Package ‘multidplyr’

December 1, 2021

Title A Multi-Process 'dplyr' Backend
Version 0.1.1
Description Partition a data frame across multiple worker processes to provide simple multicore parallelism.
License MIT + file LICENSE

URL https://multidplyr.tidyverse.org,
https://github.com/tidyverse/multidplyr

BugReports https://github.com/tidyverse/multidplyr/issues

Depends R (>= 3.5.0)
Imports callr (>= 3.5.1), crayon, dplyr (>= 1.0.0), magrittr, qs (>= 0.24.1), R6, rlang, tibble, vctrs (>= 0.3.6), tidyselect
Suggests covr, knitr, lubridate, mgcv, nycflights13, rmarkdown, testthat (>= 3.0.2), vroom, withr

VignetteBuilder knitr

Encoding UTF-8
RoxygenNote 7.1.2

Config/testthat.edition 3
Config/Needs/website tidyverse/tidytemplate

NeedsCompilation no

Author Hadley Wickham [aut, cre],
RStudio [cph]

Maintainer Hadley Wickham <hadley@rstUDIO.com>

Repository CRAN

Date/Publication 2021-12-01 21:10:02 UTC
cluster_call

Description

`cluster_call()` executes the code on each worker and returns the results; `cluster_send()` executes the code ignoring the result. Jobs are submitted to workers in parallel, and then we wait until they’re complete.

Usage

```r
cluster_call(cluster, code, ptype = list())
cluster_send(cluster, code)
```

Arguments

- `cluster` A cluster.
- `code` An expression to execute on each worker.
- `ptype` Determines the output type. The default returns a list, which will always succeed. Set to a narrower type to simplify the output.

Value

A list of results with one element for each worker in ‘cluster’.

Examples

```r
cl <- default_cluster()

# Run code on each cluster and retrieve results
cluster_call(cl, Sys.getpid())
cluster_call(cl, runif(1))

# use ptype to simplify
cluster_call(cl, runif(1), ptype = double())

# use cluster_send() to ignore results
cluster_send(cl, x <- runif(1))
cluster_call(cl, x, ptype = double())
```
Description

These functions provide useful helpers for performing common operations. `cluster_assign()` assigns the same value on each worker; `cluster_assign_each()` assigns different values on each worker; `cluster_assign_partition()` partitions vectors so that each worker gets (approximately) the same number of pieces.

Usage

```r
cluster_assign(.cluster, ...)
cluster_assign_each(.cluster, ...)
cluster_assign_partition(.cluster, ...)
cluster_copy(cluster, names, env = caller_env())
cluster_rm(cluster, names)
cluster_library(cluster, packages)
```

Arguments

- `...`: Name-value pairs
- `cluster, .cluster`: Cluster to work on
- `names`: Name of variables to copy.
- `env`: Environment in which to look for variables to copy.
- `packages`: Character vector of packages to load

Value

Functions that modify the worker environment invisibly return `cluster` so calls can be piped together. The other functions return lists with one element for each worker.

Examples

```r
c1 <- default_cluster()
c1st <- cluster_assign(c1, a = runif(1))
c1st <- cluster_call(c1, a)

# Assign different values on each cluster
c1st <- cluster_assign_each(c1, b = c(1, 10))
c1st <- cluster_call(c1, b)
```
# Partition a vector so that each worker gets approximately the
# same amount of it
cluster_assign_partition(cl, c = 1:11)
cluster_call(cl, c)

# If you want different to compute different values on each
# worker, use `cluster_call()` directly:
cluster_call(cl, d <- runif(1))
cluster_call(cl, d)

# cluster_copy() is a useful shortcut
e <- 10
cluster_copy(cl, "e")

cluster_call(cl, ls())
cluster_rm(cl, letters[1:5])
cluster_call(cl, ls())

# Use cluster_library() to load packages
cluster_call(cl, search())
cluster_library(cl, "magrittr")
cluster_call(cl, search())

---

**new_cluster**

Create a new cluster with sensible defaults.

---

**Description**

Clusters created with this function will automatically clean up after themselves.

**Usage**

new_cluster(n)

**Arguments**

- **n**
  
  Number of workers to create. Avoid setting this higher than the number of cores in your computer as it will degrade performance.

**Value**

A `multidplyr_cluster` object.

**Examples**

```r
cluster <- new_cluster(2)
cluster```

Partition data across workers in a cluster

Description
Partitioning ensures that all observations in a group end up on the same worker. To try and keep the observations on each worker balanced, `partition()` uses a greedy algorithm that iteratively assigns each group to the worker that currently has the fewest rows.

Usage

```r
partition(data, cluster)
```

Arguments

- `data` Dataset to partition, typically grouped. When grouped, all observations in a group will be assigned to the same cluster.
- `cluster` Cluster to use.

Value
A `party_df`.

Examples

```r
library(dplyr)
cl <- default_cluster()
cluster_library(cl, "dplyr")

mtcars2 <- partition(mtcars, cl)
mtcars2 %>% mutate(cyl2 = 2 * cyl)
mtcars2 %>% filter(vs == 1)
mtcars2 %>% group_by(cyl) %>% summarise(n())
mtcars2 %>% select(-cyl)
```

party_df

A `party_df` partitioned data frame

Description
This S3 class represents a data frame partitioned across workers in a cluster. You can use this constructor if you have already spread data frames spread across a cluster. If not, start with `partition()` instead.

Usage

```r
party_df(cluster, name, auto_rm = FALSE)
```
Arguments

cluster A cluster
name Name of data frame variable. Must exist on every worker, be a data frame, and have the same names.
auto_rm If ‘TRUE’, will automatically ‘rm()’ the data frame on the workers when this object is created.

Value

An S3 object with class ‘multidplyr_party_df’.

Examples

# If a real example, you might spread file names across the clusters
# and read in using data.table::fread()/vroom::vroom()/qs::qread().
cl <- default_cluster()
cluster_send(cl[1], n <- 10)
cluster_send(cl[2], n <- 15)
cluster_send(cl, df <- data.frame(x = runif(n)))

df <- party_df(cl, "df")
df
Index

cluster_assign (cluster_utils), 3
cluster_assign_each (cluster_utils), 3
cluster_assign_partition
  (cluster_utils), 3
cluster_call, 2
cluster_copy (cluster_utils), 3
cluster_library (cluster_utils), 3
cluster_rm (cluster_utils), 3
cluster_send (cluster_call), 2
cluster_utils, 3

new_cluster, 4

partition, 5
party_df, 5