Package ‘rush’

November 6, 2025

Title Rapid Asynchronous and Distributed Computing

Version 0.4.1

Description Package to tackle large-scale problems asynchronously across a distributed network. Employing a database centric model, rush enables workers to communicate tasks and their results over a shared 'Redis' database. Key features include low task overhead, efficient caching, and robust error handling. The package powers the asynchronous optimization algorithms in the 'bbotk' and 'mlr3tuning' packages.

License MIT + file LICENSE

URL `https://rush.mlr-org.com`, `https://github.com/mlr-org/rush`

BugReports `https://github.com/mlr-org/rush/issues`

Depends R (>= 3.1.0)

Imports checkmate, data.table, ids, jsonlite, lgr (>= 0.5.0), mirai (>= 2.5.0), mlr3misc, parallel, processx, R6, redux, uuid

Suggests callr, knitr, quarto, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

Config/testthat/parallel false

Encoding UTF-8

RoxygenNote 7.3.3

NeedsCompilation no

Author Marc Becker [cre, aut, cph] (ORCID: <`https://orcid.org/0000-0002-8115-0400`>)

Maintainer Marc Becker <`marcbecker@posteo.de`>

Repository CRAN

Date/Publication 2025-11-06 09:30:02 UTC

Contents

`rush-package` *rush: Rapid Asynchronous and Distributed Computing*

Description

Package to tackle large-scale problems asynchronously across a distributed network. Employing a database centric model, rush enables workers to communicate tasks and their results over a shared 'Redis' database. Key features include low task overhead, efficient caching, and robust error handling. The package powers the asynchronous optimization algorithms in the 'bbotk' and 'mlr3tuning' packages.

Author(s)

Maintainer: Marc Becker <`marcbecker@posteo.de`> (ORCID) [copyright holder]

See Also

Useful links:

- `https://rush.mlr-org.com`
- `https://github.com/mlr-org/rush`
- Report bugs at `https://github.com/mlr-org/rush/issues`

AppenderRedis *Log to Redis Database*

Description

AppenderRedis writes log messages to a Redis data base. This lgr::Appender is created internally by RushWorker when logger thresholds are passed via `rush_plan()`.

Value

Object of class R6::R6Class and `AppenderRedis` with methods for writing log events to Redis data bases.

Super classes

`lgr::Filterable` -> `lgr::Appender` -> `lgr::AppenderMemory` -> `AppenderRedis`

Methods

Public methods:

- `AppenderRedis$new()`
- `AppenderRedis$flush()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
AppenderRedis$new(
  config,
  key,
  threshold = NA_integer_,
  layout = lgr::LayoutJson$new(timestamp_fmt = "%Y-%m-%d %H:%M:%OS3"),
  buffer_size = 0,
  flush_threshold = "error",
  flush_on_exit = TRUE,
  flush_on_rotate = TRUE,
  should_flush = NULL,
  filters = NULL
)
```

Arguments:

`config` (redux::redis_config)
Redis configuration options.

`key` (`character(1)`)
Key of the list holding the log messages in the Redis data store.

`threshold` (`integer(1)` | `character(1)`)
Threshold for the log messages.

`layout` (lgr::Layout)
Layout for the log messages.

`buffer_size (integer(1))`
  Size of the buffer.

`flush_threshold (character(1))`
  Threshold for flushing the buffer.

`flush_on_exit (logical(1))`
  Flush the buffer on exit.

`flush_on_rotate (logical(1))`
  Flush the buffer on rotate.

`should_flush (function)`
  Function that determines if the buffer should be flushed.

`filters (list)`
  List of filters.

Method `flush()`**:** Sends the buffer's contents to the Redis data store, and then clears the buffer.

Usage:

```
AppenderRedis$flush()
```

Examples

```
# This example is not executed since Redis must be installed

   config_local = redux::redis_config()

   rush_plan(
     config = config_local,
     n_workers = 2,
     lgr_thresholds = c(rush = "info"))

   rush = rsh(network_id = "test_network")
   rush
```

`filter_custom_fields` *Filter Custom Fields*

Description

Filters custom fields from log events.

Usage

```
filter_custom_fields(event)
```

Arguments

`event`	(lgr::LogEvent) Log event.

`remove_rush_plan` *Remove Rush Plan*

Description

Removes the rush plan that was set by `rush_plan()`.

Usage

```
remove_rush_plan()
```

Value

Invisible `TRUE`. Function called for side effects.

Examples

```
# This example is not executed since Redis must be installed

  config_local = redux::redis_config()
  rush_plan(config = config_local, n_workers = 2)
  remove_rush_plan()
```

`rsh` *Synctactic Sugar for Rush Controller Construction*

Description

Function to construct a Rush controller.

Usage

```
rsh(network_id = NULL, config = NULL, seed = NULL, ...)
```

Arguments

`network_id`	(`character(1)`) Identifier of the rush network. Controller and workers must have the same instance id. Keys in Redis are prefixed with the instance id.
`config`	(redux::redis_config) Redis configuration options. If `NULL`, configuration set by `rush_plan()` is used. If `rush_plan()` has not been called, the `REDIS_URL` environment variable is parsed. If `REDIS_URL` is not set, a default configuration is used. See redux::redis_config for details.

`seed`	(`integer()`) Initial seed for the random number generator. Either a L'Ecuyer-CMRG seed (`integer(7)`) or a regular RNG seed (`integer(1)`). The later is converted to a L'Ecuyer-CMRG seed. If `NULL`, no seed is used for the random number generator.
`...`	(ignored).

Value

Rush controller.

Examples

```
# This example is not executed since Redis must be installed

   config_local = redux::redis_config()
   rush = rsh(network_id = "test_network", config = config_local)
   rush
```

`Rush` *Rush Controller*

Description

The `Rush` controller manages workers in a rush network.

Value

Object of class R6::R6Class and `Rush` with controller methods.

Local Workers

A local worker runs on the same machine as the controller. Local workers are spawned with the '$start_local_workers() method via the **processx** package.

Remote Workers

A remote worker runs on a different machine than the controller. Remote workers are spawned with the '$start_remote_workers() method via the **mirai** package.

Script Workers

Workers can be started with a script anywhere. The only requirement is that the worker can connect to the Redis database. The script is created with the `$worker_script()` method.

Public fields

`network_id` (`character(1)`)
  Identifier of the rush network.

`config` (redux::redis_config)
  Redis configuration options.

`connector` (redux::redis_api)
  Returns a connection to Redis.

`processes_processx` (processx::process)
  List of processes started with `$start_local_workers()`.

`processes_mirai` (mirai::mirai)
  List of mirai processes started with `$start_remote_workers()`.

Active bindings

`n_workers` (`integer(1)`)
  Number of workers.

`n_running_workers` (`integer(1)`)
  Number of running workers.

`n_terminated_workers` (`integer(1)`)
  Number of terminated workers.

`n_killed_workers` (`integer(1)`)
  Number of killed workers.

`n_lost_workers` (`integer(1)`)
  Number of lost workers. Run `$detect_lost_workers()` to update the number of lost workers.

`n_pre_workers` (`integer(1)`)
  Number of workers that are not yet completely started.

`worker_ids` (`character()`)
  Ids of workers.

`running_worker_ids` (`character()`)
  Ids of running workers.

`terminated_worker_ids` (`character()`)
  Ids of terminated workers.

`killed_worker_ids` (`character()`)
  Ids of killed workers.

`lost_worker_ids` (`character()`)
  Ids of lost workers.

`pre_worker_ids` (`character()`)
  Ids of workers that are not yet completely started.

`tasks` (`character()`)
  Keys of all tasks.

`queued_tasks` (`character()`)
  Keys of queued tasks.

`running_tasks` (`character()`)
: Keys of running tasks.

`finished_tasks` (`character()`)
: Keys of finished tasks.

`failed_tasks` (`character()`)
: Keys of failed tasks.

`n_queued_tasks` (`integer(1)`)
: Number of queued tasks.

`n_queued_priority_tasks` (`integer(1)`)
: Number of queued priority tasks.

`n_running_tasks` (`integer(1)`)
: Number of running tasks.

`n_finished_tasks` (`integer(1)`)
: Number of finished tasks.

`n_failed_tasks` (`integer(1)`)
: Number of failed tasks.

`n_tasks` (`integer(1)`)
: Number of all tasks.

`worker_info` (`data.table::data.table()`)
: Contains information about the workers.

`worker_states` (`data.table::data.table()`)
: Contains the states of the workers.

`all_workers_terminated` (`logical(1)`)
: Whether all workers are terminated.

`all_workers_lost` (`logical(1)`)
: Whether all workers are lost. Runs `$detect_lost_workers()` to detect lost workers.

`priority_info` (data.table::data.table)
: Contains the number of tasks in the priority queues.

`snapshot_schedule` (`character()`)
: Set a snapshot schedule to periodically save the data base on disk. For example, `c(60, 1000)` saves the data base every 60 seconds if there are at least 1000 changes. Overwrites the redis configuration file. Set to `NULL` to disable snapshots. For more details see redis.io.

`redis_info` (`list()`)
: Information about the Redis server.

Methods

Public methods:

- `Rush$new()`
- `Rush$format()`
- `Rush$print()`
- `Rush$reconnect()`
- `Rush$start_local_workers()`

- `Rush$start_remote_workers()`
- `Rush$worker_script()`
- `Rush$restart_workers()`
- `Rush$wait_for_workers()`
- `Rush$stop_workers()`
- `Rush$detect_lost_workers()`
- `Rush$reset()`
- `Rush$read_log()`
- `Rush$print_log()`
- `Rush$push_tasks()`
- `Rush$push_priority_tasks()`
- `Rush$push_failed()`
- `Rush$empty_queue()`
- `Rush$fetch_queued_tasks()`
- `Rush$fetch_priority_tasks()`
- `Rush$fetch_running_tasks()`
- `Rush$fetch_finished_tasks()`
- `Rush$wait_for_finished_tasks()`
- `Rush$fetch_new_tasks()`
- `Rush$wait_for_new_tasks()`
- `Rush$fetch_failed_tasks()`
- `Rush$fetch_tasks()`
- `Rush$fetch_tasks_with_state()`
- `Rush$wait_for_tasks()`
- `Rush$write_hashes()`
- `Rush$read_hashes()`
- `Rush$read_hash()`
- `Rush$is_running_task()`
- `Rush$is_failed_task()`
- `Rush$tasks_with_state()`
- `Rush$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
Rush$new(network_id = NULL, config = NULL, seed = NULL)
```

Arguments:

`network_id` (`character(1)`)
: Identifier of the rush network. Controller and workers must have the same instance id. Keys in Redis are prefixed with the instance id.

`config` (redux::redis_config)
: Redis configuration options. If `NULL`, configuration set by `rush_plan()` is used. If `rush_plan()` has not been called, the `REDIS_URL` environment variable is parsed. If `REDIS_URL` is not set, a default configuration is used. See redux::redis_config for details.

`seed` (`integer()`)
Initial seed for the random number generator. Either a L'Ecuyer-CMRG seed (`integer(7)`) or a regular RNG seed (`integer(1)`). The later is converted to a L'Ecuyer-CMRG seed. If `NULL`, no seed is used for the random number generator.

Method `format()`: Helper for print outputs.

Usage:

```
Rush$format(...)
```

Arguments:

`...` (ignored).

Returns: (`character()`).

Method `print()`: Print method.

Usage:

```
Rush$print()
```

Returns: (`character()`).

Method `reconnect()`: Reconnect to Redis. The connection breaks when the Rush object is saved to disk. Call this method to reconnect after loading the object.

Usage:

```
Rush$reconnect()
```

Method `start_local_workers()`: Start workers locally with `processx`. The processx::process are stored in `$processes_processx`. Alternatively, use `$start_remote_workers()` to start workers on remote machines with `mirai`. Parameters set by `rush_plan()` have precedence over the parameters set here.

Usage:

```
Rush$start_local_workers(
  worker_loop = NULL,
  ...,
  n_workers = NULL,
  globals = NULL,
  packages = NULL,
  lgr_thresholds = NULL,
  lgr_buffer_size = NULL,
  supervise = TRUE,
  message_log = NULL,
  output_log = NULL
)
```

Arguments:

`worker_loop` (`function`)
Loop run on the workers.

`...` (`any`)
Arguments passed to `worker_loop`.

`n_workers` (`integer(1)`)
: Number of workers to be started. Default is `NULL`, which means the number of workers is set by `rush_plan()`. If `rush_plan()` is not called, the default is `1`.

`globals` (`character()`)
: Global variables to be loaded to the workers global environment.

`packages` (`character()`)
: Packages to be loaded by the workers.

`lgr_thresholds` (named `character()` | named `numeric()`)
: Logger threshold on the workers e.g. `c("mlr3/rush" = "debug")`.

`lgr_buffer_size` (`integer(1)`)
: By default (`lgr_buffer_size = 0`), the log messages are directly saved in the Redis data store. If `lgr_buffer_size > 0`, the log messages are buffered and saved in the Redis data store when the buffer is full. This improves the performance of the logging.

`supervise` (`logical(1)`)
: Whether to kill the workers when the main R process is shut down.

`message_log` (`character(1)`)
: Path to the message log files e.g. `/tmp/message_logs/` The message log files are named `message_<worker_id>.log`. If `NULL`, no messages, warnings or errors are stored.

`output_log` (`character(1)`)
: Path to the output log files e.g. `/tmp/output_logs/` The output log files are named `output_<worker_id>.log`. If `NULL`, no output is stored.

Method `start_remote_workers()`**:** Start workers on remote machines with `mirai`. The mirai::mirai are stored in `$processes_mirai`. Parameters set by `rush_plan()` have precedence over the parameters set here.

Usage:

```
Rush$start_remote_workers(
  worker_loop,
  ...,
  n_workers = NULL,
  globals = NULL,
  packages = NULL,
  lgr_thresholds = NULL,
  lgr_buffer_size = NULL,
  message_log = NULL,
  output_log = NULL
)
```

Arguments:

`worker_loop` (`function`)
: Loop run on the workers.

`...` (any)
: Arguments passed to `worker_loop`.

`n_workers` (`integer(1)`)
: Number of workers to be started. Default is `NULL`, which means the number of workers is set by `rush_plan()`. If `rush_plan()` is not called, the default is `1`.

`globals` (`character()`)
: Global variables to be loaded to the workers global environment.

`packages` (`character()`)
: Packages to be loaded by the workers.

`lgr_thresholds` (named `character()` | named `numeric()`)
: Logger threshold on the workers e.g. `c("mlr3/rush" = "debug")`.

`lgr_buffer_size` (`integer(1)`)
: By default (`lgr_buffer_size = 0`), the log messages are directly saved in the Redis data store. If `lgr_buffer_size > 0`, the log messages are buffered and saved in the Redis data store when the buffer is full. This improves the performance of the logging.

`message_log` (`character(1)`)
: Path to the message log files e.g. `/tmp/message_logs/` The message log files are named `message_<worker_id>.log`. If `NULL`, no messages, warnings or errors are stored.

`output_log` (`character(1)`)
: Path to the output log files e.g. `/tmp/output_logs/` The output log files are named `output_<worker_id>.log`. If `NULL`, no output is stored.

Method `worker_script()`**:** Generate a script to start workers.

Usage:

```
Rush$worker_script(
  worker_loop,
  ...,
  globals = NULL,
  packages = NULL,
  lgr_thresholds = NULL,
  lgr_buffer_size = NULL,
  heartbeat_period = NULL,
  heartbeat_expire = NULL,
  message_log = NULL,
  output_log = NULL
)
```

Arguments:

`worker_loop` (`function`)
: Loop run on the workers.

`...` (any)
: Arguments passed to `worker_loop`.

`globals` (`character()`)
: Global variables to be loaded to the workers global environment.

`packages` (`character()`)
: Packages to be loaded by the workers.

`lgr_thresholds` (named `character()` | named `numeric()`)
: Logger threshold on the workers e.g. `c("mlr3/rush" = "debug")`.

`lgr_buffer_size` (`integer(1)`)
: By default (`lgr_buffer_size = 0`), the log messages are directly saved in the Redis data store. If `lgr_buffer_size > 0`, the log messages are buffered and saved in the Redis data store when the buffer is full. This improves the performance of the logging.

`heartbeat_period` (`integer(1)`)
: Period of the heartbeat in seconds. The heartbeat is updated every `heartbeat_period` seconds.

`heartbeat_expire (integer(1))`
: Time to live of the heartbeat in seconds. The heartbeat key is set to expire after `heartbeat_expire` seconds.

`message_log (character(1))`
: Path to the message log files e.g. `/tmp/message_logs/` The message log files are named `message_<worker_id>.log`. If `NULL`, no messages, warnings or errors are stored.

`output_log (character(1))`
: Path to the output log files e.g. `/tmp/output_logs/` The output log files are named `output_<worker_id>.log`. If `NULL`, no output is stored.

Method `restart_workers()`: Restart workers. If the worker is is still running, it is killed and restarted.

Usage:

```
Rush$restart_workers(worker_ids, supervise = TRUE)
```

Arguments:

`worker_ids (character())`
: Worker ids to be restarted.

`supervise (logical(1))`
: Whether to kill the workers when the main R process is shut down.

Method `wait_for_workers()`: Wait until workers are registered in the network. Either `n`, `worker_ids` or both must be provided.

Usage:

```
Rush$wait_for_workers(n = NULL, worker_ids = NULL, timeout = Inf)
```

Arguments:

`n (integer(1))`
: Number of workers to wait for. If `NULL`, wait for all workers in `worker_ids`.

`worker_ids (character())`
: Worker ids to wait for. If `NULL`, wait for any `n` workers to be registered.

`timeout (numeric(1))`
: Timeout in seconds. Default is `Inf`.

Method `stop_workers()`: Stop workers.

Usage:

```
Rush$stop_workers(type = "kill", worker_ids = NULL)
```

Arguments:

`type (character(1))`
: Type of stopping. Either `"terminate"` or `"kill"`. If `"kill"` the workers are stopped immediately. If `"terminate"` the workers evaluate the currently running task and then terminate. The `"terminate"` option must be implemented in the worker loop.

`worker_ids (character())`
: Worker ids to be stopped. Remote workers must all be killed together. If `NULL` all workers are stopped.

Method `detect_lost_workers()`: Detect lost workers. The state of the worker is changed to `"lost"`.

Usage:

```
Rush$detect_lost_workers(restart_local_workers = FALSE)
```

Arguments:

`restart_local_workers` (`logical(1)`)
    Whether to restart lost workers. Ignored for remote workers.

Method `reset()`**:** Stop workers and delete data stored in redis.

Usage:

```
Rush$reset(type = "kill")
```

Arguments:

`type` (`character(1)`)
    Type of stopping. Either `"terminate"` or `"kill"`. If `"terminate"` the workers evaluate the currently running task and then terminate. If `"kill"` the workers are stopped immediately.

Method `read_log()`**:** Read log messages written with the `lgr` package from a worker.

Usage:

```
Rush$read_log(worker_ids = NULL, time_difference = FALSE)
```

Arguments:

`worker_ids` (`character(1)`)
    Worker ids. If `NULL` all worker ids are used.

`time_difference` (`logical(1)`)
    Whether to calculate the time difference between log messages.

Returns: (`data.table::data.table()`) with level, timestamp, logger, caller and message, and optionally time difference.

Method `print_log()`**:** Print log messages written with the `lgr` package from a worker.

Usage:

```
Rush$print_log()
```

Method `push_tasks()`**:** Pushes a task to the queue. Task is added to queued tasks.

Usage:

```
Rush$push_tasks(
  xss,
  extra = NULL,
  seeds = NULL,
  timeouts = NULL,
  max_retries = NULL,
  terminate_workers = FALSE
)
```

Arguments:

`xss` (list of named `list()`)
    Lists of arguments for the function e.g. `list(list(x1, x2), list(x1, x2)))`.

`extra` (`list()`)
    List of additional information stored along with the task e.g. `list(list(timestamp), list(timestamp)))`.

`seeds` (`list()`)
  List of L'Ecuyer-CMRG seeds for each task e.g `list(list(c(104071, 490840688, 1690070564, -495119766, 503491950, 1801530932, -1629447803)))`. If `NULL` but an initial seed is set, L'Ecuyer-CMRG seeds are generated from the initial seed. If `NULL` and no initial seed is set, no seeds are used for the random number generator.

`timeouts` (`integer()`)
  Timeouts for each task in seconds e.g. `c(10, 15)`. A single number is used as the timeout for all tasks. If `NULL` no timeout is set.

`max_retries` (`integer()`)
  Number of retries for each task. A single number is used as the number of retries for all tasks. If `NULL` tasks are not retried.

`terminate_workers` (`logical(1)`)
  Whether to stop the workers after evaluating the tasks.

Returns: (`character()`)
Keys of the tasks.

Method `push_priority_tasks()`**:** Pushes a task to the queue of a specific worker. Task is added to queued priority tasks. A worker evaluates the tasks in the priority queue before the shared queue. If `priority` is `NA` the task is added to the shared queue. If the worker is lost or worker id is not known, the task is added to the shared queue.

Usage:

```
Rush$push_priority_tasks(xss, extra = NULL, priority = NULL)
```

Arguments:

`xss` (list of named `list()`)
  Lists of arguments for the function e.g. `list(list(x1, x2), list(x1, x2)))`.

`extra` (`list`)
  List of additional information stored along with the task e.g. `list(list(timestamp), list(timestamp)))`.

`priority` (`character()`)
  Worker ids to which the tasks should be pushed.

Returns: (`character()`)
Keys of the tasks.

Method `push_failed()`**:** Pushes failed tasks to the data base. Tasks are moved from queued and running to failed.

Usage:

```
Rush$push_failed(keys, conditions)
```

Arguments:

`keys` (`character(1)`)
  Keys of the associated tasks.

`conditions` (named `list()`)
  List of lists of conditions.

Method `empty_queue()`**:** Empty the queue of tasks. Moves tasks from queued to failed.

Usage:

```
Rush$empty_queue(keys = NULL, conditions = NULL)
```

Arguments:

`keys` (`character()`)
: Keys of the tasks to be moved. Defaults to all queued tasks.

`conditions` (named `list()`)
: List of lists of conditions.

Method `fetch_queued_tasks()`**:** Fetch queued tasks from the data base.

Usage:

```
Rush$fetch_queued_tasks(
  fields = c("xs", "xs_extra"),
  data_format = "data.table"
)
```

Arguments:

`fields` (`character()`)
: Fields to be read from the hashes. Defaults to `c("xs", "xs_extra")`.

`data_format` (`character()`)
: Returned data format. Choose `"data.table"` or "list". The default is `"data.table"` but `"list"` is easier when list columns are present.

Returns: `data.table()`
Table of queued tasks.

Method `fetch_priority_tasks()`**:** Fetch queued priority tasks from the data base.

Usage:

```
Rush$fetch_priority_tasks(
  fields = c("xs", "xs_extra"),
  data_format = "data.table"
)
```

Arguments:

`fields` (`character()`)
: Fields to be read from the hashes. Defaults to `c("xs", "xs_extra")`.

`data_format` (`character()`)
: Returned data format. Choose `"data.table"` or "list". The default is `"data.table"` but `"list"` is easier when list columns are present.

Returns: `data.table()`
Table of queued priority tasks.

Method `fetch_running_tasks()`**:** Fetch running tasks from the data base.

Usage:

```
Rush$fetch_running_tasks(
  fields = c("xs", "xs_extra", "worker_extra"),
  data_format = "data.table"
)
```

Arguments:

`fields` (`character()`)
: Fields to be read from the hashes. Defaults to `c("xs", "xs_extra", "worker_extra")`.

`data_format` (`character()`)
: Returned data format. Choose `"data.table"` or "list". The default is `"data.table"` but `"list"` is easier when list columns are present.

Returns: `data.table()`
Table of running tasks.

Method `fetch_finished_tasks()`**:** Fetch finished tasks from the data base. Finished tasks are cached.

Usage:

```
Rush$fetch_finished_tasks(
  fields = c("xs", "ys", "xs_extra", "worker_extra", "ys_extra", "condition"),
  reset_cache = FALSE,
  data_format = "data.table"
)
```

Arguments:

`fields` (`character()`)
: Fields to be read from the hashes. Defaults to `c("xs", "xs_extra", "worker_extra", "ys", "ys_extra")`.

`reset_cache` (`logical(1)`)
: Whether to reset the cache.

`data_format` (`character()`)
: Returned data format. Choose `"data.table"` or "list". The default is `"data.table"` but `"list"` is easier when list columns are present.

Returns: `data.table()`
Table of finished tasks.

Method `wait_for_finished_tasks()`**:** Block process until a new finished task is available. Returns all finished tasks or `NULL` if no new task is available after `timeout` seconds.

Usage:

```
Rush$wait_for_finished_tasks(
  fields = c("xs", "ys", "xs_extra", "worker_extra", "ys_extra"),
  timeout = Inf,
  data_format = "data.table"
)
```

Arguments:

`fields` (`character()`)
: Fields to be read from the hashes. Defaults to `c("xs", "xs_extra", "worker_extra", "ys", "ys_extra")`.

`timeout` (`numeric(1)`)
: Time to wait for a result in seconds.

`data_format` (`character()`)
: Returned data format. Choose `"data.table"` or "list". The default is `"data.table"` but `"list"` is easier when list columns are present.

Returns: `data.table()`
Table of finished tasks.

Method `fetch_new_tasks()`**:** Fetch finished tasks from the data base that finished after the last fetch. Updates the cache of the finished tasks.

Usage:

```
Rush$fetch_new_tasks(
  fields = c("xs", "ys", "xs_extra", "worker_extra", "ys_extra", "condition"),
  data_format = "data.table"
)
```

Arguments:

`fields` (`character()`)
Fields to be read from the hashes.

`data_format` (`character()`)
Returned data format. Choose `"data.table"` or "list". The default is `"data.table"` but `"list"` is easier when list columns are present.

Returns: `data.table()`
Latest results.

Method `wait_for_new_tasks()`**:** Block process until a new finished task is available. Returns new tasks or `NULL` if no new task is available after `timeout` seconds.

Usage:

```
Rush$wait_for_new_tasks(
  fields = c("xs", "ys", "xs_extra", "worker_extra", "ys_extra", "condition"),
  timeout = Inf,
  data_format = "data.table"
)
```

Arguments:

`fields` (`character()`)
Fields to be read from the hashes. Defaults to `c("xs", "xs_extra", "worker_extra", "ys", "ys_extra")`.

`timeout` (`numeric(1)`)
Time to wait for new result in seconds.

`data_format` (`character()`)
Returned data format. Choose `"data.table"` or "list". The default is `"data.table"` but `"list"` is easier when list columns are present.

Returns: `data.table() | list()`.

Method `fetch_failed_tasks()`**:** Fetch failed tasks from the data base.

Usage:

```
Rush$fetch_failed_tasks(
  fields = c("xs", "worker_extra", "condition"),
  data_format = "data.table"
)
```

Arguments:

`fields` (`character()`)
Fields to be read from the hashes. Defaults to `c("xs", "xs_extra", "worker_extra", "condition"`.

`data_format` (`character()`)
Returned data format. Choose `"data.table"` or "list". The default is `"data.table"` but `"list"` is easier when list columns are present.

Returns: `data.table()`
Table of failed tasks.

Method `fetch_tasks()`**:** Fetch all tasks from the data base.

Usage:

```
Rush$fetch_tasks(
  fields = c("xs", "ys", "xs_extra", "worker_extra", "ys_extra", "condition"),
  data_format = "data.table"
)
```

Arguments:

`fields` (`character()`)
Fields to be read from the hashes. Defaults to `c("xs", "xs_extra", "worker_extra", "ys", "ys_extra", "condition", "state")`.

`data_format` (`character()`)
Returned data format. Choose `"data.table"` or "list". The default is `"data.table"` but `"list"` is easier when list columns are present.

Returns: `data.table()`
Table of all tasks.

Method `fetch_tasks_with_state()`**:** Fetch tasks with different states from the data base. If tasks with different states are to be queried at the same time, this function prevents tasks from appearing twice. This could be the case if a worker changes the state of a task while the tasks are being fetched. Finished tasks are cached.

Usage:

```
Rush$fetch_tasks_with_state(
  fields = c("xs", "ys", "xs_extra", "worker_extra", "ys_extra", "condition"),
  states = c("queued", "running", "finished", "failed"),
  reset_cache = FALSE,
  data_format = "data.table"
)
```

Arguments:

`fields` (`character()`)
Fields to be read from the hashes. Defaults to `c("xs", "ys", "xs_extra", "worker_extra", "ys_extra")`.

`states` (`character()`)
States of the tasks to be fetched. Defaults to `c("queued", "running", "finished", "failed")`.

`reset_cache` (`logical(1)`)
Whether to reset the cache of the finished tasks.

`data_format` (`character()`)
Returned data format. Choose `"data.table"` or "list". The default is `"data.table"` but `"list"` is easier when list columns are present.

Method `wait_for_tasks()`**:** Wait until tasks are finished. The function also unblocks when no worker is running or all tasks failed.

Usage:

```
Rush$wait_for_tasks(keys, detect_lost_workers = FALSE)
```

Arguments:

`keys` (`character()`)
 Keys of the tasks to wait for.

`detect_lost_workers` (`logical(1)`)
 Whether to detect failed tasks. Comes with an overhead.

Method `write_hashes()`**:** Writes R objects to Redis hashes. The function takes the vectors in `...` as input and writes each element as a field-value pair to a new hash. The name of the argument defines the field into which the serialized element is written. For example, `xs = list(list(x1 = 1, x2 = 2), list(x1 = 3, x2 = 4))` writes `serialize(list(x1 = 1, x2 = 2))` at field `xs` into a hash and `serialize(list(x1 = 3, x2 = 4))` at field `xs` into another hash. The function can iterate over multiple vectors simultaneously. For example, `xs = list(list(x1 = 1, x2 = 2), list(x1 = 3, x2 = 4)), ys =` creates two hashes with the fields `xs` and `ys`. The vectors are recycled to the length of the longest vector. Both lists and atomic vectors are supported. Arguments that are `NULL` are ignored.

Usage:

```
Rush$write_hashes(..., .values = list(), keys = NULL)
```

Arguments:

`...` (named `list()`)
 Lists to be written to the hashes. The names of the arguments are used as fields.

`.values` (named `list()`)
 Lists to be written to the hashes. The names of the list are used as fields.

`keys` (character())
 Keys of the hashes. If `NULL` new keys are generated.

Returns: (`character()`)
Keys of the hashes.

Method `read_hashes()`**:** Reads R Objects from Redis hashes. The function reads the field-value pairs of the hashes stored at `keys`. The values of a hash are deserialized and combined to a list. If `flatten` is `TRUE`, the values are flattened to a single list e.g. list(xs = list(x1 = 1, x2 = 2), ys = list(y = 3)) becomes list(x1 = 1, x2 = 2, y = 3). The reading functions combine the hashes to a table where the names of the inner lists are the column names. For example, `xs = list(list(x1 = 1, x2 = 2), list(x1 = 3, x2 = 4)), ys = list(list(y = 3), list(y = 7))` becomes `data.table(x1 = c(1, 3), x2 = c(2, 4), y = c(3, 7))`.

Usage:

```
Rush$read_hashes(keys, fields, flatten = TRUE)
```

Arguments:

`keys` (`character()`)
 Keys of the hashes.

`fields` (`character()`)
 Fields to be read from the hashes.

`flatten` (`logical(1)`)
Whether to flatten the list.

Returns: (list of `list()`)
The outer list contains one element for each key. The inner list is the combination of the lists stored at the different fields.

Method `read_hash()`**:** Reads a single Redis hash and returns the values as a list named by the fields.

Usage:

```
Rush$read_hash(key, fields)
```

Arguments:

`key` (`character(1)`)
Key of the hash.

`fields` (`character()`)
Fields to be read from the hash.

Returns: (list of `list()`)
The outer list contains one element for each key. The inner list is the combination of the lists stored at the different fields.

Method `is_running_task()`**:** Checks whether tasks have the status `"running"`.

Usage:

```
Rush$is_running_task(keys)
```

Arguments:

`keys` (`character()`)
Keys of the tasks.

Method `is_failed_task()`**:** Checks whether tasks have the status `"failed"`.

Usage:

```
Rush$is_failed_task(keys)
```

Arguments:

`keys` (`character()`)
Keys of the tasks.

Method `tasks_with_state()`**:** Returns keys of requested states.

Usage:

```
Rush$tasks_with_state(states)
```

Arguments:

`states` (`character()`)
States of the tasks.

Returns: (Named list of `character()`).

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Rush$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# This example is not executed since Redis must be installed

   config_local = redux::redis_config()
   rush = rsh(network_id = "test_network", config = config_local)
   rush
```

RushWorker *Rush Worker*

Description

RushWorker evaluates tasks and writes results to the data base. The worker inherits from Rush.

Value

Object of class R6::R6Class and `RushWorker` with worker methods.

Super class

`rush::Rush` -> `RushWorker`

Public fields

`worker_id (character(1))`
Identifier of the worker.

`remote (logical(1))`
Whether the worker is on a remote machine.

`heartbeat (callr::r_bg)`
Background process for the heartbeat.

Active bindings

`terminated (logical(1))`
Whether to shutdown the worker. Used in the worker loop to determine whether to continue.

`terminated_on_idle (logical(1))`
Whether to shutdown the worker if no tasks are queued. Used in the worker loop to determine whether to continue.

Methods

Public methods:

- `RushWorker$new()`
- `RushWorker$push_running_tasks()`
- `RushWorker$pop_task()`

- `RushWorker$push_results()`
- `RushWorker$set_terminated()`
- `RushWorker$clone()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
RushWorker$new(
  network_id,
  config = NULL,
  remote,
  worker_id = NULL,
  heartbeat_period = NULL,
  heartbeat_expire = NULL,
  seed = NULL
)
```

Arguments:

`network_id (character(1))`
Identifier of the rush network. Controller and workers must have the same instance id. Keys in Redis are prefixed with the instance id.

`config` (redux::redis_config)
Redis configuration options. If `NULL`, configuration set by `rush_plan()` is used. If `rush_plan()` has not been called, the `REDIS_URL` environment variable is parsed. If `REDIS_URL` is not set, a default configuration is used. See redux::redis_config for details.

`remote (logical(1))`
Whether the worker is started on a remote machine. See Rush for details.

`worker_id (character(1))`
Identifier of the worker. Keys in redis specific to the worker are prefixed with the worker id.

`heartbeat_period (integer(1))`
Period of the heartbeat in seconds. The heartbeat is updated every `heartbeat_period` seconds.

`heartbeat_expire (integer(1))`
Time to live of the heartbeat in seconds. The heartbeat key is set to expire after `heartbeat_expire` seconds.

`seed (integer())`
Initial seed for the random number generator. Either a L'Ecuyer-CMRG seed (`integer(7)`) or a regular RNG seed (`integer(1)`). The later is converted to a L'Ecuyer-CMRG seed. If `NULL`, no seed is used for the random number generator.

Method `push_running_tasks()`**:** Push a task to running tasks without queue.

Usage:

```
RushWorker$push_running_tasks(xss, extra = NULL)
```

Arguments:

`xss` (list of named `list()`)
Lists of arguments for the function e.g. `list(list(x1, x2), list(x1, x2)))`.

`extra (list)`
List of additional information stored along with the task e.g. `list(list(timestamp), list(timestamp)))`.

Returns: `(character())`
Keys of the tasks.

Method `pop_task()`**:** Pop a task from the queue. Task is moved to the running tasks.

Usage:

```
RushWorker$pop_task(timeout = 1, fields = "xs")
```

Arguments:

`timeout` (`numeric(1)`)
Time to wait for task in seconds.

`fields` (`character()`)
Fields to be returned.

Method `push_results()`**:** Pushes results to the data base.

Usage:

```
RushWorker$push_results(keys, yss, extra = NULL)
```

Arguments:

`keys` (`character(1)`)
Keys of the associated tasks.

`yss` (named `list()`)
List of lists of named results.

`extra` (named `list()`)
List of lists of additional information stored along with the results.

Method `set_terminated()`**:** Mark the worker as terminated. Last step in the worker loop before the worker terminates.

Usage:

```
RushWorker$set_terminated()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
RushWorker$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Note

The worker registers itself in the data base of the rush network.

`rush_available` *Rush Available*

Description

Returns `TRUE` if a redis config file (redux::redis_config) has been set by `rush_plan()`.

Usage

```
rush_available()
```

Value

`logical(1)`

Examples

```
# This example is not executed since Redis must be installed

  config_local = redux::redis_config()
  rush_plan(config = config_local, n_workers = 2)
  rush_available()
```

`rush_config` *Get Rush Config*

Description

Returns the rush config that was set by `rush_plan()`.

Usage

```
rush_config()
```

Value

`list()` with the stored configuration.

Examples

```
# This example is not executed since Redis must be installed

  config_local = redux::redis_config()
  rush_plan(config = config_local, n_workers = 2)
  rush_config()
```

rush_plan *Create Rush Plan*

Description

Stores the number of workers and Redis configuration options (redux::redis_config) for Rush. The function tests the connection to Redis and throws an error if the connection fails. This function is usually used in third-party packages to setup how workers are started.

Usage

```
rush_plan(
  n_workers = NULL,
  config = NULL,
  lgr_thresholds = NULL,
  lgr_buffer_size = NULL,
  large_objects_path = NULL,
  worker_type = "local"
)
```

Arguments

`n_workers`	(`integer(1)`) Number of workers to be started.
`config`	(redux::redis_config) Configuration options used to connect to Redis. If `NULL`, the `REDIS_URL` environment variable is parsed. If `REDIS_URL` is not set, a default configuration is used. See redux::redis_config for details.
`lgr_thresholds`	(named `character()` | named `numeric()`) Logger threshold on the workers e.g. `c("mlr3/rush" = "debug")`.
`lgr_buffer_size`	(`integer(1)`) By default (`lgr_buffer_size = 0`), the log messages are directly saved in the Redis data store. If `lgr_buffer_size > 0`, the log messages are buffered and saved in the Redis data store when the buffer is full. This improves the performance of the logging.
`large_objects_path`	(`character(1)`) The path to the directory where large objects are stored.
`worker_type`	(`character(1)`) The type of worker to use. Options are `"local"` to start with **processx**, `"remote"` to use **mirai** or `"script"` to get a script to run.

Value

`list()` with the stored configuration.

Examples

```
# This example is not executed since Redis must be installed

   config_local = redux::redis_config()
   rush_plan(config = config_local, n_workers = 2)

   rush = rsh(network_id = "test_network")
   rush
```

`start_worker` *Start a worker*

Description

Starts a worker. The function loads the globals and packages, initializes the RushWorker instance and invokes the worker loop. This function is called by `$start_local_workers()` or by the user after creating the worker script with `$create_worker_script()`. Use with caution. The global environment is changed.

Usage

```
start_worker(
  worker_id = NULL,
  network_id,
  config = NULL,
  remote = TRUE,
  lgr_thresholds = NULL,
  lgr_buffer_size = 0,
  heartbeat_period = NULL,
  heartbeat_expire = NULL,
  message_log = NULL,
  output_log = NULL
)
```

Arguments

`worker_id`	(`character(1)`) Identifier of the worker. Keys in redis specific to the worker are prefixed with the worker id.
`network_id`	(`character(1)`) Identifier of the rush network. Controller and workers must have the same instance id. Keys in Redis are prefixed with the instance id.
`config`	(`list()`) Configuration for the Redis connection.
`remote`	(`logical(1)`) Whether the worker is on a remote machine.

`lgr_thresholds`	(named `character()` | named `numeric()`) Logger threshold on the workers e.g. `c("mlr3/rush" = "debug")`.
`lgr_buffer_size`	(`integer(1)`) By default (`lgr_buffer_size = 0`), the log messages are directly saved in the Redis data store. If `lgr_buffer_size > 0`, the log messages are buffered and saved in the Redis data store when the buffer is full. This improves the performance of the logging.
`heartbeat_period`	(`integer(1)`) Period of the heartbeat in seconds. The heartbeat is updated every `heartbeat_period` seconds.
`heartbeat_expire`	(`integer(1)`) Time to live of the heartbeat in seconds. The heartbeat key is set to expire after `heartbeat_expire` seconds.
`message_log`	(`character(1)`) Path to the message log files e.g. `/tmp/message_logs/` The message log files are named `message_<worker_id>.log`. If `NULL`, no messages, warnings or errors are stored.
`output_log`	(`character(1)`) Path to the output log files e.g. `/tmp/output_logs/` The output log files are named `output_<worker_id>.log`. If `NULL`, no output is stored.

Value

`NULL`

Note

The function initializes the connection to the Redis data base. It loads the packages and copies the globals to the global environment of the worker. The function initialize the RushWorker instance and starts the worker loop.

Examples

```
# This example is not executed since Redis must be installed
## Not run:
  rush::start_worker(
   network_id = 'test-rush',
   remote = TRUE,
   url = 'redis://127.0.0.1:6379',
   scheme = 'redis',
   host = '127.0.0.1',
   port = '6379')

## End(Not run)
```

`store_large_object` *Store Large Objects*

Description

Store large objects to disk and return a reference to the object.

Usage

```
store_large_object(obj, path)
```

Arguments

`obj`	`(any)` Object to store.
`path`	`(character(1))` Path to store the object.

Value

`list()` of class `"rush_large_object"` with the name and path of the stored object.

Examples

```
obj = list(a = 1, b = 2)
rush_large_object = store_large_object(obj, tempdir())
```

Index