Package 'crew.cluster'

June 9, 2025

Title Crew Launcher Plugins for Traditional High-Performance Computing Clusters

Description In computationally demanding analysis projects, statisticians and data scientists asynchronously deploy long-running tasks to distributed systems, ranging from traditional clusters to cloud services. The 'crew.cluster' package extends the 'mirai'-powered 'crew' package with worker launcher plugins for traditional high-performance computing systems. Inspiration also comes from packages 'mirai' by Gao (2023) <https://github.com/r-lib/mirai>, 'future' by Bengtsson (2021) <doi:10.32614/RJ-2021-048>, 'rrq' by FitzJohn and Ashton (2023) <https://github.com/mrc-ide/rrq>, 'clustermq' by Schubert (2019) <doi:10.1093/bioinformatics/btz284>), and 'batchtools' by Lang, Bischl, and Surmann (2017). <doi:10.21105/joss.00135>.

Version 0.3.8

License MIT + file LICENSE

URL https://wlandau.github.io/crew.cluster/, https://github.com/wlandau/crew.cluster

BugReports https://github.com/wlandau/crew.cluster/issues

Depends R (>= 4.0.0)

Imports crew (>= 1.2.0), ps, lifecycle, R6, rlang, utils, vctrs, xml2, yaml

Suggests knitr (>= 1.30), markdown (>= 1.1), rmarkdown (>= 2.4), testthat (>= 3.0.0)

Encoding UTF-8

Language en-US

Config/testthat/edition 3

RoxygenNote 7.3.2

NeedsCompilation no

Author William Michael Landau [aut, cre] (ORCID:
<https://orcid.org/0000-0003-1878-3253>),
Michael Gilbert Levin [aut] (ORCID:
<https://orcid.org/0000-0002-9937-9932>),
Brendan Furneaux [aut] (ORCID: <https://orcid.org/0000-0003-3522-7363>),
Eli Lilly and Company [cph, fnd]

Maintainer William Michael Landau <will.landau.oss@gmail.com>

Repository CRAN

Date/Publication 2025-06-09 14:40:02 UTC

Contents

`crew.cluster-package` *crew.cluster: crew launcher plugins for traditional high-performance computing clusters*

Description

In computationally demanding analysis projects, statisticians and data scientists asynchronously deploy long-running tasks to distributed systems, ranging from traditional clusters to cloud services. The `crew.cluster` package extends the `mirai`-powered `crew` package with worker launcher plugins for traditional high-performance computing systems. Inspiration also comes from packages `mirai`, `future`, `rrq`, `clustermq`, and `batchtools`.

Attribution

The template files at https://github.com/mschubert/clustermq/tree/master/inst informed the development of the crew launcher plugins in crew.cluster, and we would like to thank Michael Schubert for developing clustermq and releasing it under the permissive Apache License 2.0. See the NOTICE and README.md files in the crew.cluster source code for additional attribution.

---

crew_class_launcher_lsf

[Experimental] *LSF launcher class*

Description

R6 class to launch and manage LSF workers.

Details

See crew_launcher_lsf().

Attribution

The template files at https://github.com/mschubert/clustermq/tree/master/inst informed the development of the crew launcher plugins in crew.cluster, and we would like to thank Michael Schubert for developing clustermq and releasing it under the permissive Apache License 2.0. See the NOTICE and README.md files in the crew.cluster source code for additional attribution.

Super classes

crew::crew_class_launcher -> crew.cluster::crew_class_launcher_cluster -> crew_class_launcher_lsf

Methods

Public methods:

- crew_class_launcher_lsf$validate()
- crew_class_launcher_lsf$script()

Method `validate()`**:** Validate the launcher.

Usage:

```
crew_class_launcher_lsf$validate()
```

Returns: NULL (invisibly). Throws an error if a field is invalid.

Method `script()`**:** Generate the job script.

Usage:

```
crew_class_launcher_lsf$script(name)
```

Arguments:

`name` Character of length 1, name of the job. For inspection purposes, you can supply a mock job name.

Details: Includes everything except the worker-instance-specific job name and the worker-instance-specific call to `crew::crew_worker()`, both of which get inserted at the bottom of the script at launch time.

Returns: Character vector of the lines of the job script.

Examples:

```
if (identical(Sys.getenv("CREW_EXAMPLES"), "true")) {
launcher <- crew_launcher_lsf(
  lsf_cwd = getwd(),
  lsf_log_output = "log_file_%J.log",
  lsf_log_error = NULL,
  lsf_memory_gigabytes_limit = 4
)
launcher$script(name = "my_job_name")
}
```

See Also

Other lsf: `crew_controller_lsf()`, `crew_launcher_lsf()`, `crew_options_lsf()`

Examples

```
## ------------------------------------------------
## Method `crew_class_launcher_lsf$script`
## ------------------------------------------------

if (identical(Sys.getenv("CREW_EXAMPLES"), "true")) {
launcher <- crew_launcher_lsf(
  lsf_cwd = getwd(),
  lsf_log_output = "log_file_%J.log",
  lsf_log_error = NULL,
  lsf_memory_gigabytes_limit = 4
)
launcher$script(name = "my_job_name")
}
```

`crew_class_launcher_pbs`

[Maturing] *PBS/TORQUE launcher class*

Description

`R6` class to launch and manage PBS/TORQUE workers.

Details

See crew_launcher_pbs().

Attribution

The template files at https://github.com/mschubert/clustermq/tree/master/inst informed the development of the crew launcher plugins in crew.cluster, and we would like to thank Michael Schubert for developing clustermq and releasing it under the permissive Apache License 2.0. See the NOTICE and README.md files in the crew.cluster source code for additional attribution.

Super classes

crew::crew_class_launcher -> crew.cluster::crew_class_launcher_cluster -> crew_class_launcher_pbs

Methods

Public methods:

- crew_class_launcher_pbs$validate()
- crew_class_launcher_pbs$script()

Method validate(): Validate the launcher.

Usage:

```
crew_class_launcher_pbs$validate()
```

Returns: NULL (invisibly). Throws an error if a field is invalid.

Method script(): Generate the job script.

Usage:

```
crew_class_launcher_pbs$script(name)
```

Arguments:

name Character of length 1, name of the job. For inspection purposes, you can supply a mock job name.

Details: Includes everything except the worker-instance-specific job name and the worker-instance-specific call to crew::crew_worker(), both of which get inserted at the bottom of the script at launch time.

Returns: Character vector of the lines of the job script.

Examples:

```
if (identical(Sys.getenv("CREW_EXAMPLES"), "true")) {
launcher <- crew_launcher_pbs(
  pbs_cores = 2,
  pbs_memory_gigabytes_required = 4
)
launcher$script(name = "my_job_name")
}
```

See Also

Other pbs: crew_controller_pbs(), crew_launcher_pbs(), crew_options_pbs()

Examples

```
## ------------------------------------------------
## Method `crew_class_launcher_pbs$script`
## ------------------------------------------------

if (identical(Sys.getenv("CREW_EXAMPLES"), "true")) {
launcher <- crew_launcher_pbs(
  pbs_cores = 2,
  pbs_memory_gigabytes_required = 4
)
launcher$script(name = "my_job_name")
}
```

---

crew_class_launcher_sge

[Maturing] *SGE launcher class*

Description

R6 class to launch and manage SGE workers.

Details

See crew_launcher_sge().

Attribution

The template files at https://github.com/mschubert/clustermq/tree/master/inst informed the development of the crew launcher plugins in crew.cluster, and we would like to thank Michael Schubert for developing clustermq and releasing it under the permissive Apache License 2.0. See the NOTICE and README.md files in the crew.cluster source code for additional attribution.

Super classes

crew::crew_class_launcher -> crew.cluster::crew_class_launcher_cluster -> crew_class_launcher_sge

Methods

Public methods:

- crew_class_launcher_sge$validate()
- crew_class_launcher_sge$script()

Method validate(): Validate the launcher.

Usage:

```
crew_class_launcher_sge$validate()
```

Returns: `NULL` (invisibly). Throws an error if a field is invalid.

Method `script()`**:** Generate the job script.

Usage:

```
crew_class_launcher_sge$script(name)
```

Arguments:

`name` Character of length 1, name of the job. For inspection purposes, you can supply a mock job name.

Details: Includes everything except the worker-instance-specific job name and the worker-instance-specific call to `crew::crew_worker()`, both of which get inserted at the bottom of the script at launch time.

Returns: Character vector of the lines of the job script.

Examples:

```
if (identical(Sys.getenv("CREW_EXAMPLES"), "true")) {
launcher <- crew_launcher_sge(
  sge_cores = 2,
  sge_memory_gigabytes_required = 4
)
launcher$script(name = "my_job_name")
}
```

See Also

Other sge: `crew_class_monitor_sge`, `crew_controller_sge()`, `crew_launcher_sge()`, `crew_monitor_sge()`, `crew_options_sge()`

Examples

```
## ------------------------------------------------
## Method `crew_class_launcher_sge$script`
## ------------------------------------------------

if (identical(Sys.getenv("CREW_EXAMPLES"), "true")) {
launcher <- crew_launcher_sge(
  sge_cores = 2,
  sge_memory_gigabytes_required = 4
)
launcher$script(name = "my_job_name")
}
```

crew_class_launcher_slurm **[Experimental]** *SLURM launcher class*

Description

R6 class to launch and manage SLURM workers.

Details

See crew_launcher_slurm().

Attribution

The template files at https://github.com/mschubert/clustermq/tree/master/inst informed the development of the crew launcher plugins in crew.cluster, and we would like to thank Michael Schubert for developing clustermq and releasing it under the permissive Apache License 2.0. See the NOTICE and README.md files in the crew.cluster source code for additional attribution.

Super classes

crew::crew_class_launcher -> crew.cluster::crew_class_launcher_cluster -> crew_class_launcher_slurm

Methods

Public methods:

- crew_class_launcher_slurm$validate()
- crew_class_launcher_slurm$script()

Method validate(): Validate the launcher.

Usage:

```
crew_class_launcher_slurm$validate()
```

Returns: NULL (invisibly). Throws an error if a field is invalid.

Method script(): Generate the job script.

Usage:

```
crew_class_launcher_slurm$script(name)
```

Arguments:

name Character of length 1, name of the job. For inspection purposes, you can supply a mock job name.

Details: Includes everything except the worker-instance-specific job name and the worker-instance-specific call to crew::crew_worker(), both of which get inserted at the bottom of the script at launch time.

Returns: Character vector of the lines of the job script.

Examples:

```
if (identical(Sys.getenv("CREW_EXAMPLES"), "true")) {
launcher <- crew_launcher_slurm(
  slurm_log_output = "log_file_%A.log",
  slurm_log_error = NULL,
  slurm_memory_gigabytes_per_cpu = 4096
)
launcher$script(name = "my_job_name")
}
```

See Also

Other slurm: crew_class_monitor_slurm, crew_controller_slurm(), crew_launcher_slurm(), crew_monitor_slurm(), crew_options_slurm()

Examples

```
## ------------------------------------------------
## Method `crew_class_launcher_slurm$script`
## ------------------------------------------------

if (identical(Sys.getenv("CREW_EXAMPLES"), "true")) {
launcher <- crew_launcher_slurm(
  slurm_log_output = "log_file_%A.log",
  slurm_log_error = NULL,
  slurm_memory_gigabytes_per_cpu = 4096
)
launcher$script(name = "my_job_name")
}
```

---

crew_class_monitor_sge

[Experimental] *SGE monitor class*

Description

SGE monitor R6 class

Details

See crew_monitor_sge().

Super class

crew.cluster::crew_class_monitor_cluster -> crew_class_monitor_sge

Methods

Public methods:

- crew_class_monitor_sge$jobs()
- crew_class_monitor_sge$terminate()

Method `jobs()`**:** List SGE jobs.

Usage:

```
crew_class_monitor_sge$jobs(user = ps::ps_username())
```

Arguments:

`user` Character of length 1, user name of the jobs to list.

Returns: A `tibble` with one row per SGE job and columns with specific details.

Method `terminate()`**:** Terminate one or more SGE jobs.

Usage:

```
crew_class_monitor_sge$terminate(jobs = NULL, all = FALSE)
```

Arguments:

`jobs` Character vector of job names or job IDs to terminate. Ignored if `all` is set to `TRUE`.

`all` Logical of length 1, whether to terminate all the jobs under your user name. This terminates ALL your SGE jobs, regardless of whether `crew.cluster` launched them, so use with caution!

Returns: `NULL` (invisibly).

See Also

Other sge: crew_class_launcher_sge, crew_controller_sge(), crew_launcher_sge(), crew_monitor_sge(), crew_options_sge()

`crew_class_monitor_slurm`

[Experimental] *SLURM monitor class*

Description

SLURM monitor `R6` class

Details

See crew_monitor_slurm().

Super class

crew.cluster::crew_class_monitor_cluster -> `crew_class_monitor_slurm`

Methods

Public methods:

- `crew_class_monitor_slurm$jobs()`
- `crew_class_monitor_slurm$terminate()`

Method `jobs()`**:** List SLURM jobs.

Usage:

```
crew_class_monitor_slurm$jobs(user = ps::ps_username())
```

Arguments:

`user` Character of length 1, user name of the jobs to list.

Details: This function loads the entire SLURM queue for all users, so it may take several seconds to execute. It is intended for interactive use, and should especially be avoided in scripts where it is called frequently. It requires SLURM version 20.02 or higher, along with the YAML plugin.

Returns: A `tibble` with one row per SLURM job and columns with specific details.

Method `terminate()`**:** Terminate one or more SLURM jobs.

Usage:

```
crew_class_monitor_slurm$terminate(jobs = NULL, all = FALSE)
```

Arguments:

`jobs` Character vector of job names or job IDs to terminate. Ignored if `all` is set to `TRUE`.

`all` Logical of length 1, whether to terminate all the jobs under your user name. This terminates ALL your SLURM jobs, regardless of whether `crew.cluster` launched them, so use with caution!

Returns: `NULL` (invisibly).

See Also

Other slurm: `crew_class_launcher_slurm`, `crew_controller_slurm()`, `crew_launcher_slurm()`, `crew_monitor_slurm()`, `crew_options_slurm()`

`crew_controller_lsf` **[Experimental]** *Create a controller with a LSF launcher.*

Description

Create an `R6` object to submit tasks and launch workers on LSF workers.

Usage

```
crew_controller_lsf(
  name = NULL,
  workers = 1L,
  host = NULL,
  port = NULL,
  tls = crew::crew_tls(mode = "automatic"),
  tls_enable = NULL,
  tls_config = NULL,
  serialization = NULL,
  seconds_interval = 0.25,
  seconds_timeout = 60,
  seconds_launch = 86400,
  seconds_idle = 300,
  seconds_wall = Inf,
  seconds_exit = NULL,
  retry_tasks = NULL,
  tasks_max = Inf,
  tasks_timers = 0L,
  reset_globals = TRUE,
  reset_packages = FALSE,
  reset_options = FALSE,
  garbage_collection = FALSE,
  crashes_error = NULL,
  r_arguments = c("--no-save", "--no-restore"),
  crashes_max = 5L,
  backup = NULL,
  options_metrics = crew::crew_options_metrics(),
  options_cluster = crew.cluster::crew_options_lsf(),
  verbose = NULL,
  command_submit = NULL,
  command_terminate = NULL,
  command_delete = NULL,
  script_directory = NULL,
  script_lines = NULL,
  lsf_cwd = NULL,
  lsf_log_output = NULL,
  lsf_log_error = NULL,
  lsf_memory_gigabytes_limit = NULL,
  lsf_memory_gigabytes_required = NULL,
  lsf_cores = NULL
)
```

Arguments

name	Character string, name of the launcher. If the name is NULL, then a name is automatically generated when the launcher starts.
workers	Maximum number of workers to run concurrently when auto-scaling, excluding

task retries and manual calls to `launch()`. Special workers allocated for task retries do not count towards this limit, so the number of workers running at a given time may exceed this maximum. A smaller number of workers may run if the number of executing tasks is smaller than the supplied value of the `workers` argument.

`host` IP address of the `mirai` client to send and receive tasks. If `NULL`, the host defaults to `nanonext::ip_addr()[1]`.

`port` TCP port to listen for the workers. If `NULL`, then an available ephemeral port is automatically chosen. Controllers running simultaneously on the same computer (as in a controller group) must not share the same TCP port.

`tls` A TLS configuration object from `crew_tls()`.

`tls_enable` Deprecated on 2023-09-15 in version 0.4.1. Use argument `tls` instead.

`tls_config` Deprecated on 2023-09-15 in version 0.4.1. Use argument `tls` instead.

`serialization` Either `NULL` (default) or an object produced by `mirai::serial_config()` to control the serialization of data sent to workers. This can help with either more efficient data transfers or to preserve attributes of otherwise non-exportable objects (such as `torch` tensors or `arrow` tables). See `?mirai::serial_config` for details.

`seconds_interval`
Number of seconds between polling intervals waiting for certain internal synchronous operations to complete. In certain cases, exponential backoff is used with this argument passed to `seconds_max` in a `crew_throttle()` object.

`seconds_timeout`
Number of seconds until timing out while waiting for certain synchronous operations to complete, such as checking `mirai::status()`.

`seconds_launch` Seconds of startup time to allow. A worker is unconditionally assumed to be alive from the moment of its launch until `seconds_launch` seconds later. After `seconds_launch` seconds, the worker is only considered alive if it is actively connected to its assign websocket.

`seconds_idle` Maximum number of seconds that a worker can idle since the completion of the last task. If exceeded, the worker exits. But the timer does not launch until `tasks_timers` tasks have completed. See the `idletime` argument of `mirai::daemon()`. `crew` does not excel with perfectly transient workers because it does not micromanage the assignment of tasks to workers, so please allow enough idle time for a new worker to be delegated a new task.

`seconds_wall` Soft wall time in seconds. The timer does not launch until `tasks_timers` tasks have completed. See the `walltime` argument of `mirai::daemon()`.

`seconds_exit` Deprecated on 2023-09-21 in version 0.1.2.9000. No longer necessary.

`retry_tasks` Deprecated on 2025-01-13 (`crew` version 0.10.2.9002).

`tasks_max` Maximum number of tasks that a worker will do before exiting. See the `maxtasks` argument of `mirai::daemon()`. `crew` does not excel with perfectly transient workers because it does not micromanage the assignment of tasks to workers, it is recommended to set `tasks_max` to a value greater than 1.

`tasks_timers` Number of tasks to do before activating the timers for `seconds_idle` and `seconds_wall`. See the `timerstart` argument of `mirai::daemon()`.

`reset_globals` TRUE to reset global environment variables between tasks, FALSE to leave them alone.

`reset_packages` TRUE to detach any packages loaded during a task (runs between each task), FALSE to leave packages alone. In either case, the namespaces are not detached.

`reset_options` TRUE to reset global options to their original state between each task, FALSE otherwise. It is recommended to only set `reset_options = TRUE` if `reset_packages` is also TRUE because packages sometimes rely on options they set at loading time. for this and other reasons, `reset_options` only resets options that were nonempty at the beginning of the task. If your task sets an entirely new option not already in `options()`, then `reset_options = TRUE` does not delete the option.

`garbage_collection`
TRUE to run garbage collection after each task task, FALSE to skip.

`crashes_error` Deprecated on 2025-01-13 (crew version 0.10.2.9002).

`r_arguments` Optional character vector of command line arguments to pass to `Rscript` (non-Windows) or `Rscript.exe` (Windows) when starting a worker. Example: `r_arguments = c("--vanilla", "--max-connections=32")`.

`crashes_max` In rare cases, a worker may exit unexpectedly before it completes its current task. If this happens, `pop()` returns a status of `"crash"` instead of `"error"` for the task. The controller does not automatically retry the task, but you can retry it manually by calling `push()` again and using the same task name as before. (However, `targets` pipelines running `crew` do automatically retry tasks whose workers crashed.)

`crashes_max` is a non-negative integer, and it sets the maximum number of allowable consecutive crashes for a given task. If a task's worker crashes more than `crashes_max` times in a row, then `pop()` throws an error when it tries to return the results of the task.

`backup` An optional `crew` controller object, or NULL to omit. If supplied, the `backup` controller runs any pushed tasks that have already reached `crashes_max` consecutive crashes. Using `backup`, you can create a chain of controllers with different levels of resources (such as worker memory and CPUs) so that a task that fails on one controller can retry using incrementally more powerful workers. All controllers in a backup chain should be part of the same controller group (see `crew_controller_group()`) so you can call the group-level `pop()` and `collect()` methods to make sure you get results regardless of which controller actually ended up running the task.

Limitations of `backup`: * `crashes_max` needs to be positive in order for `backup` to be used. Otherwise, every task would always skip the current controller and go to `backup`. * `backup` cannot be a controller group. It must be an ordinary controller.

`options_metrics`
Either NULL to opt out of resource metric logging for workers, or an object from `crew_options_metrics()` to enable and configure resource metric logging for workers. For resource logging to run, the `autometric` R package version 0.1.0 or higher must be installed.

`options_cluster`
: An options list from `crew_options_lsf()` with cluster-specific configuration options.

`verbose`
: Deprecated. Use `options_cluster` instead.

`command_submit`
: Deprecated. Use `options_cluster` instead.

`command_terminate`
: Deprecated. Use `options_cluster` instead.

`command_delete`
: Deprecated on 2024-01-08 (version 0.1.4.9001). Use `command_terminate` instead.

`script_directory`
: Deprecated. Use `options_cluster` instead.

`script_lines`
: Deprecated. Use `options_cluster` instead.

`lsf_cwd`
: Deprecated. Use `options_cluster` instead.

`lsf_log_output`
: Deprecated. Use `options_cluster` instead.

`lsf_log_error`
: Deprecated. Use `options_cluster` instead.

`lsf_memory_gigabytes_limit`
: Deprecated. Use `options_cluster` instead.

`lsf_memory_gigabytes_required`
: Deprecated. Use `options_cluster` instead.

`lsf_cores`
: Deprecated. Use `options_cluster` instead.

Details

WARNING: the `crew.cluster` LSF plugin is experimental and has not actually been tested on a LSF cluster. Please proceed with caution and report bugs to https://github.com/wlandau/crew.cluster.

Attribution

The template files at https://github.com/mschubert/clustermq/tree/master/inst informed the development of the `crew` launcher plugins in `crew.cluster`, and we would like to thank Michael Schubert for developing `clustermq` and releasing it under the permissive Apache License 2.0. See the `NOTICE` and `README.md` files in the `crew.cluster` source code for additional attribution.

See Also

Other lsf: `crew_class_launcher_lsf`, `crew_launcher_lsf()`, `crew_options_lsf()`

Examples

```
if (identical(Sys.getenv("CREW_EXAMPLES"), "true")) {
controller <- crew_controller_lsf()
controller$start()
controller$push(name = "task", command = sqrt(4))
controller$wait()
controller$pop()$result
```

```
controller$terminate()
}
```

`crew_controller_pbs` **[Experimental]** *Create a controller with a PBS/TORQUE launcher.*

Description

Create an `R6` object to submit tasks and launch workers on a PBS or TORQUE cluster.

Usage

```
crew_controller_pbs(
  name = NULL,
  workers = 1L,
  host = NULL,
  port = NULL,
  tls = crew::crew_tls(mode = "automatic"),
  tls_enable = NULL,
  tls_config = NULL,
  serialization = NULL,
  seconds_interval = 0.25,
  seconds_timeout = 60,
  seconds_launch = 86400,
  seconds_idle = 300,
  seconds_wall = Inf,
  seconds_exit = NULL,
  retry_tasks = NULL,
  tasks_max = Inf,
  tasks_timers = 0L,
  reset_globals = TRUE,
  reset_packages = FALSE,
  reset_options = FALSE,
  garbage_collection = FALSE,
  crashes_error = NULL,
  r_arguments = c("--no-save", "--no-restore"),
  crashes_max = 5L,
  backup = NULL,
  options_metrics = crew::crew_options_metrics(),
  options_cluster = crew.cluster::crew_options_pbs(),
  verbose = NULL,
  command_submit = NULL,
  command_terminate = NULL,
  command_delete = NULL,
  script_directory = NULL,
  script_lines = NULL,
  pbs_cwd = NULL,
```

```
  pbs_log_output = NULL,
  pbs_log_error = NULL,
  pbs_log_join = NULL,
  pbs_memory_gigabytes_required = NULL,
  pbs_cores = NULL,
  pbs_walltime_hours = NULL
)
```

Arguments

name	Character string, name of the launcher. If the name is NULL, then a name is automatically generated when the launcher starts.
workers	Maximum number of workers to run concurrently when auto-scaling, excluding task retries and manual calls to launch(). Special workers allocated for task retries do not count towards this limit, so the number of workers running at a given time may exceed this maximum. A smaller number of workers may run if the number of executing tasks is smaller than the supplied value of the workers argument.
host	IP address of the mirai client to send and receive tasks. If NULL, the host defaults to nanonext::ip_addr()[1].
port	TCP port to listen for the workers. If NULL, then an available ephemeral port is automatically chosen. Controllers running simultaneously on the same computer (as in a controller group) must not share the same TCP port.
tls	A TLS configuration object from crew_tls().
tls_enable	Deprecated on 2023-09-15 in version 0.4.1. Use argument tls instead.
tls_config	Deprecated on 2023-09-15 in version 0.4.1. Use argument tls instead.
serialization	Either NULL (default) or an object produced by mirai::serial_config() to control the serialization of data sent to workers. This can help with either more efficient data transfers or to preserve attributes of otherwise non-exportable objects (such as torch tensors or arrow tables). See ?mirai::serial_config for details.
seconds_interval	Number of seconds between polling intervals waiting for certain internal synchronous operations to complete. In certain cases, exponential backoff is used with this argument passed to seconds_max in a crew_throttle() object.
seconds_timeout	Number of seconds until timing out while waiting for certain synchronous operations to complete, such as checking mirai::status().
seconds_launch	Seconds of startup time to allow. A worker is unconditionally assumed to be alive from the moment of its launch until seconds_launch seconds later. After seconds_launch seconds, the worker is only considered alive if it is actively connected to its assign websocket.
seconds_idle	Maximum number of seconds that a worker can idle since the completion of the last task. If exceeded, the worker exits. But the timer does not launch until tasks_timers tasks have completed. See the idletime argument of mirai::daemon().

crew does not excel with perfectly transient workers because it does not micromanage the assignment of tasks to workers, so please allow enough idle time for a new worker to be delegated a new task.

seconds_wall
: Soft wall time in seconds. The timer does not launch until tasks_timers tasks have completed. See the walltime argument of mirai::daemon().

seconds_exit
: Deprecated on 2023-09-21 in version 0.1.2.9000. No longer necessary.

retry_tasks
: Deprecated on 2025-01-13 (crew version 0.10.2.9002).

tasks_max
: Maximum number of tasks that a worker will do before exiting. See the maxtasks argument of mirai::daemon(). crew does not excel with perfectly transient workers because it does not micromanage the assignment of tasks to workers, it is recommended to set tasks_max to a value greater than 1.

tasks_timers
: Number of tasks to do before activating the timers for seconds_idle and seconds_wall. See the timerstart argument of mirai::daemon().

reset_globals
: TRUE to reset global environment variables between tasks, FALSE to leave them alone.

reset_packages
: TRUE to detach any packages loaded during a task (runs between each task), FALSE to leave packages alone. In either case, the namespaces are not detached.

reset_options
: TRUE to reset global options to their original state between each task, FALSE otherwise. It is recommended to only set reset_options = TRUE if reset_packages is also TRUE because packages sometimes rely on options they set at loading time. for this and other reasons, reset_options only resets options that were nonempty at the beginning of the task. If your task sets an entirely new option not already in options(), then reset_options = TRUE does not delete the option.

garbage_collection
: TRUE to run garbage collection after each task task, FALSE to skip.

crashes_error
: Deprecated on 2025-01-13 (crew version 0.10.2.9002).

r_arguments
: Optional character vector of command line arguments to pass to Rscript (non-Windows) or Rscript.exe (Windows) when starting a worker. Example: r_arguments = c("--vanilla", "--max-connections=32").

crashes_max
: In rare cases, a worker may exit unexpectedly before it completes its current task. If this happens, pop() returns a status of "crash" instead of "error" for the task. The controller does not automatically retry the task, but you can retry it manually by calling push() again and using the same task name as before. (However, targets pipelines running crew do automatically retry tasks whose workers crashed.)

  crashes_max is a non-negative integer, and it sets the maximum number of allowable consecutive crashes for a given task. If a task's worker crashes more than crashes_max times in a row, then pop() throws an error when it tries to return the results of the task.

backup
: An optional crew controller object, or NULL to omit. If supplied, the backup controller runs any pushed tasks that have already reached crashes_max consecutive crashes. Using backup, you can create a chain of controllers with different levels of resources (such as worker memory and CPUs) so that a task that fails on one controller can retry using incrementally more powerful workers.

All controllers in a backup chain should be part of the same controller group (see crew_controller_group()) so you can call the group-level pop() and collect() methods to make sure you get results regardless of which controller actually ended up running the task.

Limitations of backup: * crashes_max needs to be positive in order for backup to be used. Otherwise, every task would always skip the current controller and go to backup. * backup cannot be a controller group. It must be an ordinary controller.

options_metrics
: Either NULL to opt out of resource metric logging for workers, or an object from crew_options_metrics() to enable and configure resource metric logging for workers. For resource logging to run, the autometric R package version 0.1.0 or higher must be installed.

options_cluster
: An options list from crew_options_pbs() with cluster-specific configuration options.

verbose
: Deprecated. Use options_cluster instead.

command_submit
: Deprecated. Use options_cluster instead.

command_terminate
: Deprecated. Use options_cluster instead.

command_delete
: Deprecated on 2024-01-08 (version 0.1.4.9001). Use command_terminate instead.

script_directory
: Deprecated. Use options_cluster instead.

script_lines
: Deprecated. Use options_cluster instead.

pbs_cwd
: Deprecated. Use options_cluster instead.

pbs_log_output
: Deprecated. Use options_cluster instead.

pbs_log_error
: Deprecated. Use options_cluster instead.

pbs_log_join
: Deprecated. Use options_cluster instead.

pbs_memory_gigabytes_required
: Deprecated. Use options_cluster instead.

pbs_cores
: Deprecated. Use options_cluster instead.

pbs_walltime_hours
: Deprecated. Use options_cluster instead.

Attribution

The template files at https://github.com/mschubert/clustermq/tree/master/inst informed the development of the crew launcher plugins in crew.cluster, and we would like to thank Michael Schubert for developing clustermq and releasing it under the permissive Apache License 2.0. See the NOTICE and README.md files in the crew.cluster source code for additional attribution.

See Also

Other pbs: crew_class_launcher_pbs, crew_launcher_pbs(), crew_options_pbs()

Examples

```
if (identical(Sys.getenv("CREW_EXAMPLES"), "true")) {
controller <- crew_controller_pbs()
controller$start()
controller$push(name = "task", command = sqrt(4))
controller$wait()
controller$pop()$result
controller$terminate()
}
```

`crew_controller_sge` **[Maturing]** *Create a controller with a Sun Grid Engine (SGE) launcher.*

Description

Create an R6 object to submit tasks and launch workers on Sun Grid Engine (SGE) workers.

Usage

```
crew_controller_sge(
  name = NULL,
  workers = 1L,
  host = NULL,
  port = NULL,
  tls = crew::crew_tls(mode = "automatic"),
  tls_enable = NULL,
  tls_config = NULL,
  serialization = NULL,
  seconds_interval = 0.25,
  seconds_timeout = 60,
  seconds_launch = 86400,
  seconds_idle = 300,
  seconds_wall = Inf,
  seconds_exit = NULL,
  retry_tasks = NULL,
  tasks_max = Inf,
  tasks_timers = 0L,
  reset_globals = TRUE,
  reset_packages = FALSE,
  reset_options = FALSE,
  garbage_collection = FALSE,
  crashes_error = NULL,
  r_arguments = c("--no-save", "--no-restore"),
  crashes_max = 5L,
  backup = NULL,
  options_metrics = crew::crew_options_metrics(),
```

```
  options_cluster = crew.cluster::crew_options_sge(),
  verbose = NULL,
  command_submit = NULL,
  command_terminate = NULL,
  command_delete = NULL,
  script_directory = NULL,
  script_lines = NULL,
  sge_cwd = NULL,
  sge_envvars = NULL,
  sge_log_output = NULL,
  sge_log_error = NULL,
  sge_log_join = NULL,
  sge_memory_gigabytes_limit = NULL,
  sge_memory_gigabytes_required = NULL,
  sge_cores = NULL,
  sge_gpu = NULL
)
```

Arguments

name	Character string, name of the launcher. If the name is NULL, then a name is automatically generated when the launcher starts.
workers	Maximum number of workers to run concurrently when auto-scaling, excluding task retries and manual calls to launch(). Special workers allocated for task retries do not count towards this limit, so the number of workers running at a given time may exceed this maximum. A smaller number of workers may run if the number of executing tasks is smaller than the supplied value of the workers argument.
host	IP address of the mirai client to send and receive tasks. If NULL, the host defaults to nanonext::ip_addr()[1].
port	TCP port to listen for the workers. If NULL, then an available ephemeral port is automatically chosen. Controllers running simultaneously on the same computer (as in a controller group) must not share the same TCP port.
tls	A TLS configuration object from crew_tls().
tls_enable	Deprecated on 2023-09-15 in version 0.4.1. Use argument tls instead.
tls_config	Deprecated on 2023-09-15 in version 0.4.1. Use argument tls instead.
serialization	Either NULL (default) or an object produced by mirai::serial_config() to control the serialization of data sent to workers. This can help with either more efficient data transfers or to preserve attributes of otherwise non-exportable objects (such as torch tensors or arrow tables). See ?mirai::serial_config for details.
seconds_interval	Number of seconds between polling intervals waiting for certain internal synchronous operations to complete. In certain cases, exponential backoff is used with this argument passed to seconds_max in a crew_throttle() object.

seconds_timeout
: Number of seconds until timing out while waiting for certain synchronous operations to complete, such as checking `mirai::status()`.

seconds_launch
: Seconds of startup time to allow. A worker is unconditionally assumed to be alive from the moment of its launch until `seconds_launch` seconds later. After `seconds_launch` seconds, the worker is only considered alive if it is actively connected to its assign websocket.

seconds_idle
: Maximum number of seconds that a worker can idle since the completion of the last task. If exceeded, the worker exits. But the timer does not launch until `tasks_timers` tasks have completed. See the `idletime` argument of `mirai::daemon()`. `crew` does not excel with perfectly transient workers because it does not micromanage the assignment of tasks to workers, so please allow enough idle time for a new worker to be delegated a new task.

seconds_wall
: Soft wall time in seconds. The timer does not launch until `tasks_timers` tasks have completed. See the `walltime` argument of `mirai::daemon()`.

seconds_exit
: Deprecated on 2023-09-21 in version 0.1.2.9000. No longer necessary.

retry_tasks
: Deprecated on 2025-01-13 (`crew` version 0.10.2.9002).

tasks_max
: Maximum number of tasks that a worker will do before exiting. See the `maxtasks` argument of `mirai::daemon()`. `crew` does not excel with perfectly transient workers because it does not micromanage the assignment of tasks to workers, it is recommended to set `tasks_max` to a value greater than 1.

tasks_timers
: Number of tasks to do before activating the timers for `seconds_idle` and `seconds_wall`. See the `timerstart` argument of `mirai::daemon()`.

reset_globals
: `TRUE` to reset global environment variables between tasks, `FALSE` to leave them alone.

reset_packages
: `TRUE` to detach any packages loaded during a task (runs between each task), `FALSE` to leave packages alone. In either case, the namespaces are not detached.

reset_options
: `TRUE` to reset global options to their original state between each task, `FALSE` otherwise. It is recommended to only set `reset_options = TRUE` if `reset_packages` is also `TRUE` because packages sometimes rely on options they set at loading time. for this and other reasons, `reset_options` only resets options that were nonempty at the beginning of the task. If your task sets an entirely new option not already in `options()`, then `reset_options = TRUE` does not delete the option.

garbage_collection
: `TRUE` to run garbage collection after each task task, `FALSE` to skip.

crashes_error
: Deprecated on 2025-01-13 (`crew` version 0.10.2.9002).

r_arguments
: Optional character vector of command line arguments to pass to `Rscript` (non-Windows) or `Rscript.exe` (Windows) when starting a worker. Example: `r_arguments = c("--vanilla", "--max-connections=32")`.

crashes_max
: In rare cases, a worker may exit unexpectedly before it completes its current task. If this happens, `pop()` returns a status of `"crash"` instead of `"error"` for the task. The controller does not automatically retry the task, but you can retry it manually by calling `push()` again and using the same task name as before.

(However, targets pipelines running crew do automatically retry tasks whose workers crashed.)

crashes_max is a non-negative integer, and it sets the maximum number of allowable consecutive crashes for a given task. If a task's worker crashes more than crashes_max times in a row, then pop() throws an error when it tries to return the results of the task.

backup An optional crew controller object, or NULL to omit. If supplied, the backup controller runs any pushed tasks that have already reached crashes_max consecutive crashes. Using backup, you can create a chain of controllers with different levels of resources (such as worker memory and CPUs) so that a task that fails on one controller can retry using incrementally more powerful workers. All controllers in a backup chain should be part of the same controller group (see crew_controller_group()) so you can call the group-level pop() and collect() methods to make sure you get results regardless of which controller actually ended up running the task.

Limitations of backup: * crashes_max needs to be positive in order for backup to be used. Otherwise, every task would always skip the current controller and go to backup. * backup cannot be a controller group. It must be an ordinary controller.

options_metrics

Either NULL to opt out of resource metric logging for workers, or an object from crew_options_metrics() to enable and configure resource metric logging for workers. For resource logging to run, the autometric R package version 0.1.0 or higher must be installed.

options_cluster

An options list from crew_options_sge() with cluster-specific configuration options.

verbose Deprecated. Use options_cluster instead.

command_submit Deprecated. Use options_cluster instead.

command_terminate

Deprecated. Use options_cluster instead.

command_delete Deprecated on 2024-01-08 (version 0.1.4.9001). Use command_terminate instead.

script_directory

Deprecated. Use options_cluster instead.

script_lines Deprecated. Use options_cluster instead.

sge_cwd Deprecated. Use options_cluster instead.

sge_envvars Deprecated. Use options_cluster instead.

sge_log_output Deprecated. Use options_cluster instead.

sge_log_error Deprecated. Use options_cluster instead.

sge_log_join Deprecated. Use options_cluster instead.

sge_memory_gigabytes_limit

Deprecated. Use options_cluster instead.

sge_memory_gigabytes_required

Deprecated. Use options_cluster instead.

sge_cores	Deprecated. Use options_cluster instead.
sge_gpu	Deprecated. Use options_cluster instead.

Attribution

The template files at https://github.com/mschubert/clustermq/tree/master/inst informed the development of the crew launcher plugins in crew.cluster, and we would like to thank Michael Schubert for developing clustermq and releasing it under the permissive Apache License 2.0. See the NOTICE and README.md files in the crew.cluster source code for additional attribution.

See Also

Other sge: crew_class_launcher_sge, crew_class_monitor_sge, crew_launcher_sge(), crew_monitor_sge(), crew_options_sge()

Examples

```
if (identical(Sys.getenv("CREW_EXAMPLES"), "true")) {
controller <- crew_controller_sge()
controller$start()
controller$push(name = "task", command = sqrt(4))
controller$wait()
controller$pop()$result
controller$terminate()
}
```

---

crew_controller_slurm **[Experimental]** *Create a controller with a SLURM launcher.*

Description

Create an R6 object to submit tasks and launch workers on SLURM workers.

Usage

```
crew_controller_slurm(
  name = NULL,
  workers = 1L,
  host = NULL,
  port = NULL,
  tls = crew::crew_tls(mode = "automatic"),
  tls_enable = NULL,
  tls_config = NULL,
  serialization = NULL,
  seconds_interval = 0.25,
  seconds_timeout = 60,
  seconds_launch = 86400,
```

```
  seconds_idle = 300,
  seconds_wall = Inf,
  seconds_exit = NULL,
  retry_tasks = NULL,
  tasks_max = Inf,
  tasks_timers = 0L,
  reset_globals = TRUE,
  reset_packages = FALSE,
  reset_options = FALSE,
  garbage_collection = FALSE,
  crashes_error = NULL,
  r_arguments = c("--no-save", "--no-restore"),
  crashes_max = 5L,
  backup = NULL,
  options_metrics = crew::crew_options_metrics(),
  options_cluster = crew.cluster::crew_options_slurm(),
  verbose = NULL,
  command_submit = NULL,
  command_terminate = NULL,
  command_delete = NULL,
  script_directory = NULL,
  script_lines = NULL,
  slurm_log_output = NULL,
  slurm_log_error = NULL,
  slurm_memory_gigabytes_required = NULL,
  slurm_memory_gigabytes_per_cpu = NULL,
  slurm_cpus_per_task = NULL,
  slurm_time_minutes = NULL,
  slurm_partition = NULL
)
```

Arguments

`name`	Character string, name of the launcher. If the name is `NULL`, then a name is automatically generated when the launcher starts.
`workers`	Maximum number of workers to run concurrently when auto-scaling, excluding task retries and manual calls to `launch()`. Special workers allocated for task retries do not count towards this limit, so the number of workers running at a given time may exceed this maximum. A smaller number of workers may run if the number of executing tasks is smaller than the supplied value of the `workers` argument.
`host`	IP address of the `mirai` client to send and receive tasks. If `NULL`, the host defaults to `nanonext::ip_addr()[1]`.
`port`	TCP port to listen for the workers. If `NULL`, then an available ephemeral port is automatically chosen. Controllers running simultaneously on the same computer (as in a controller group) must not share the same TCP port.
`tls`	A TLS configuration object from `crew_tls()`.

`tls_enable` Deprecated on 2023-09-15 in version 0.4.1. Use argument `tls` instead.

`tls_config` Deprecated on 2023-09-15 in version 0.4.1. Use argument `tls` instead.

`serialization` Either `NULL` (default) or an object produced by `mirai::serial_config()` to control the serialization of data sent to workers. This can help with either more efficient data transfers or to preserve attributes of otherwise non-exportable objects (such as `torch` tensors or `arrow` tables). See `?mirai::serial_config` for details.

`seconds_interval` Number of seconds between polling intervals waiting for certain internal synchronous operations to complete. In certain cases, exponential backoff is used with this argument passed to `seconds_max` in a `crew_throttle()` object.

`seconds_timeout` Number of seconds until timing out while waiting for certain synchronous operations to complete, such as checking `mirai::status()`.

`seconds_launch` Seconds of startup time to allow. A worker is unconditionally assumed to be alive from the moment of its launch until `seconds_launch` seconds later. After `seconds_launch` seconds, the worker is only considered alive if it is actively connected to its assign websocket.

`seconds_idle` Maximum number of seconds that a worker can idle since the completion of the last task. If exceeded, the worker exits. But the timer does not launch until `tasks_timers` tasks have completed. See the `idletime` argument of `mirai::daemon()`. `crew` does not excel with perfectly transient workers because it does not micromanage the assignment of tasks to workers, so please allow enough idle time for a new worker to be delegated a new task.

`seconds_wall` Soft wall time in seconds. The timer does not launch until `tasks_timers` tasks have completed. See the `walltime` argument of `mirai::daemon()`.

`seconds_exit` Deprecated on 2023-09-21 in version 0.1.2.9000. No longer necessary.

`retry_tasks` Deprecated on 2025-01-13 (`crew` version 0.10.2.9002).

`tasks_max` Maximum number of tasks that a worker will do before exiting. See the `maxtasks` argument of `mirai::daemon()`. `crew` does not excel with perfectly transient workers because it does not micromanage the assignment of tasks to workers, it is recommended to set `tasks_max` to a value greater than 1.

`tasks_timers` Number of tasks to do before activating the timers for `seconds_idle` and `seconds_wall`. See the `timerstart` argument of `mirai::daemon()`.

`reset_globals` `TRUE` to reset global environment variables between tasks, `FALSE` to leave them alone.

`reset_packages` `TRUE` to detach any packages loaded during a task (runs between each task), `FALSE` to leave packages alone. In either case, the namespaces are not detached.

`reset_options` `TRUE` to reset global options to their original state between each task, `FALSE` otherwise. It is recommended to only set `reset_options = TRUE` if `reset_packages` is also `TRUE` because packages sometimes rely on options they set at loading time. for this and other reasons, `reset_options` only resets options that were nonempty at the beginning of the task. If your task sets an entirely new option not already in `options()`, then `reset_options = TRUE` does not delete the option.

garbage_collection
: TRUE to run garbage collection after each task task, FALSE to skip.

crashes_error
: Deprecated on 2025-01-13 (crew version 0.10.2.9002).

r_arguments
: Optional character vector of command line arguments to pass to Rscript (non-Windows) or Rscript.exe (Windows) when starting a worker. Example: r_arguments = c("--vanilla", "--max-connections=32").

crashes_max
: In rare cases, a worker may exit unexpectedly before it completes its current task. If this happens, pop() returns a status of "crash" instead of "error" for the task. The controller does not automatically retry the task, but you can retry it manually by calling push() again and using the same task name as before. (However, targets pipelines running crew do automatically retry tasks whose workers crashed.)

  crashes_max is a non-negative integer, and it sets the maximum number of allowable consecutive crashes for a given task. If a task's worker crashes more than crashes_max times in a row, then pop() throws an error when it tries to return the results of the task.

backup
: An optional crew controller object, or NULL to omit. If supplied, the backup controller runs any pushed tasks that have already reached crashes_max consecutive crashes. Using backup, you can create a chain of controllers with different levels of resources (such as worker memory and CPUs) so that a task that fails on one controller can retry using incrementally more powerful workers. All controllers in a backup chain should be part of the same controller group (see crew_controller_group()) so you can call the group-level pop() and collect() methods to make sure you get results regardless of which controller actually ended up running the task.

  Limitations of backup: * crashes_max needs to be positive in order for backup to be used. Otherwise, every task would always skip the current controller and go to backup. * backup cannot be a controller group. It must be an ordinary controller.

options_metrics
: Either NULL to opt out of resource metric logging for workers, or an object from crew_options_metrics() to enable and configure resource metric logging for workers. For resource logging to run, the autometric R package version 0.1.0 or higher must be installed.

options_cluster
: An options list from crew_options_slurm() with cluster-specific configuration options.

verbose
: Deprecated. Use options_cluster instead.

command_submit
: Deprecated. Use options_cluster instead.

command_terminate
: Deprecated. Use options_cluster instead.

command_delete
: Deprecated on 2024-01-08 (version 0.1.4.9001). Use command_terminate instead.

script_directory
: Deprecated. Use options_cluster instead.

script_lines
: Deprecated. Use options_cluster instead.

`slurm_log_output`
: Deprecated. Use `options_cluster` instead.

`slurm_log_error`
: Deprecated. Use `options_cluster` instead.

`slurm_memory_gigabytes_required`
: Deprecated. Use `options_cluster` instead.

`slurm_memory_gigabytes_per_cpu`
: Deprecated. Use `options_cluster` instead.

`slurm_cpus_per_task`
: Deprecated. Use `options_cluster` instead.

`slurm_time_minutes`
: Deprecated. Use `options_cluster` instead.

`slurm_partition`
: Deprecated. Use `options_cluster` instead.

Details

WARNING: the `crew.cluster` SLURM plugin is experimental and has not actually been tested on a SLURM cluster. Please proceed with caution and report bugs to https://github.com/wlandau/crew.cluster.

Attribution

The template files at https://github.com/mschubert/clustermq/tree/master/inst informed the development of the `crew` launcher plugins in `crew.cluster`, and we would like to thank Michael Schubert for developing `clustermq` and releasing it under the permissive Apache License 2.0. See the `NOTICE` and `README.md` files in the `crew.cluster` source code for additional attribution.

See Also

Other slurm: `crew_class_launcher_slurm`, `crew_class_monitor_slurm`, `crew_launcher_slurm()`, `crew_monitor_slurm()`, `crew_options_slurm()`

Examples

```
if (identical(Sys.getenv("CREW_EXAMPLES"), "true")) {
controller <- crew_controller_slurm()
controller$start()
controller$push(name = "task", command = sqrt(4))
controller$wait()
controller$pop()$result
controller$terminate()
}
```

crew_launcher_lsf **[Experimental]** *Create a launcher with LSF workers.*

Description

Create an R6 object to launch and maintain workers as LSF jobs.

Usage

```
crew_launcher_lsf(
  name = NULL,
  workers = 1L,
  seconds_interval = 0.5,
  seconds_timeout = 60,
  seconds_launch = 86400,
  seconds_idle = 300,
  seconds_wall = Inf,
  tasks_max = Inf,
  tasks_timers = 0L,
  reset_globals = NULL,
  reset_packages = NULL,
  reset_options = NULL,
  garbage_collection = NULL,
  crashes_error = NULL,
  tls = crew::crew_tls(mode = "automatic"),
  r_arguments = c("--no-save", "--no-restore"),
  options_metrics = crew::crew_options_metrics(),
  options_cluster = crew.cluster::crew_options_lsf(),
  verbose = NULL,
  command_submit = NULL,
  command_terminate = NULL,
  command_delete = NULL,
  script_directory = NULL,
  script_lines = NULL,
  lsf_cwd = NULL,
  lsf_log_output = NULL,
  lsf_log_error = NULL,
  lsf_memory_gigabytes_limit = NULL,
  lsf_memory_gigabytes_required = NULL,
  lsf_cores = NULL
)
```

Arguments

name	Character string, name of the launcher. If the name is NULL, then a name is automatically generated when the launcher starts.

`workers` Maximum number of workers to run concurrently when auto-scaling, excluding task retries and manual calls to `launch()`. Special workers allocated for task retries do not count towards this limit, so the number of workers running at a given time may exceed this maximum. A smaller number of workers may run if the number of executing tasks is smaller than the supplied value of the `workers` argument.

`seconds_interval` Number of seconds between polling intervals waiting for certain internal synchronous operations to complete. In certain cases, exponential backoff is used with this argument passed to `seconds_max` in a `crew_throttle()` object.

`seconds_timeout` Number of seconds until timing out while waiting for certain synchronous operations to complete, such as checking `mirai::status()`.

`seconds_launch` Seconds of startup time to allow. A worker is unconditionally assumed to be alive from the moment of its launch until `seconds_launch` seconds later. After `seconds_launch` seconds, the worker is only considered alive if it is actively connected to its assign websocket.

`seconds_idle` Maximum number of seconds that a worker can idle since the completion of the last task. If exceeded, the worker exits. But the timer does not launch until `tasks_timers` tasks have completed. See the `idletime` argument of `mirai::daemon()`. `crew` does not excel with perfectly transient workers because it does not micromanage the assignment of tasks to workers, so please allow enough idle time for a new worker to be delegated a new task.

`seconds_wall` Soft wall time in seconds. The timer does not launch until `tasks_timers` tasks have completed. See the `walltime` argument of `mirai::daemon()`.

`tasks_max` Maximum number of tasks that a worker will do before exiting. See the `maxtasks` argument of `mirai::daemon()`. `crew` does not excel with perfectly transient workers because it does not micromanage the assignment of tasks to workers, it is recommended to set `tasks_max` to a value greater than 1.

`tasks_timers` Number of tasks to do before activating the timers for `seconds_idle` and `seconds_wall`. See the `timerstart` argument of `mirai::daemon()`.

`reset_globals` Deprecated on 2025-05-30 (`crew` version 1.1.2.9004). Please use the `reset_globals` option of `crew_controller()` instead.

`reset_packages` Deprecated on 2025-05-30 (`crew` version 1.1.2.9004). Please use the `reset_packages` option of `crew_controller()` instead.

`reset_options` Deprecated on 2025-05-30 (`crew` version 1.1.2.9004). Please use the `reset_options` option of `crew_controller()` instead.

`garbage_collection` Deprecated on 2025-05-30 (`crew` version 1.1.2.9004). Please use the `garbage_collection` option of `crew_controller()` instead.

`crashes_error` Deprecated on 2025-01-13 (`crew` version 0.10.2.9002).

`tls` A TLS configuration object from `crew_tls()`.

`r_arguments` Optional character vector of command line arguments to pass to `Rscript` (non-Windows) or `Rscript.exe` (Windows) when starting a worker. Example: `r_arguments = c("--vanilla", "--max-connections=32")`.

options_metrics
: Either NULL to opt out of resource metric logging for workers, or an object from crew_options_metrics() to enable and configure resource metric logging for workers. For resource logging to run, the autometric R package version 0.1.0 or higher must be installed.

options_cluster
: An options list from crew_options_lsf() with cluster-specific configuration options.

verbose
: Deprecated. Use options_cluster instead.

command_submit
: Deprecated. Use options_cluster instead.

command_terminate
: Deprecated. Use options_cluster instead.

command_delete
: Deprecated on 2024-01-08 (version 0.1.4.9001). Use command_terminate instead.

script_directory
: Deprecated. Use options_cluster instead.

script_lines
: Deprecated. Use options_cluster instead.

lsf_cwd
: Deprecated. Use options_cluster instead.

lsf_log_output
: Deprecated. Use options_cluster instead.

lsf_log_error
: Deprecated. Use options_cluster instead.

lsf_memory_gigabytes_limit
: Deprecated. Use options_cluster instead.

lsf_memory_gigabytes_required
: Deprecated. Use options_cluster instead.

lsf_cores
: Deprecated. Use options_cluster instead.

Details

WARNING: the crew.cluster LSF plugin is experimental. Please proceed with caution and report bugs to https://github.com/wlandau/crew.cluster.

To launch a LSF worker, this launcher creates a temporary job script with a call to crew::crew_worker() and submits it as an LSF job with sbatch. To see most of the lines of the job script in advance, use the script() method of the launcher. It has all the lines except for the job name and the call to crew::crew_worker(), both of which will be inserted at the last minute when it is time to actually launch a worker.

Attribution

The template files at https://github.com/mschubert/clustermq/tree/master/inst informed the development of the crew launcher plugins in crew.cluster, and we would like to thank Michael Schubert for developing clustermq and releasing it under the permissive Apache License 2.0. See the NOTICE and README.md files in the crew.cluster source code for additional attribution.

See Also

Other lsf: crew_class_launcher_lsf, crew_controller_lsf(), crew_options_lsf()

crew_launcher_pbs **[Experimental]** *Create a launcher with PBS or TORQUE workers.*

Description

Create an R6 object to launch and maintain workers as jobs on a PBS or TORQUE cluster.

Usage

```
crew_launcher_pbs(
  name = NULL,
  workers = 1L,
  seconds_interval = 0.5,
  seconds_timeout = 60,
  seconds_launch = 86400,
  seconds_idle = 300,
  seconds_wall = Inf,
  tasks_max = Inf,
  tasks_timers = 0L,
  reset_globals = NULL,
  reset_packages = NULL,
  reset_options = NULL,
  garbage_collection = NULL,
  crashes_error = NULL,
  tls = crew::crew_tls(mode = "automatic"),
  r_arguments = c("--no-save", "--no-restore"),
  options_metrics = crew::crew_options_metrics(),
  options_cluster = crew.cluster::crew_options_pbs(),
  verbose = NULL,
  command_submit = NULL,
  command_terminate = NULL,
  command_delete = NULL,
  script_directory = NULL,
  script_lines = NULL,
  pbs_cwd = NULL,
  pbs_log_output = NULL,
  pbs_log_error = NULL,
  pbs_log_join = NULL,
  pbs_memory_gigabytes_required = NULL,
  pbs_cores = NULL,
  pbs_walltime_hours = NULL
)
```

Arguments

name	Character string, name of the launcher. If the name is NULL, then a name is automatically generated when the launcher starts.

`workers` Maximum number of workers to run concurrently when auto-scaling, excluding task retries and manual calls to `launch()`. Special workers allocated for task retries do not count towards this limit, so the number of workers running at a given time may exceed this maximum. A smaller number of workers may run if the number of executing tasks is smaller than the supplied value of the `workers` argument.

`seconds_interval` Number of seconds between polling intervals waiting for certain internal synchronous operations to complete. In certain cases, exponential backoff is used with this argument passed to `seconds_max` in a `crew_throttle()` object.

`seconds_timeout` Number of seconds until timing out while waiting for certain synchronous operations to complete, such as checking `mirai::status()`.

`seconds_launch` Seconds of startup time to allow. A worker is unconditionally assumed to be alive from the moment of its launch until `seconds_launch` seconds later. After `seconds_launch` seconds, the worker is only considered alive if it is actively connected to its assign websocket.

`seconds_idle` Maximum number of seconds that a worker can idle since the completion of the last task. If exceeded, the worker exits. But the timer does not launch until `tasks_timers` tasks have completed. See the `idletime` argument of `mirai::daemon()`. `crew` does not excel with perfectly transient workers because it does not micromanage the assignment of tasks to workers, so please allow enough idle time for a new worker to be delegated a new task.

`seconds_wall` Soft wall time in seconds. The timer does not launch until `tasks_timers` tasks have completed. See the `walltime` argument of `mirai::daemon()`.

`tasks_max` Maximum number of tasks that a worker will do before exiting. See the `maxtasks` argument of `mirai::daemon()`. `crew` does not excel with perfectly transient workers because it does not micromanage the assignment of tasks to workers, it is recommended to set `tasks_max` to a value greater than 1.

`tasks_timers` Number of tasks to do before activating the timers for `seconds_idle` and `seconds_wall`. See the `timerstart` argument of `mirai::daemon()`.

`reset_globals` Deprecated on 2025-05-30 (`crew` version 1.1.2.9004). Please use the `reset_globals` option of `crew_controller()` instead.

`reset_packages` Deprecated on 2025-05-30 (`crew` version 1.1.2.9004). Please use the `reset_packages` option of `crew_controller()` instead.

`reset_options` Deprecated on 2025-05-30 (`crew` version 1.1.2.9004). Please use the `reset_options` option of `crew_controller()` instead.

`garbage_collection` Deprecated on 2025-05-30 (`crew` version 1.1.2.9004). Please use the `garbage_collection` option of `crew_controller()` instead.

`crashes_error` Deprecated on 2025-01-13 (`crew` version 0.10.2.9002).

`tls` A TLS configuration object from `crew_tls()`.

`r_arguments` Optional character vector of command line arguments to pass to `Rscript` (non-Windows) or `Rscript.exe` (Windows) when starting a worker. Example: `r_arguments = c("--vanilla", "--max-connections=32")`.

options_metrics
: Either NULL to opt out of resource metric logging for workers, or an object from crew_options_metrics() to enable and configure resource metric logging for workers. For resource logging to run, the autometric R package version 0.1.0 or higher must be installed.

options_cluster
: An options list from crew_options_pbs() with cluster-specific configuration options.

verbose
: Deprecated. Use options_cluster instead.

command_submit
: Deprecated. Use options_cluster instead.

command_terminate
: Deprecated. Use options_cluster instead.

command_delete
: Deprecated on 2024-01-08 (version 0.1.4.9001). Use command_terminate instead.

script_directory
: Deprecated. Use options_cluster instead.

script_lines
: Deprecated. Use options_cluster instead.

pbs_cwd
: Deprecated. Use options_cluster instead.

pbs_log_output
: Deprecated. Use options_cluster instead.

pbs_log_error
: Deprecated. Use options_cluster instead.

pbs_log_join
: Deprecated. Use options_cluster instead.

pbs_memory_gigabytes_required
: Deprecated. Use options_cluster instead.

pbs_cores
: Deprecated. Use options_cluster instead.

pbs_walltime_hours
: Deprecated. Use options_cluster instead.

Details

WARNING: the crew.cluster PBS plugin is experimental and has not actually been tested on a PBS cluster. Please proceed with caution and report bugs to https://github.com/wlandau/crew.cluster.

To launch a PBS/TORQUE worker, this launcher creates a temporary job script with a call to crew::crew_worker() and submits it as an PBS job with qsub. To see most of the lines of the job script in advance, use the script() method of the launcher. It has all the lines except for the job name and the call to crew::crew_worker(), both of which will be inserted at the last minute when it is time to actually launch a worker.

Attribution

The template files at https://github.com/mschubert/clustermq/tree/master/inst informed the development of the crew launcher plugins in crew.cluster, and we would like to thank Michael Schubert for developing clustermq and releasing it under the permissive Apache License 2.0. See the NOTICE and README.md files in the crew.cluster source code for additional attribution.

See Also

Other pbs: crew_class_launcher_pbs, crew_controller_pbs(), crew_options_pbs()

---

crew_launcher_sge **[Maturing]** *Create a launcher with Sun Grid Engine (SGE) workers.*

Description

Create an R6 object to launch and maintain workers as Sun Grid Engine (SGE) jobs.

Usage

```
crew_launcher_sge(
  name = NULL,
  workers = 1L,
  seconds_interval = 0.5,
  seconds_timeout = 60,
  seconds_launch = 86400,
  seconds_idle = 300,
  seconds_wall = Inf,
  tasks_max = Inf,
  tasks_timers = 0L,
  reset_globals = NULL,
  reset_packages = NULL,
  reset_options = NULL,
  garbage_collection = NULL,
  crashes_error = NULL,
  tls = crew::crew_tls(mode = "automatic"),
  r_arguments = c("--no-save", "--no-restore"),
  options_metrics = crew::crew_options_metrics(),
  options_cluster = crew.cluster::crew_options_sge(),
  verbose = NULL,
  command_submit = NULL,
  command_terminate = NULL,
  command_delete = NULL,
  script_directory = NULL,
  script_lines = NULL,
  sge_cwd = NULL,
  sge_envvars = NULL,
  sge_log_output = NULL,
  sge_log_error = NULL,
  sge_log_join = NULL,
  sge_memory_gigabytes_limit = NULL,
  sge_memory_gigabytes_required = NULL,
  sge_cores = NULL,
  sge_gpu = NULL
)
```

Arguments

name	Character string, name of the launcher. If the name is NULL, then a name is automatically generated when the launcher starts.
workers	Maximum number of workers to run concurrently when auto-scaling, excluding task retries and manual calls to launch(). Special workers allocated for task retries do not count towards this limit, so the number of workers running at a given time may exceed this maximum. A smaller number of workers may run if the number of executing tasks is smaller than the supplied value of the workers argument.
seconds_interval	Number of seconds between polling intervals waiting for certain internal synchronous operations to complete. In certain cases, exponential backoff is used with this argument passed to seconds_max in a crew_throttle() object.
seconds_timeout	Number of seconds until timing out while waiting for certain synchronous operations to complete, such as checking mirai::status().
seconds_launch	Seconds of startup time to allow. A worker is unconditionally assumed to be alive from the moment of its launch until seconds_launch seconds later. After seconds_launch seconds, the worker is only considered alive if it is actively connected to its assign websocket.
seconds_idle	Maximum number of seconds that a worker can idle since the completion of the last task. If exceeded, the worker exits. But the timer does not launch until tasks_timers tasks have completed. See the idletime argument of mirai::daemon(). crew does not excel with perfectly transient workers because it does not micromanage the assignment of tasks to workers, so please allow enough idle time for a new worker to be delegated a new task.
seconds_wall	Soft wall time in seconds. The timer does not launch until tasks_timers tasks have completed. See the walltime argument of mirai::daemon().
tasks_max	Maximum number of tasks that a worker will do before exiting. See the maxtasks argument of mirai::daemon(). crew does not excel with perfectly transient workers because it does not micromanage the assignment of tasks to workers, it is recommended to set tasks_max to a value greater than 1.
tasks_timers	Number of tasks to do before activating the timers for seconds_idle and seconds_wall. See the timerstart argument of mirai::daemon().
reset_globals	Deprecated on 2025-05-30 (crew version 1.1.2.9004). Please use the reset_globals option of crew_controller() instead.
reset_packages	Deprecated on 2025-05-30 (crew version 1.1.2.9004). Please use the reset_packages option of crew_controller() instead.
reset_options	Deprecated on 2025-05-30 (crew version 1.1.2.9004). Please use the reset_options option of crew_controller() instead.
garbage_collection	Deprecated on 2025-05-30 (crew version 1.1.2.9004). Please use the garbage_collection option of crew_controller() instead.
crashes_error	Deprecated on 2025-01-13 (crew version 0.10.2.9002).

`tls` A TLS configuration object from `crew_tls()`.

`r_arguments` Optional character vector of command line arguments to pass to `Rscript` (non-Windows) or `Rscript.exe` (Windows) when starting a worker. Example: `r_arguments = c("--vanilla", "--max-connections=32")`.

`options_metrics`
Either `NULL` to opt out of resource metric logging for workers, or an object from `crew_options_metrics()` to enable and configure resource metric logging for workers. For resource logging to run, the `autometric` R package version 0.1.0 or higher must be installed.

`options_cluster`
An options list from `crew_options_sge()` with cluster-specific configuration options.

`verbose` Deprecated. Use `options_cluster` instead.

`command_submit` Deprecated. Use `options_cluster` instead.

`command_terminate`
Deprecated. Use `options_cluster` instead.

`command_delete` Deprecated on 2024-01-08 (version 0.1.4.9001). Use `command_terminate` instead.

`script_directory`
Deprecated. Use `options_cluster` instead.

`script_lines` Deprecated. Use `options_cluster` instead.

`sge_cwd` Deprecated. Use `options_cluster` instead.

`sge_envvars` Deprecated. Use `options_cluster` instead.

`sge_log_output` Deprecated. Use `options_cluster` instead.

`sge_log_error` Deprecated. Use `options_cluster` instead.

`sge_log_join` Deprecated. Use `options_cluster` instead.

`sge_memory_gigabytes_limit`
Deprecated. Use `options_cluster` instead.

`sge_memory_gigabytes_required`
Deprecated. Use `options_cluster` instead.

`sge_cores` Deprecated. Use `options_cluster` instead.

`sge_gpu` Deprecated. Use `options_cluster` instead.

Details

To launch a Sun Grid Engine (SGE) worker, this launcher creates a temporary job script with a call to `crew::crew_worker()` and submits it as an SGE job with `qsub`. To see most of the lines of the job script in advance, use the `script()` method of the launcher. It has all the lines except for the job name and the call to `crew::crew_worker()`, both of which will be inserted at the last minute when it is time to actually launch a worker.

Attribution

The template files at https://github.com/mschubert/clustermq/tree/master/inst informed the development of the crew launcher plugins in crew.cluster, and we would like to thank Michael Schubert for developing clustermq and releasing it under the permissive Apache License 2.0. See the NOTICE and README.md files in the crew.cluster source code for additional attribution.

See Also

Other sge: crew_class_launcher_sge, crew_class_monitor_sge, crew_controller_sge(), crew_monitor_sge(), crew_options_sge()

---

crew_launcher_slurm **[Experimental]** *Create a launcher with SLURM workers.*

Description

Create an R6 object to launch and maintain workers as SLURM jobs.

Usage

```
crew_launcher_slurm(
  name = NULL,
  workers = 1L,
  seconds_interval = 0.5,
  seconds_timeout = 60,
  seconds_launch = 86400,
  seconds_idle = 300,
  seconds_wall = Inf,
  tasks_max = Inf,
  tasks_timers = 0L,
  reset_globals = NULL,
  reset_packages = NULL,
  reset_options = NULL,
  garbage_collection = NULL,
  crashes_error = NULL,
  tls = crew::crew_tls(mode = "automatic"),
  r_arguments = c("--no-save", "--no-restore"),
  options_metrics = crew::crew_options_metrics(),
  options_cluster = crew.cluster::crew_options_slurm(),
  verbose = NULL,
  command_submit = NULL,
  command_terminate = NULL,
  command_delete = NULL,
  script_directory = NULL,
  script_lines = NULL,
  slurm_log_output = NULL,
```

```
  slurm_log_error = NULL,
  slurm_memory_gigabytes_required = NULL,
  slurm_memory_gigabytes_per_cpu = NULL,
  slurm_cpus_per_task = NULL,
  slurm_time_minutes = NULL,
  slurm_partition = NULL
)
```

Arguments

name	Character string, name of the launcher. If the name is NULL, then a name is automatically generated when the launcher starts.
workers	Maximum number of workers to run concurrently when auto-scaling, excluding task retries and manual calls to launch(). Special workers allocated for task retries do not count towards this limit, so the number of workers running at a given time may exceed this maximum. A smaller number of workers may run if the number of executing tasks is smaller than the supplied value of the workers argument.
seconds_interval	Number of seconds between polling intervals waiting for certain internal synchronous operations to complete. In certain cases, exponential backoff is used with this argument passed to seconds_max in a crew_throttle() object.
seconds_timeout	Number of seconds until timing out while waiting for certain synchronous operations to complete, such as checking mirai::status().
seconds_launch	Seconds of startup time to allow. A worker is unconditionally assumed to be alive from the moment of its launch until seconds_launch seconds later. After seconds_launch seconds, the worker is only considered alive if it is actively connected to its assign websocket.
seconds_idle	Maximum number of seconds that a worker can idle since the completion of the last task. If exceeded, the worker exits. But the timer does not launch until tasks_timers tasks have completed. See the idletime argument of mirai::daemon(). crew does not excel with perfectly transient workers because it does not micromanage the assignment of tasks to workers, so please allow enough idle time for a new worker to be delegated a new task.
seconds_wall	Soft wall time in seconds. The timer does not launch until tasks_timers tasks have completed. See the walltime argument of mirai::daemon().
tasks_max	Maximum number of tasks that a worker will do before exiting. See the maxtasks argument of mirai::daemon(). crew does not excel with perfectly transient workers because it does not micromanage the assignment of tasks to workers, it is recommended to set tasks_max to a value greater than 1.
tasks_timers	Number of tasks to do before activating the timers for seconds_idle and seconds_wall. See the timerstart argument of mirai::daemon().
reset_globals	Deprecated on 2025-05-30 (crew version 1.1.2.9004). Please use the reset_globals option of crew_controller() instead.
reset_packages	Deprecated on 2025-05-30 (crew version 1.1.2.9004). Please use the reset_packages option of crew_controller() instead.

`reset_options` Deprecated on 2025-05-30 (`crew` version 1.1.2.9004). Please use the `reset_options` option of `crew_controller()` instead.

`garbage_collection`
Deprecated on 2025-05-30 (`crew` version 1.1.2.9004). Please use the `garbage_collection` option of `crew_controller()` instead.

`crashes_error` Deprecated on 2025-01-13 (`crew` version 0.10.2.9002).

`tls` A TLS configuration object from `crew_tls()`.

`r_arguments` Optional character vector of command line arguments to pass to `Rscript` (non-Windows) or `Rscript.exe` (Windows) when starting a worker. Example: `r_arguments = c("--vanilla", "--max-connections=32")`.

`options_metrics`
Either `NULL` to opt out of resource metric logging for workers, or an object from `crew_options_metrics()` to enable and configure resource metric logging for workers. For resource logging to run, the `autometric` R package version 0.1.0 or higher must be installed.

`options_cluster`
An options list from `crew_options_slurm()` with cluster-specific configuration options.

`verbose` Deprecated. Use `options_cluster` instead.

`command_submit` Deprecated. Use `options_cluster` instead.

`command_terminate`
Deprecated. Use `options_cluster` instead.

`command_delete` Deprecated on 2024-01-08 (version 0.1.4.9001). Use `command_terminate` instead.

`script_directory`
Deprecated. Use `options_cluster` instead.

`script_lines` Deprecated. Use `options_cluster` instead.

`slurm_log_output`
Deprecated. Use `options_cluster` instead.

`slurm_log_error`
Deprecated. Use `options_cluster` instead.

`slurm_memory_gigabytes_required`
Deprecated. Use `options_cluster` instead.

`slurm_memory_gigabytes_per_cpu`
Deprecated. Use `options_cluster` instead.

`slurm_cpus_per_task`
Deprecated. Use `options_cluster` instead.

`slurm_time_minutes`
Deprecated. Use `options_cluster` instead.

`slurm_partition`
Deprecated. Use `options_cluster` instead.

Details

WARNING: the crew.cluster SLURM plugin is experimental and has not actually been tested on a SLURM cluster. Please proceed with caution and report bugs to https://github.com/wlandau/crew.cluster.

To launch a SLURM worker, this launcher creates a temporary job script with a call to crew::crew_worker() and submits it as an SLURM job with sbatch. To see most of the lines of the job script in advance, use the script() method of the launcher. It has all the lines except for the job name and the call to crew::crew_worker(), both of which will be inserted at the last minute when it is time to actually launch a worker.

Attribution

The template files at https://github.com/mschubert/clustermq/tree/master/inst informed the development of the crew launcher plugins in crew.cluster, and we would like to thank Michael Schubert for developing clustermq and releasing it under the permissive Apache License 2.0. See the NOTICE and README.md files in the crew.cluster source code for additional attribution.

See Also

Other slurm: crew_class_launcher_slurm, crew_class_monitor_slurm, crew_controller_slurm(), crew_monitor_slurm(), crew_options_slurm()

---

crew_monitor_sge **[Experimental]** *Create a SGE monitor object.*

Description

Create an R6 object to monitor SGE cluster jobs.

Usage

```
crew_monitor_sge(
  verbose = TRUE,
  command_list = as.character(Sys.which("qstat")),
  command_terminate = as.character(Sys.which("qdel"))
)
```

Arguments

verbose	Deprecated. Use options_cluster instead.
command_list	Character of length 1, file path to the executable to list jobs.
command_terminate	Deprecated. Use options_cluster instead.

See Also

Other sge: crew_class_launcher_sge, crew_class_monitor_sge, crew_controller_sge(), crew_launcher_sge(), crew_options_sge()

`crew_monitor_slurm` **[Experimental]** *Create a SLURM monitor object.*

Description

Create an R6 object to monitor SLURM cluster jobs.

Usage

```
crew_monitor_slurm(
  verbose = TRUE,
  command_list = as.character(Sys.which("squeue")),
  command_terminate = as.character(Sys.which("scancel"))
)
```

Arguments

verbose	Deprecated. Use options_cluster instead.
command_list	Character of length 1, file path to the executable to list jobs.
command_terminate	Deprecated. Use options_cluster instead.

See Also

Other slurm: crew_class_launcher_slurm, crew_class_monitor_slurm, crew_controller_slurm(), crew_launcher_slurm(), crew_options_slurm()

`crew_options_lsf` **[Experimental]** *LSF options.*

Description

Set options for LSF job management.

Usage

```
crew_options_lsf(
  verbose = FALSE,
  command_submit = as.character(Sys.which("bsub")),
  command_terminate = as.character(Sys.which("bkill")),
  script_directory = tempdir(),
  script_lines = character(0L),
  cwd = getwd(),
  log_output = "/dev/null",
  log_error = "/dev/null",
  memory_gigabytes_limit = NULL,
  memory_gigabytes_required = NULL,
  cores = NULL
)
```

Arguments

verbose	Logical, whether to see console output and error messages when submitting worker.
command_submit	Character of length 1, file path to the executable to submit a worker job.
command_terminate	Character of length 1, file path to the executable to terminate a worker job. Set to "" to skip manually terminating the worker. Unless there is an issue with the platform, the job should still exit thanks to the NNG-powered network programming capabilities of mirai. Still, if you set command_terminate = "", you are assuming extra responsibility for manually monitoring your jobs on the cluster and manually terminating jobs as appropriate.
script_directory	Character of length 1, directory path to the job scripts. Just before each job submission, a job script is created in this folder. Script base names are unique to each launcher and worker, and the launcher deletes the script when the worker is manually terminated. tempdir() is the default, but it might not work for some systems. tools::R_user_dir("crew.cluster", which = "cache") is another reasonable choice.
script_lines	Optional character vector of additional lines to be added to the job script just after the more common flags. An example would be script_lines = "module load R" if your cluster supports R through an environment module.
cwd	Character of length 1, directory to launch the worker from (as opposed to the system default). cwd = "/home" translates to a line of #BSUB -cwd /home in the LSF job script. cwd = getwd() is the default, which launches workers from the current working directory. Set cwd = NULL to omit this line from the job script.
log_output	Character of length 1, file pattern to control the locations of the LSF worker log files. By default, both standard output and standard error go to the same file. log_output = "crew_log_%J.log" translates to a line of #BSUB -o crew_log_%J.log in the LSF job script, where %J is replaced by the job ID of the worker. The default is /dev/null to omit these logs. Set log_output = NULL to omit this line from the job script.

log_error — Character of length 1, file pattern for standard error. log_error = "crew_error_%J.err" translates to a line of #BSUB -e crew_error_%J.err in the LSF job script, where %J is replaced by the job ID of the worker. The default is /dev/null to omit these logs. Set log_error = NULL to omit this line from the job script.

memory_gigabytes_limit — Positive numeric scalar, memory limit in gigabytes of the worker. memory_gigabytes_limit = 4 translates to a line of #BSUB -M 4G in the LSF job script. memory_gigabytes_limit = NULL omits this line.

memory_gigabytes_required — Positive numeric scalar, memory requirement in gigabytes. memory_gigabytes_required = 4 translates to a line of #BSUB -R 'rusage[mem=4G]' in the LSF job script. memory_gigabytes_required = NULL omits this line.

cores — Optional positive integer scalar, number of CPU cores for the worker. cores = 4 translates to a line of #BSUB -n 4 in the LSF job script. cores = NULL omits this line.

Value

A classed list of options.

Retryable options

Retryable options are deprecated in crew.cluster as of 2025-01-27 (version 0.3.4).

Attribution

The template files at https://github.com/mschubert/clustermq/tree/master/inst informed the development of the crew launcher plugins in crew.cluster, and we would like to thank Michael Schubert for developing clustermq and releasing it under the permissive Apache License 2.0. See the NOTICE and README.md files in the crew.cluster source code for additional attribution.

See Also

Other lsf: crew_class_launcher_lsf, crew_controller_lsf(), crew_launcher_lsf()

Examples

```
crew_options_lsf()
```

---

crew_options_pbs **[Experimental]** *PBS options.*

Description

Set options for PBS job management.

Usage

```
crew_options_pbs(
  verbose = FALSE,
  command_submit = as.character(Sys.which("qsub")),
  command_terminate = as.character(Sys.which("qdel")),
  script_directory = tempdir(),
  script_lines = character(0L),
  cwd = TRUE,
  log_output = "/dev/null",
  log_error = NULL,
  log_join = TRUE,
  memory_gigabytes_required = NULL,
  cores = NULL,
  walltime_hours = 12
)
```

Arguments

verbose	Logical, whether to see console output and error messages when submitting worker.
command_submit	Character of length 1, file path to the executable to submit a worker job.
command_terminate	Character of length 1, file path to the executable to terminate a worker job. Set to "" to skip manually terminating the worker. Unless there is an issue with the platform, the job should still exit thanks to the NNG-powered network programming capabilities of mirai. Still, if you set command_terminate = "", you are assuming extra responsibility for manually monitoring your jobs on the cluster and manually terminating jobs as appropriate.
script_directory	Character of length 1, directory path to the job scripts. Just before each job submission, a job script is created in this folder. Script base names are unique to each launcher and worker, and the launcher deletes the script when the worker is manually terminated. tempdir() is the default, but it might not work for some systems. tools::R_user_dir("crew.cluster", which = "cache") is another reasonable choice.
script_lines	Optional character vector of additional lines to be added to the job script just after the more common flags. An example would be script_lines = "module load R" if your cluster supports R through an environment module.
cwd	Logical of length 1, whether to set the working directory of the worker to the working directory it was launched from. cwd = TRUE is translates to a line of cd "$O_WORKDIR" in the job script. This line is inserted after the content of script_lines to make sure the #PBS directives are above system commands. cwd = FALSE omits this line.
log_output	Character of length 1, file or directory path to PBS worker log files for standard output. log_output = "VALUE" translates to a line of #PBS -o VALUE in the PBS job script. The default is /dev/null to omit the logs. If you do supply

a non-/dev/null value, it is recommended to supply a directory path with a trailing slash so that each worker gets its own set of log files.

log_error Character of length 1, file or directory path to PBS worker log files for standard error. log_error = "VALUE" translates to a line of #PBS -e VALUE in the PBS job script. The default of NULL omits this line. If you do supply a non-/dev/null value, it is recommended to supply a directory path with a trailing slash so that each worker gets its own set of log files.

log_join Logical, whether to join the stdout and stderr log files together into one file. log_join = TRUE translates to a line of #PBS -j oe in the PBS job script, while log_join = FALSE is equivalent to #PBS -j n. If log_join = TRUE, then log_error should be NULL.

memory_gigabytes_required
Optional positive numeric scalar, gigabytes of memory required to run the worker. memory_gigabytes_required = 2.4 translates to a line of #PBS -l mem=2.4gb in the PBS job script. memory_gigabytes_required = NULL omits this line.

cores Optional positive integer scalar, number of cores for the worker ("slots" in PBS lingo). cores = 4 translates to a line of #PBS -l ppn=4 in the PBS job script. cores = NULL omits this line.

walltime_hours Numeric scalar, hours of wall time to request for the worker. walltime_hours = 23 translates to a line of #PBS -l walltime=23:00:00 in the job script. walltime_hours = NULL omits this line.

Value

A classed list of options.

Retryable options

Retryable options are deprecated in crew.cluster as of 2025-01-27 (version 0.3.4).

Attribution

The template files at https://github.com/mschubert/clustermq/tree/master/inst informed the development of the crew launcher plugins in crew.cluster, and we would like to thank Michael Schubert for developing clustermq and releasing it under the permissive Apache License 2.0. See the NOTICE and README.md files in the crew.cluster source code for additional attribution.

See Also

Other pbs: crew_class_launcher_pbs, crew_controller_pbs(), crew_launcher_pbs()

Examples

```
crew_options_pbs()
```

`crew_options_sge` **[Maturing]** *SGE options.*

Description

Set options for SGE job management.

Usage

```
crew_options_sge(
  verbose = FALSE,
  command_submit = as.character(Sys.which("qsub")),
  command_terminate = as.character(Sys.which("qdel")),
  script_directory = tempdir(),
  script_lines = character(0L),
  cwd = TRUE,
  envvars = FALSE,
  log_output = "/dev/null",
  log_error = NULL,
  log_join = TRUE,
  memory_gigabytes_limit = NULL,
  memory_gigabytes_required = NULL,
  cores = NULL,
  gpu = NULL
)
```

Arguments

`verbose` Logical, whether to see console output and error messages when submitting worker.

`command_submit` Character of length 1, file path to the executable to submit a worker job.

`command_terminate`
Character of length 1, file path to the executable to terminate a worker job. Set to `""` to skip manually terminating the worker. Unless there is an issue with the platform, the job should still exit thanks to the NNG-powered network programming capabilities of `mirai`. Still, if you set `command_terminate = ""`, you are assuming extra responsibility for manually monitoring your jobs on the cluster and manually terminating jobs as appropriate.

`script_directory`
Character of length 1, directory path to the job scripts. Just before each job submission, a job script is created in this folder. Script base names are unique to each launcher and worker, and the launcher deletes the script when the worker is manually terminated. `tempdir()` is the default, but it might not work for some systems. `tools::R_user_dir("crew.cluster", which = "cache")` is another reasonable choice.

`script_lines` Optional character vector of additional lines to be added to the job script just after the more common flags. An example would be `script_lines = "module load R"` if your cluster supports R through an environment module.

`cwd` Logical of length 1, whether to launch the worker from the current working directory (as opposed to the user home directory). `cwd = TRUE` translates to a line of `#$ -cwd` in the SGE job script. `cwd = FALSE` omits this line.

`envvars` Logical of length 1, whether to forward the environment variables of the current session to the SGE worker. `envvars = TRUE` translates to a line of `#$ -V` in the SGE job script. `envvars = FALSE` omits this line.

`log_output` Character of length 1, file or directory path to SGE worker log files for standard output. `log_output = "VALUE"` translates to a line of `#$ -o VALUE` in the SGE job script. The default is `/dev/null` to omit the logs. If you do supply a non-`/dev/null` value, it is recommended to supply a directory path with a trailing slash so that each worker gets its own set of log files.

`log_error` Character of length 1, file or directory path to SGE worker log files for standard error. `log_error = "VALUE"` translates to a line of `#$ -e VALUE` in the SGE job script. The default of `NULL` omits this line. If you do supply a non-`/dev/null` value, it is recommended to supply a directory path with a trailing slash so that each worker gets its own set of log files.

`log_join` Logical, whether to join the stdout and stderr log files together into one file. `log_join = TRUE` translates to a line of `#$ -j y` in the SGE job script, while `log_join = FALSE` is equivalent to `#$ -j n`. If `log_join = TRUE`, then `log_error` should be `NULL`.

`memory_gigabytes_limit` Optional numeric scalar, maximum number of gigabytes of memory a worker is allowed to consume. If the worker consumes more than this level of memory, then SGE will terminate it. `memory_gigabytes_limit = 5.7"` translates to a line of `"#$ -l h_rss=5.7G"` in the SGE job script. `memory_gigabytes_limit = NULL` omits this line.

`memory_gigabytes_required` Optional positive numeric scalar, gigabytes of memory required to run the worker. `memory_gigabytes_required = 2.4` translates to a line of `#$ -l m_mem_free=2.4G` in the SGE job script. `memory_gigabytes_required = NULL` omits this line.

`cores` Optional positive integer scalar, number of cores per worker ("slots" in SGE lingo). `cores = 4` translates to a line of `#$ -pe smp 4` in the SGE job script. `cores = NULL` omits this line.

`gpu` Optional integer scalar, number of GPUs to request for the worker. `gpu = 1` translates to a line of `"#$ -l gpu=1"` in the SGE job script. `gpu = NULL` omits this line.

Value

A classed list of options.

Retryable options

Retryable options are deprecated in `crew.cluster` as of 2025-01-27 (version `0.3.4`).

Attribution

The template files at https://github.com/mschubert/clustermq/tree/master/inst informed the development of the crew launcher plugins in crew.cluster, and we would like to thank Michael Schubert for developing clustermq and releasing it under the permissive Apache License 2.0. See the NOTICE and README.md files in the crew.cluster source code for additional attribution.

See Also

Other sge: crew_class_launcher_sge, crew_class_monitor_sge, crew_controller_sge(), crew_launcher_sge(), crew_monitor_sge()

Examples

```
crew_options_sge()
```

---

crew_options_slurm **[Experimental]** *SLURM options.*

Description

Set options for SLURM job management.

Usage

```
crew_options_slurm(
  verbose = FALSE,
  command_submit = as.character(Sys.which("sbatch")),
  command_terminate = as.character(Sys.which("scancel")),
  script_directory = tempdir(),
  script_lines = character(0L),
  log_output = "/dev/null",
  log_error = "/dev/null",
  memory_gigabytes_required = NULL,
  memory_gigabytes_per_cpu = NULL,
  cpus_per_task = NULL,
  time_minutes = NULL,
  partition = NULL,
  n_tasks = 1
)
```

Arguments

verbose	Logical, whether to see console output and error messages when submitting worker.
command_submit	Character of length 1, file path to the executable to submit a worker job.

command_terminate
: Character of length 1, file path to the executable to terminate a worker job. Set to "" to skip manually terminating the worker. Unless there is an issue with the platform, the job should still exit thanks to the NNG-powered network programming capabilities of mirai. Still, if you set command_terminate = "", you are assuming extra responsibility for manually monitoring your jobs on the cluster and manually terminating jobs as appropriate.

script_directory
: Character of length 1, directory path to the job scripts. Just before each job submission, a job script is created in this folder. Script base names are unique to each launcher and worker, and the launcher deletes the script when the worker is manually terminated. tempdir() is the default, but it might not work for some systems. tools::R_user_dir("crew.cluster", which = "cache") is another reasonable choice.

script_lines
: Optional character vector of additional lines to be added to the job script just after the more common flags. An example would be script_lines = "module load R" if your cluster supports R through an environment module.

log_output
: Character of length 1, file pattern to control the locations of the SLURM worker log files. By default, both standard output and standard error go to the same file. log_output = "crew_log_%A.txt" translates to a line of #SBATCH --output=crew_log_%A.txt in the SLURM job script, where %A is replaced by the job ID of the worker. The default is /dev/null to omit these logs. Set log_output = NULL to omit this line from the job script.

log_error
: Character of length 1, file pattern for standard error. log_error = "crew_log_%A.txt" translates to a line of #SBATCH --error=crew_log_%A.txt in the SLURM job script, where %A is replaced by the job ID of the worker. The default is /dev/null to omit these logs. Set log_error = NULL to omit this line from the job script.

memory_gigabytes_required
: Positive numeric scalar, total number of gigabytes of memory required per node. memory_gigabytes_required = 2.40123 translates to a line of #SBATCH --mem=2041M in the SLURM job script. memory_gigabytes_required = NULL omits this line.

memory_gigabytes_per_cpu
: Positive numeric scalar, gigabytes of memory required per CPU. memory_gigabytes_per_cpu = 2.40123 translates to a line of #SBATCH --mem-per-cpu=2041M in the SLURM job script. memory_gigabytes_per_cpu = NULL omits this line.

cpus_per_task
: Optional positive integer scalar, number of CPUs for the worker. cpus_per_task = 4 translates to a line of #SBATCH --cpus-per-task=4 in the SLURM job script. cpus_per_task = NULL omits this line.

time_minutes
: Numeric scalar, number of minutes to designate as the wall time of crew each worker instance on the SLURM cluster. time_minutes = 60 translates to a line of #SBATCH --time=60 in the SLURM job script. time_minutes = NULL omits this line.

partition
: Character string, name of the SLURM partition to create workers on. partition = "partition1,partition2" translates to a line of #SBATCH --partition=partition1,partition2 in the SLURM job script. partition = NULL omits this line.

`n_tasks` Numeric scalar, number of SLURM tasks to run within the job. `n_tasks = 1` translates to a line of `#SBATCH --ntasks=1` in the SLURM job script. `n_tasks = 0` omits this line.

Value

A classed list of options.

Retryable options

Retryable options are deprecated in `crew.cluster` as of 2025-01-27 (version `0.3.4`).

Attribution

The template files at https://github.com/mschubert/clustermq/tree/master/inst informed the development of the `crew` launcher plugins in `crew.cluster`, and we would like to thank Michael Schubert for developing `clustermq` and releasing it under the permissive Apache License 2.0. See the `NOTICE` and `README.md` files in the `crew.cluster` source code for additional attribution.

See Also

Other slurm: `crew_class_launcher_slurm`, `crew_class_monitor_slurm`, `crew_controller_slurm()`, `crew_launcher_slurm()`, `crew_monitor_slurm()`

Examples

```
crew_options_slurm()
```

Index