Package ‘crew.cluster’

July 10, 2024

Title Crew Launcher Plugins for Traditional High-Performance Computing Clusters


Version 0.3.2

License MIT + file LICENSE


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

Depends R (>= 4.0.0)

Imports crew (>= 0.9.5), 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 7.3.1

NeedsCompilation no
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.
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/instruct 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

Active bindings

  lsf_cwd See crew_launcher_lsf().
  lsf_log_output See crew_launcher_lsf().
  lsf_log_error See crew_launcher_lsf().
  lsf_memory_gigabytes_limit See crew_launcher_lsf().
  lsf_memory_gigabytes_required See crew_launcher_lsf().
  lsf_cores See crew_launcher_lsf().
Methods

Public methods:

• crew_class_launcher_lsf\$new()
• crew_class_launcher_lsf\$validate()
• crew_class_launcher_lsf\$script()

Method new(): LSF launcher constructor.

Usage:
crew_class_launcher_lsf\$new(
  name = NULL,
  seconds_interval = NULL,
  seconds_timeout = NULL,
  seconds_launch = NULL,
  seconds_idle = NULL,
  seconds_wall = NULL,
  tasks_max = NULL,
  tasks_timers = NULL,
  reset_globals = NULL,
  reset_packages = NULL,
  reset_options = NULL,
  garbage_collection = NULL,
  launch_max = NULL,
  tls = NULL,
  verbose = NULL,
  command_submit = NULL,
  command_terminate = 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 See crew_launcher_lsf().
seconds_interval See crew_launcher_lsf().
seconds_timeout See crew_launcher_lsf().
seconds_launch See crew_launcher_lsf().
seconds_idle See crew_launcher_lsf().
seconds_wall See crew_launcher_lsf().
tasks_max See crew_launcher_lsf().
tasks_timers See crew_launcher_lsf().
reset_globals See crew_launcher_lsf().
reset_packages See `crew_launcher_lsf()`.
reset_options See `crew_launcher_lsf()`.
garbage_collection See `crew_launcher_lsf()`.
launch_max See `crew_launcher_lsf()`.
tls See `crew_launcher_lsf()`.
verbose See `crew_launcher_lsf()`.
command_submit See `crew_launcher_lsf()`.
command_terminate See `crew_launcher_lsf()`.
script_directory See `crew_launcher_lsf()`.
script_lines See `crew_launcher_lsf()`.
lsf_cwd See `crew_launcher_lsf()`.
lsf_log_output See `crew_launcher_lsf()`.
lsf_log_error See `crew_launcher_lsf()`.
lsf_memory_gigabytes_limit See `crew_launcher_lsf()`.
lsf_memory_gigabytes_required See `crew_launcher_lsf()`.
lsf_cores See `crew_launcher_lsf()`.

Returns: an LSF launcher object.

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

*Usage:*

```r
crew_class_launcher_lsf$validate()
```

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

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

*Usage:*

```r
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:*

```r
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()`

Examples

```r
## ------------------------------------------------
## 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` $\rightarrow$ `crew.cluster::crew_class_launcher_cluster` $\rightarrow$ `crew_class_launcher_pbs`
**Active bindings**

- `pbs_cwd` See `crew_launcher_pbs()`.
- `pbs_log_output` See `crew_launcher_pbs()`.
- `pbs_log_error` See `crew_launcher_pbs()`.
- `pbs_log_join` See `crew_launcher_pbs()`.
- `pbs_memory_gigabytes_required` See `crew_launcher_pbs()`.
- `pbs_cores` See `crew_launcher_pbs()`.
- `pbs_walltime_hours` See `crew_launcher_pbs()`.

**Methods**

**Public methods:**

- `crew_class_launcher_pbs$new()`
- `crew_class_launcher_pbs$validate()`
- `crew_class_launcher_pbs$script()`

**Method** `new()`: PBS/TORQUE launcher constructor.

**Usage:**

```r
crew_class_launcher_pbs$new(
  name = NULL,
  seconds_interval = NULL,
  seconds_timeout = NULL,
  seconds_launch = NULL,
  seconds_idle = NULL,
  seconds_wall = NULL,
  tasks_max = NULL,
  tasks_timers = NULL,
  resetGlobals = NULL,
  reset_packages = NULL,
  reset_options = NULL,
  garbage_collection = NULL,
  launch_max = NULL,
  tls = NULL,
  verbose = NULL,
  command_submit = NULL,
  command_terminate = 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 See crew_launcher_pbs().
seconds_interval See crew_launcher_slurm().
seconds_timeout See crew_launcher_slurm().
seconds_launch See crew_launcher_pbs().
seconds_idle See crew_launcher_pbs().
seconds_wall See crew_launcher_pbs().
tasks_max See crew_launcher_pbs().
tasks_timers See crew_launcher_pbs().
reset_globals See crew_launcher_pbs().
reset_packages See crew_launcher_pbs().
reset_options See crew_launcher_pbs().
garbage_collection See crew_launcher_pbs().
l腔_max See crew_launcher_pbs().
 tls See crew_launcher_pbs().
verbose See crew_launcher_pbs().
command_submit See crew_launcher_pbs().
command_terminate See crew_launcher_pbs().
script_directory See crew_launcher_pbs().
script_lines See crew_launcher_pbs().
pbs_cwd See crew_launcher_sge().
pbs_log_output See crew_launcher_pbs().
pbs_log_error See crew_launcher_pbs().
pbs_log_join See crew_launcher_pbs().
pbs_memory_gigabytes_required See crew_launcher_pbs().
pbs_cores See crew_launcher_pbs().
pbs_walltime_hours See crew_launcher_pbs().

Returns: an PBS/TORQUE launcher object.

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.
crew_class_launcher_sge

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()

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

Active bindings
sge_cwd  See crew_launcher_sge().
sgenvs  See crew_launcher_sge().
sge_log_output  See crew_launcher_sge().
sgelog_error  See crew_launcher_sge().
sgelog_join  See crew_launcher_sge().
sgememory_gigabytes_limit  See crew_launcher_sge().
sgememory_gigabytes_required  See crew_launcher_sge().
sgecores  See crew_launcher_sge().
sgegpu  See crew_launcher_sge().

Methods
Public methods:
• crew_class_launcher_sge$new()
• crew_class_launcher_sge$validate()
• crew_class_launcher_sge$script()

Method new(): SGE launcher constructor.
Usage:
crew_class_launcher_sge$new(
  name = NULL,
  seconds_interval = NULL,
  seconds_timeout = NULL,
  seconds_launch = NULL,
  seconds_idle = NULL,
  seconds_wall = NULL,
  tasks_max = NULL,
  tasks_timers = NULL,
  reset_globals = NULL,
  reset_packages = NULL,
  reset_options = NULL,
  garbage_collection = NULL,
  launch_max = NULL,
  tls = NULL,
  verbose = NULL,
  command_submit = NULL,
  command_terminate = 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 See crew_launcher_sge().
seconds_interval See crew_launcher_slurm().
seconds_timeout See crew_launcher_slurm().
seconds_launch See crew_launcher_sge().
seconds_idle See crew_launcher_sge().
seconds_wall See crew_launcher_sge().
tasks_max See crew_launcher_sge().
tasks_timers See crew_launcher_sge().
reset_globals See crew_launcher_sge().
reset_packages See crew_launcher_sge().
reset_options See crew_launcher_sge().
garbage_collection See crew_launcher_sge().
launch_max See crew_launcher_sge().
tls See crew_launcher_sge().
verbose See crew_launcher_sge().
command_submit See crew_launcher_sge().
command_terminate See crew_launcher_sge().
script_directory See crew_launcher_sge().
script_lines See crew_launcher_sge().
sge_cwd See crew_launcher_sge().
sge_envvars See crew_launcher_sge().
sge_log_output See crew_launcher_sge().
sge_log_error See crew_launcher_sge().
sge_log_join See crew_launcher_sge().
sge_memory_gigabytes_limit See crew_launcher_sge().
sge_memory_gigabytes_required See crew_launcher_sge().
sge_cores See crew_launcher_sge().
sge_gpu See crew_launcher_sge().

Returns: an SGE launcher object.

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()

Examples

```r
### 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.
crew_class_launcher_slurm

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

Active bindings

slurm_log_output See crew_launcher_slurm().
slurm_log_error See crew_launcher_slurm().
slurm_memory_gigabytes_per_cpu See crew_launcher_slurm().
slurm_cpus_per_task See crew_launcher_slurm().
slurm_time_minutes See crew_launcher_slurm().
slurm_partition See crew_launcher_slurm().

Methods

Public methods:

• crew_class_launcher_slurm$new()
• crew_class_launcher_slurm$validate()
• crew_class_launcher_slurm$script()

Method new(): SLURM launcher constructor.

Usage:

crew_class_launcher_slurm$new(
  name = NULL,
  seconds_interval = NULL,
  seconds_timeout = NULL,
  seconds_launch = NULL,
  seconds_idle = NULL,
  seconds_wall = NULL,
  tasks_max = NULL,
  tasks_timers = NULL,
  resetGlobals = NULL,
  reset_packages = NULL,
  reset_options = NULL,
  garbage_collection = NULL,
launch_max = NULL,
tls = NULL,
verbose = NULL,
command_submit = NULL,
command_terminate = NULL,
script_directory = NULL,
script_lines = NULL,
slurm_log_output = NULL,
slurm_log_error = NULL,
slurm_memory_gigabytes_per_cpu = NULL,
slurm_cpus_per_task = NULL,
slurm_time_minutes = NULL,
slurm_partition = NULL
)

Arguments:
name See crew_launcher_slurm().
seconds_interval See crew_launcher_slurm().
seconds_timeout See crew_launcher_slurm().
seconds_launch See crew_launcher_slurm().
seconds_idle See crew_launcher_slurm().
seconds_wall See crew_launcher_slurm().
tasks_max See crew_launcher_slurm().
tasks_timers See crew_launcher_slurm().
reset_globals See crew_launcher_slurm().
reset_packages See crew_launcher_slurm().
reset_options See crew_launcher_slurm().
garbage_collection See crew_launcher_slurm().
launch_max See crew_launcher_slurm().
tls See crew_launcher_slurm().
verbose See crew_launcher_slurm().
command_submit See crew_launcher_sge().
command_terminate See crew_launcher_sge().
script_directory See crew_launcher_sge().
script_lines See crew_launcher_sge().
slurm_log_output See crew_launcher_slurm().
slurm_log_error See crew_launcher_slurm().
slurm_memory_gigabytes_per_cpu See crew_launcher_slurm().
slurm_cpus_per_task See crew_launcher_slurm().
slurm_time_minutes See crew_launcher_slurm().
slurm_partition See crew_launcher_slurm().

Returns: an SLURM launcher object.

Method validate(): Validate the launcher.

Usage:
crew_class_launcher_slurm

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:**

```r
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()`

**Examples**

```r
## ------------------------------------------------
## 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_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_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**

```r
crew_controller_lsf(
  name = NULL,
  workers = 1L,
  host = NULL,
  port = NULL,
  tls = crew::crew_tls(mode = "automatic"),
  tls_enable = NULL,
  tls_config = NULL,
  seconds_interval = 0.25,
  seconds_timeout = 60,
  seconds_launch = 86400,
  secondsIdle = Inf,
  seconds_wall = Inf,
  seconds_exit = NULL,
  retry_tasks = TRUE,
  tasks_max = Inf,
  tasks_timers = 0L,
  reset_globals = TRUE,
  reset_packages = FALSE,
  reset_options = FALSE,
  garbage_collection = FALSE,
  launch_max = 5L,
  verbose = FALSE,
  command_submit = as.character(Sys.which("bsub")),
  command_terminate = as.character(Sys.which("bkill")),
  command_delete = NULL,
  script_directory = tempdir(),
  script_lines = character(0L),
  lsf_cwd = getwd(),
  lsf_log_output = "/dev/null",
  lsf_log_error = "/dev/null",
  lsf_memory_gigabytes_limit = NULL,
  lsf_memory_gigabytes_required = NULL,
  lsf_cores = NULL
)
```
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the client object. If NULL, a name is automatically generated.</td>
</tr>
<tr>
<td>workers</td>
<td>Integer, maximum number of parallel workers to run.</td>
</tr>
<tr>
<td>host</td>
<td>IP address of the mirai client to send and receive tasks. If NULL, the host defaults to the local IP address.</td>
</tr>
<tr>
<td>port</td>
<td>TCP port to listen for the workers. If NULL, then an available ephemeral port is automatically chosen.</td>
</tr>
<tr>
<td>tls</td>
<td>A TLS configuration object from crew_tls().</td>
</tr>
<tr>
<td>tls_enable</td>
<td>Deprecated on 2023-09-15 in version 0.4.1. Use argument tls instead.</td>
</tr>
<tr>
<td>tls_config</td>
<td>Deprecated on 2023-09-15 in version 0.4.1. Use argument tls instead.</td>
</tr>
<tr>
<td>seconds_interval</td>
<td>Number of seconds between polling intervals waiting for certain internal synchronous operations to complete, such as checking mirai::status()</td>
</tr>
<tr>
<td>seconds_timeout</td>
<td>Number of seconds until timing out while waiting for certain synchronous operations to complete, such as checking mirai::status().</td>
</tr>
<tr>
<td>seconds_launch</td>
<td>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.</td>
</tr>
<tr>
<td>seconds_idle</td>
<td>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.</td>
</tr>
<tr>
<td>seconds_wall</td>
<td>Soft wall time in seconds. The timer does not launch until tasks_timers tasks have completed. See the walltime argument of mirai::daemon().</td>
</tr>
<tr>
<td>seconds_exit</td>
<td>Deprecated on 2023-09-21 in version 0.1.2.9000. No longer necessary.</td>
</tr>
<tr>
<td>retry_tasks</td>
<td>TRUE to automatically retry a task in the event of an unexpected worker exit. FALSE to give up on the first exit and return a mirai error code (code number 19). TRUE (default) is recommended in most situations. Use FALSE for debugging purposes, e.g. to confirm that a task is causing a worker to run out of memory or crash in some other way.</td>
</tr>
<tr>
<td>tasks_max</td>
<td>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.</td>
</tr>
<tr>
<td>tasks_timers</td>
<td>Number of tasks to do before activating the timers for seconds_idle and seconds_wall. See the timerstart argument of mirai::daemon().</td>
</tr>
<tr>
<td>resetGlobals</td>
<td>TRUE to reset global environment variables between tasks, FALSE to leave them alone.</td>
</tr>
<tr>
<td>resetPackages</td>
<td>TRUE to unload any packages loaded during a task (runs between each task), FALSE to leave packages alone.</td>
</tr>
</tbody>
</table>
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.

garbage_collection TRUE to run garbage collection between tasks, FALSE to skip.

launch_max Positive integer of length 1, maximum allowed consecutive launch attempts which do not complete any tasks. Enforced on a worker-by-worker basis. The futile launch count resets to back 0 for each worker that completes a task. It is recommended to set launch_max above 0 because sometimes workers are unproductive under perfectly ordinary circumstances. But launch_max should still be small enough to detect errors in the underlying platform.

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.

cmd_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.

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

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.

lsf_cwd Character of length 1, directory to launch the worker from (as opposed to the system default). lsf_cwd = "/home" translates to a line of #SUB -cwd /home in the LSF job script. lsf_cwd = getwd() is the default, which launches workers from the current working directory. Set lsf_cwd = NULL to omit this line from the job script.

lsf_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. lsf_log_output = "crew_log_%J.log" translates to a line of #SUB -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 lsf_log_output = NULL to omit this line from the job script.
**lsf_log_error**  Character of length 1, file pattern for standard error. `lsf_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 `lsf_log_error = NULL` to omit this line from the job script.

**lsf_memory_gigabytes_limit**  Positive numeric of length 1 with the limit in gigabytes `lsf_memory_gigabytes_limit = 4` translates to a line of `#BSUB -M 4G` in the LSF job script. `lsf_memory_gigabytes_limit = NULL` omits this line.

**lsf_memory_gigabytes_required**  Positive numeric of length 1 with the memory requirement in gigabytes `lsf_memory_gigabytes_required = 4` translates to a line of `#BSUB -R 'rusage[mem=4G]'` in the LSF job script. `lsf_memory_gigabytes_required = NULL` omits this line.

**lsf_cores**  Optional positive integer of length 1, number of CPU cores for the worker. `lsf_cores = 4` translates to a line of `#BSUB -n 4` in the LSF job script. `lsf_cores = NULL` omits this line.

### 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](https://github.com/wlandau/crew.cluster).

### Attribution

The template files at [https://github.com/mschubert/clustermq/tree/master/inst](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()](#)

### Examples

```r
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

```r
crew_controller_pbs(
  name = NULL,
  workers = 1L,
  host = NULL,
  port = NULL,
  tls = crew::crew_tls(mode = "automatic"),
  tls_enable = NULL,
  tls_config = NULL,
  seconds_interval = 0.25,
  seconds_timeout = 60,
  seconds_launch = 86400,
  seconds_idle = Inf,
  seconds_wall = Inf,
  seconds_exit = NULL,
  retry_tasks = TRUE,
  tasks_max = Inf,
  tasks_timers = 0L,
  reset_globals = TRUE,
  reset_packages = FALSE,
  reset_options = FALSE,
  garbage_collection = FALSE,
  launch_max = 5L,
  verbose = FALSE,
  command_submit = as.character(Sys.which("qsub")),
  command_terminate = as.character(Sys.which("qdel")),
  command_delete = NULL,
  script_directory = tempdir(),
  script_lines = character(0L),
  pbs_cwd = TRUE,
  pbs_log_output = "/dev/null",
  pbs_log_error = NULL,
  pbs_log_join = TRUE,
  pbs_memory_gigabytes_required = NULL,
  pbs_cores = NULL,
  pbs_walltime_hours = 12
)
```
**Arguments**

- **name**: Name of the client object. If NULL, a name is automatically generated.
- **workers**: Integer, maximum number of parallel workers to run.
- **host**: IP address of the mirai client to send and receive tasks. If NULL, the host defaults to the local IP address.
- **port**: TCP port to listen for the workers. If NULL, then an available ephemeral port is automatically chosen.
- **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.
- **seconds_interval**: Number of seconds between polling intervals waiting for certain internal synchronous operations to complete, such as checking `mirai::status()`.
- **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()`.
- **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**: TRUE to automatically retry a task in the event of an unexpected worker exit. FALSE to give up on the first exit and return a mirai error code (code number 19). TRUE (default) is recommended in most situations. Use FALSE for debugging purposes, e.g. to confirm that a task is causing a worker to run out of memory or crash in some other way.
- **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 unload any packages loaded during a task (runs between each task), FALSE to leave packages alone.
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.

garbage_collection  TRUE to run garbage collection between tasks, FALSE to skip.

launch_max  Positive integer of length 1, maximum allowed consecutive launch attempts which do not complete any tasks. Enforced on a worker-by-worker basis. The futile launch count resets to back 0 for each worker that completes a task. It is recommended to set launch_max above 0 because sometimes workers are unproductive under perfectly ordinary circumstances. But launch_max should still be small enough to detect errors in the underlying platform.

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.

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

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.

pbs_cwd  Logical of length 1, whether to set the working directory of the worker to the working directory it was launched from. pbs_cwd = TRUE is translates to a line of cd "$PBS_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. pbs_cwd = FALSE omits this line.

pbs_log_output  Character of length 1, file or directory path to PBS worker log files for standard output. pbs_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.

pbs_log_error  Character of length 1, file or directory path to PBS worker log files for standard error. pbs_log_error = "VALUE" translates to a line of #PBS -e VALUE in the
crew_controller_pbs

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.

**pbs_log_join**

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

**pbs_memory_gigabytes_required**

Optional positive numeric of length 1 with the gigabytes of memory required to run the worker. pbs_memory_gigabytes_required = 2.4 translates to a line of #PBS -l mem=2.4gb in the PBS job script. pbs_memory_gigabytes_required = NULL omits this line.

**pbs_cores**

Optional positive integer of length 1, number of cores per worker ("slots" in PBS lingo). pbs_cores = 4 translates to a line of #PBS -l pppn=4 in the PBS job script. pbs_cores = NULL omits this line.

**pbs_walltime_hours**

Numeric of length 1 with the hours of wall time to request for the job. pbs_walltime_hours = 23 translates to a line of #PBS -l walltime=23:00:00 in the job script. pbs_walltime_hours = NULL omits this line.

Attrition

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()

Examples

```r
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

```r
crew_controller_sge(
  name = NULL,
  workers = 1L,
  host = NULL,
  port = NULL,
  tls = crew::crew_tls(mode = "automatic"),
  tls_enable = NULL,
  tls_config = NULL,
  seconds_interval = 0.25,
  seconds_timeout = 60,
  seconds_launch = 86400,
  seconds_idle = Inf,
  seconds_wall = Inf,
  seconds_exit = NULL,
  retry_tasks = TRUE,
  tasks_max = Inf,
  tasks_timers = 0L,
  reset_globals = TRUE,
  reset_packages = FALSE,
  reset_options = FALSE,
  garbage_collection = FALSE,
  launch_max = 5L,
  verbose = FALSE,
  command_submit = as.character(Sys.which("qsub")),
  command_terminate = as.character(Sys.which("qdel")),
  command_delete = NULL,
  script_directory = tempdir(),
  script_lines = character(0L),
  sge_cwd = TRUE,
  sge_envvars = FALSE,
  sge_log_output = "/dev/null",
  sge_log_error = NULL,
  sge_log_join = TRUE,
  sge_memory_gigabytes_limit = NULL,
  sge_memory_gigabytes_required = NULL,
  sge_cores = NULL,
  sge_gpu = NULL
)
```
Arguments

name
Name of the client object. If NULL, a name is automatically generated.

workers
Integer, maximum number of parallel workers to run.

host
IP address of the mirai client to send and receive tasks. If NULL, the host defaults to the local IP address.

port
TCP port to listen for the workers. If NULL, then an available ephemeral port is automatically chosen.

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.

seconds_interval
Number of seconds between polling intervals waiting for certain internal synchronous operations to complete, such as checking mirai::status().

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
TRUE to automatically retry a task in the event of an unexpected worker exit. FALSE to give up on the first exit and return a mirai error code (code number 19). TRUE (default) is recommended in most situations. Use FALSE for debugging purposes, e.g. to confirm that a task is causing a worker to run out of memory or crash in some other way.

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 unload any packages loaded during a task (runs between each task), FALSE to leave packages alone.

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.

garbage_collection TRUE to run garbage collection between tasks, FALSE to skip.

launch_max Positive integer of length 1, maximum allowed consecutive launch attempts which do not complete any tasks. Enforced on a worker-by-worker basis. The futile launch count resets to back 0 for each worker that completes a task. It is recommended to set launch_max above 0 because sometimes workers are unproductive under perfectly ordinary circumstances. But launch_max should still be small enough to detect errors in the underlying platform.

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.

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

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.

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

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

sg_log_output Character of length 1, file or directory path to SGE worker log files for standard output. sg_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.
crew_controller_sge

sge_log_error Character of length 1, file or directory path to SGE worker log files for standard error. sge_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.

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

sge_memory_gigabytes_limit Optional numeric of length 1 with the 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. sge_memory_gigabytes_limit = 5.7" translates to a line of "#$ -l h_rss=5.7G" in the SGE job script. sge_memory_gigabytes_limit = NULL omits this line.

sge_memory_gigabytes_required Optional positive numeric of length 1 with the gigabytes of memory required to run the worker. sge_memory_gigabytes_required = 2.4 translates to a line of #$ -l m_mem_free=2.4G in the SGE job script. sge_memory_gigabytes_required = NULL omits this line.

sge_cores Optional positive integer of length 1, number of cores per worker ("slots" in SGE lingo). sge_cores = 4 translates to a line of #$ -pe smp 4 in the SGE job script. sge_cores = NULL omits this line.

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

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()

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**

```r
crew_controller_slurm(
  name = NULL,
  workers = 1L,
  host = NULL,
  port = NULL,
  tls = crew::crew_tls(mode = "automatic"),
  tls_enable = NULL,
  tls_config = NULL,
  seconds_interval = 0.25,
  seconds_timeout = 60,
  seconds_launch = 86400,
  seconds_idle = Inf,
  seconds_wall = Inf,
  seconds_exit = NULL,
  retry_tasks = TRUE,
  tasks_max = Inf,
  tasks_timers = 0L,
  reset_globals = TRUE,
  reset_packages = FALSE,
  reset_options = FALSE,
  garbage_collection = FALSE,
  launch_max = 5L,
  verbose = FALSE,
  command_submit = as.character(Sys.which("sbatch")),
  command_terminate = as.character(Sys.which("scancel")),
  command_delete = NULL,
  script_directory = tempdir(),
  script_lines = character(0L),
  slurm_log_output = "/dev/null",
  slurm_log_error = "/dev/null",
  slurm_memory_gigabytes_per_cpu = NULL,
  slurm_cpus_per_task = NULL,
  slurm_time_minutes = 1440,
  slurm_partition = NULL
)
```

**Arguments**

- **name**  
  Name of the client object. If NULL, a name is automatically generated.
crew_controller_slurm

workers  Integer, maximum number of parallel workers to run.
host     IP address of the mirai client to send and receive tasks. If NULL, the host defaults to the local IP address.
port     TCP port to listen for the workers. If NULL, then an available ephemeral port is automatically chosen.
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.
seconds_interval  Number of seconds between polling intervals waiting for certain internal synchronous operations to complete, such as checking mirai::status().
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     TRUE to automatically retry a task in the event of an unexpected worker exit. FALSE to give up on the first exit and return a mirai error code (code number 19). TRUE (default) is recommended in most situations. Use FALSE for debugging purposes, e.g. to confirm that a task is causing a worker to run out of memory or crash in some other way.
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().
resetGlobals    TRUE to reset global environment variables between tasks, FALSE to leave them alone.
resetPackages   TRUE to unload any packages loaded during a task (runs between each task), FALSE to leave packages alone.
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.

garbage_collection  TRUE to run garbage collection between tasks, FALSE to skip.

launch_max  Positive integer of length 1, maximum allowed consecutive launch attempts which do not complete any tasks. Enforced on a worker-by-worker basis. The futile launch count resets to back 0 for each worker that completes a task. It is recommended to set launch_max above 0 because sometimes workers are unproductive under perfectly ordinary circumstances. But launch_max should still be small enough to detect errors in the underlying platform.

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.

custom_launch_attempts  Positive integer of length 1, maximum allowed consecutive worker launch attempts which do not complete any tasks. Enforced on a worker-by-worker basis. The futile launch count resets to back 0 for each worker that completes a task. It is recommended to set launch_max above 0 because sometimes workers are unproductive under perfectly ordinary circumstances. But launch_max should still be small enough to detect errors in the underlying platform.

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

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.

slurm_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. slurm_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 slurm_log_output = NULL to omit this line from the job script.

slurm_log_error  Character of length 1, file pattern for standard error. slurm_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 slurm_log_error = NULL to omit this line from the job script.
crew_controller_slurm

slurm_memory_gigabytes_per_cpu
Positive numeric of length 1 with the gigabytes of memory required per CPU.
slurm_memory_gigabytes_per_cpu = 2.40123 translates to a line of 

```
#SBATCH --mem-per-cpu=2041M
```
in the SLURM job script. slurm_memory_gigabytes_per_cpu = NULL omits this line.

slurm_cpus_per_task
Optional positive integer of length 1, number of CPUs for the worker. 

```
#SBATCH --cpus-per-task=4
```
slurm_cpus_per_task = NULL omits this line.

slurm_time_minutes
Numeric of length 1, number of minutes to designate as the wall time of crew each worker instance on the SLURM cluster.

```
#SBATCH --time=60
```
slurm_time_minutes = NULL omits this line.

slurm_partition
Character of length 1, name of the SLURM partition to create workers on.

```
#SBATCH --partition=partition1,partition2
```
slurm_partition = NULL omits this line.

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()

Examples

```r
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**

```r
crew_launcher_lsf(
  name = NULL,
  seconds_interval = 0.5,
  seconds_timeout = 60,
  seconds_launch = 86400,
  seconds_idle = Inf,
  seconds_wall = Inf,
  tasks_max = Inf,
  tasks_timers = 0L,
  reset_globals = TRUE,
  reset_packages = FALSE,
  reset_options = FALSE,
  garbage_collection = FALSE,
  launch_max = 5L,
  tls = crew::crew_tls(mode = "automatic"),
  verbose = FALSE,
  command_submit = as.character(Sys.which("bsub")),
  command_terminate = as.character(Sys.which("bkill")),
  command_delete = NULL,
  script_directory = tempdir(),
  script_lines = character(0L),
  lsf_cwd = getwd(),
  lsf_log_output = "/dev/null",
  lsf_log_error = "/dev/null",
  lsf_memory_gigabytes_limit = NULL,
  lsf_memory_gigabytes_required = NULL,
  lsf_cores = NULL
)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the launcher.</td>
</tr>
<tr>
<td>seconds_interval</td>
<td>Number of seconds between polling intervals waiting for certain internal synchronous operations to complete, such as checking <code>mirai::status()</code>.</td>
</tr>
<tr>
<td>seconds_timeout</td>
<td>Number of seconds until timing out while waiting for certain synchronous operations to complete, such as checking <code>mirai::status()</code>.</td>
</tr>
</tbody>
</table>
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 micro-manage 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 micro-manage 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 unload any packages loaded during a task (runs between each task), FALSE to leave packages alone.

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.

garbage_collection  TRUE to run garbage collection between tasks, FALSE to skip.

launch_max  Positive integer of length 1, maximum allowed consecutive launch attempts which do not complete any tasks. Enforced on a worker-by-worker basis. The futile launch count resets to back 0 for each worker that completes a task. It is recommended to set launch_max above 0 because sometimes workers are unproductive under perfectly ordinary circumstances. But launch_max should still be small enough to detect errors in the underlying platform.

tls  A TLS configuration object from crew_tls().

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

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.

lsf_cwd  Character of length 1, directory to launch the worker from (as opposed to the system default). lsf.cwd = "/home" translates to a line of #BSUB -cwd /home in the LSF job script. lsf.cwd = getwd() is the default, which launches workers from the current working directory. Set lsf.cwd = NULL to omit this line from the job script.

lsf_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. lsf_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 lsf_log_output = NULL to omit this line from the job script.

lsf_log_error  Character of length 1, file pattern for standard error. lsf_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 lsf_log_error = NULL to omit this line from the job script.

lsf_memory_gigabytes_limit  Positive numeric of length 1 with the limit in gigabytes lsf_memory_gigabytes_limit = 4 translates to a line of #BSUB -M 4G in the LSF job script. lsf_memory_gigabytes_limit = NULL omits this line.

lsf_memory_gigabytes_required  Positive numeric of length 1 with the memory requirement in gigabytes lsf_memory_gigabytes_required = 4 translates to a line of #BSUB -R 'rusage[mem=4G]' in the LSF job script. lsf_memory_gigabytes_required = NULL omits this line.

lsf_cores  Optional positive integer of length 1, number of CPU cores for the worker. lsf_cores = 4 translates to a line of #BSUB -n 4 in the LSF job script. lsf_cores = NULL omits this line.

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.
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.

See Also

Other lsf: crew_class_launcher_lsf, crew_controller_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,
  seconds_interval = 0.5,
  seconds_timeout = 60,
  seconds_launch = 86400,
  seconds_idle = Inf,
  seconds_wall = Inf,
  tasks_max = Inf,
  tasks_timers = 0L,
  reset_globals = TRUE,
  reset_packages = FALSE,
  reset_options = FALSE,
  garbage_collection = FALSE,
  launch_max = 5L,
  tls = crew::crew_tls(mode = "automatic"),
  verbose = FALSE,
  command_submit = as.character(Sys.which("qsub")),
  command_terminate = as.character(Sys.which("qdel")),
  command_delete = NULL,
  script_directory = tempdir(),
  script_lines = character(0L),
  pbs_cwd = TRUE,
  pbs_log_output = "/dev/null",
  pbs_log_error = NULL,
  pbs_log_join = TRUE,
  pbs_memory_gigabytes_required = NULL,
  pbs_cores = NULL,
)```

pbs_walltime_hours = 12
)

Arguments

name
Name of the launcher.

seconds_interval
Number of seconds between polling intervals waiting for certain internal synchronous operations to complete, such as checking mirai::status().

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

reset_packages
TRUE to unload any packages loaded during a task (runs between each task), FALSE to leave packages alone.

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.

garbage_collection
TRUE to run garbage collection between tasks, FALSE to skip.

launch_max
Positive integer of length 1, maximum allowed consecutive launch attempts which do not complete any tasks. Enforced on a worker-by-worker basis. The futile launch count resets to back 0 for each worker that completes a task. It is recommended to set launch_max above 0 because sometimes workers are unproductive under perfectly ordinary circumstances. But launch_max should still be small enough to detect errors in the underlying platform.
tls
  A TLS configuration object from \texttt{crew_tls()}. 

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 \texttt{""} 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 program-
ing capabilities of \texttt{mirai}. Still, if you set \texttt{command_terminate = ""}, you are
assuming extra responsibility for manually monitoring your jobs on the cluster
and manually terminating jobs as appropriate.

command_delete
  Deprecated on 2024-01-08 (version 0.1.4.9001). Use \texttt{command_terminate}
instead.

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. \texttt{tempdir()} is the default, but it might not work for
some systems. \texttt{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 \texttt{script_lines = "module
load R"} if your cluster supports R through an environment module.

pbs_cwd
  Logical of length 1, whether to set the working directory of the worker to the
working directory it was launched from. \texttt{pbs_cwd = TRUE} translates to a line
of \texttt{cd "$PBS_O_WORKDIR"} in the job script. This line is inserted after the content
of \texttt{script_lines} to make sure the \#PBS directives are above system commands.
\texttt{pbs_cwd = FALSE} omits this line.

pbs_log_output
  Character of length 1, file or directory path to PBS worker log files for standard
output. \texttt{pbs_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.

pbs_log_error
  Character of length 1, file or directory path to PBS worker log files for standard
error. \texttt{pbs_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.

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

pbs_memory_gigabytes_required
  Optional positive numeric of length 1 with the gigabytes of memory required to
run the worker. \texttt{pbs_memory_gigabytes_required = 2.4} translates to a line of
\#PBS \ -l \texttt{mem=2.4gb} in the PBS job script. \texttt{pbs_memory_gigabytes_required
= NULL} omits this line.
crew_launcher_sge

Optional positive integer of length 1, number of cores per worker ("slots" in PBS lingo). `pbs_cores = 4` translates to a line of `#PBS -l ppn=4` in the PBS job script. `pbs_cores = NULL` omits this line.

Numeric of length 1 with the hours of wall time to request for the job. `pbs_walltime_hours = 23` translates to a line of `#PBS -l walltime=23:00:00` in the job script. `pbs_walltime_hours = NULL` omits this line.

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.

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.

Other pbs: crew_class_launcher_pbs, crew_controller_pbs()

crew_launcher_sge

Create a launcher with Sun Grid Engine (SGE) workers.

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

Usage

```r
crew_launcher_sge(
  name = NULL,
  seconds_interval = 0.5,
  seconds_timeout = 60,
  seconds_launch = 86400,
  seconds_idle = Inf,
  seconds_wall = Inf,
)```

[Maturing]
tasks_max = Inf,
tasks_timers = 0L,
resetGlobals = TRUE,
reset_packages = FALSE,
reset_options = FALSE,
garbage_collection = FALSE,
launch_max = 5L,

tls = crew::crew_tls(mode = "automatic"),
verbose = FALSE,
command_submit = as.character(Sys.which("qsub")),
command_terminate = as.character(Sys.which("qdel")),
command_delete = NULL,
script_directory = tempdir(),
script_lines = character(0L),
sge_cwd = TRUE,
sge_envvars = FALSE,
sge_log_output = "/dev/null",
sge_log_error = NULL,
sge_log_join = TRUE,
sge_memory_gigabytes_limit = NULL,
sge_memory_gigabytes_required = NULL,
sge_cores = NULL,
sge_gpu = NULL
)

Arguments

name
Name of the launcher.

seconds_interval
Number of seconds between polling intervals waiting for certain internal syn-
chronous operations to complete, such as checking mirai::status().

seconds_timeout
Number of seconds until timing out while waiting for certain synchronous oper-
ations 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 micro-
manage 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().

**resetGlobals**
TRUE to reset global environment variables between tasks, FALSE to leave them alone.

**resetPackages**
TRUE to unload any packages loaded during a task (runs between each task), FALSE to leave packages alone.

**resetOptions**
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.

**garbageCollection**
TRUE to run garbage collection between tasks, FALSE to skip.

**launchMax**
Positive integer of length 1, maximum allowed consecutive launch attempts which do not complete any tasks. Enforced on a worker-by-worker basis. The futile launch count resets to back 0 for each worker that completes a task. It is recommended to set launch_max above 0 because sometimes workers are unproductive under perfectly ordinary circumstances. But launch_max should still be small enough to detect errors in the underlying platform.

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

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

**commandSubmit**
Character of length 1, file path to the executable to submit a worker job.

**commandTerminate**
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.

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

**scriptDirectory**
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.

**scriptLines**
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.

**sgecwd**
Logical of length 1, whether to launch the worker from the current working directory (as opposed to the user home directory). sge_cwd = TRUE translates to a line of #$ -cwd in the SGE job script. sge_cwd = FALSE omits this line.
sge_envvars  Logical of length 1, whether to forward the environment variables of the current session to the SGE worker. sge_envvars = TRUE translates to a line of #$ -V in the SGE job script. sge_envvars = FALSE omits this line.

sge_log_output  Character of length 1, file or directory path to SGE worker log files for standard output. sge_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.

sge_log_error  Character of length 1, file or directory path to SGE worker log files for standard error. sge_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.

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

sge_memory_gigabytes_limit  Optional numeric of length 1 with the 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. sge_memory_gigabytes_limit = 5.7" translates to a line of "#$ -l h_rss=5.7G" in the SGE job script. sge_memory_gigabytes_limit = NULL omits this line.

sge_memory_gigabytes_required  Optional positive numeric of length 1 with the gigabytes of memory required to run the worker. sge_memory_gigabytes_required = 2.4 translates to a line of #$ -1 m_mem_free=2.4G in the SGE job script. sge_memory_gigabytes_required = NULL omits this line.

sge_cores  Optional positive integer of length 1, number of cores per worker ("slots" in SGE lingo). sge_cores = 4 translates to a line of #$ -pe smp 4 in the SGE job script. sge_cores = NULL omits this line.

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

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_launcher_slurm  [Experimental] Create a launcher with SLURM workers.

Description

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

Usage

```r
crew_launcher_slurm(
  name = NULL,
  seconds_interval = 0.5,
  seconds_timeout = 60,
  seconds_launch = 86400,
  seconds_idle = Inf,
  seconds_wall = Inf,
  tasks_max = Inf,
  tasks_timers = 0L,
  reset_globals = TRUE,
  reset_packages = FALSE,
  reset_options = FALSE,
  garbage_collection = FALSE,
  launch_max = 5L,
  tls = crew::crew_tls(mode = "automatic"),
  verbose = FALSE,
  command_submit = as.character(Sys.which("sbatch")),
  command_terminate = as.character(Sys.which("scancel")),
  command_delete = NULL,
  script_directory = tempdir(),
  script_lines = character(0L),
  slurm_log_output = "/dev/null",
  slurm_log_error = "/dev/null",
  slurm_memory_gigabytes_per_cpu = NULL,
  slurm_cpus_per_task = NULL,
  slurm_time_minutes = 1440,
  slurm_partition = NULL
)
```
Arguments

- **name**: Name of the launcher.

- **seconds_interval**: Number of seconds between polling intervals waiting for certain internal synchronous operations to complete, such as checking `mirai::status()`.

- **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()`.

- **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, so please allow enough idle time for a new worker to be delegated a new task.

- **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 unload any packages loaded during a task (runs between each task), FALSE to leave packages alone.

- **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.

- **garbage_collection**: TRUE to run garbage collection between tasks, FALSE to skip.

- **launch_max**: Positive integer of length 1, maximum allowed consecutive launch attempts which do not complete any tasks. Enforced on a worker-by-worker basis. The futile launch count resets to back 0 for each worker that completes a task. It is recommended to set `launch_max` above 0 because sometimes workers are unproductive under perfectly ordinary circumstances. But `launch_max` should still be small enough to detect errors in the underlying platform.

- **tls**: A TLS configuration object from `crew_tls()`.

- **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.

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

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.

slurm_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. slurm_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 slurm_log_output = NULL to omit this line from the job script.

slurm_log_error  Character of length 1, file pattern for standard error. slurm_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 slurm_log_error = NULL to omit this line from the job script.

slurm_memory_gigabytes_per_cpu  Positive numeric of length 1 with the gigabytes of memory required per CPU. slurm_memory_gigabytes_per_cpu = 2.40123 translates to a line of #SBATCH --mem-per-cpu=2041M in the SLURM job script. slurm_memory_gigabytes_per_cpu = NULL omits this line.

slurm_cpus_per_task  Optional positive integer of length 1, number of CPUs for the worker. slurm_cpus_per_task = 4 translates to a line of #SBATCH --cpus-per-task=4 in the SLURM job script. slurm_cpus_per_task = NULL omits this line.

slurm_time_minutes  Numeric of length 1, number of minutes to designate as the wall time of crew each worker instance on the SLURM cluster. slurm_time_minutes = 60 translates to a line of #SBATCH --time=60 in the SLURM job script. slurm_time_minutes = NULL omits this line.
crew_monitor_sge

slurm_partition

Character of length 1, name of the SLURM partition to create workers on. 
slurm_partition = "partition1,partition2" translates to a line of 
#SBATCH --partition=partition1,partition2 
in the SLURM job script. slurm_partition = NULL omits this line.

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

command_list Character of length 1, file path to the executable to list jobs.

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.

See Also
Other sge: crew_class_launcher_sge, crew_class_monitor_sge, crew_controller_sge(), crew_launcher_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 Logical, whether to see console output and error messages when submitting worker.

command_list Character of length 1, file path to the executable to list jobs.

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.
See Also

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

** help  
crew.cluster-package, 2

** lsf  
crew_class_launcher_lsf, 3  
crew_controller_lsf, 18  
crew_launcher_lsf, 34

** pbs  
crew_class_launcher_pbs, 6  
crew_controller_pbs, 22  
crew_launcher_pbs, 37

** sge  
crew_class_launcher_sge, 9  
crew_class_monitor_sge, 16  
crew_controller_sge, 26  
crew_launcher_sge, 40  
crew_monitor_sge, 47

** slurm  
crew_class_launcher_slurm, 12  
crew_class_monitor_slurm, 17  
crew_controller_slurm, 30  
crew_launcher_slurm, 44  
crew_monitor_slurm, 48

crew.cluster-package, 2  
crew.cluster::crew_class_launcher_cluster,  
3, 6, 10, 13  
crew.cluster::crew_class_monitor_cluster,  
16, 17  
crew::crew_class_launcher, 3, 6, 10, 13  
crew_class_launcher_lsf, 3, 21, 37  
crew_class_launcher_pbs, 6, 25, 40  
crew_class_launcher_sge, 9, 16, 29, 44, 48  
crew_class_monitor_slurm, 12, 18, 33, 47,  
49  
crew_class_monitor_sge, 12, 16, 29, 44, 48  
crew_class_monitor_slurm, 15, 17, 33, 47,  
49  
crew_controller_lsf, 6, 18, 37  
crew_controller_pbs, 9, 22, 40  
crew_controller_sge, 12, 16, 26, 44, 48  
crew_controller_slurm, 15, 18, 30, 47, 49  
crew_launcher_lsf, 6, 21, 34  
crew_launcher_lsf(), 3–5  
crew_launcher_pbs, 9, 25, 37  
crew_launcher_pbs(), 6–8  
crew_launcher_sge, 12, 16, 29, 40, 48  
crew_launcher_sge(), 8–11, 14  
crew_launcher_slurm, 15, 18, 33, 44, 49  
crew_launcher_slurm(), 8, 11, 13, 14  
crew_monitor_sge, 12, 16, 29, 44, 47  
crew_monitor_sge(), 16  
crew_monitor_slurm, 15, 18, 33, 47, 48  
crew_monitor_slurm(), 17  
crew_tls(), 19, 23, 27, 31, 35, 39, 42, 45