Package ‘parallelMap’

March 26, 2020

Title  Unified Interface to Parallelization Back-Ends
Version  1.5.0
Maintainer  Bernd Bischl <bernd_bischl@gmx.net>
Description  Unified parallelization framework for multiple back-end, designed for internal package and interactive usage. The main operation is parallel mapping over lists. Supports 'local', 'multicore', 'mpi' and 'BatchJobs' mode. Allows tagging of the parallel operation with a level name that can be later selected by the user to switch on parallel execution for exactly this operation.
License  BSD_2_clause + file LICENSE
BugReports  https://github.com/berndbischl/parallelMap
Depends  R (>= 3.0.0)
Imports  BBmisc (>= 1.8), checkmate (>= 1.8.0), parallel, stats, utils
Suggests  BatchJobs (>= 1.8), batchtools (>= 0.9.6), data.table, Rmpi, rpart, snow, testthat
ByteCompile  yes
Encoding  UTF-8
LazyData  yes
RoxygenNote  7.1.0
NeedsCompilation  no
Author  Bernd Bischl [cre, aut],
Michel Lang [aut] (<https://orcid.org/0000-0001-9754-0393>),
Patrick Schratz [aut] (<https://orcid.org/0000-0003-0748-6624>)
Repository  CRAN
Date/Publication  2020-03-26 11:00:02 UTC
R topics documented:

parallelExport .................................................. 2
parallelGetOptions .............................................. 3
parallelGetRegisteredLevels ................................. 3
parallelLapply .................................................... 4
parallelLibrary .................................................... 5
parallelMap .......................................................... 6
parallelRegisterLevels .......................................... 7
parallelSource ..................................................... 8
parallelStart ........................................................ 9
parallelStop ........................................................ 12

Index 13

parallelExport Export R objects for parallelization.

Description

Makes sure that the objects are exported to slave process so that they can be used in a job function which is later run with parallelMap().

Usage

parallelExport(
  ..., 
  objnames, 
  master = TRUE, 
  level = NA_character_, 
  show.info = NA
)

Arguments

... character
  Names of objects to export.

objnames (character(1))
  Names of objects to export. Alternative way to pass arguments.

master (logical(1))
  Really export to package environment on master for local and multicore mode? If you do not do this your objects might not get exported for the mapping function call. Only disable when you are really sure. Default is TRUE.

level (character(1))
  If a (non-missing) level is specified in parallelStart(), the function only exports if the level specified here matches. See parallelMap(). Useful if this function is used in a package. Default is NA.
parallelGetOptions

show.info (logical(1))
Verbose output on console? Can be used to override setting from options / parallelStart(). Default is NA which means no overriding.

Value
Nothing.

Description
Returned are current and default settings, both as lists. The return value has slots elements settings and defaults, which are both lists of the same structure, named by option names.
A printer exists to display this object.
For details on the configuration procedure please read parallelStart() and https://github.com/mlr-org/parallelMap.

Usage
parallelGetOptions()

Value
ParallelMapOptions. See above.

parallelGetRegisteredLevels
Get registered parallelization levels for all currently loaded packages.

Description
With flatten = FALSE, a structured S3 object is returned. The S3 object only has one slot, which is called levels. This contains a named list. Each name refers to package from the call to parallelRegisterLevels(), while the entries are character vectors of the form “package.level”.
With flatten = TRUE, a simple character vector is returned that contains all concatenated entries of levels from above.

Usage
parallelGetRegisteredLevels(flatten = FALSE)
**Arguments**

flattened (logical(1))

Flatten to character vector or not? See description. Default is FALSE.

**Value**

RegisteredLevels | character. See above.

---

**parallelLapply**

*Parallel versions of apply-family functions.*

**Description**

parallelLapply: A parallel `lapply()` version.

parallelSapply: A parallel `sapply()` version.

All functions are simple wrappers for `parallelMap()`.

**Usage**

```r
callParallelLapply(xs, fun, ..., impute.error = NULL, level = NA_character_)
callParallelSapply(  
  xs,  
  fun,  
  ...,  
  simplify = TRUE,  
  use.names = TRUE,  
  impute.error = NULL,  
  level = NA_character_  
)
```

**Arguments**

xs (vector | list)

fun is applied to the elements of this argument.

fun function

Function to map over xs.

... (any)

Further arguments passed to fun.

impute.error (NULL | function(x))

See `parallelMap()`.

level (character(1))

See `parallelMap()`.

simplify (logical(1))

See `sapply()`. Default is TRUE.

use.names (logical(1))

See `sapply()`. Default is TRUE.
parallelLibrary

Value
For parallelLapply a named list, for parallelSapply it depends on the return value of fun and the settings of simplify and use.names.

Description
Makes sure that the packages are loaded in slave process so that they can be used in a job function which is later run with parallelMap().
For all modes, the packages are also (potentially) loaded on the master.

Usage
parallelLibrary(
  ..., packages, master = TRUE, level = NA_character_, show.info = NA)

Arguments

... character Names of packages to load.
packages (character(1)) Names of packages to load. Alternative way to pass arguments.
master (logical(1)) Load packages also on master for any mode? Default is TRUE.
level (character(1)) If a (non-missing) level is specified in parallelStart(), the function only loads the packages if the level specified here matches. See parallelMap(). Useful if this function is used in a package. Default is NA.
show.info (logical(1)) Verbose output on console? Can be used to override setting from options/parallelStart(). Default is NA which means no overriding.

Value
Nothing.
parallelMap

Maps a function over lists or vectors in parallel.

Description

Uses the parallelization mode and the other options specified in parallelStart(). Libraries and source file can be initialized on slaves with parallelLibrary() and parallelSource(). Large objects can be separately exported via parallelExport(), they can be simply used under their exported name in slave body code. Regarding error handling, see the argument impute.error.

Usage

parallelMap(
  fun,
  ...,  # any
  more.args = list(),
  simplify = FALSE,
  use.names = FALSE,
  impute.error = NULL,
  level = NA_character_,
  show.info = NA
)

Arguments

fun function
Function to map over ....

... (any)
Arguments to vectorize over (list or vector).

more.args list
A list of other arguments passed to fun. Default is empty list.

simplify (logical(1))
Should the result be simplified? See simplify2array. If TRUE, simplify2array(higher = TRUE) will be called on the result object. Default is FALSE.

use.names (logical(1))
Should result be named? Use names if the first ... argument has names, or if it is a character vector, use that character vector as the names.

impute.error (NULL | function(x))
This argument can be used for improved error handling. NULL means that, if an exception is generated on one of the slaves, it is also thrown on the master. Usually all slave jobs will have to terminate until this exception on the master can be thrown. If you pass a constant value or a function, all jobs are guaranteed to return a result object, without generating an exception on the master for slave
errors. In case of an error, this is a `simpleError()` object containing the error message. If you passed a constant object, the error-objects will be substituted with this object. If you passed a function, it will be used to operate on these error-objects (it will ONLY be applied to the error results). For example, using identity would keep and return the `simpleError`-object, or `function(x) 99` would impute a constant value (which could be achieved more easily by simply passing 99). Default is NULL.

level (character(1))
If a (non-missing) level is specified in `parallelStart()`, this call is only parallelized if the level specified here matches. Useful if this function is used in a package. Default is NA.

show.info (logical(1))
Verbose output on console? Can be used to override setting from options / `parallelStart()`. Default is NA which means no overriding.

Value
Result.

Examples
```
parallelStart()
parallelMap(identity, 1:2)
parallelStop()
```
parallelSource

Arguments

package (character(1))
Name of your package. Default is “custom” (we are not in a package).

levels (character(1))
Available levels that are used in the parallelMap() operations of your package or code. If package is not missing, all levels will be prefixed with “package.”.

Value

Nothing.

parallelSource Source R files for parallelization.

Description

Makes sure that the files are sourced in slave process so that they can be used in a job function which is later run with parallelMap().

For all modes, the files are also (potentially) loaded on the master.

Usage

parallelSource(
  ..., 
  files, 
  master = TRUE, 
  level = NA_character_, 
  show.info = NA 
)

Arguments

... character
File paths to sources.

files character
File paths to sources. Alternative way to pass arguments.

master (logical(1))
Source files also on master for any mode? Default is TRUE.

level (character(1))
If a (non-missing) level is specified in parallelStart(), the function only sources the files if the level specified here matches. See parallelMap(). Useful if this function is used in a package. Default is NA.

show.info (logical(1))
Verbose output on console? Can be used to override setting from options / parallelStart(). Default is NA which means no overriding.
**parallelStart**

**Value**

Nothing.

---

**parallelStart**  
*Parallelization setup for parallelMap.*

**Description**

Defines the underlying parallelization mode for `parallelMap()`. Also allows to set a “level” of parallelization. Only calls to `parallelMap()` with a matching level are parallelized. The defaults of all settings are taken from your options, which you can also define in your R profile. For an introductory tutorial and information on the options configuration, please go to the project’s github page at https://github.com/mlr-org/parallelMap.

**Usage**

```r
parallelStart(
  mode,
  cpus,
  socket.hosts,
  bj.resources = list(),
  bt.resources = list(),
  logging,
  storagedir,
  level,
  load.balancing = FALSE,
  show.info,
  suppress.local.errors = FALSE,
  reproducible,
  ...
)
```

```r
parallelStartLocal(show.info, suppress.local.errors = FALSE, ...)
```

```r
parallelStartMulticore(
  cpus,
  logging,
  storagedir,
  level,
  load.balancing = FALSE,
  show.info,
  reproducible,
  ...
)
```

```r
parallelStartSocket(
```
parallelStart

```r
parallelStart(cpus, socket.hosts, logging, storagedir, level, load.balancing = FALSE, show.info, reproducible, ...
)
```

```r
parallelStartMPI(cpus, logging, storagedir, level, load.balancing = FALSE, show.info, reproducible, ...
)
```

```r
parallelStartBatchJobs(bj.resources = list(), logging, storagedir, level, show.info, ...
)
```

```r
parallelStartBatchtools(bt.resources = list(), logging, storagedir, level, show.info, ...
)
```

**Arguments**

- **mode** (character(1))
  Which parallel mode should be used: “local”, “multicore”, “socket”, “mpi”, “BatchJobs”. Default is the option parallelMap.default.mode or, if not set, “local” without parallel execution.

- **cpus** (integer(1))
  Number of used cpus. For local and BatchJobs mode this argument is ignored.
For socket mode, this is the number of processes spawned on localhost, if you want processes on multiple machines use socket.hosts. Default is the option parallelMap.default.cpus or, if not set, parallel::detectCores() for multicore mode, \( \max(1, [\text{mpi.universe.size}][\text{Rmpi::mpi.universe.size}]-1) \) for mpi mode and 1 for socket mode.

**socket.hosts** character

Only used in socket mode, otherwise ignored. Names of hosts where parallel processes are spawned. Default is the option parallelMap.default.socket.hosts, if this option exists.

**bj.resources** list

Resources like walltime for submitting jobs on HPC clusters via BatchJobs. See BatchJobs::submitJobs(). Defaults are taken from your BatchJobs config file.

**bt.resources** list

Analog to bj.resources. See batchtools::submitJobs().

**logging** (logical(1))

Should slave output be logged to files via sink() under the storagedir? Files are named <iteration_number>-log and put into unique subdirectories named parallelMap_log_<nr> for each subsequent parallelMap() operation. Previous logging directories are removed on parallelStart if logging is enabled. Logging is not supported for local mode, because you will see all output on the master and can also run stuff like traceback() in case of errors. Default is the option parallelMap.default.logging or, if not set, FALSE.

**storagedir** (character(1))

Existing directory where log files and intermediate objects for BatchJobs mode are stored. Note that all nodes must have write access to exactly this path. Default is the current working directory.

**level** (character(1))

You can set this so only calls to parallelMap() that have exactly the same level are parallelized. Default is the option parallelMap.default.level or, if not set, NA which means all calls to parallelMap() are are potentially parallelized.

**load.balancing** (logical(1))

Enables load balancing for multicore, socket and mpi. Set this to TRUE if you have heterogeneous runtimes. Default is FALSE.

**show.info** (logical(1))

Verbose output on console for all further package calls? Default is the option parallelMap.default.show.info or, if not set, TRUE.

**suppress.local.errors**

(logical(1))

Should reporting of error messages during function evaluations in local mode be suppressed? Default ist FALSE, i.e. every error message is shown.

**reproducible** (logical(1))

Should parallel jobs produce reproducible results when setting a seed? With this option, parallelMap() calls will be reproducible when using set.seed() with the default RNG kind. This is not the case by default when parallelizing in R,
since the default RNG kind "Mersenne-Twister" is not honored by parallel processes. Instead RNG kind "L'Ecyer-CMRG" needs to be used to ensure parallel reproducibility. Default is the option parallelMap.default.reproducible or, if not set, TRUE.

... Optional parameters, for socket mode passed to parallel::makePSOCKcluster(), for mpi mode passed to parallel::makeCluster() and for multicore passed to parallel::mcmapply() (mc.preschedule (overwriting load.balancing), mc.set.seed, mc.silent and mc.cleanup are supported for multicore).

Details

Currently the following modes are supported, which internally dispatch the mapping operation to functions from different parallelization packages:

local No parallelization with mapply().
multicore Multicore execution on a single machine with parallel::mclapply().
mpi Snow MPI cluster on one or multiple machines with parallel::makeCluster() and parallel::clusterMap().
BatchJobs Parallelization on batch queuing HPC clusters, e.g., Torque, SLURM, etc., with BatchJobs::batchMap().

For BatchJobs mode you need to define a storage directory through the argument storagedir or the option parallelMap.default.storagedir.

Value

Nothing.

parallelStop

Stops parallelization.

Description

Sets mode to “local”, i.e., parallelization is turned off and all necessary stuff is cleaned up.

For socket and mpi mode parallel::stopCluster() is called.

For BatchJobs mode the subdirectory of the storagedir containing the exported objects is removed.

After a subsequent call of parallelStart(), no exported objects are present on the slaves and no libraries are loaded, i.e., you have clean R sessions on the slaves.

Usage

parallelStop()

Value

Nothing.
Index

base::onLoad(), 7
BatchJobs::batchMap(), 12
BatchJobs::submitJobs(), 11
batchtools::submitJobs(), 11

character, 2, 5, 8, 11

function, 4, 6

lapply(), 4
list, 6, 11

mapply(), 12

parallel::clusterMap(), 12
parallel::detectCores(), 11
parallel::makeCluster(), 12
parallel::makePSOCKcluster(), 12
parallel::mclapply(), 12
parallel::mcmapply(), 12
parallel::stopCluster(), 12
parallelExport, 2
parallelExport(), 6
parallelGetOptions, 3
parallelGetRegisteredLevels, 3
parallelLapply, 4
parallelLibrary, 5
parallelLibrary(), 6
parallelMap, 6
parallelMap(), 2, 4, 5, 7–9, 11
parallelRegisterLevels, 7
parallelRegisterLevels(), 3
parallelSapply (parallelLapply), 4
parallelSource, 8
parallelSource(), 6
parallelStart, 9
parallelStart(), 2, 3, 5–8, 12
parallelStartBatchJobs (parallelStart), 9
parallelStartBatchtools (parallelStart), 9
parallelStartLocal (parallelStart), 9
parallelStartMPI (parallelStart), 9
parallelStartMulticore (parallelStart), 9
parallelStartSocket (parallelStart), 9
parallelStop, 12

sapply(), 4
simpleError(), 7
simplify2array, 6
sink(), 11

traceback(), 11