Package 'future.batchtools'

May 8, 2026

Version 0.22.0

Depends R (>= 3.2.0), future (>= 1.58.0)

Imports parallelly, batchtools (>= 0.9.17), utils, checkmate, stringi

Suggests globals, future.apply, listenv, markdown, R.rsp

VignetteBuilder R.rsp

Title A Future API for Parallel and Distributed Processing using 'batchtools'

Description Implementation of the Future API <doi:10.32614/RJ-2021-048> on top of the 'batchtools' package.
This allows you to process futures, as defined by the 'future' package, in parallel out of the box, not only on your local machine or ad-hoc cluster of machines, but also via high-performance compute ('HPC') job schedulers such as 'LSF', 'OpenLava', 'Slurm', 'SGE', and 'TORQUE' / 'PBS', e.g. 'y <- future.apply::future_lapply(files, FUN = process)'.

License LGPL (>= 2.1)

LazyLoad TRUE

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

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

Language en-US

Encoding UTF-8

RoxygenNote 7.3.3

NeedsCompilation no

Author Henrik Bengtsson [aut, cre, cph] (ORCID: <https://orcid.org/0000-0002-7579-5165>)

Maintainer Henrik Bengtsson <henrikb@braju.com>

Repository CRAN

Date/Publication 2026-04-24 23:00:02 UTC

Contents

`batchtools_bash` *A batchtools Bash backend that resolves futures sequentially via a Bash template script*

Description

The `batchtools_bash` backend was added to illustrate how to write a custom **future.batchtools** backend that uses a templated job script. Please see the source code, for details.

Usage

```
batchtools_bash(
  ...,
  template = "bash",
  fs.latency = 0,
  resources = list(),
  delete = getOption("future.batchtools.delete", "on-success")
)

makeClusterFunctionsBash(template = "bash", fs.latency = 0, ...)
```

Arguments

`template`	(optional) Name of job-script template to be searched for by `batchtools::findTemplateFile()`. If not found, it defaults to the `templates/bash.tmpl` part of this package (see below).
`fs.latency`	[`numeric(1)`] Expected maximum latency of the file system, in seconds. Set to a positive number for network file systems like NFS which enables more robust (but also more expensive) mechanisms to access files and directories. Usually safe to set to `0` to disable the heuristic, e.g. if you are working on a local file system.

`resources` (optional) A named list passed to the **batchtools** job-script template as variable `resources`. This is based on how `batchtools::submitJobs()` works, with the exception for specially reserved names defined by the **future.batchtools** package;

- `resources[["details"]]`, if TRUE, results in the job script outputting job details and job summaries at the beginning and at the end.
- `resources[["startup"]]` and `resources[["shutdown"]]` are character vectors of shell code to be injected to the job script as-is.
- `resources[["modules"]]` is character vector of Linux environment modules to be loaded.
- `resources[["envs"]]`, is an optional names character vector specifying environment variables to be set.
- `resources[["rscript"]]` is an optional character vector specifying how the 'Rscript' is launched. The `resources[["rscript_args"]]` field is an optional character vector specifying the 'Rscript' command-line arguments.
- `resources[["asis"]]` is a character vector that are passed as-is to the job script and are injected as job resource declarations.
- All remaining `resources` named elements are injected as named resource specification for the scheduler.

`delete` Controls if and when the batchtools job registry folder is deleted. If `"on-success"` (default), it is deleted if the future was resolved successfully *and* the expression did not produce an error. If `"never"`, then it is never deleted. If `"always"`, then it is always deleted.

`...` Not used.

Details

Batchtools Bash futures use **batchtools** cluster functions created by `makeClusterFunctionsBash()` and requires that `bash` is installed on the current machine and the `timeout` command is available.

The default template script `templates/bash.tmpl` can be found in:

```
system.file("templates", "bash.tmpl", package = "future.batchtools")
```

and comprises:

```
#!/bin/bash
##############################################################################
# A batchtools launch script template
#
# Author: Henrik Bengtsson
##############################################################################

## Bash settings
set -e          # exit on error
set -u          # error on unset variables
set -o pipefail # fail a pipeline if any command fails
```

```
trap 'echo "ERROR: future.batchtools job script failed on line $LINENO" >&2; exit 1' ERR

## Redirect stdout and stderr to the batchtools log file
exec > <%= log.file %> 2>&1

<%
  ## Shell "startup" code to evaluate
  startup <- resources[["startup"]]
  resources[["startup"]] <- NULL

  ## Shell "shutdown" code to evaluate
  shutdown <- resources[["shutdown"]]
  resources[["shutdown"]] <- NULL

  ## Environment modules specifications
  modules <- resources[["modules"]]
  resources[["modules"]] <- NULL

  ## Environment variables to be set
  envs <- resources[["envs"]]
  if (length(envs) > 0) {
    stopifnot(is.character(envs), !is.null(names(envs)))
  }
  resources[["envs"]] <- NULL

  ## Custom "Rscript" command and Rscript arguments
  rscript <- resources[["rscript"]]
  if (is.null(rscript)) {
    rscript <- "Rscript"
  } else if (length(rscript) == 0 || !nzchar(rscript)[1]) {
    stop("Argument 'resources' specifies an empty 'rscript' field")
  }
  resources[["rscript"]] <- NULL

  ## Maximum runtime?
  timeout <- resources[["timeout"]]
  resources[["timeout"]] <- NULL
  if (length(timeout) > 0) {
      rscript <- c("timeout", timeout, rscript)
  }
  rscript_args <- resources[["rscript_args"]]
  resources[["rscript_args"]] <- NULL
  rscript_call <- paste(c(rscript, rscript_args), collapse = " ")
%>

<% if (length(startup) > 0) {
  writeLines(startup)
} %>
```

```
<% if (length(modules) > 0) {
  writeLines(c(
    'echo "Load environment modules:"',
    sprintf('echo "- modules: %s"', paste(modules, collapse = ", ")),
    sprintf("module load %s", paste(modules, collapse = " ")),
    "module list"
  ))
} %>

<% if (length(envs) > 0) {
  writeLines(c(
    sprintf("echo 'Setting environment variables: [n=%d]'", length(envs)),
    sprintf("echo ' - %s=%s'", names(envs), shQuote(envs)),
    sprintf("export %s=%s", names(envs), shQuote(envs))
  ))
} %>

echo "Session information:"
echo "- timestamp: $(date +"%Y-%m-%d %H:%M:%S%z")"
echo "- hostname: $(hostname)"
echo "- Rscript: <%= paste(rscript, collapse = " ") %>"
echo "- Rscript args: <%= paste(rscript_args, collapse = " ") %>"
echo "- Rscript call: <%= rscript_call %>"
if ! command -v <%= rscript[1] %> &> /dev/null; then
   >&2 echo "ERROR: Argument 'resources' specifies a non-existing 'Rscript' launch command: <%= rscript[
    exit 1
fi
echo "- Rscript version: $(<%= paste(rscript, collapse = " ") %> --version)"
echo "- R_LIBS_USER=${R_LIBS_USER:-<not set>}"
echo "- R_LIBS_SITE=${R_LIBS_SITE:-<not set>}"
echo "- R_LIBS=${R_LIBS:-<not set>}"
echo "- Rscript library paths: $(<%= rscript_call %> -e "cat(shQuote(.libPaths()), sep = ' ')")"
echo

# Launch R and evaluate the batchtools R job
echo "Calling 'batchtools::doJobCollection()' ..."
echo "- job name: '<%= job.name %>'"
echo "- job log file: '<%= log.file %>'"
echo "- job uri: '<%= uri %>'"
<%= rscript_call %> -e 'batchtools::doJobCollection("<%= uri %>")'
res=$?
echo " - exit code: ${res}"
echo "Calling 'batchtools::doJobCollection()' ... done"
echo

<% if (length(shutdown) > 0) {
  writeLines(shutdown)
```

```
} %>

echo "End time: $(date +"%Y-%m-%d %H:%M:%S%z")"

## Relay the exit code from Rscript
exit "${res}"
```

Value

Nothing.

`makeClusterFunctionsBash()` returns a ClusterFunctions object.

Examples

```
library(future)

# Limit runtime to 30 seconds per future
plan(future.batchtools::batchtools_bash, resources = list(runtime = 30))

message("Main process ID: ", Sys.getpid())

f <- future({
  data.frame(
    hostname = Sys.info()[["nodename"]],
          os = Sys.info()[["sysname"]],
       cores = unname(parallelly::availableCores()),
         pid = Sys.getpid(),
     modules = Sys.getenv("LOADEDMODULES")
  )
})
info <- value(f)
print(info)
```

---

batchtools_interactive

A batchtools backend that resolves futures sequentially in the current R session

---

Description

The batchtools interactive backend is useful for verifying parts of your **batchtools** setup locally, while still being able to do interactive debugging.

Usage

```
batchtools_interactive(
  ...,
  fs.latency = 0,
```

```
  delete = getOption("future.batchtools.delete", "on-success")
)
```

Arguments

`fs.latency`	`[numeric(1)]` Expected maximum latency of the file system, in seconds. Set to a positive number for network file systems like NFS which enables more robust (but also more expensive) mechanisms to access files and directories. Usually safe to set to `0` to disable the heuristic, e.g. if you are working on a local file system.
`delete`	Controls if and when the batchtools job registry folder is deleted. If `"on-success"` (default), it is deleted if the future was resolved successfully *and* the expression did not produce an error. If `"never"`, then it is never deleted. If `"always"`, then it is always deleted.
`...`	Not used.

Details

Batchtools interactive futures use **batchtools** cluster functions created by `batchtools::makeClusterFunctionsInteracti` with `external = TRUE`.

An alternative to the batchtools interactive backend is to use `plan(future::sequential)`, which is a faster way process futures sequentially and that also can be debugged interactively.

Value

Nothing.

Examples

```
library(future)
plan(future.batchtools::batchtools_interactive)

message("Main process ID: ", Sys.getpid())

f <- future({
  data.frame(
    hostname = Sys.info()[["nodename"]],
          os = Sys.info()[["sysname"]],
       cores = unname(parallelly::availableCores()),
         pid = Sys.getpid(),
     modules = Sys.getenv("LOADEDMODULES")
  )
})
info <- value(f)
print(info)
```

batchtools_local *A batchtools backend that resolves futures sequentially in transient background R sessions*

Description

The batchtools local backend is useful for verifying parts of your **batchtools** setup locally, before using a more advanced backend such as the job-scheduler backends.

Usage

```
batchtools_local(
  ...,
  fs.latency = 0,
  delete = getOption("future.batchtools.delete", "on-success")
)
```

Arguments

`fs.latency`	`[numeric(1)]` Expected maximum latency of the file system, in seconds. Set to a positive number for network file systems like NFS which enables more robust (but also more expensive) mechanisms to access files and directories. Usually safe to set to `0` to disable the heuristic, e.g. if you are working on a local file system.
`delete`	Controls if and when the batchtools job registry folder is deleted. If `"on-success"` (default), it is deleted if the future was resolved successfully *and* the expression did not produce an error. If `"never"`, then it is never deleted. If `"always"`, then it is always deleted.
`...`	Not used.

Details

Batchtools local futures use **batchtools** cluster functions created by `batchtools::makeClusterFunctionsInteractive()` with `external = TRUE`.

An alternative to the batchtools interactive backend is to use `plan(future::cluster, workers = I(1))`.

Value

Nothing.

Examples

```
library(future)
plan(future.batchtools::batchtools_local)

message("Main process ID: ", Sys.getpid())
```

```
f <- future({
  data.frame(
    hostname = Sys.info()[["nodename"]],
          os = Sys.info()[["sysname"]],
       cores = unname(parallelly::availableCores()),
         pid = Sys.getpid(),
     modules = Sys.getenv("LOADEDMODULES")
  )
})
info <- value(f)
print(info)
```

`batchtools_lsf` *A batchtools LSF backend resolves futures in parallel via a Load Sharing Facility (LSF) job scheduler*

Description

A batchtools LSF backend resolves futures in parallel via a Load Sharing Facility (LSF) job scheduler

Usage

```
batchtools_lsf(
  ...,
  template = "lsf",
  scheduler.latency = 1,
  fs.latency = 65,
  resources = list(),
  delete = getOption("future.batchtools.delete", "on-success"),
  workers = getOption("future.batchtools.workers", default = 100L)
)
```

Arguments

`template`	(optional) Name of job-script template to be searched for by `batchtools::findTemplateFile()`. If not found, it defaults to the `templates/lsf.tmpl` part of this package (see below).
`scheduler.latency`	[`numeric(1)`] Time to sleep after important interactions with the scheduler to ensure a sane state. Currently only triggered after calling `submitJobs`.
`fs.latency`	[`numeric(1)`] Expected maximum latency of the file system, in seconds. Set to a positive number for network file systems like NFS which enables more robust (but also more expensive) mechanisms to access files and directories. Usually safe to set to `0` to disable the heuristic, e.g. if you are working on a local file system.

`resources` (optional) A named list passed to the **batchtools** job-script template as variable `resources`. This is based on how `batchtools::submitJobs()` works, with the exception for specially reserved names defined by the **future.batchtools** package;

- `resources[["details"]]`, if TRUE, results in the job script outputting job details and job summaries at the beginning and at the end.
- `resources[["startup"]]` and `resources[["shutdown"]]` are character vectors of shell code to be injected to the job script as-is.
- `resources[["modules"]]` is character vector of Linux environment modules to be loaded.
- `resources[["envs"]]`, is an optional names character vector specifying environment variables to be set.
- `resources[["rscript"]]` is an optional character vector specifying how the 'Rscript' is launched. The `resources[["rscript_args"]]` field is an optional character vector specifying the 'Rscript' command-line arguments.
- `resources[["asis"]]` is a character vector that are passed as-is to the job script and are injected as job resource declarations.
- All remaining `resources` named elements are injected as named resource specification for the scheduler.

`delete` Controls if and when the batchtools job registry folder is deleted. If `"on-success"` (default), it is deleted if the future was resolved successfully *and* the expression did not produce an error. If `"never"`, then it is never deleted. If `"always"`, then it is always deleted.

`workers` The maximum number of workers the batchtools backend may use at any time, which for HPC schedulers corresponds to the maximum number of queued jobs. The default is `getOption("future.batchtools.workers", 100)`.

`...` Not used.

Details

Batchtools Load Sharing Facility (LSF) futures use **batchtools** cluster functions created by `batchtools::makeClusterFunc` which are used to interact with the LSF job scheduler. This requires that LSF commands `bsub`, `bjobs`, and `bkill` are available on the current machine.

The default template script `templates/lsf.tmpl` can be found in:

```
system.file("templates", "lsf.tmpl", package = "future.batchtools")
```

and comprises:

```
#!/bin/bash
##############################################################################
# A batchtools launch script template for LSF and OpenLava
#
# Author: Henrik Bengtsson
##############################################################################
```

```
## Job name
#BSUB -J <%= job.name %>

## Direct streams to logfile
#BSUB -o <%= log.file %>

## Resources needed
<%
  ## Should scheduler "details" be seen?
  details <- isTRUE(resources[["details"]])
  resources[["details"]] <- NULL

  ## Shell "startup" code to evaluate
  startup <- resources[["startup"]]
  resources[["startup"]] <- NULL

  ## Shell "shutdown" code to evaluate
  shutdown <- resources[["shutdown"]]
  resources[["shutdown"]] <- NULL

  ## Environment modules specifications
  modules <- resources[["modules"]]
  resources[["modules"]] <- NULL

  ## Environment variables to be set
  envs <- resources[["envs"]]
  if (length(envs) > 0) {
    stopifnot(is.character(envs), !is.null(names(envs)))
  }
  resources[["envs"]] <- NULL

  ## Custom "Rscript" command and Rscript arguments
  rscript <- resources[["rscript"]]
  if (is.null(rscript)) {
    rscript <- "Rscript"
  } else if (length(rscript) == 0 || !nzchar(rscript)[1]) {
    stop("Argument 'resources' specifies an empty 'rscript' field")
  }
  resources[["rscript"]] <- NULL
  rscript_args <- resources[["rscript_args"]]
  resources[["rscript_args"]] <- NULL
  rscript_call <- paste(c(rscript, rscript_args), collapse = " ")

  ## As-is resource specifications
  job_declarations <- resources[["asis"]]
  resources[["asis"]] <- NULL

  ## Remaining resources are assumed to be of type '<key>=<value>'
```

```
  opts <- unlist(resources, use.names = TRUE)
  opts <- sprintf("%s=%s", names(opts), opts)
 job_declarations <- sprintf("#BSUB %s", c(job_declarations, sprintf("-%s", opts)))
  writeLines(job_declarations)
%>

## Bash settings
set -e          # exit on error
set -u          # error on unset variables
set -o pipefail # fail a pipeline if any command fails
trap 'echo "ERROR: future.batchtools job script failed on line $LINENO" >&2; exit 1' ERR

<% if (length(job_declarations) > 0) {
  writeLines(c(
    "echo 'Job submission declarations:'",
    sprintf("echo '%s'", job_declarations),
    "echo"
  ))
} %>

<% if (details) { %>
if command -v bjobs > /dev/null; then
  echo "Job information:"
  bjobs -l "${LSB_JOBID}"
  echo
fi
<% } %>

<% if (length(startup) > 0) {
  writeLines(startup)
} %>

<% if (length(modules) > 0) {
  writeLines(c(
    "echo 'Load environment modules:'",
    sprintf("echo '- modules: %s'", paste(modules, collapse = ", ")),
    sprintf("module load %s", paste(modules, collapse = " ")),
    "module list"
  ))
} %>

<% if (length(envs) > 0) {
  writeLines(c(
    sprintf("echo 'Setting environment variables: [n=%d]'", length(envs)),
    sprintf("echo ' - %s=%s'", names(envs), shQuote(envs)),
    sprintf("export %s=%s", names(envs), shQuote(envs))
  ))
} %>
```

```
echo "Session information:"
echo "- timestamp: $(date +"%Y-%m-%d %H:%M:%S%z")"
echo "- hostname: $(hostname)"
echo "- Rscript call: <%= rscript_call %>"
if ! command -v <%= rscript[1] %> &> /dev/null; then
   >&2 echo "ERROR: Argument 'resources' specifies a non-existing 'Rscript' launch command: <%= rscript[
    exit 1
fi
echo "- Rscript version: $(<%= paste(rscript, collapse = " ") %> --version)"
echo "- R_LIBS_USER=${R_LIBS_USER:-<not set>}"
echo "- R_LIBS_SITE=${R_LIBS_SITE:-<not set>}"
echo "- R_LIBS=${R_LIBS:-<not set>}"
echo "- Rscript library paths: $(<%= rscript_call %> -e "cat(shQuote(.libPaths()), sep = ' ')")"
echo


## Launch R and evaluate the batchtools R job
echo "Calling 'batchtools::doJobCollection()' ..."
echo "- job name: '<%= job.name %>'"
echo "- job log file: '<%= log.file %>'"
echo "- job uri: '<%= uri %>'"
<%= rscript_call %> -e 'batchtools::doJobCollection("<%= uri %>")'
res=$?
echo " - exit code: ${res}"
echo "Calling 'batchtools::doJobCollection()' ... done"
echo

<% if (details) { %>
if command -v bjobs > /dev/null; then
  echo "Job summary:"
  bjobs -l "${LSB_JOBID}"
fi
<% } %>

<% if (length(shutdown) > 0) {
  writeLines(shutdown)
} %>

echo "End time: $(date +"%Y-%m-%d %H:%M:%S%z")"

## Relay the exit code from Rscript
exit "${res}"
```

Value

Nothing.

References

- https://en.wikipedia.org/wiki/IBM_Spectrum_LSF

Examples

```
library(future)

# Limit runtime to 10 minutes and total memory to 400 MiB per future,
# request a parallel environment with four slots on a single host.
# Submit to the 'freecycle' queue. Load environment modules 'r' and
# 'jags'. Report on job details at startup and at the end of the job.
plan(future.batchtools::batchtools_lsf, resources = list(
  W = "00:10:00", M = "400",
  asis = c("-n 4", "-R 'span[hosts=1]'", "-q freecycle"),
  modules = c("r", "jags"),
  details = TRUE
))

f <- future({
  data.frame(
     hostname = Sys.info()[["nodename"]],
           os = Sys.info()[["sysname"]],
    osVersion = utils::osVersion,
        cores = unname(parallelly::availableCores()),
      modules = Sys.getenv("LOADEDMODULES")
  )
})
info <- value(f)
print(info)
```

---

batchtools_multicore *A batchtools backend that resolves futures in parallel via forked background R processes*

Description

A batchtools backend that resolves futures in parallel via forked background R processes

Usage

```
batchtools_multicore(
  ...,
  workers = availableCores(constraints = "multicore"),
  fs.latency = 0,
  delete = getOption("future.batchtools.delete", "on-success")
)
```

Arguments

workers	The number of multicore processes to be available for concurrent batchtools multicore futures.
fs.latency	[numeric(1)] Expected maximum latency of the file system, in seconds. Set to a positive number for network file systems like NFS which enables more robust (but also more expensive) mechanisms to access files and directories. Usually safe to set to 0 to disable the heuristic, e.g. if you are working on a local file system.
delete	Controls if and when the batchtools job registry folder is deleted. If "on-success" (default), it is deleted if the future was resolved successfully *and* the expression did not produce an error. If "never", then it is never deleted. If "always", then it is always deleted.
...	Not used.

Details

Batchtools multicore futures use **batchtools** cluster functions created by batchtools::makeClusterFunctionsMulticore(with ncpus = workers.

An alternative to the batchtools multicore backend is to use plan(future::multicore).

Value

Nothing.

Examples

```
library(future)
plan(future.batchtools::batchtools_multicore, workers = 2)

message("Main process ID: ", Sys.getpid())

f <- future({
  data.frame(
    hostname = Sys.info()[["nodename"]],
          os = Sys.info()[["sysname"]],
       cores = unname(parallelly::availableCores()),
         pid = Sys.getpid(),
     modules = Sys.getenv("LOADEDMODULES")
  )
})
info <- value(f)
print(info)
```

batchtools_openlava *A batchtools OpenLava backend resolves futures in parallel via an OpenLava job scheduler*

Description

A batchtools OpenLava backend resolves futures in parallel via an OpenLava job scheduler

Usage

```
batchtools_openlava(
  ...,
  template = "openlava",
  scheduler.latency = 1,
  fs.latency = 65,
  resources = list(),
  delete = getOption("future.batchtools.delete", "on-success"),
  workers = getOption("future.batchtools.workers", default = 100L)
)
```

Arguments

`template`	(optional) Name of job-script template to be searched for by `batchtools::findTemplateFile()`. If not found, it defaults to the `templates/openlava.tmpl` part of this package (see below).
`scheduler.latency`	[`numeric(1)`] Time to sleep after important interactions with the scheduler to ensure a sane state. Currently only triggered after calling `submitJobs`.
`fs.latency`	[`numeric(1)`] Expected maximum latency of the file system, in seconds. Set to a positive number for network file systems like NFS which enables more robust (but also more expensive) mechanisms to access files and directories. Usually safe to set to `0` to disable the heuristic, e.g. if you are working on a local file system.
`resources`	(optional) A named list passed to the **batchtools** job-script template as variable `resources`. This is based on how `batchtools::submitJobs()` works, with the exception for specially reserved names defined by the **future.batchtools** package; • `resources[["details"]]`, if TRUE, results in the job script outputting job details and job summaries at the beginning and at the end. • `resources[["startup"]]` and `resources[["shutdown"]]` are character vectors of shell code to be injected to the job script as-is. • `resources[["modules"]]` is character vector of Linux environment modules to be loaded. • `resources[["envs"]]`, is an optional names character vector specifying environment variables to be set.

- `resources[["rscript"]]` is an optional character vector specifying how the 'Rscript' is launched. The `resources[["rscript_args"]]` field is an optional character vector specifying the 'Rscript' command-line arguments.
- `resources[["asis"]]` is a character vector that are passed as-is to the job script and are injected as job resource declarations.
- All remaining `resources` named elements are injected as named resource specification for the scheduler.

`delete` Controls if and when the batchtools job registry folder is deleted. If `"on-success"` (default), it is deleted if the future was resolved successfully *and* the expression did not produce an error. If `"never"`, then it is never deleted. If `"always"`, then it is always deleted.

`workers` The maximum number of workers the batchtools backend may use at any time, which for HPC schedulers corresponds to the maximum number of queued jobs. The default is `getOption("future.batchtools.workers", 100)`.

`...` Not used.

Details

Batchtools OpenLava futures use **batchtools** cluster functions created by `batchtools::makeClusterFunctionsOpenLava()` which are used to interact with the OpenLava job scheduler. This requires that OpenLava commands `bsub`, `bjobs`, and `bkill` are available on the current machine.

The default template script `templates/openlava.tmpl` can be found in:

```
system.file("templates", "openlava.tmpl", package = "future.batchtools")
```

and comprises:

```
#!/bin/bash
##############################################################################
# A batchtools launch script template for LSF and OpenLava
#
# Author: Henrik Bengtsson
##############################################################################

## Job name
#BSUB -J <%= job.name %>

## Direct streams to logfile
#BSUB -o <%= log.file %>

## Resources needed
<%
  ## Should scheduler "details" be seen?
  details <- isTRUE(resources[["details"]])
  resources[["details"]] <- NULL

  ## Shell "startup" code to evaluate
```

```
  startup <- resources[["startup"]]
  resources[["startup"]] <- NULL

  ## Shell "shutdown" code to evaluate
  shutdown <- resources[["shutdown"]]
  resources[["shutdown"]] <- NULL

  ## Environment modules specifications
  modules <- resources[["modules"]]
  resources[["modules"]] <- NULL

  ## Environment variables to be set
  envs <- resources[["envs"]]
  if (length(envs) > 0) {
    stopifnot(is.character(envs), !is.null(names(envs)))
  }
  resources[["envs"]] <- NULL

  ## Custom "Rscript" command and Rscript arguments
  rscript <- resources[["rscript"]]
  if (is.null(rscript)) {
    rscript <- "Rscript"
  } else if (length(rscript) == 0 || !nzchar(rscript)[1]) {
    stop("Argument 'resources' specifies an empty 'rscript' field")
  }
  resources[["rscript"]] <- NULL
  rscript_args <- resources[["rscript_args"]]
  resources[["rscript_args"]] <- NULL
  rscript_call <- paste(c(rscript, rscript_args), collapse = " ")

  ## As-is resource specifications
  job_declarations <- resources[["asis"]]
  resources[["asis"]] <- NULL

  ## Remaining resources are assumed to be of type '<key>=<value>'
  opts <- unlist(resources, use.names = TRUE)
  opts <- sprintf("%s=%s", names(opts), opts)
 job_declarations <- sprintf("#BSUB %s", c(job_declarations, sprintf("-%s", opts)))
  writeLines(job_declarations)
%>

## Bash settings
set -e          # exit on error
set -u          # error on unset variables
set -o pipefail # fail a pipeline if any command fails
trap 'echo "ERROR: future.batchtools job script failed on line $LINENO" >&2; exit 1' ERR

<% if (length(job_declarations) > 0) {
```

```
  writeLines(c(
    "echo 'Job submission declarations:'",
    sprintf("echo '%s'", job_declarations),
    "echo"
  ))
} %>

<% if (details) { %>
if command -v bjobs > /dev/null; then
  echo "Job information:"
  bjobs -l "${LSB_JOBID}"
  echo
fi
<% } %>

<% if (length(startup) > 0) {
  writeLines(startup)
} %>

<% if (length(modules) > 0) {
  writeLines(c(
    "echo 'Load environment modules:'",
    sprintf("echo '- modules: %s'", paste(modules, collapse = ", ")),
    sprintf("module load %s", paste(modules, collapse = " ")),
    "module list"
  ))
} %>

<% if (length(envs) > 0) {
  writeLines(c(
    sprintf("echo 'Setting environment variables: [n=%d]'", length(envs)),
    sprintf("echo ' - %s=%s'", names(envs), shQuote(envs)),
    sprintf("export %s=%s", names(envs), shQuote(envs))
  ))
} %>

echo "Session information:"
echo "- timestamp: $(date +"%Y-%m-%d %H:%M:%S%z")"
echo "- hostname: $(hostname)"
echo "- Rscript call: <%= rscript_call %>"
if ! command -v <%= rscript[1] %> &> /dev/null; then
   >&2 echo "ERROR: Argument 'resources' specifies a non-existing 'Rscript' launch command: <%= rscript[
    exit 1
fi
echo "- Rscript version: $(<%= paste(rscript, collapse = " ") %> --version)"
echo "- R_LIBS_USER=${R_LIBS_USER:-<not set>}"
echo "- R_LIBS_SITE=${R_LIBS_SITE:-<not set>}"
echo "- R_LIBS=${R_LIBS:-<not set>}"
```

```
echo "- Rscript library paths: $(<%= rscript_call %> -e "cat(shQuote(.libPaths()), sep = ' ')")"
echo


## Launch R and evaluate the batchtools R job
echo "Calling 'batchtools::doJobCollection()' ..."
echo "- job name: '<%= job.name %>'"
echo "- job log file: '<%= log.file %>'"
echo "- job uri: '<%= uri %>'"
<%= rscript_call %> -e 'batchtools::doJobCollection("<%= uri %>")'
res=$?
echo " - exit code: ${res}"
echo "Calling 'batchtools::doJobCollection()' ... done"
echo

<% if (details) { %>
if command -v bjobs > /dev/null; then
  echo "Job summary:"
  bjobs -l "${LSB_JOBID}"
fi
<% } %>

<% if (length(shutdown) > 0) {
  writeLines(shutdown)
} %>

echo "End time: $(date +"%Y-%m-%d %H:%M:%S%z")"

## Relay the exit code from Rscript
exit "${res}"
```

Value

Nothing.

References

- https://en.wikipedia.org/wiki/OpenLava

Examples

```
library(future)

# Limit runtime to 10 minutes and total memory to 400 MiB per future,
# request a parallel environment with four slots on a single host.
# Submit to the 'freecycle' queue. Load environment modules 'r' and
# 'jags'. Report on job details at startup and at the end of the job.
plan(future.batchtools::batchtools_openlava, resources = list(
  W = "00:10:00", M = "400",
  asis = c("-n 4", "-R 'span[hosts=1]'", "-q freecycle"),
```

```
  modules = c("r", "jags"),
  details = TRUE
))

f <- future({
  data.frame(
     hostname = Sys.info()[["nodename"]],
           os = Sys.info()[["sysname"]],
    osVersion = utils::osVersion,
        cores = unname(parallelly::availableCores()),
      modules = Sys.getenv("LOADEDMODULES")
  )
})
info <- value(f)
print(info)
```

`batchtools_sge` *A batchtools SGE backend resolves futures in parallel via a Sun/Son of/Oracle/Univa/Altair Grid Engine job scheduler*

Description

A batchtools SGE backend resolves futures in parallel via a Sun/Son of/Oracle/Univa/Altair Grid Engine job scheduler

Usage

```
batchtools_sge(
  ...,
  template = "sge",
  scheduler.latency = 1,
  fs.latency = 65,
  resources = list(),
  delete = getOption("future.batchtools.delete", "on-success"),
  workers = getOption("future.batchtools.workers", default = 100L)
)
```

Arguments

`template`	(optional) Name of job-script template to be searched for by `batchtools::findTemplateFile()`. If not found, it defaults to the `templates/sge.tmpl` part of this package (see below).
`scheduler.latency`	[`numeric(1)`] Time to sleep after important interactions with the scheduler to ensure a sane state. Currently only triggered after calling `submitJobs`.

`fs.latency` [numeric(1)]
Expected maximum latency of the file system, in seconds. Set to a positive number for network file systems like NFS which enables more robust (but also more expensive) mechanisms to access files and directories. Usually safe to set to `0` to disable the heuristic, e.g. if you are working on a local file system.

`resources` (optional) A named list passed to the **batchtools** job-script template as variable `resources`. This is based on how `batchtools::submitJobs()` works, with the exception for specially reserved names defined by the **future.batchtools** package;

- `resources[["details"]]`, if TRUE, results in the job script outputting job details and job summaries at the beginning and at the end.
- `resources[["startup"]]` and `resources[["shutdown"]]` are character vectors of shell code to be injected to the job script as-is.
- `resources[["modules"]]` is character vector of Linux environment modules to be loaded.
- `resources[["envs"]]`, is an optional names character vector specifying environment variables to be set.
- `resources[["rscript"]]` is an optional character vector specifying how the 'Rscript' is launched. The `resources[["rscript_args"]]` field is an optional character vector specifying the 'Rscript' command-line arguments.
- `resources[["asis"]]` is a character vector that are passed as-is to the job script and are injected as job resource declarations.
- All remaining `resources` named elements are injected as named resource specification for the scheduler.

`delete` Controls if and when the batchtools job registry folder is deleted. If `"on-success"` (default), it is deleted if the future was resolved successfully *and* the expression did not produce an error. If `"never"`, then it is never deleted. If `"always"`, then it is always deleted.

`workers` The maximum number of workers the batchtools backend may use at any time, which for HPC schedulers corresponds to the maximum number of queued jobs. The default is `getOption("future.batchtools.workers", 100)`.

`...` Not used.

Details

Batchtools SGE futures use **batchtools** cluster functions created by `batchtools::makeClusterFunctionsSGE()`, which are used to interact with the SGE job scheduler. This requires that SGE commands `qsub`, `qstat`, and `qdel` are available on the current machine.

The default template script `templates/sge.tmpl` can be found in:

```
system.file("templates", "sge.tmpl", package = "future.batchtools")
```

and comprises:

```
#!/bin/bash
##############################################################################
```

```
# A batchtools launch script template for SGE
#
# Author: Henrik Bengtsson
##############################################################################
## Shell
#$ -S /bin/bash

## Job name
#$ -N <%= job.name %>

## Direct streams to logfile
#$ -o <%= log.file %>

## Merge standard error and output
#$ -j y

## Tell the queue system to use the current directory
## as the working directory
#$ -cwd

## Resources needed
<%
  ## Should scheduler "details" be seen?
  details <- isTRUE(resources[["details"]])
  resources[["details"]] <- NULL

  ## Shell "startup" code to evaluate
  startup <- resources[["startup"]]
  resources[["startup"]] <- NULL

  ## Shell "shutdown" code to evaluate
  shutdown <- resources[["shutdown"]]
  resources[["shutdown"]] <- NULL

  ## Environment modules specifications
  modules <- resources[["modules"]]
  resources[["modules"]] <- NULL

  ## Environment variables to be set
  envs <- resources[["envs"]]
  if (length(envs) > 0) {
    stopifnot(is.character(envs), !is.null(names(envs)))
  }
  resources[["envs"]] <- NULL

  ## Custom "Rscript" command and Rscript arguments
  rscript <- resources[["rscript"]]
  if (is.null(rscript)) {
```

```
    rscript <- "Rscript"
  } else if (length(rscript) == 0 || !nzchar(rscript)[1]) {
    stop("Argument 'resources' specifies an empty 'rscript' field")
  }
  resources[["rscript"]] <- NULL
  rscript_args <- resources[["rscript_args"]]
  resources[["rscript_args"]] <- NULL
  rscript_call <- paste(c(rscript, rscript_args), collapse = " ")

  ## As-is resource specifications
  job_declarations <- resources[["asis"]]
  resources[["asis"]] <- NULL

  ## Remaining resources are assumed to be of type '<key>=<value>'
  opts <- unlist(resources, use.names = TRUE)
  opts <- sprintf("%s=%s", names(opts), opts)
 job_declarations <- sprintf("#$ %s", c(job_declarations, sprintf("-l %s", opts)))
  writeLines(job_declarations)
%>

## Bash settings
set -e          # exit on error
set -u          # error on unset variables
set -o pipefail # fail a pipeline if any command fails
trap 'echo "ERROR: future.batchtools job script failed on line $LINENO" >&2; exit 1' ERR

<% if (length(job_declarations) > 0) {
  writeLines(c(
    "echo 'Job submission declarations:'",
    sprintf("echo '%s'", job_declarations),
    "echo"
  ))
} %>

<% if (details) { %>
if command -v qstat > /dev/null; then
  echo "Job information:"
  qstat -j "${JOB_ID}"
  echo
fi
<% } %>

<% if (length(startup) > 0) {
  writeLines(startup)
} %>

<% if (length(modules) > 0) {
  writeLines(c(
```

```
    "echo 'Load environment modules:'",
    sprintf("echo '- modules: %s'", paste(modules, collapse = ", ")),
    sprintf("module load %s", paste(modules, collapse = " ")),
    "module list"
  ))
} %>

<% if (length(envs) > 0) {
  writeLines(c(
    sprintf("echo 'Setting environment variables: [n=%d]'", length(envs)),
    sprintf("echo ' - %s=%s'", names(envs), shQuote(envs)),
    sprintf("export %s=%s", names(envs), shQuote(envs))
  ))
} %>

echo "Session information:"
echo "- timestamp: $(date +"%Y-%m-%d %H:%M:%S%z")"
echo "- hostname: $(hostname)"
echo "- Rscript call: <%= rscript_call %>"
if ! command -v <%= rscript[1] %> &> /dev/null; then
   >&2 echo "ERROR: Argument 'resources' specifies a non-existing 'Rscript' launch command: <%= rscript[
    exit 1
fi
echo "- Rscript version: $(<%= paste(rscript, collapse = " ") %> --version)"
echo "- R_LIBS_USER=${R_LIBS_USER:-<not set>}"
echo "- R_LIBS_SITE=${R_LIBS_SITE:-<not set>}"
echo "- R_LIBS=${R_LIBS:-<not set>}"
echo "- Rscript library paths: $(<%= rscript_call %> -e "cat(shQuote(.libPaths()), sep = ' ')")"
echo


## Launch R and evaluate the batchtools R job
echo "Calling 'batchtools::doJobCollection()' ..."
echo "- job name: '<%= job.name %>'"
echo "- job log file: '<%= log.file %>'"
echo "- job uri: '<%= uri %>'"
<%= rscript_call %> -e 'batchtools::doJobCollection("<%= uri %>")'
res=$?
echo " - exit code: ${res}"
echo "Calling 'batchtools::doJobCollection()' ... done"
echo

<% if (details) { %>
if command -v qstat > /dev/null; then
  echo "Job summary:"
  qstat -j "${JOB_ID}"
fi
<% } %>
```

```
<% if (length(shutdown) > 0) {
  writeLines(shutdown)
} %>

echo "End time: $(date +"%Y-%m-%d %H:%M:%S%z")"

## Relay the exit code from Rscript
exit "${res}"
```

This template and the built-in `batchtools::makeClusterFunctionsSGE()` have been verified to work on a few different Grid Engine HPC clusters;

1. SGE 8.1.9 (Son of Grid Engine), Rocky 8 Linux, BeeGFS global filesystem (August 2025)
2. AGE 2024.1.0 (8.9.0), Rocky 9 Linux, NSF global filesystem (August 2025)

Value

Nothing.

References

- https://en.wikipedia.org/wiki/Oracle_Grid_Engine

Examples

```
library(future)

# Limit runtime to 10 minutes and memory to 400 MiB per future,
# request a parallel environment with four slots on a single host.
# Submit to the 'freecycle' queue. Load environment modules 'r' and
# 'jags'. Report on job details at startup and at the end of the job.
plan(future.batchtools::batchtools_sge, resources = list(
  h_rt = "00:10:00", mem_free = "100M",  ## memory is per process
  asis = c("-pe smp 4", "-q freecycle.q"),
  modules = c("r", "jags"),
  details = TRUE
))

f <- future({
  data.frame(
     hostname = Sys.info()[["nodename"]],
           os = Sys.info()[["sysname"]],
    osVersion = utils::osVersion,
        cores = unname(parallelly::availableCores()),
      modules = Sys.getenv("LOADEDMODULES")
  )
})
info <- value(f)
print(info)
```

batchtools_slurm *A batchtools Slurm backend resolves futures in parallel via a Slurm job scheduler*

Description

A batchtools Slurm backend resolves futures in parallel via a Slurm job scheduler

Usage

```
batchtools_slurm(
  ...,
  template = "slurm",
  scheduler.latency = 1,
  fs.latency = 65,
  resources = list(),
  delete = getOption("future.batchtools.delete", "on-success"),
  workers = getOption("future.batchtools.workers", default = 100L)
)
```

Arguments

`template`	(optional) Name of job-script template to be searched for by `batchtools::findTemplateFile()`. If not found, it defaults to the `templates/slurm.tmpl` part of this package (see below).
`scheduler.latency`	[`numeric(1)`] Time to sleep after important interactions with the scheduler to ensure a sane state. Currently only triggered after calling `submitJobs`.
`fs.latency`	[`numeric(1)`] Expected maximum latency of the file system, in seconds. Set to a positive number for network file systems like NFS which enables more robust (but also more expensive) mechanisms to access files and directories. Usually safe to set to `0` to disable the heuristic, e.g. if you are working on a local file system.
`resources`	(optional) A named list passed to the **batchtools** job-script template as variable `resources`. This is based on how `batchtools::submitJobs()` works, with the exception for specially reserved names defined by the **future.batchtools** package; • `resources[["details"]]`, if TRUE, results in the job script outputting job details and job summaries at the beginning and at the end. • `resources[["startup"]]` and `resources[["shutdown"]]` are character vectors of shell code to be injected to the job script as-is. • `resources[["modules"]]` is character vector of Linux environment modules to be loaded. • `resources[["envs"]]`, is an optional names character vector specifying environment variables to be set.

- `resources[["rscript"]]` is an optional character vector specifying how the 'Rscript' is launched. The `resources[["rscript_args"]]` field is an optional character vector specifying the 'Rscript' command-line arguments.
- `resources[["asis"]]` is a character vector that are passed as-is to the job script and are injected as job resource declarations.
- All remaining `resources` named elements are injected as named resource specification for the scheduler.

`delete` Controls if and when the batchtools job registry folder is deleted. If `"on-success"` (default), it is deleted if the future was resolved successfully *and* the expression did not produce an error. If `"never"`, then it is never deleted. If `"always"`, then it is always deleted.

`workers` The maximum number of workers the batchtools backend may use at any time, which for HPC schedulers corresponds to the maximum number of queued jobs. The default is `getOption("future.batchtools.workers", 100)`.

`...` Not used.

Details

Batchtools Slurm futures use **batchtools** cluster functions created by `makeClusterFunctionsSlurm2()`, which are used to interact with the Slurm job scheduler. This requires that Slurm commands `sbatch`, `squeue`, `sacct`, and `scancel` are available on the current machine.

The default template script `templates/slurm.tmpl` can be found in:

```
system.file("templates", "slurm.tmpl", package = "future.batchtools")
```

and comprises:

```
#!/bin/bash
######################################################################
# A batchtools launch script template for Slurm
#
# Author: Henrik Bengtsson
######################################################################

## Job name
#SBATCH --job-name=<%= job.name %>
## Direct streams to logfile
#SBATCH --output=<%= log.file %>

## Resources needed
<%
  ## Should scheduler "details" be seen?
  details <- isTRUE(resources[["details"]])
  resources[["details"]] <- NULL

  ## Shell "startup" code to evaluate
  startup <- resources[["startup"]]
```

```
  resources[["startup"]] <- NULL

  ## Shell "shutdown" code to evaluate
  shutdown <- resources[["shutdown"]]
  resources[["shutdown"]] <- NULL

  ## Environment modules specifications
  modules <- resources[["modules"]]
  resources[["modules"]] <- NULL

  ## Environment variables to be set
  envs <- resources[["envs"]]
  if (length(envs) > 0) {
    stopifnot(is.character(envs), !is.null(names(envs)))
  }
  resources[["envs"]] <- NULL

  ## Custom "Rscript" command and Rscript arguments
  rscript <- resources[["rscript"]]
  if (is.null(rscript)) {
    rscript <- "Rscript"
  } else if (length(rscript) == 0 || !nzchar(rscript)[1]) {
    stop("Argument 'resources' specifies an empty 'rscript' field")
  }
  resources[["rscript"]] <- NULL
  rscript_args <- resources[["rscript_args"]]
  resources[["rscript_args"]] <- NULL
  rscript_call <- paste(c(rscript, rscript_args), collapse = " ")

  ## As-is resource specifications
  job_declarations <- resources[["asis"]]
  resources[["asis"]] <- NULL

  ## Remaining resources are assumed to be of type '<key>=<value>'
  opts <- unlist(resources, use.names = TRUE)
  opts <- sprintf("%s=%s", names(opts), opts)
 job_declarations <- sprintf("#SBATCH %s", c(job_declarations, sprintf("--%s", opts)))
  writeLines(job_declarations)
%>

## Bash settings
set -e          # exit on error
set -u          # error on unset variables
set -o pipefail # fail a pipeline if any command fails
trap 'echo "ERROR: future.batchtools job script failed on line $LINENO" >&2; exit 1' ERR

<% if (length(job_declarations) > 0) {
  writeLines(c(
```

```
    "echo 'Job submission declarations:'",
    sprintf("echo '%s'", job_declarations),
    "echo"
  ))
} %>

<% if (details) { %>
if command -v scontrol > /dev/null; then
  echo "Job information:"
  scontrol show job "${SLURM_JOB_ID}"
  echo
fi
<% } %>

<% if (length(startup) > 0) {
  writeLines(startup)
} %>

<% if (length(modules) > 0) {
  writeLines(c(
    "echo 'Load environment modules:'",
    sprintf("echo '- modules: %s'", paste(modules, collapse = ", ")),
    sprintf("module load %s", paste(modules, collapse = " ")),
    "module list"
  ))
} %>

<% if (length(envs) > 0) {
  writeLines(c(
    sprintf("echo 'Setting environment variables: [n=%d]'", length(envs)),
    sprintf("echo ' - %s=%s'", names(envs), shQuote(envs)),
    sprintf("export %s=%s", names(envs), shQuote(envs))
  ))
} %>

echo "Session information:"
echo "- timestamp: $(date +"%Y-%m-%d %H:%M:%S%z")"
echo "- hostname: $(hostname)"
echo "- Rscript call: <%= rscript_call %>"
if ! command -v <%= rscript[1] %> &> /dev/null; then
   >&2 echo "ERROR: Argument 'resources' specifies a non-existing 'Rscript' launch command: <%= rscript[
    exit 1
fi
echo "- Rscript version: $(<%= paste(rscript, collapse = " ") %> --version)"
echo "- R_LIBS_USER=${R_LIBS_USER:-<not set>}"
echo "- R_LIBS_SITE=${R_LIBS_SITE:-<not set>}"
echo "- R_LIBS=${R_LIBS:-<not set>}"
echo "- Rscript library paths: $(<%= rscript_call %> -e "cat(shQuote(.libPaths()), sep = ' ')")"
```

```
echo

## Launch R and evaluate the batchtools R job
echo "Calling 'batchtools::doJobCollection()' ..."
echo "- job name: '<%= job.name %>'"
echo "- job log file: '<%= log.file %>'"
echo "- job uri: '<%= uri %>'"
<%= rscript_call %> -e 'batchtools::doJobCollection("<%= uri %>")'
res=$?
echo " - exit code: ${res}"
echo "Calling 'batchtools::doJobCollection()' ... done"
echo

<% if (details) { %>
if command -v sstat > /dev/null; then
  echo "Job summary:"
 sstat --format="JobID,AveCPU,MaxRSS,MaxPages,MaxDiskRead,MaxDiskWrite" --allsteps --jobs="${SLURM_J
fi
<% } %>

<% if (length(shutdown) > 0) {
  writeLines(shutdown)
} %>

echo "End time: $(date +"%Y-%m-%d %H:%M:%S%z")"

## Relay the exit code from Rscript
exit "${res}"
```

This template and the built-in `makeClusterFunctionsSlurm2()` have been verified to work on a few different Slurm HPC clusters;

1. Slurm 21.08.4, Rocky Linux 8, NFS global filesystem (September 2025)
2. Slurm 22.05.10, Rocky Linux 9, Lustre global filesystem (September 2025)
3. Slurm 22.05.11, Rocky Linux 8, NFS global filesystem (September 2025)
4. Slurm 23.02.6, Ubuntu 24.04 LTS, NFS global filesystem (September 2025)
5. Slurm 24.11.3, AlmaLinux 9, Lustre global filesystem (September 2025)*
6. Slurm 24.11.5, Rocky Linux 9, VAST global filesystem (February 2026)

(*) Verified with **future.batchtools** 0.20.0, which used `batchtools::makeClusterFunctionsSlurm()`, which the new `makeClusterFunctionsSlurm2()` enhances.

Value

Nothing.

References

- https://en.wikipedia.org/wiki/Slurm_Workload_Manager

Examples

```
library(future)

# Limit runtime to 10 minutes and memory to 400 MiB per future,
# request a parallel environment with four slots on a single host.
# Submit to the 'freecycle' partition. Load environment modules 'r' and
# 'jags'. Report on job details at startup and at the end of the job.
plan(future.batchtools::batchtools_slurm, resources = list(
  time = "00:10:00", mem = "400M",
  asis = c("--nodes=1", "--ntasks=4", "--partition=freecycle"),
  modules = c("r", "jags"),
  details = TRUE
))

f <- future({
  data.frame(
     hostname = Sys.info()[["nodename"]],
           os = Sys.info()[["sysname"]],
    osVersion = utils::osVersion,
        cores = unname(parallelly::availableCores())
  )
})
info <- value(f)
print(info)

# As above, but use R from the Rocker 'r-base' Linux container;
#
#   mkdir -p ~/lxc
#   apptainer build ~/lxc/rocker_r-base.sif docker://rocker/r-base
#
# Example assumes that 'future.batchtools' has already been installed in
# the container to the 'R_LIBS_USER' package folder living on the host;
#
#   R_LIBS_USER=~/R/rocker-%p-library/%v ~/lxc/rocker_r-base.sif
#   ...
#   > chooseCRANmirror(ind = 1)
#   > install.packages("future.batchtools")
#
plan(future.batchtools::batchtools_slurm, resources = list(
  time = "00:10:00", mem = "400M",
  asis = c("--nodes=1", "--ntasks=4", "--partition=freecycle"),
  details = TRUE,
  envs = c(R_LIBS_USER = "~/R/rocker-%p-library/%v"),
  rscript = c("apptainer", "exec", "~/lxc/rocker_r-base.sif", "Rscript")
))

f <- future({
  data.frame(
     hostname = Sys.info()[["nodename"]],
           os = Sys.info()[["sysname"]],
    osVersion = utils::osVersion,
        cores = unname(parallelly::availableCores()),
```

```
      modules = Sys.getenv("LOADEDMODULES")
  )
})
info <- value(f)
print(info)
```

---

batchtools_torque *A batchtools TORQUE backend resolves futures in parallel via a TORQUE/PBS job scheduler*

Description

A batchtools TORQUE backend resolves futures in parallel via a TORQUE/PBS job scheduler

Usage

```
batchtools_torque(
  ...,
  template = "torque",
  scheduler.latency = 1,
  fs.latency = 65,
  resources = list(),
  delete = getOption("future.batchtools.delete", "on-success"),
  workers = getOption("future.batchtools.workers", default = 100L)
)
```

Arguments

`template`	(optional) Name of job-script template to be searched for by `batchtools::findTemplateFile()`. If not found, it defaults to the `templates/torque.tmpl` part of this package (see below).
`scheduler.latency`	[`numeric(1)`] Time to sleep after important interactions with the scheduler to ensure a sane state. Currently only triggered after calling `submitJobs`.
`fs.latency`	[`numeric(1)`] Expected maximum latency of the file system, in seconds. Set to a positive number for network file systems like NFS which enables more robust (but also more expensive) mechanisms to access files and directories. Usually safe to set to `0` to disable the heuristic, e.g. if you are working on a local file system.
`resources`	(optional) A named list passed to the **batchtools** job-script template as variable `resources`. This is based on how `batchtools::submitJobs()` works, with the exception for specially reserved names defined by the **future.batchtools** package; • `resources[["details"]]`, if TRUE, results in the job script outputting job details and job summaries at the beginning and at the end.

- `resources[["startup"]]` and `resources[["shutdown"]]` are character vectors of shell code to be injected to the job script as-is.
- `resources[["modules"]]` is character vector of Linux environment modules to be loaded.
- `resources[["envs"]]`, is an optional names character vector specifying environment variables to be set.
- `resources[["rscript"]]` is an optional character vector specifying how the 'Rscript' is launched. The `resources[["rscript_args"]]` field is an optional character vector specifying the 'Rscript' command-line arguments.
- `resources[["asis"]]` is a character vector that are passed as-is to the job script and are injected as job resource declarations.
- All remaining `resources` named elements are injected as named resource specification for the scheduler.

`delete` Controls if and when the batchtools job registry folder is deleted. If `"on-success"` (default), it is deleted if the future was resolved successfully *and* the expression did not produce an error. If `"never"`, then it is never deleted. If `"always"`, then it is always deleted.

`workers` The maximum number of workers the batchtools backend may use at any time, which for HPC schedulers corresponds to the maximum number of queued jobs. The default is `getOption("future.batchtools.workers", 100)`.

`...` Not used.

Details

Batchtools TORQUE/PBS futures use **batchtools** cluster functions created by `batchtools::makeClusterFunctionsTORQUE` which are used to interact with the TORQUE/PBS job scheduler. This requires that TORQUE/PBS commands `qsub`, `qselect`, and `qdel` are available on the current machine.

The default template script `templates/torque.tmpl` can be found in:

```
system.file("templates", "torque.tmpl", package = "future.batchtools")
```

and comprises:

```
#!/bin/bash
##############################################################################
# A batchtools launch script template for TORQUE/PBS
#
# Author: Henrik Bengtsson
##############################################################################

## Job name
#PBS -N <%= job.name %>

## Direct streams to logfile
#PBS -o <%= log.file %>
```

```
## Merge standard error and output
#PBS -j oe

## Resources needed
<%
  ## Should scheduler "details" be seen?
  details <- isTRUE(resources[["details"]])
  resources[["details"]] <- NULL

  ## Shell "startup" code to evaluate
  startup <- resources[["startup"]]
  resources[["startup"]] <- NULL

  ## Shell "shutdown" code to evaluate
  shutdown <- resources[["shutdown"]]
  resources[["shutdown"]] <- NULL

  ## Environment modules specifications
  modules <- resources[["modules"]]
  resources[["modules"]] <- NULL

  envs <- resources[["envs"]]
  if (length(envs) > 0) {
    stopifnot(is.character(envs), !is.null(names(envs)))
  }
  resources[["envs"]] <- NULL

  ## Custom "Rscript" command and Rscript arguments
  rscript <- resources[["rscript"]]
  if (is.null(rscript)) {
    rscript <- "Rscript"
  } else if (length(rscript) == 0 || !nzchar(rscript)[1]) {
    stop("Argument 'resources' specifies an empty 'rscript' field")
  }
  resources[["rscript"]] <- NULL
  rscript_args <- resources[["rscript_args"]]
  resources[["rscript_args"]] <- NULL
  rscript_call <- paste(c(rscript, rscript_args), collapse = " ")

  ## Environment variables to be set
  ## As-is resource specifications
  job_declarations <- resources[["asis"]]
  resources[["asis"]] <- NULL

  ## Remaining resources are assumed to be of type '<key>=<value>'
  opts <- unlist(resources, use.names = TRUE)
  opts <- sprintf("%s=%s", names(opts), opts)
 job_declarations <- sprintf("#PBS %s", c(job_declarations, sprintf("-l %s", opts)))
```

```
  writeLines(job_declarations)
%>

## Bash settings
set -e          # exit on error
set -u          # error on unset variables
set -o pipefail # fail a pipeline if any command fails
trap 'echo "ERROR: future.batchtools job script failed on line $LINENO" >&2; exit 1' ERR

<% if (length(job_declarations) > 0) {
  writeLines(c(
    "echo 'Job submission declarations:'",
    sprintf("echo '%s'", job_declarations),
    "echo"
  ))
} %>

<% if (details) { %>
if command -v qstat > /dev/null; then
  echo "Job information:"
  qstat -f "${PBS_JOBID}"
  echo
fi
<% } %>

<% if (length(startup) > 0) {
  writeLines(startup)
} %>

<% if (length(modules) > 0) {
  writeLines(c(
    "echo 'Load environment modules:'",
    sprintf("echo '- modules: %s'", paste(modules, collapse = ", ")),
    sprintf("module load %s", paste(modules, collapse = " ")),
    "module list"
  ))
} %>

<% if (length(envs) > 0) {
  writeLines(c(
    sprintf("echo 'Setting environment variables: [n=%d]'", length(envs)),
    sprintf("echo ' - %s=%s'", names(envs), shQuote(envs)),
    sprintf("export %s=%s", names(envs), shQuote(envs))
  ))
} %>

echo "Session information:"
echo "- timestamp: $(date +"%Y-%m-%d %H:%M:%S%z")"
```

```
echo "- hostname: $(hostname)"
echo "- Rscript call: <%= rscript_call %>"
if ! command -v <%= rscript[1] %> &> /dev/null; then
   >&2 echo "ERROR: Argument 'resources' specifies a non-existing 'Rscript' launch command: <%= rscript[
    exit 1
fi
echo "- Rscript version: $(<%= paste(rscript, collapse = " ") %> --version)"
echo "- R_LIBS_USER=${R_LIBS_USER:-<n/a>}"
echo "- R_LIBS_SITE=${R_LIBS_SITE:-<n/a>}"
echo "- R_LIBS=${R_LIBS:-<n/a>}"
echo "- Rscript library paths: $(<%= rscript_call %> -e "cat(shQuote(.libPaths()), sep = ' ')")"
echo

## Launch R and evaluate the batchtools R job
echo "Calling 'batchtools::doJobCollection()' ..."
echo "- job name: '<%= job.name %>'"
echo "- job log file: '<%= log.file %>'"
echo "- job uri: '<%= uri %>'"
<%= rscript_call %> -e 'batchtools::doJobCollection("<%= uri %>")'
res=$?
echo " - exit code: ${res}"
echo "Calling 'batchtools::doJobCollection()' ... done"
echo

<% if (details) { %>
if command -v qstat > /dev/null; then
  echo "Job summary:"
  qstat -f "${PBS_JOBID}"
fi
<% } %>

<% if (length(shutdown) > 0) {
  writeLines(shutdown)
} %>

echo "End time: $(date +"%Y-%m-%d %H:%M:%S%z")"

## Relay the exit code from Rscript
exit "${res}"
```

This template and the built-in `batchtools::makeClusterFunctionsTORQUE()` have been verified to work on the following PBS/TORQUE HPC cluster;

1. PBSPro 2024.1.2, Rocky 8 Linux, Lustre global filesystem (September 2025)

Value

Nothing.

References

- https://en.wikipedia.org/wiki/TORQUE

Examples

```
library(future)

# Limit runtime to 10 minutes and total memory to 400 MiB per future,
# request a parallel environment with four slots on a single host.
# Submit to the 'freecycle' queue. Load environment modules 'r' and
# 'jags'. Report on job details at startup and at the end of the job.
plan(future.batchtools::batchtools_torque, resources = list(
  walltime = "00:10:00", mem = "100mb",  ## memory is per process
  asis = c("-l nodes=1:ppn=4", "-q freecycle"),
  modules = c("r", "jags"),
  details = TRUE
))

f <- future({
  data.frame(
     hostname = Sys.info()[["nodename"]],
           os = Sys.info()[["sysname"]],
    osVersion = utils::osVersion,
        cores = unname(parallelly::availableCores()),
      modules = Sys.getenv("LOADEDMODULES")
  )
})
info <- value(f)
print(info)
```

`future.batchtools` *future.batchtools: A Future for batchtools*

Description

The **future.batchtools** package implements the Future API on top of **batchtools** such that futures can be resolved on for instance high-performance compute (HPC) clusters via job schedulers. The Future API is defined by the **future** package.

Details

To use batchtools futures, load **future.batchtools**, and select the type of future you wish to use via `future::plan()`.

Author(s)

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

See Also

Useful links:

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

Examples

```
library(future)
plan(future.batchtools::batchtools_local)
demo("mandelbrot", package = "future", ask = FALSE)
```

`makeClusterFunctionsSlurm2`

ClusterFunctions for Slurm Systems (patched)

Description

This function enhances `batchtools::makeClusterFunctionsSlurm()` with a few patches. Firstly, it patches the `listJobsQueued()` cluster function such that it falls back to querying Slurm's account database (`sacct`), if the future was *not* found in the Slurm job queue (`squeue`), which might be the case when Slurm provisions a job that was just submitted to the scheduler. Secondly, it patches the `submitJob()` cluster function such that the system call to `sbatch` does not capture stderr together with stdout, but rather separately such that any extra INFO messages from `sbatch` do not corrupt the output intended to come from stdout only.

Usage

```
makeClusterFunctionsSlurm2(
  template = "slurm",
  array.jobs = TRUE,
  nodename = "localhost",
  scheduler.latency = 1,
  fs.latency = 65
)
```

Arguments

`template` [character(1)]
Either a path to a **brew** template file (with extension "tmpl"), or a short descriptive name enabling the following heuristic for the file lookup:

1. "batchtools.[template].tmpl" in the path specified by the environment variable "R_BATCHTOOLS_SEARCH_PATH".
2. "batchtools.[template].tmpl" in the current working directory.

3. "[template].tmpl" in the user config directory (see `user_config_dir`); on linux this is usually "~/.config/batchtools/[template].tmpl".
4. ".batchtools.[template].tmpl" in the home directory.
5. "[template].tmpl" in the package installation directory in the subfolder "templates".

`array.jobs` [`logical(1)`]
If array jobs are disabled on the computing site, set to `FALSE`.

`nodename` [`character(1)`]
Nodename of the master host. All commands are send via SSH to this host. Only works iff

1. Passwordless authentication (e.g., via SSH public key authentication) is set up.
2. The file directory is shared across machines, e.g. mounted via SSHFS.
3. Either the absolute path to the `file.dir` is identical on the machines, or paths are provided relative to the home directory. Symbolic links should work.

`scheduler.latency` [`numeric(1)`]
Time to sleep after important interactions with the scheduler to ensure a sane state. Currently only triggered after calling `submitJobs`.

`fs.latency` [`numeric(1)`]
Expected maximum latency of the file system, in seconds. Set to a positive number for network file systems like NFS which enables more robust (but also more expensive) mechanisms to access files and directories. Usually safe to set to `0` to disable the heuristic, e.g. if you are working on a local file system.

Value

A batchtools::ClusterFunctions object.

`zzz-future.batchtools.options`
Options used for batchtools futures

Description

Below are the R options and environment variables that are used by the **future.batchtools** package. See future::future.options for additional ones that apply to futures in general.

WARNING: Note that the names and the default values of these options may change in future versions of the package. Please use with care until further notice.

Settings for batchtools futures

'`future.batchtools.workers`': (a positive numeric or `+Inf`) The default number of workers available on HPC schedulers with job queues. (Default: `100`)

'`future.batchtools.output`': (logical) If TRUE, **batchtools** will produce extra output. If FALSE, such output will be disabled by setting **batchtools** options '`batchtools.verbose`' and '`batchtools.progress`' to FALSE. (Default: `getOption("future.debug", FALSE)`)

'`future.batchtools.expiration.tail`': (a positive numeric) When a **batchtools** job expires, the last few lines will be relayed by batchtools futures to help troubleshooting. This option controls how many lines are displayed. (Default: `48L`)

'`future.cache.path`': (character string) An absolute or relative path specifying the root folder in which **batchtools** registry folders are stored. This folder needs to be accessible from all hosts ("workers"). Specifically, it must *not* be a folder that is only local to the machine such as `file.path(tempdir(), ".future")` if a job scheduler on an HPC environment is used. (Default: `.future` in the current working directory)

'`future.batchtools.delete`': (character string) Controls whether or not the future's **batchtools** registry folder is deleted after the future result has been collected. If `"always"`, it is always deleted. If `"never"`, it is never deleted. If `"on-success"`, it is deleted if the future resolved successfully, whereas if it failed, it is left as-is to help with troubleshooting. (Default: `"on-success"`)

Environment variables that set R options

All of the above R '`future.batchtools.*`' options can be set by corresponding environment variable `R_FUTURE_BATCHTOOLS_*` *when the* **future.batchtools** *package is loaded*. This means that those environment variables must be set before the **future.batchtools** package is loaded in order to have an effect. For example, if `R_FUTURE_BATCHTOOLS_WORKERS="200"` is set, then option '`future.batchtools.workers`' is set to `200` (numeric).

Examples

```
# Set an R option:
options(future.cache.path = "/cluster-wide/folder/.future")
```

Index