Package 'future.mirai'

May 8, 2026

Version 0.10.1

Depends future (>= 1.49.0)

Imports mirai (>= 2.2.0), parallelly, utils

Suggests future.tests, future.apply, listenv

Title A 'Future' API for Parallel Processing using 'mirai'

Description Implementation of the 'Future' API <doi:10.32614/RJ-2021-048> on top of the 'mirai' package <doi:10.5281/zenodo.7912722>. By using this package, you get to take advantage of the benefits of 'mirai' plus everything else that 'future' and the 'Futureverse' adds on top of it. It allows you to process futures, as defined by the 'future' package, in parallel out of the box, on your local machine or across remote machines. Contrary to back-ends relying on the 'parallel' package (e.g. 'multisession') and socket connections, 'mirai_cluster' and 'mirai_multisession', provided here, can run more than 125 parallel R processes. As a reminder, regardless which future backend is used by the user, the code does not have to change, it gives identical results, and behaves exactly the same.

License GPL (>= 3)

Language en-US

Encoding UTF-8

URL https://future.mirai.futureverse.org,
https://github.com/futureverse/future.mirai

BugReports https://github.com/futureverse/future.mirai/issues

RoxygenNote 7.3.2

NeedsCompilation no

Author Henrik Bengtsson [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0002-7579-5165>),
Charlie Gao [ctb] (ORCID: <https://orcid.org/0000-0002-0750-061X>,
note: For 'mirai'-related patches and implementing feature requests
in 'mirai')

Maintainer Henrik Bengtsson <henrikb@braju.com>

Repository CRAN

Date/Publication 2025-07-10 10:30:01 UTC

Contents

`future.mirai` *future.mirai: A Future API for Parallel Processing using 'mirai'*

Description

The **future.mirai** package implements the Future API using the **mirai** package.

Author(s)

Maintainer: Henrik Bengtsson <henrikb@braju.com> (ORCID) [copyright holder]

Other contributors:

- Charlie Gao <charlie.gao@shikokuchuo.net> (ORCID) (For 'mirai'-related patches and implementing feature requests in 'mirai') [contributor]

See Also

Useful links:

- https://future.mirai.futureverse.org
- https://github.com/futureverse/future.mirai
- Report bugs at https://github.com/futureverse/future.mirai/issues

Examples

```
plan(mirai_multisession)

# A function that returns a future
# (note that N is a global variable)
f <- function() future({
  4 * sum((runif(N) ^ 2 + runif(N) ^ 2) < 1) / N
}, seed = TRUE)

# Run a simple sampling approximation of pi in parallel using  M * N points:
N <- 1e6  # samples per worker
M <- 10   # iterations
pi_est <- Reduce(sum, Map(value, replicate(M, f()))) / M
print(pi_est)

## Switch back to sequential processing, which also
## shuts down the automatically launched mirai workers
plan(sequential)
```

MiraiFutureBackend *A future backend based based on the 'mirai' framework*

Description

Set up the future parameters.

Usage

```
MiraiFutureBackend(...)
```

Arguments

`...`	Additional arguments passed to `Future()`.

Value

An object of class `MiraiFutureBackend`.

mirai_cluster *Mirai-based cluster futures*

Description

WARNING: This function must never be called. It may only be used with `future::plan()`

Usage

```
mirai_cluster(..., envir = parent.frame())
```

Arguments

`envir`	The environment from where global objects should be identified.
`...`	Not used.

Value

Nothing.

Examples

```
# Manually launch mirai workers
mirai::daemons(parallelly::availableCores())

plan(mirai_cluster)

# A function that returns a future
# (note that N is a global variable)
f <- function() future({
  4 * sum((runif(N) ^ 2 + runif(N) ^ 2) < 1) / N
}, seed = TRUE)

# Run a simple sampling approximation of pi in parallel using  M * N points:
N <- 1e6  # samples per worker
M <- 10   # iterations
pi_est <- Reduce(sum, Map(value, replicate(M, f()))) / M
print(pi_est)

## Switch back to sequential processing
plan(sequential)

## Shut down manually launched mirai workers
invisible(mirai::daemons(0))
```

`mirai_multisession` *Mirai-based localhost multisession futures*

Description

WARNING: This function must never be called. It may only be used with `future::plan()`

Usage

```
mirai_multisession(..., workers = availableCores(), envir = parent.frame())
```

Arguments

`workers`	The number of parallel processes to use. If a function, it is called without arguments *when the future is created* and its value is used to configure the workers. If `workers == 1`, then all processing using done in the current/main R session and we therefore fall back to using a sequential future. To override this fallback, use `workers = I(1)`.
`envir`	The environment from where global objects should be identified.
`...`	Not used.

Value

Nothing.

Examples

```
plan(mirai_multisession)

# A function that returns a future
# (note that N is a global variable)
f <- function() future({
  4 * sum((runif(N) ^ 2 + runif(N) ^ 2) < 1) / N
}, seed = TRUE)

# Run a simple sampling approximation of pi in parallel using  M * N points:
N <- 1e6  # samples per worker
M <- 10   # iterations
pi_est <- Reduce(sum, Map(value, replicate(M, f()))) / M
print(pi_est)

## Switch back to sequential processing, which also
## shuts down the automatically launched mirai workers
plan(sequential)
```

Index