

Package: clusters (via r-universe)

October 9, 2026

Type Package

Title Utilities to manage, estimate speed, optimize use of parallel clusters

Description Miscellaneous utilities developed by the Predictive Ecology Group (<<http://predictiveecology.org>>).

URL <http://clusters.predictiveecology.org>,
<https://github.com/PredictiveEcology/clusters>

Date 2026-09-16

Version 0.0.51

Depends R (>= 4.2)

Imports cli, data.table, DEoptim, filelock, ggplot2, graphics,
parallel, parallelly, qs2, reproducible (>= 2.1.2.9045),
Require, rstudioapi, stats, tools, utils

Suggests codetools, ggpubr, knitr, roxygen2, rmarkdown, sf,
SpaDES.core, terra, testthat, withr

Remotes PredictiveEcology/reproducible@development,
PredictiveEcology/Require@development

Encoding UTF-8

Language en-CA

License GPL-3

VignetteBuilder knitr, rmarkdown

BugReports <https://github.com/PredictiveEcology/clusters/issues>

ByteCompile yes

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.1.0

Config/pak/sysreqs cmake make libuv1-dev

Repository <https://predictiveecology.r-universe.dev>

Date/Publication 2026-09-29 00:58:55 UTC

```
RemoteUrl https://github.com/PredictiveEcology/clusters
RemoteRef HEAD
RemoteSha 483d517e61998c985a851bd96644e135dad4cf55
```

Contents

.sshKeepaliveOpts	2
clusterSetup	3
DEoptimIterative2	5
dirNew	6
freeCoresLessReserved	6
getHostCombination	7
makeClusterPSOCK	8
makeHosts	9
mirrorTerraOptions	10
monitorCluster	11
numActiveThreads	12
plan_psock_min	13
plotMachine	15
ramUsageGB	15
readTransferredObjects	16
reservationsPath	17
resourcesUsed	18
rmIncompleteDups	18
runTests	19
shipSystemLibs	20
summaryOutputFolder	21
tableFiles	22
termsInDEoptim	22
testMachine	23
transferDirName	23
verifyClusterHosts	24
visualizeDE	25
Index	26

.sshKeepaliveOpts	SSH options that let the master notice a worker has died
-------------------	--

Description

PSOCK workers are reached over SSH, and the master reads results with `unserialize()` on the socket. That read is bounded by `parallelly`’s `timeout`, which we set to 30 days so that a legitimately long computation is never cut off. The consequence is that if a worker’s R process dies *after* connecting, SSH holds the tunnel open, the socket never closes, and the master blocks for up to 30 days – in practice, forever.

Usage

```
.sshKeepaliveOpts(extra = NULL, interval = 30L, countMax = 4L)
```

Arguments

extra	Character vector of additional SSH options to append.
interval	Seconds between keepalive probes.
countMax	Number of unanswered probes before SSH closes the connection.

Details

`connectTimeout` does not help: it only bounds the initial connect, which already succeeded.

`ServerAliveInterval/ServerAliveCountMax` close the door from the SSH side instead. SSH probes an idle connection every `ServerAliveInterval` seconds and tears it down after `ServerAliveCountMax` unanswered probes, so a dead worker surfaces as a closed connection and an R error within ~2 minutes rather than hanging the run. A busy worker answers the probes, so long computations are unaffected – the probes run at the SSH layer, not in R.

Observed 2026-09-19/20: two re-scores hung for hours after a worker died at startup with `Error in unserialize(node$con) inside workRSTACK`, and the masters had to be killed by hand.

Value

A character vector of SSH options for `rshopts`.

<code>clusterSetup</code>	<i>Setup a cluster for DEoptim</i>
---------------------------	------------------------------------

Description

This includes copying files over to `unique(cores)` machines, then loading all objects from disk in each of the parallel cores.

Usage

```
clusterSetup(
  messagePrefix = "DEoptim_",
  itermmax = 500,
  trace = TRUE,
  strategy = 3,
  initialpop = NULL,
  NP = NULL,
  cores,
  logPath,
  libPath,
  objsNeeded,
```

```

    pkgsNeeded,
    nCoresNeeded = 100,
    envir = parent.frame(),
    controlArgs = list()
)

```

Arguments

messagePrefix	Character prefix for this cluster's messages, so concurrent builds can be told apart in a shared log.
itermax	Integer; maximum DEoptim iterations. See DEoptim::DEoptim.control() .
trace	Logical or integer passed to DEoptim.control() ; how often to report progress.
strategy	Integer in [1, 10]; the DEoptim strategy variant.
initialpop	Optional matrix of starting parameter sets, one row per population member.
NP	Integer; number of population members. Defaults to DEoptim's own rule when NULL.
cores	Character vector of host names, repeated once per worker wanted on that host, or a number for localhost only.
logPath	Path for worker output (outfile of the PSOCK cluster). Each host writes its own copy.
libPath	The project library. This is the master copy: it is mirrored to every host, and workers load from it.
objsNeeded	Character vector naming objects in envir to send to the workers.
pkgsNeeded	Character vector of packages the workers must be able to load. Hosts that cannot are reported by verifyClusterHosts() .
nCoresNeeded	Integer; how many workers to aim for across all hosts.
envir	Environment holding objsNeeded ; defaults to the caller's.
controlArgs	Named list of further DEoptim::DEoptim.control() settings (for example CR , F , c , p , reltol), added to the returned control so they reach DEoptim. A name DEoptim.control() does not have is an error. NP is still the number of workers built, and cluster and parallelType are set here.

Value

A list of items that can be passed to [DEoptim.control\(\)](#)

DEoptimIterative2 *Run DEoptim in cached chunks of generations*

Description

Runs `DEoptim::DEoptim()` a few generations at a time (`iterStep` in ..., default 1), starting each call from the previous call's population, and caches every chunk with `reproducible::Cache()`, so a fit that is interrupted resumes from its last finished chunk. Between chunks it can re-score the surviving population (`options(clusters.deoptimRescoreEvery)`), draws progress figures with `SpaDES.core::Plots()` when `figurePath` is a path, and stops early once the population has stopped improving (see `.deoptimPopulationConverged()`).

Usage

```
DEoptimIterative2(
  fn,
  lower,
  upper,
  control,
  ...,
  .plots,
  figurePath,
  cachePath,
  runName = 1,
  .verbose = TRUE
)
```

Arguments

<code>fn</code>	The objective function, as for <code>DEoptim::DEoptim()</code> .
<code>lower, upper</code>	Parameter bounds, as for <code>DEoptim::DEoptim()</code> .
<code>control</code>	A list of <code>DEoptim::DEoptim.control()</code> settings. Only what is set here is used; the rest get this function's defaults. <code>itermax</code> is the total number of generations.
<code>...</code>	Passed to <code>fn</code> , except <code>iterStep</code> (generations per DEoptim call) and <code>rep</code> .
<code>.plots</code>	Passed to <code>SpaDES.core::Plots()</code> as <code>types</code> for the progress figures.
<code>figurePath</code>	Folder for the progress figures, or <code>FALSE</code> for none.
<code>cachePath</code>	Passed to the progress figures that read the cache.
<code>runName</code>	Distinguishes this fit's cached chunks from another fit's.
<code>.verbose</code>	Passed to <code>reproducible::Cache()</code> .

Value

A list with one `DEoptim::DEoptim()` result per chunk run, in order.

dirNew	<i>Remove duplicate figures, keeping most recent duplicate only</i>
---------------	---

Description

This is for cleaning up cases where an interrupted optimization is leading to multiple files for the same `.runName`. This will remove duplicates, keeping only the most recent.

Usage

```
dirNew(
  path,
  secsAgo = Inf,
  after = Sys.time() - secsAgo,
  pattern = "^(.+)\_\\_[:digit:]{6,8}.*\\.png"
)
```

Arguments

path	A folder in which to search for duplicates
secsAgo	Numeric; consider files modified within this many seconds. Ignored if after is given.
after	A time; consider files modified after it. Defaults to secsAgo before now.
pattern	The regular expression to search for, to identify the files. This must have 1 set of parentheses (), as only the content between the () will be used for duplicate assessment, i.e., remove anything in the file that shouldn't be used.

Value

For side effects: removed files

freeCoresLessReserved	<i>Subtract live reservations from measured free cores</i>
------------------------------	--

Description

Subtract live reservations from measured free cores

Usage

```
freeCoresLessReserved(
  nodes,
  path = getOption("clusters.reservationsPath"),
  loadWindowMinutes = 5,
  graceMinutes = 2
)
```

Arguments

nodes	A data.frame with host and free_est , as built by <code>plan_psock_min()</code> from its probe.
path	Path to the reservations ledger; defaults to <code>getOption("clusters.reservationsPath")</code> .
loadWindowMinutes	The averaging window, in minutes, of the load average free_est was measured from (5 for <code>parallelly::freeCores(memory = "5min")</code>).
graceMinutes	Minutes after it is made during which a reservation counts in full. A reservation is recorded when the cluster has started, but its workers only load the hosts once objects are copied to them and the work begins (2.5-4 minutes for the FireSense spread fits).

Details

free_est already contains the load of clusters that have been running for a while, so subtracting their whole reservation counts those cores twice. A load average is exponentially damped: t minutes after a cluster's load starts it shows $1 - \exp(-t / \text{loadWindowMinutes})$ of it. Each reservation is therefore subtracted only by the share the average has not absorbed yet, $\text{workers} * \exp(-\max(\text{age} - \text{graceMinutes}, 0) / \text{loadWindowMinutes})$: all of it for a cluster built moments ago, which is what stops concurrent builds double-booking, and almost none of it for a cluster that has been running for half an hour.

Value

nodes with **free_est** reduced by the unabsorbed share of every live reservation on that host, floored at zero, plus a **reserved** column (the reserved workers, for reporting). Reservations held by this process count too: a master that already built one cluster is genuinely using those cores while it builds the next.

getHostCombination *Pick how many workers to run on each host*

Description

Takes the per-host timings from `runTests()` and fits, for each host, a linear model of the slowest worker's time against the number of workers. The models are then used to choose an allocation of **Npops** workers across the hosts that finishes soonest, since a host whose per-worker time degrades quickly is worth fewer workers than its core count suggests.

Usage

```
getHostCombination(outs, Npops = 100)
```

Arguments

outs	A list of <code>testMachine()</code> results, one per host, named by host.
Npops	Integer; total number of workers to allocate.

Value

A `data.table` with one row per host, giving the workers assigned and the predicted time.

<code>makeClusterPSOCK</code>	<i>Lightweight wrapper for <code>parallelly::makeClusterPSOCK</code></i>
-------------------------------	--

Description

Lightweight wrapper for `parallelly::makeClusterPSOCK`

Usage

```
makeClusterPSOCK(
  workers,
  outfile = NULL,
  rscript_libs = .libPaths(),
  ...,
  rscript = NULL,
  default_packages = c("datasets", "utils", "grDevices", "graphics", "stats", "methods"),
  port = NULL,
  rshopts = .sshKeepaliveOpts(c("-o", "ForwardX11=no", "-o", "ExitOnForwardFailure=yes")),
  tries = 5L,
  delay = 5,
  renice = 20,
  revtunnel = TRUE
)
```

Arguments

workers	The hostnames of workers (as a character vector) or the number of local-host workers (as a positive integer).
outfile	Optional log file path.
rscript_libs	Optional library paths for workers.
...	Additional arguments passed to <code>parallelly::makeClusterPSOCK</code> .
rscript	The command that starts a worker, passed to <code>parallelly::makeClusterPSOCK()</code> . It is an argument here, not left to <code>...</code> : otherwise <code>rscript</code> = partially matched <code>rscript_libs</code> and became the workers' library path whenever <code>rscript_libs</code> was not also given.
default_packages	Packages R attaches in each worker; the default is <code>parallelly</code> 's.
port	Optional port or port block start.
rshopts	Character vector of options passed to <code>ssh</code> . The defaults disable X11 forwarding and fail fast when a tunnel cannot be established.

tries, delay	Maximum number of attempts done to launch each node with <code>makeNode()</code> and the delay (in seconds) in-between attempts. If argument <code>port</code> specifies more than one port, e.g. <code>port = "random"</code> then a random port will be drawn and validated at most tries times. Arguments tries and delay are used only when <code>setup_strategy == "sequential"</code> .
renice	A numerical 'niceness' (priority) to set for the worker processes.
revtunnel	If TRUE, a reverse SSH tunnel is set up for each worker such that the worker R process sets up a socket connection to its local port (<code>port + rank - 1</code>) which then reaches the master on port <code>port</code> . If FALSE, then the worker will try to connect directly to port <code>port</code> on <code>master</code> . If NA, then TRUE or FALSE is inferred from inspection of <code>rshcmd[1]</code> . For more details, see below.

Details

A worker command that starts with `env` (the OpenBLAS cap from `.workerRscript()`, or an `LD_LIBRARY_PATH` prefix) is not recognised by `parallelly` as `Rscript`, so `parallelly` passes the default packages as an `R_DEFAULT_PACKAGES=` assignment in front of the command. With `renice` that assignment ends up after `nice`, which then fails to run it ("nice: 'R_DEFAULT_PACKAGES=...': No such file or directory"). For such a command the default packages are put among `env`'s own assignments instead.

makeHosts

Build host names from the last octets of their addresses

Description

A convenience for clusters whose hosts sit on one subnet: give the final octets and get back full addresses.

Usage

```
makeHosts(ips, ipbase = "10.20.0.")
```

Arguments

ips	Numeric or character vector of final octets, e.g. <code>c(11, 12)</code> .
ipbase	Character; everything before the final octet, with its trailing dot.

Value

Character vector of addresses.

Examples

```
makeHosts(c(11, 12))
```

<code>mirrorTerraOptions</code>	<i>Give cluster workers the master's terra memory settings</i>
---------------------------------	--

Description

A PSOCK worker is a fresh R session, so it starts at terra's defaults no matter what the master set: `memfrac = 0.5`, `memmax = 16`, `todisk = FALSE`. `memfrac` is a fraction of the machine's *total* RAM and terra applies it per process, so a hundred workers each sizing a working buffer at half of total RAM is not a policy anyone chose – and it silently overrides whatever the master was careful to set.

Usage

```
mirrorTerraOptions(cl)
```

Arguments

`cl` A cluster object, as returned by `parallel::makeCluster()` and friends.

Details

Called by `clusterSetup()` once the workers are up. Does nothing when terra is not installed, on either side.

Value

Invisibly, the named list of settings sent, or NULL if terra is unavailable on the master.

Examples

```
if (requireNamespace("terra", quietly = TRUE)) {  
  cl <- parallel::makeCluster(1L)  
  terra::terraOptions(memfrac = 0, todisk = TRUE)  
  mirrorTerraOptions(cl)  
  parallel::clusterEvalQ(cl, terra::terraOptions(print = FALSE)$memfrac) # 0  
  parallel::stopCluster(cl)  
}
```

monitorCluster
Watch active threads and memory across a cluster's hosts

Description

Polls every worker for its active-thread count and its memory use, and redraws two aligned rows in place: threads on the first, **used/totalGB** on the second. Interrupt it (Ctrl-C) to stop; it then prints the per-host peaks it saw and returns them.

Neither probe needs **clusters** installed on the hosts. The functions are re-homed in the global environment before being sent, so they travel whole rather than as a reference to this package's namespace.

Usage

```
monitorCluster(cl, cores, pad = 2, interval = 1)
```

Arguments

cl	A running cluster with one worker per host, in the same order as cores . If missing, one is built over SSH from cores and stopped on exit.
cores	Character vector of host names, in the cluster's worker order.
pad	Integer; spaces between columns.
interval	Numeric; seconds between polls.

Value

Invisibly, a list with **threads** (peak active threads per host) and **ram** (peak memory used per host, GB). Returned on interrupt.

Note

The display uses ANSI erase sequences to redraw in place. Redirect it to a file and you get one block per tick instead of a live display.

Examples

```
## Not run:
hosts <- c("birds", "biomass", "camas")
peaks <- monitorCluster(cores = hosts)  # Ctrl-C to stop
peaks$ram

## End(Not run)
```

numActiveThreads	<i>Estimate the number of active threads currently being used</i>
------------------	---

Description

This only works on non-linux operating systems, as it uses `ps`

Usage

```
numActiveThreads(pattern = "", minCPU = 50)
```

Arguments

pattern	An optional search pattern to look for when identifying threads that are active. If left at default, then all threads that are active will count.
minCPU	The minimum CPU (in percent , i.e., 0 to 100) that a thread must be using for it to count as actively being used.

Value

An integer representing the current number of threads that are being used as a CPU% greater than `minCPU`. This can be used, e.g., with `parallely::availableCores()` to estimate the number of cores that are available to be used.

Note

This does not address memory or disk use issues.

Examples

```
## Not run:
## Runs until interrupted, so R CMD check must not execute it.

# This will show the active number, updated every 0.5 seconds
cat("Number Active CPUs right now:\n");
while(TRUE) {
  numAC <- clusters::numActiveThreads();
  cat("\r"); cat(numAC); cat(" ");
  Sys.sleep(0.5)
}

## End(Not run)
```

plan_psock_min	<i>Minimal HT-aware PSOCK cluster planner (parallelly-only)</i>
----------------	---

Description

Launches a probe cluster (1 worker per host) via `parallelly::makeClusterPSOCK()` using **reverse SSH tunneling** (`revtunnel = TRUE`) and **sequential** setup (on Rstudio), **parallel** otherwise, configures a **per-user library** on each worker (created via `dir.create(recursive=TRUE)`), installs & loads `pkgsNeeded` in that user library, measures per-host capacity via `parallelly::availableCores()` and `parallelly::freeCores()`, performs a **hyper-threading-aware** allocation of up to **total** workers (penalty beyond ~50% logical occupancy), and optionally launches the final cluster with the computed worker counts.

This is intentionally minimal—no `master` argument, no diagnostics or fallbacks.

Usage

```
plan_psock_min(
  hosts,
  total = 100L,
  beta = 0.5,
  load_memory = "5min",
  fraction = 0.9,
  pkgsNeeded = c("parallelly", "future", "foreach"),
  user_lib_template = "~/local/R/%v/library",
  install_repos = c("https://cloud.r-project.org", "https://cran.r-project.org"),
  rshopts = .sshKeepaliveOpts(),
  rscript = "Rscript",
  auto_stop = TRUE,
  logPath = NULL,
  libPath = NULL,
  build_final_cluster = TRUE
)
```

Arguments

<code>hosts</code>	Character vector of hostnames reachable by SSH.
<code>total</code>	Integer total desired workers (default 100).
<code>beta</code>	Numeric in (0,1], HT penalty beyond 50% logical occupancy (default 0.5).
<code>load_memory</code>	Character; "1min" "5min" "15min" window for <code>freeCores()</code> (default "5min").
<code>fraction</code>	Numeric 0; headroom scale for <code>freeCores()</code> (default 0.9).
<code>pkgsNeeded</code>	Character vector of packages to ensure on workers (default <code>c("parallelly", "future", "foreach")</code>).
<code>user_lib_template</code>	Character; per-user library path template; %v expands to R version (default <code>~/local/R/%v/library</code>).

<code>install_repos</code>	Character vector of CRAN-like repos tried in order (default <code>c("https://cloud.r-project.org")</code>)
<code>rshopts</code>	Character vector of SSH options passed by parallelly. Defaults to <code>.sshKeepaliveOpts()</code> , which bounds how long the master waits on a worker that has died.
<code>rscript</code>	Character; path to <code>Rscript</code> on workers (default <code>"Rscript"</code>).
<code>auto_stop</code>	Logical; auto-stop clusters on GC (default <code>TRUE</code>).
<code>logPath</code>	Optional local file to append compact bootstrap / probe info.
<code>libPath</code>	Optional site library path to prepend and use if writable.
<code>build_final_cluster</code>	Logical; if <code>TRUE</code> launch the final cluster (default <code>TRUE</code>).

Value

A list with:

- `probe`: per-host capacity & load,
- `allocation`: per-host assignments (HT-aware),
- `workers`: hostname vector repeated per assigned workers,
- `total_requested`, `total_assigned`,
- `cluster` (if built).

References

parallelly PSOCK cluster setup & worker startup controls: <https://parallelly.futureverse.org/reference/makeCluster>
 availableCores / freeCores: <https://cran.r-project.org/web/packages/parallelly/parallelly.pdf>
<https://rdrr.io/cran/parallelly/man/freeCores.html> Base parallel PSOCK notes (background):
<https://web.mit.edu/r/current/lib/R/library/parallel/html/makeCluster.html>

Examples

```
## Not run:
plan <- plan_psock_min(
  hosts = c("nodeA", "nodeB", "nodeC", "nodeD"),
  total = 100,
  pkgsNeeded = c("parallelly", "future", "foreach"),
  user_lib_template = "~/local/R/%v/library",
  build_final_cluster = TRUE
)
plan$allocation
parallel::stopCluster(plan$cluster)

## End(Not run)
```

plotMachine	<i>Plot the outputs from testMachine</i>
-------------	--

Description

Plot the outputs from `testMachine`

Usage

```
plotMachine(
  Ncores,
  coreTimes,
  estTotTime,
  optimisticTotTime,
  systemTimes,
  N = 100,
  nam,
  detectedCores
)
```

Arguments

<code>Ncores</code>	Integer vector of worker counts that were timed.
<code>coreTimes</code>	List of per-core <code>range()</code> timings, one element per entry of <code>Ncores</code> .
<code>estTotTime</code>	List of estimated total times, one per entry of <code>Ncores</code> .
<code>optimisticTotTime</code>	List of best-case total times, one per entry of <code>Ncores</code> .
<code>systemTimes</code>	List of measured elapsed times, one per entry of <code>Ncores</code> .
<code>N</code>	Integer; the workload multiplier the times are scaled to.
<code>nam</code>	Character; host name, used in the plot title.
<code>detectedCores</code>	Integer; what the host reports as its core count, drawn as a reference line.

ramUsageGB	<i>Get RAM usage in GB on the current machine</i>
------------	---

Description

Reads `/proc/meminfo` and returns used and total memory in gigabytes. Used is `MemTotal - MemAvailable`, that is, everything the kernel cannot hand out immediately, so page cache in active use counts as used.

Usage

```
ramUsageGB()
```

Value

A list with `used_gb` and `total_gb`, both `NA_real_` where `‘/proc/meminfo’` is absent (any non-Linux host) or unreadable.

Examples

```
ramUsageGB()
```

```
readTransferredObjects
```

Read objects transferred to a worker, failing loudly if the transfer did not arrive

Description

Tries the local (rsynced) copy first, then the original shared path. A failed read used to fall through to `reproducible::.unwrap()` on a `try-error`, which produced objects whose terra external pointers were dangling; the job then died much later with "`external pointer is not valid`", an error saying nothing about a file transfer. Verifying here keeps the error where the cause is.

Usage

```
readTransferredObjects(therePath, filenameForTransfer)
```

Arguments

`therePath` Path to the worker-local copy of the transferred objects.

`filenameForTransfer`

Path to the original, used as a fallback.

Value

The list of transferred objects.

reservationsPath	Core reservations shared between concurrent cluster builds
------------------	--

Description

`plan_psock_min()` sizes a cluster from `parallelly::freeCores()`, which reads a *trailing* load average (5 minutes by default). Two clusters built inside that window therefore both observe the same idle cores and both claim them: the load produced by the first has not yet reached the average the second reads. Staggering the builds by more than the averaging window papers over this; it does not fix it, and it forces callers to hand-tune both the stagger and the number of concurrent jobs against the size of the fleet.

These functions keep a small ledger of what has actually been handed out, so allocation is based on *reservations* rather than on a lagging measurement. Each successful build records `host -> workers` under a reservation id owned by the building process; `freeCoresLessReserved()` subtracts those from the measured free cores. Releases are keyed by id, because one process may hold several clusters at once.

The ledger is self-healing: every read drops entries whose owning process is no longer alive, so a master that is killed without releasing cannot leak a reservation forever.

Usage

```
reservationsPath(path = getOption("clusters.reservationsPath"))
```

```
liveReservations(path = getOption("clusters.reservationsPath"))
```

```
reserveCores(
  alloc,
  id = basename(tempfile("resv")),
  pid = Sys.getpid(),
  path = getOption("clusters.reservationsPath")
)
```

```
releaseCores(
  id = NULL,
  pid = Sys.getpid(),
  path = getOption("clusters.reservationsPath")
)
```

Arguments

<code>path</code>	Directory holding the ledger. Defaults to <code>getOption("clusters.reservationsPath")</code> , else a per-user data directory.
<code>alloc</code>	A data.frame with columns <code>host</code> and <code>assign</code> , as returned in the <code>allocation</code> element of <code>plan_psock_min()</code> .
<code>id</code>	Reservation id. One process may hold several clusters at once, so releases are keyed by reservation rather than by process.
<code>pid</code>	Owning process id; defaults to this process.

Value

`reservationsPath()` a file path; `liveReservations()` a data.frame with columns `id`, `pid`, `host`, `workers`, `created`; `reserveCores()` returns its reservation `id`; `releaseCores()` returns invisibly.

Scope

The ledger is a file, so it is shared by every process that can see that file – which is what is needed when several worker panes on one machine each build their own cluster over a shared fleet. If *different* machines build clusters over the same fleet, point them all at one path on shared storage via `options(clusters.reservationsPath = )`.

<code>resourcesUsed</code>	<i>Assess the machine resources used</i>
----------------------------	--

Description

Uses `vmstat` (must be installed; it is by default on linux).

Usage

```
resourcesUsed(machines = "localhost", resource = "us")
```

Arguments

<code>machines</code>	Character vector of the name(s) of the PSOCK resource to query, e.g., "n168"
<code>resource</code>	Character; which <code>ps</code> field to report, e.g. "us" for user CPU.

Value

Returns the outputs from `vmstat`

<code>rmIncompleteDups</code>	<i>This is for cleaning up cases where an interrupted optimization is leading to multiple files for the same .runName. This will remove duplicates, keeping only the most recent.</i>
-------------------------------	---

Description

This is for cleaning up cases where an interrupted optimization is leading to multiple files for the same `.runName`. This will remove duplicates, keeping only the most recent.

Usage

```
rmIncompleteDups(
  path,
  pattern = "^(.+)\_\_[:digit:]]{5,8}.*\\.png",
  delete = FALSE
)
```

Arguments

path	A folder in which to search for duplicates
pattern	The regular expression to search for, to identify the files. This must have 1 set of parentheses (), as only the content between the () will be used for duplicate assessment, i.e., remove anything in the file that shouldn't be used.
delete	Logical. Default FALSE , which will only list the files that will be deleted. If TRUE , then the identified files will be deleted

Value

For side effects: removed files

runTests	<i>Runs test on each machine in hosts</i>
-----------------	--

Description

Runs test on each machine in **hosts**

Usage

```
runTests(
  hosts,
  repos = c("predictiveecology.r-universe.dev", getOption("repos")),
  clustersBranch = "main",
  RscriptPath = "/usr/local/bin/Rscript",
  Npops = 100
)
```

Arguments

hosts	Character vector of host names to test.
repos	Repositories to install from on each host.
clustersBranch	Branch of this package to install on the hosts.
RscriptPath	Path to Rscript on the hosts.
Npops	Integer; population size the timings are scaled to.

Value

a list; same as `getHostCombination` return.

<code>shipSystemLibs</code>	<i>Ship missing system shared libraries to worker hosts, without root</i>
-----------------------------	---

Description

The project library is rsynced to each host, but the *system* libraries those compiled packages link against are not. A host missing one fails at load with e.g. `libtbb.so.12: cannot open shared object file – RcppParallel`, and so everything that depends on it. The obvious fix, `sudo apt-get install`, needs root on every host, which is exactly what a shared research cluster tends not to give you.

It is not needed. A shared library is just a file: copy it into a user-writable directory on the host and point `LD_LIBRARY_PATH` at it. This detects what is actually missing, ships only that, and returns the directory so the caller can add it to the workers' environment.

Usage

```
shipSystemLibs(cl, hosts, libs, dest = "~/.local/lib/clusters", verbose = TRUE)
```

Arguments

<code>cl</code>	A running cluster with one worker per host (the probe cluster).
<code>hosts</code>	Character vector of host labels, in the same order as <code>cl</code> .
<code>libs</code>	Library path(s) whose compiled code should be checked.
<code>dest</code>	Directory on each host to copy into. Must be user-writable; defaults to <code>~/.local/lib/clusters</code> .
<code>verbose</code>	Logical; report what is shipped.

Value

Invisibly, a named list with `dest` (as given), `destResolved` (the absolute directory on each host), `ldPath` (each host's existing `LD_LIBRARY_PATH`), `shipped` (per host, the sonames copied) and `unmet` (per host, anything still missing that this will not ship).

What is deliberately not shipped

The core runtime – `libc`, `libstdc++`, `libgcc_s`, `libm`, `libpthread`, `libdl`, `librt` and the loader itself. Those must be the host's own: they are what everything else on the machine is already linked against, and overriding them via `LD_LIBRARY_PATH` breaks far more than it fixes. If one of those is genuinely missing, the host is not usable and should be reported, not patched.

When this refuses

Only when the master and the host agree on architecture and libc version is copying a binary between them sound. Hosts that disagree are skipped with a message rather than given a library that cannot load, or worse, one that can.

Setting LD_LIBRARY_PATH

It must be in the worker's environment **before R starts**. glibc parses LD_LIBRARY_PATH once at process startup, so `Sys.setenv()` from inside R – which is what `parallelly`'s `rscrip_envs` does, via `Rscript -e 'Sys.setenv(...)'` – has no effect on later `dlopen()`. Verified: with the library shipped but only `Sys.setenv()`, `library(fireSenseUtils)` still fails; with `env LD_LIBRARY_PATH=... Rscript`, it loads. Prefix the worker command instead.

summaryOutputFolder *Summary – wrapper around dirNew and tableFiles*

Description

Convenient wrapper.

Usage

```
summaryOutputFolder(
  path,
  pattern = "^.+hists/(.+)_\\_iter.+\\_[:digit:]{6,8}.*\\.png"
)
```

Arguments

path	A folder in which to search for duplicates
pattern	The regular expression to base the table on. It should have one and only one parenthesis i.e., <code>(.+)</code> , which will be the basis of the table . Everything outside of the <code>(.+)</code> will be removed

Value

Tabulation of the files, based on the pattern.

tableFiles	<i>Tabulate the filenames into groups based on pattern</i>
------------	--

Description

Using a regular expression, with a single `(.+)` identifying the parts of the filenames to keep, and therefore to base the `table` on. Everything before

Usage

```
tableFiles(  
  files,  
  pattern = "^.+hists/(.+)\$_iter.+\\_\\[[[:digit:]]{6,8}\\}.*\\.png"  
)
```

Arguments

- `files` A vector of full filenames
- `pattern` The regular expression to base the `table` on. It should have one and only one parenthesis i.e., `(.+)`, which will be the basis of the `table`. Everything outside of the `(.+)` will be removed

Value

Tabulation of the files, based on the pattern.

termsInDEoptim	termsInDEoptim
----------------	----------------

Description

termsInDEoptim

Usage

```
termsInDEoptim(fireSense_spreadFormula, thresh, numParams)
```

Arguments

- `fireSense_spreadFormula` The formula to be submitted to `DEoptim::DEoptim()`, from e.g., `sim$fireSense_spreadFormula`
- `thresh` The threshold for accepting fits; e.g., from `mod$thresh`.
- `numParams` The number of parameters (TODO: improve description)

testMachine	<i>Test machines</i>
-------------	----------------------

Description

Test machines

Usage

```
testMachine(NcoresMax = parallel::detectCores(), N = 100, thinning = 5)
```

Arguments

NcoresMax	Integer; the largest worker count to time.
N	Integer; workload multiplier for the reported totals.
thinning	Integer; step between the worker counts that are timed.

transferDirName	<i>Build the per-job worker transfer directory name</i>
-----------------	---

Description

Unique per master process. This was a single shared `/tmp/fireSense_SpreadFit` for every job, while the cleanup step removes `dirname(therePath)` recursively – so concurrent jobs deleted each other’s staged objects mid-read.

Usage

```
transferDirName()
```

Value

A directory path, as a length-one character vector.

verifyClusterHosts	<i>Verify that every cluster host can actually run a worker</i>
--------------------	---

Description

`plan_psock_min()` rsyncs the master's library to the same absolute path on each host and prepends it. Three things can silently go wrong with that, and each produces a worker that dies later rather than a build that fails now:

- **R version drift.** A library built under one R major.minor is ignored by another, which then falls back to the system library and cannot find the packages at all. (`plan_psock_min()` has always checked this one.)
- **Package version drift.** The rsync only carries the dependency closure of `pkgsNeeded`, so anything outside it can differ between master and worker – including the packages doing the orchestrating.
- **A library that is present but unloadable**, typically a missing system library (`libtbb.so.12` for `RcppParallel`, `libgdal.so.NN` for `sf/terra`) or a binary built against a different GDAL.

This checks all three against the master and reports every offending host at once, rather than failing on the first or – worse – succeeding and letting the workers die one at a time inside the objective function.

Usage

```
verifyClusterHosts(cl, hosts, pkgs, libs, action = c("stop", "drop"))
```

Arguments

<code>cl</code>	A running cluster with one worker per host (the probe cluster).
<code>hosts</code>	Character vector of host labels, in the same order as <code>cl</code> .
<code>pkgs</code>	Character vector of packages that a worker must be able to load.
<code>libs</code>	Library paths to prepend on each worker (the master's).
<code>action</code>	"stop" (default) to error listing the bad hosts, or "drop" to return only the hosts that passed.

Value

Character vector of hosts that passed, invisibly. With `action = "stop"` that is always `hosts`.

visualizeDE	<i>Make histograms of DEoptim object pars</i>
-------------	---

Description

Make histograms of DEoptim object `pars`

Usage

```
visualizeDE(DE, cachePath, titles, lower, upper)
```

Arguments

<code>DE</code>	An object from a DEoptim call
<code>cachePath</code>	A <code>cacheRepo</code> to pass to <code>showCache</code> and <code>loadFromCache</code> if <code>DE</code> is missing.
<code>titles</code>	Character vector of parameter names, one per histogram.
<code>lower</code>	Named numeric of lower bounds, used to fix each panel's x range.
<code>upper</code>	Named numeric of upper bounds, used to fix each panel's x range.

Index

`.sshKeepaliveOpts`, 2
`.sshKeepaliveOpts()`, 14
`.workerRscript()`, 9

`clusterSetup`, 3
`clusterSetup()`, 10

`DEoptim::DEoptim()`, 5, 22
`DEoptim::DEoptim.control()`, 4, 5
`DEoptimIterative2`, 5
`dirNew`, 6

`freeCoresLessReserved`, 6
`freeCoresLessReserved()`, 17

`getHostCombination`, 7

`liveReservations (reservationsPath)`, 17

`makeClusterPSOCK`, 8
`makeHosts`, 9
`mirrorTerraOptions`, 10
`monitorCluster`, 11

`numActiveThreads`, 12

`parallel::makeCluster()`, 10
`parallelly::availableCores()`, 13
`parallelly::freeCores()`, 13, 17
`parallelly::makeClusterPSOCK()`, 8, 13
`plan_psock_min`, 13
`plan_psock_min()`, 7, 17
`plotMachine`, 15

`ramUsageGB`, 15
`readTransferredObjects`, 16
`releaseCores (reservationsPath)`, 17
`reproducible::unwrap()`, 16
`reproducible::Cache()`, 5
`reservationsPath`, 17

`reserveCores (reservationsPath)`, 17
`resourcesUsed`, 18
`rmIncompleteDups`, 18
`runTests`, 19
`runTests()`, 7

`shipSystemLibs`, 20
`summaryOutputFolder`, 21

`tableFiles`, 22
`termsInDEoptim`, 22
`testMachine`, 23
`testMachine()`, 7
`transferDirName`, 23

`verifyClusterHosts`, 24
`verifyClusterHosts()`, 4
`visualizeDE`, 25