message at the specified level. The message will be sent to all the registered loggers.
**logFatal**

*Log a message at the FATAL level*

**Description**

Log a message at the FATAL level

**Usage**

`logFatal(...)`

**Arguments**

`...`  
Zero or more objects which can be coerced to character (and which are pasted together with no separator).

**Details**

Log a message at the specified level. The message will be sent to all the registered loggers. This function is be automatically called when an error occurs, and should not be called directly. Use `stop()` instead.

---

**logInfo**

*Log a message at the INFO level*

**Description**

Log a message at the INFO level

**Usage**

`logInfo(...)`

**Arguments**

`...`  
Zero or more objects which can be coerced to character (and which are pasted together with no separator).

**Details**

Log a message at the specified level. The message will be sent to all the registered loggers. This is equivalent to calling R’s native `message()` function.
Examples

appender <- createConsoleAppender(layout = layoutTimestamp)

logger <- createLogger(name = "SIMPLE",
threshold = "INFO",
appenders = list(appender))

registerLogger(logger)
logTrace("This event is below the threshold (INFO)"
logInfo("Hello world")
unregisterLogger("SIMPLE")

logTrace  \hspace{1cm} Log a message at the TRACE level

Description

Log a message at the TRACE level

Usage

logTrace(...)  

Arguments

...  \hspace{1cm} Zero or more objects which can be coerced to character (and which are pasted together with no separator).

Details

Log a message at the specified level. The message will be sent to all the registered loggers.

Examples

appender <- createConsoleAppender(layout = layoutTimestamp)

logger <- createLogger(name = "SIMPLE",
threshold = "INFO",
appenders = list(appender))

registerLogger(logger)
logTrace("This event is below the threshold (INFO)"
logInfo("Hello world")
unregisterLogger("SIMPLE")
**logWarn**  
*Log a message at the WARN level*

**Description**  
Log a message at the WARN level

**Usage**  
`logWarn(...)`

**Arguments**  
...  
Zero or more objects which can be coerced to character (and which are pasted together with no separator).

**Details**  
Log a message at the specified level. The message will be sent to all the registered loggers. This function is automatically called when a warning is thrown, and should not be called directly. Use `warning()` instead.

**makeCluster**  
*Create a cluster of nodes for parallel computation*

**Description**  
Create a cluster of nodes for parallel computation

**Usage**  
`makeCluster(  
  numberOfThreads,  
  singleThreadToMain = TRUE,  
  setAndromedaTempFolder = TRUE  
)`

**Arguments**  
`numberOfThreads`  
Number of parallel threads.

`singleThreadToMain`  
If `numberOfThreads` is 1, should we fall back to running the process in the main thread?

`setAndromedaTempFolder`  
When TRUE, the andromedaTempFolder option will be copied to each thread.
Value
An object representing the cluster.

Examples
fun <- function(x) {
  return (x^2)
}
cluster <- makeCluster(numberOfThreads = 3)
clusterApply(cluster, 1:10, fun)
stopCluster(cluster)

matchInList

Description
In a list of object of the same type, find those that match the input

Usage
matchInList(x, toMatch)

Arguments

x
A list of objects of the same type.

toMatch
The object to match.

Details
Typically, toMatch will contain a subset of the variables that are in the objects in the list. Any object matching all variables in toMatch will be included in the result.

Value
A list of objects that match the toMatch object.

Examples
x <- list(
  a = list(name = "John", age = 25, gender = "M"),
  b = list(name = "Mary", age = 24, gender = "F")
)
matchInList(x, list(name = "Mary"))

# $a
$\texttt{registerLogger}$

```
# $a$\texttt{name}
# [1] "John"
# 
# $a$\texttt{age}
# [1] 25
#
#
# $b$
# $b$\texttt{name}
# [1] "Mary"
# 
# $b$\texttt{age}
# [1] 24
```

---

**registerLogger**  
*Register a logger*

**Description**

Register a logger

**Usage**

```
registerLogger(logger)
```

**Arguments**

- **logger**  
  An object of type `Logger` as created using the `createLogger` function.

**Details**

Registers a logger as created using the `createLogger` function to the logging system.

**Examples**

```
appender <- createConsoleAppender(layout = layoutTimestamp)

logger <- createLogger(name = "SIMPLE",  
                        threshold = "INFO",  
                        appenders = list(appender))

registerLogger(logger)
logTrace("This event is below the threshold (INFO)")
logInfo("Hello world")
unregisterLogger("SIMPLE")
```
### saveSettingsToJson

**Description**

Save a settings object as JSON file

**Usage**

```r
saveSettingsToJson(object, fileName)
```

**Arguments**

- `object` R object to be saved.
- `fileName` File name where the object should be saved.

**Details**

Save a setting object as a JSON file, using pretty formatting and preserving object classes and attributes.

### selectFromList

**Description**

Select variables from a list of objects of the same type

**Usage**

```r
selectFromList(x, select)
```

**Arguments**

- `x` A list of objects of the same type.
- `select` A character vector of names of variables to select.
Examples

```r
x <- list(
  a = list(name = "John", age = 25, gender = "M"),
  b = list(name = "Mary", age = 24, gender = "F")
)
selectFromList(x, c("name", "age"))

# $a
# $a$name
# [1] "John"
#
# $a$age
# [1] 25
#
#
# $b
# $b$name
# [1] "Mary"
#
# $b$age
# [1] 24
```

Description

Stop the cluster

Usage

```r
stopCluster(cluster)
```

Arguments

- `cluster` The cluster to stop

Examples

```r
fun <- function(x) {
  return (x^2)
}

cluster <- makeCluster(numberOfThreads = 3)
clusterApply(cluster, 1:10, fun)
stopCluster(cluster)
```
**unregisterLogger**  

*Unregister a logger*

**Description**

Unregister a logger

**Usage**

`unregisterLogger(x, silent = FALSE)`

**Arguments**

- `x`  
  Can either be an integer (e.g. 2 to remove the second logger), the name of the logger, or the logger object itself.

- `silent`  
  If TRUE, no warning will be issued if the logger is not found.

**Details**

Unregisters a logger from the logging system.

**Value**

Returns TRUE if the logger was removed.

**Examples**

```r
appender <- createConsoleAppender(layout = layoutTimestamp)

logger <- createLogger(name = "SIMPLE",
                       threshold = "INFO",
                       appenders = list(appender))

registerLogger(logger)
logTrace("This event is below the threshold (INFO)")
logInfo("Hello world")
unregisterLogger("SIMPLE")
```