Package 'autometric'

October 10, 2024

Title Background Resource Logging

Description Intense parallel workloads can be difficult to monitor.
Packages 'crew.cluster', 'clustermq', and 'future.batchtools'
distribute hundreds of worker processes over multiple computers.
If a worker process exhausts its available memory,
it may terminate silently, leaving the underlying problem difficult
to detect or troubleshoot.
Using the 'autometric' package, a worker can proactively monitor
itself in a detached background thread.
The worker process itself runs normally,
and the thread writes to a log every few seconds.
If the worker terminates unexpectedly, 'autometric' can read and
visualize the log file to reveal potential resource-related
reasons for the crash. The 'autometric' package borrows heavily from
the methods of packages 'ps' <doi:10.32614/CRAN.package.ps> and 'psutil'.

Version 0.0.5

License MIT + file LICENSE

URL https://wlandau.github.io/autometric/,
https://github.com/wlandau/autometric

BugReports https://github.com/wlandau/autometric/issues

Depends R (>= 3.5.0)

Imports graphics, utils

Suggests grDevices, ps, tinytest

Encoding UTF-8

Language en-US

RoxygenNote 7.3.2

NeedsCompilation yes

Author William Michael Landau [aut, cre]
(<https://orcid.org/0000-0003-1878-3253>),
Eli Lilly and Company [cph, fnd],
Posit Software, PBC [cph] (For the 'ps' package. See LICENSE.note.),

Jay Loden [cph] (For the 'psutil' package. See LICENSE.note.),
Dave Daeschler [cph] (For the 'psutil' package. See LICENSE.note.),
Giampaolo Rodola [cph] (For the 'psutil' package. See LICENSE.note.)

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

Repository CRAN

Date/Publication 2024-10-10 16:30:02 UTC

Contents

`log_active` *Check the log thread.*

Description

Check if the log is running.

Usage

```
log_active()
```

Value

`TRUE` if a background thread is actively writing to the log, `FALSE` otherwise. The result is based on a static C variable, so the information is second-hand.

Examples

```
path <- tempfile()
log_active()
log_start(seconds = 0.5, path = path)
log_active()
Sys.sleep(2)
log_stop()
Sys.sleep(2)
log_active()
unlink(path)
```

log_plot *Plot a metric of a process over time*

Description

Visualize a metric of a log over time for a single process ID in a single log file.

Usage

```
log_plot(log, pid = NULL, name = NULL, metric = "resident", ...)
```

Arguments

log	Data frame returned by log_read(). Must be nonempty. log_plot() only includes rows with status code equal to 0.
pid	Either NULL or a non-negative integer with the process ID to plot. At least one of pid or name must be NULL.
name	Either NULL or a non-negative integer with the name of the process to plot. The name was previously specified in the names of the pid argument of log_start() or log_print(). At least one of pid or name must be NULL.
metric	Character string with the name of a metric to plot against time. Must be only a single string. Defaults to the resident set size (RSS), the total amount of memory used by the process. See log_read() for descriptions of the metrics available.
...	Named optional parameters to pass to the base function plot().

Value

A base plot of a metric of a log over time.

Examples

```
path <- tempfile()
log_start(seconds = 0.5, path = path)
Sys.sleep(1)
log_stop()
Sys.sleep(2)
log <- log_read(path)
log_plot(log, metric = "cpu")
unlink(path)
```

log_print *Print once to the log.*

Description

Sample CPU load metrics and print a single line to the log for each process in `pids`. Used for debugging and testing only. Not for users.

Usage

```
log_print(
  path,
  seconds = 1,
  pids = c(local = Sys.getpid()),
  error = getOption("autometric_error", TRUE)
)
```

Arguments

`path`	Character string, path to a file to log resource usage. On Windows, the path must point to a physical file on disk. On Unix-like systems, `path` can be `"/dev/stdout"` to print to standard output or `"/dev/stderr"` to print to standard error. Standard output is the most convenient option for high-performance computing scenarios where worker processes already write to log files. Such workers usually already redirect standard output to a physical file, as with a cluster like SLURM, or capture messages with Amazon CloudWatch. Normally, standard output and standard error are discouraged because of how they interact with the R API and R console. However, the exported user interface of `autometric` only ever prints from a detached POSIX thread where it is unsafe to use `Rprintf()` or `REprintf()`.
`seconds`	Positive number, number of seconds to sample CPU load metrics before printing to the log. This number should be at least 1, usually more. A low number of seconds could burden the operating system and generate large log files.
`pids`	Nonempty vector of non-negative integers of process IDs to monitor. NOTE: On Mac OS, only the currently running process can be monitored. This is due to security restrictions around certain system calls, c.f. https://os-tres.net/blog/2010/02/17/mac-os-x-and-task-for-pid-mach-call/. # nolint If the `pids` vector is named, then the names will show alongside the process IDs in the log entries.
`error`	`TRUE` to throw an error if the thread is not supported on the current platform. (See `log_support()`.)

Value

`NULL` (invisibly). Called for its side effects.

Examples

```
path = tempfile()
log_print(path = path)
log_read(path)
unlink(path)
```

`log_read` *Read a log.*

Description

Read a log file into R.

Usage

```
log_read(
  path,
  units_cpu = c("percentage", "fraction"),
  units_memory = c("megabytes", "bytes", "kilobytes", "gigabytes"),
  units_time = c("seconds", "minutes", "hours", "days"),
  hidden = FALSE
)
```

Arguments

`path`	Character vector of paths to files and/or directories of logs to read.
`units_cpu`	Character string with the units of the `cpu` field. Defaults to `"percentage"` and must be in `c("percentage", "fraction")`.
`units_memory`	Character string with the units of the `memory` field. Defaults to `"megabytes"` and must be in `c("megabytes", "bytes", "kilobytes", "gigabytes")`.
`units_time`	Character string, units of the `time` field. Defaults to `"seconds"` and must be in `c("seconds", "minutes", "hours", "days")`.
`hidden`	`TRUE` to include hidden files in the files and directories listed in `path`, `FALSE` to omit.

Details

`log_read()` is capable of reading a log file where both `autometric` and other processes have printed. Whenever `autometric` writes to a log, it bounds the beginning and end of the text with the keyword `"__AUTOMETRIC__"`. that way, `log_read()` knows to only read and process the correct lines of the file.

In addition, it automatically converts the log data into the units `units_time`, `units_cpu`, and `units_memory` arguments.

Value

A data frame of metrics from the log with one row per log entry and columns with metadata and resource usage metrics. `log_read()` automatically converts the data into the units chosen with arguments `units_time`, `units_cpu`, and `units_memory`. The returned data frame has the following columns:

- `version`: Version of the package used to write the log entry.
- `pid`: Process ID monitored.
- `status`: A status code for the log entry. Status 0 means logging succeeded. A status code not equal to 0 indicates something went wrong and the metrics should not be trusted.
- `time`: numeric time stamp at which the entry was logged. `log_read()` automatically recenters this column so that time 0 indicates the first logged entry. Use the `units_time` argument to customize the units of this field.
- `core`: CPU load of the process scaled relative to a single CPU core. Measures the amount of time the process spends running during a given interval of elapsed time.

  On Mac OS, the package uses native system calls to get CPU core usage. On Linux and Windows, the package calculates it manually using. user + kernel clock cycles that ran during a sampling interval. It measures the clock cycles that the process executed during the interval, converts the clock cycles into seconds, then divides the result by the elapsed time of the interval. The length of the sampling interval is the `seconds` argument supplied to `log_start()`, or length of time between calls to `log_print()`. The first `core` measurement is 0 to reflect that a full sampling interval has not elapsed yet.

  `core` can be read in as a percentage or fraction, depending on the `units_cpu` argument.
- `cpu`: `core` divided by the number of logical CPU cores. This metric measures the load on the machine as a whole, not just the CPU core it runs on. Use the `units_cpu` argument to customize the units of this field.
- `rss`: resident set size, the total amount of memory used by the process at the time of logging. This include the memory unique to the process (unique set size USS) and shared memory. Use the `units_memory` argument to customize the units of this field.
- `virtual`: total virtual memory available to the process. The process does not necessarily use all this memory, but it can request more virtual memory throughout its life cycle. Use the `units_memory` argument to customize the units of this field.

Examples

```
path <- tempfile()
log_start(seconds = 0.5, path = path)
Sys.sleep(2)
log_stop()
Sys.sleep(2)
log_read(path)
unlink(path)
```

log_start *Start the log thread.*

Description

Start a background thread that periodically writes system usage metrics of the current R process to a log file. See `log_read()` for explanations of the specific metrics.

Usage

```
log_start(
  path,
  seconds = 1,
  pids = c(local = Sys.getpid()),
  error = getOption("autometric_error", TRUE)
)
```

Arguments

`path`	Character string, path to a file to log resource usage. On Windows, the path must point to a physical file on disk. On Unix-like systems, `path` can be `"/dev/stdout"` to print to standard output or `"/dev/stderr"` to print to standard error. Standard output is the most convenient option for high-performance computing scenarios where worker processes already write to log files. Such workers usually already redirect standard output to a physical file, as with a cluster like SLURM, or capture messages with Amazon CloudWatch. Normally, standard output and standard error are discouraged because of how they interact with the R API and R console. However, the exported user interface of `autometric` only ever prints from a detached POSIX thread where it is unsafe to use `Rprintf()` or `REprintf()`.
`seconds`	Positive number, number of seconds between writes to the log file. This number should be noticeably large, anywhere between half a second to several seconds or minutes. A low number of seconds could burden the operating system and generate large log files. Because of the way CPU usage measurements work, the first log entry starts only after after the first interval of `seconds` has passed.
`pids`	Nonempty vector of non-negative integers of process IDs to monitor. NOTE: On Mac OS, only the currently running process can be monitored. This is due to security restrictions around certain system calls, c.f. `https://os-tres.net/blog/2010/02/17/mac-os-x-and-task-for-pid-mach-call/`. # nolint If the `pids` vector is named, then the names will show alongside the process IDs in the log entries.
`error`	`TRUE` to throw an error if the thread is not supported on the current platform. (See `log_support()`.)

Details

Only one thread can run at a time. If the thread is already running, then `log_start()` does not start an additional one. Before creating a new thread, call `log_stop()` to terminate the first one.

Value

`NULL` (invisibly). Called for its side effects.

Examples

```
path <- tempfile()
log_start(seconds = 0.5, path = path)
Sys.sleep(2)
log_stop()
Sys.sleep(2)
log_read(path)
unlink(path)
```

---

log_stop *Stop the log thread.*

Description

Stop the background thread that periodically writes system usage metrics of the current R process to a log file.

Usage

```
log_stop()
```

Details

The background thread is detached, so is there no way to directly terminate it (other than terminating the main thread, i.e. restarting R). `log_stop()` merely signals to the thread using a static C variable protected by a mutex. It may take time for the thread to notice, depending on the value of `seconds` you supplied to `log_start()`. For this reason, you may see one or two lines in the log even after you call `log_stop()`.

Value

`NULL` (invisibly). Called for its side effects.

Examples

```
path <- tempfile()
log_start(seconds = 0.5, path = path)
Sys.sleep(2)
log_stop()
log_read(path)
unlink(path)
```

log_support *Log support*

Description

Check if your system supports background logging.

Usage

```
log_support()
```

Details

The background logging functionality requires a Linux, Mac, or Windows computer, It also requires POSIX thread support and the `nanosleep()` C function.

Value

`TRUE` if your system supports background logging, `FALSE` otherwise.

Examples

```
log_support()
```

Index