Package 'mirai'

January 17, 2023

Type Package

Title Minimalist Async Evaluation Framework for R

Version 0.7.2

Description Lightweight parallel code execution, local or distributed across the network. Designed for simplicity, a 'mirai' evaluates an arbitrary expression asynchronously, resolving automatically upon completion. Built on 'nanonext' and 'NNG' (Nanomsg Next Gen), uses scalability protocols not subject to R connection limits and transports faster than TCP/IP where applicable.

License GPL (>= 3)

BugReports https://github.com/shikokuchuo/mirai/issues

URL https://shikokuchuo.net/mirai/, https://github.com/shikokuchuo/mirai/

Encoding UTF-8

Depends R (>= 2.12)

Imports nanonext (>= 0.7.0)

RoxygenNote 7.2.3

NeedsCompilation no

Author Charlie Gao [aut, cre] (<https://orcid.org/0000-0002-0750-061X>), Hibiki AI Limited [cph]

Maintainer Charlie Gao <charlie.gao@shikokuchuo.net>

Repository CRAN

Date/Publication 2023-01-17 15:20:02 UTC

R topics documented:

`mirai-package` *mirai: Minimalist Async Evaluation Framework for R*

Description

Lightweight parallel code execution, local or distributed across the network. Designed for simplicity, a 'mirai' evaluates an arbitrary expression asynchronously, resolving automatically upon completion. Built on 'nanonext' and 'NNG' (Nanomsg Next Gen), uses scalability protocols not subject to R connection limits and transports faster than TCP/IP where applicable.

Notes

For local mirai processes, the default transport for intra-process communications is platform-dependent: abstract sockets on Linux, Unix domain sockets on MacOS, Solaris and other POSIX platforms, and named pipes on Windows.

This may be overriden if required by specifying a custom client URL in the `daemons` interface, and starting server processes manually with `server` on the same machine.

Links

mirai website: https://shikokuchuo.net/mirai/
mirai on CRAN: https://cran.r-project.org/package=mirai

nanonext website: https://shikokuchuo.net/nanonext/
nanonext on CRAN: https://cran.r-project.org/package=nanonext

NNG website: https://nng.nanomsg.org/

Author(s)

Charlie Gao <charlie.gao@shikokuchuo.net> (ORCID)

`call_mirai` *mirai (Call Value)*

Description

Call the value of a mirai, waiting for the the asynchronous operation to resolve if it is still in progress.

Usage

```
call_mirai(aio)
```

Arguments

`aio`	a 'mirai' object.

Details

This function will wait for the async operation to complete if still in progress (blocking).

A blocking call can be sent a user interrupt with e.g. ctrl+c. If the ongoing execution in the mirai is interruptible, it will resolve into an object of class 'miraiInterrupt' and 'errorValue'. `is_mirai_interrupt` may be used to handle such cases.

If an error occurs in evaluation, the error message is returned as a character string of class 'miraiError' and 'errorValue'. `is_mirai_error` may be used to test for this.

`is_error_value` tests for all error conditions including mirai errors, interrupts, and timeouts.

The mirai updates itself in place, so to access the value of a mirai `x` directly, use `call_mirai(x)$data`.

Value

The passed mirai (invisibly). The retrieved value is stored at `$data`.

Alternatively

The value of a mirai may be accessed at any time at `$data`, and if yet to resolve, an 'unresolved' logical NA will be returned instead.

Using `unresolved` on a mirai returns TRUE only if a mirai has yet to resolve and FALSE otherwise. This is suitable for use in control flow statements such as `while` or `if`.

Examples

```
if (interactive()) {
# Only run examples in interactive R sessions

m <- mirai(x + y + 1, x = 2, y = 3)
m
m$data
Sys.sleep(0.2)
```

```
m$data

df1 <- data.frame(a = 1, b = 2)
df2 <- data.frame(a = 3, b = 1)
m <- mirai(as.matrix(rbind(df1, df2)), .args = list(df1, df2), .timeout = 1000)
call_mirai(m)$data

m <- mirai({
  res <- rnorm(n)
  res / rev(res)
}, n = 1e6)
while (unresolved(m)) {
  cat("unresolved\n")
  Sys.sleep(0.1)
}
str(m$data)

file <- tempfile()
cat("r <- rnorm(n)", file = file)
n <- 10L
m <- mirai({source(file, local = TRUE); r}, .args = list(file, n))
call_mirai(m)[["data"]]
unlink(file)

}
```

---

daemons *daemons (Persistent Server Processes)*

Description

Set or view the number of 'daemons' or persistent server processes receiving `mirai` requests. These are, by default, automatically created on the local machine. Alternatively, a client URL may be set to receive connections from remote servers started with `server`, for distributing tasks across the network.

Usage

```
daemons(n, .url)
```

Arguments

`n`	integer number of daemons to set | 'view' to view the current number of daemons.
`.url`	(optional) for distributing tasks across the network: character client URL and port accepting incoming connections e.g. 'tcp://192.168.0.2:5555' at which server processes started using `server` should connect to. To listen to port 5555 (for example) on all interfaces on the host, specify one of 'tcp://:5555', 'tcp://*:5555' or 'tcp://0.0.0.0:5555'.

Details

Set 'n' to 0 to reset all daemon connections. {mirai} will revert to the default behaviour of creating a new background process for each request.

Specifying '.url' without 'n' assumes a value for 'n' of 1. After setting '.url', further calls specifying 'n' can be used to update the number of connected daemons (this is not strictly necessary as daemons are detected automatically, but will ensure that the correct number of shutdown signals are sent when the session is ended).

Setting a new '.url' value will attempt to shutdown existing daemons connected at the existing address before opening a connection at the new address.

Value

Depending on 'n' specified:

- integer: integer change in number of daemons (created or destroyed).
- 'view': integer number of currently set daemons.

Calling `daemons()` without any arguments returns the 'nanoSocket' for connecting to the daemons, or NULL if it is yet to be created.

About

Daemons provide a potentially more efficient solution for asynchronous operations as new processes no longer need to be created on an ad hoc basis.

Specifying '.url' allows tasks to be distributed across the network. The network topology is such that server daemons (started with `server`) dial into the client, which listens at the '.url' address. In this way, network resources may be added or removed at any time. The client automatically distributes tasks to all available servers.

The current implementation is low-level and ensures tasks are evenly-distributed amongst daemons without actively managing a task queue. This approach provides a robust and resource-light solution, particularly well-suited to working with similar-length tasks, or where the number of concurrent tasks typically does not exceed the number of available daemons.

Examples

```
if (interactive()) {
# Only run examples in interactive R sessions

# Create 2 daemons
daemons(2)
# View the number of active daemons
daemons("view")
# Reset to zero
daemons(0)

}
```

`eval_mirai` *mirai (Evaluate Async)*

Description

Evaluate an expression asynchronously in a new background R process or persistent daemon (local or remote). This function will return immediately with a 'mirai', which will resolve to the evaluated result once complete.

Usage

```
eval_mirai(.expr, ..., .args = list(), .timeout = NULL)

mirai(.expr, ..., .args = list(), .timeout = NULL)
```

Arguments

`.expr`	an expression to evaluate asynchronously. This may be of arbitrary length, wrapped in { } if necessary.
`...`	(optional) named arguments specifying objects referenced in '.expr'.
`.args`	(optional) list supplying objects referenced in '.expr' (used in addition to or instead of named arguments specified as '...').
`.timeout`	(optional) integer value in milliseconds or NULL for no timeout. A mirai will resolve to an 'errorValue' 5 (timed out) if evaluation exceeds this limit.

Details

This function will return a 'mirai' object immediately.

The value of a mirai may be accessed at any time at `$data`, and if yet to resolve, an 'unresolved' logical NA will be returned instead.

`unresolved` may be used on a mirai, returning TRUE if a 'mirai' has yet to resolve and FALSE otherwise. This is suitable for use in control flow statements such as `while` or `if`.

Alternatively, to call (and wait for) the result, use `call_mirai` on the returned mirai. This will block until the result is returned (although interruptible with e.g. ctrl+c).

The expression '.expr' will be evaluated in a separate R process in a clean environment consisting only of the named objects passed as '...' and/or the list supplied to '.args'.

If an error occurs in evaluation, the error message is returned as a character string of class 'miraiError' and 'errorValue'. `is_mirai_error` may be used to test for this.

`is_error_value` tests for all error conditions including 'mirai' errors, interrupts, and timeouts.

`mirai` is an alias for `eval_mirai`.

Value

A 'mirai' object.

Examples

```
if (interactive()) {
# Only run examples in interactive R sessions

m <- mirai(x + y + 1, x = 2, y = 3)
m
m$data
Sys.sleep(0.2)
m$data

df1 <- data.frame(a = 1, b = 2)
df2 <- data.frame(a = 3, b = 1)
m <- mirai(as.matrix(rbind(df1, df2)), .args = list(df1, df2), .timeout = 1000)
call_mirai(m)$data

m <- mirai({
  res <- rnorm(n)
  res / rev(res)
}, n = 1e6)
while (unresolved(m)) {
  cat("unresolved\n")
  Sys.sleep(0.1)
}
str(m$data)

file <- tempfile()
cat("r <- rnorm(n)", file = file)
n <- 10L
m <- mirai({source(file, local = TRUE); r}, .args = list(file, n))
call_mirai(m)[["data"]]
unlink(file)

}
```

---

is_error_value *Is Error Value*

Description

Is the object an error value, such as a mirai timeout, a 'miraiError' from failed execution within a mirai or a 'miraiInterrupt' resulting from the user interrupt of an ongoing mirai evaluation.

Usage

```
is_error_value(x)
```

Arguments

x	an object.

Value

Logical TRUE if 'x' is of class 'errorValue', FALSE otherwise.

Examples

```
is_error_value(1L)
```

`is_mirai` *Is mirai*

Description

Is the object a 'mirai'.

Usage

```
is_mirai(x)
```

Arguments

`x`	an object.

Value

Logical TRUE if 'x' is of class 'mirai', FALSE otherwise.

Examples

```
if (interactive()) {
# Only run examples in interactive R sessions

m <- mirai(as.matrix(df), df = data.frame())
is_mirai(m)
is_mirai(df)

}
```

is_mirai_error *Is mirai Error*

Description

Is the object a 'miraiError'. When execution in a mirai process fails, the error message is returned as a character string of class 'miraiError' and 'errorValue'. To test for all error conditions, including timeouts etc., `is_error_value` should be used instead.

Usage

```
is_mirai_error(x)
```

Arguments

x	an object.

Value

Logical TRUE if 'x' is of class 'miraiError', FALSE otherwise.

Examples

```
if (interactive()) {
# Only run examples in interactive R sessions

m <- mirai(stop())
call_mirai(m)
is_mirai_error(m$data)

}
```

is_mirai_interrupt *Is mirai Interrupt*

Description

Is the object a 'miraiInterrupt'. When a mirai is sent a user interrupt, e.g. by ctrl+c during an ongoing `call_mirai`, the mirai will resolve to an empty character string classed as 'miraiInterrupt' and 'errorValue'. To test for all error conditions, including timeouts etc., `is_error_value` should be used instead.

Usage

```
is_mirai_interrupt(x)
```

Arguments

x	an object.

Value

Logical TRUE if 'x' is of class 'miraiInterrupt', FALSE otherwise.

Examples

```
if (interactive()) {
# Only run examples in interactive R sessions

m <- mirai(stop())
call_mirai(m)
is_mirai_interrupt(m$data)

}
```

---

server *mirai Server (Async Executor [Daemon])*

Description

Implements a [persistent] executor/server for the remote process. Awaits data, evaluates an expression in an environment containing the supplied data, and returns the result to the caller/client.

Usage

```
server(.url, daemon = TRUE)
```

Arguments

.url	the client URL and port to connect to as a character string e.g. 'tcp://192.168.0.2:5555'.
daemon	[default TRUE] launch as a persistent daemon or, if FALSE, an ephemeral process.

Value

Invisible NULL.

About

The network topology is such that server daemons dial into the client, which listens at the '.url' address. In this way, network resources may be added or removed at any time and the client automatically distributes tasks to all available servers.

stop_mirai *mirai (Stop Evaluation)*

Description

Stop evaluation of a mirai that is in progress.

Usage

```
stop_mirai(aio)
```

Arguments

aio	a 'mirai' object.

Details

Stops the asynchronous operation associated with the mirai by aborting, and then waits for it to complete or to be completely aborted. The mirai is then deallocated and attempting to access the value at `$data` will result in an error.

Value

Invisible NULL.

Examples

```
if (interactive()) {
# Only run examples in interactive R sessions

s <- mirai(Sys.sleep(n), n = 5)
stop_mirai(s)

}
```

unresolved *Query if a mirai is Unresolved*

Description

Query whether a mirai or mirai value remains unresolved. Unlike `call_mirai`, this function does not wait for completion.

Usage

```
unresolved(aio)
```

Arguments

aio	a 'mirai' object or 'mirai' value stored at $data.

Details

Suitable for use in control flow statements such as while or if.

Note: querying resolution may cause a previously unresolved 'mirai' to resolve.

Value

Logical TRUE if 'aio' is an unresolved mirai or mirai value, or FALSE otherwise.

Examples

```
if (interactive()) {
# Only run examples in interactive R sessions

m <- mirai(Sys.sleep(0.1))
unresolved(m)
Sys.sleep(0.3)
unresolved(m)

}
```

%>>% *Deferred Evaluation Pipe*

Description

Pipe a possibly unresolved value forward into a function.

Usage

```
x %>>% f
```

Arguments

x	a value that is possibly an 'unresolvedValue'.
f	a function that accepts 'x' as its first argument.

Details

An 'unresolvedExpr' encapsulates the eventual evaluation result. Query its $data element for resolution. Once resolved, the object changes into a 'resolvedExpr' and the evaluated result will be available at $data.

Supports stringing together a series of piped expressions (as per the below example).

unresolved may be used on an 'unresolvedExpr' or its $data element to test for resolution.

Value

The evaluated result, or if x is an 'unresolvedValue', an 'unresolvedExpr'.

Usage

Usage is similar to R's native `|>` pipe.

`x %>>% f` is equivalent to `f(x)`

`x %>>% f()` is equivalent to `f(x)`

`x %>>% f(y)` is equivalent to `f(x, y)`

Please note that other usage is not supported and it is not a drop-in replacement for magrittr's `%>%` pipe.

Examples

```
if (interactive()) {
# Only run examples in interactive R sessions

m <- mirai({Sys.sleep(0.5); 1})
b <- m$data %>>% c(2, 3) %>>% as.character()
b
b$data
call_mirai(m)
b$data
b

}
```

Index