Package 'doRedis'

February 19, 2015

Type Package

Title Foreach parallel adapter for the rredis package

Version 1.1.1

Date 2014-2-25

Author B. W. Lewis <blewis@illposed.net>

Maintainer B. W. Lewis <blewis@illposed.net>

Description A Redis parallel backend for the %dopar% function

BugReports https://github.com/bwlewis/doRedis/issues

Depends R (>= 2.5), rredis (>= 1.6.8), foreach(>= 1.3.0), iterators(>= 1.0.0), utils

Suggests boot

License GPL-2

LazyLoad yes

NeedsCompilation yes

Repository CRAN

Date/Publication 2014-04-03 18:30:30

R topics documented:

doRedis-package *A Redis-based back end for parallel computing with foreach.*

Description

The doRedis package supplies a lightweight, cross-platform parallel back end for the foreach `%dopar%` function.

Details

The foreach package for R defines a modular interface for back end parallel processing implementations. The doRedis package imlpements a simple but very flexible parallel back end that uses Redis for inter-process communication.

The doRedis package requires a connection to an available Redis server (not included with the package).

Two foreach parameters are specific to the doRedis back end: `chunkSize` (default value 1), and `ftinterval` (default value 30). The `chunkSize` option sets the default number of jobs that are doled out to each worker process. Jobs are doled out one at a time by default. Setting the chunk size larger for shorter-running jobs can substantially improve performance. Setting this value too high can negatively impact load-balancing across workers, however.

The `ftinterval` option sets the number of seconds between checks for back end worker failures. Failed jobs will be re-submitted after this interval.

Author(s)

B. W. Lewis <blewis@illposed.net>

References

http://cran.r-project.org/web/packages/foreach/index.html

See Also

foreach

Examples

```
## Not run:
# The example assumes that a Redis server is running on the local host
# and standard port.

# 1. Open one or more 'worker' R sessions and run:
require('doRedis')
redisWorker('jobs')

# We use the name 'jobs' to identify a work queue.
# 2. Open another R session acting as a 'master' and run this simple
```

```
#    sampling approximation of pi:
require('doRedis')
registerDoRedis('jobs')
foreach(j=1:10,.combine=sum,.multicombine=TRUE) %dopar%
          4*sum((runif(1000000)^2 + runif(1000000)^2)<1)/10000000

## End(Not run)
```

---

redisWorker *redisWorker*

Description

Initialize a doRedis worker R process.

Usage

```
redisWorker(queue,
            host = "localhost",
            port = 6379,
            iter = Inf,
            timeout = 30,
            log = stdout(),
            connected=FALSE,
            password=NULL)
```

Arguments

queue	A (character) work queue name, or a list or character vector of queue names.
host	The Redis server host name or (character) I. P. address.
port	The Redis server port number.
iter	The maximum number of jobs to execute before exiting the worker loop (defaults to infinity).
timeout	The worker loop terminates if the work queue is deleted after the specified timeout interval.
log	Log messages to the specified destination (defaults to stdout()).
connected	Is the R session creating the worker already connected to Redis?
password	The Redis server password.

Details

The redisWorker function enrolls the current R session in one or more doRedis worker pools specified by the work queue names. The worker loop takes over the R session until either the work queue(s) are deleted or the worker times out waiting for a new task.

Value

Nothing is returned but status messages are printed to the log during operation of the worker loop.

Note

All doRedis functions require network access to a Redis server (not included with the doRedis package).

Author(s)

B. W. Lewis <blewis@illposed.net>

See Also

registerDoRedis

Examples

```
## Not run:
require('doRedis')
redisWorker('jobs')

## End(Not run)
```

---

registerDoRedis *Register the doRedis parallel back end with foreach.*

Description

The doRedis package supplies a simple and lightweight parallel back end for the foreach %dopar% function.

Usage

```
registerDoRedis(queue, host = "localhost", port = 6379, password=NULL)
```

Arguments

queue	A (character) work queue name.
host	The Redis server host or (character) I. P. address.
port	The Redis server port number.
password	The Redis server password.

Details

The doRedis package imlpements a simple but flexible parallel back end for foreach that uses Redis for inter-process communication. The work queue name specifies the base name of a small set of Redis keys that the master and worker processes use to exchange data.

Back-end worker R processes advertise their availablility for work with the `redisWorker` function.

The doRedis parallel back end tolerates faults among the worker processes and automatically re-submits failed tasks. It is also portable and supports heterogeneous sets of workers, even across operative systems. The back end supports dynamic pools of worker processes. New workers may be added to work queues at any time and can be immediately used by in-flight foreach computations.

Value

Nothing is returned.

Note

All doRedis functions require network access to a Redis server (not included with the doRedis package).

Author(s)

B. W. Lewis <blewis@illposed.net>

References

http://cran.r-project.org/web/packages/foreach/index.html

See Also

`redisWorker` `removeQueue`

Examples

```
## Not run:
## The example assumes that a Redis server is running on the local host
## and standard port.

## 1. Open one or more 'worker' R sessions and run:
require('doRedis')
redisWorker('jobs')

## 2. Open another R session acting as a 'master' and run this simple
##    sampling approximation of pi:
require('doRedis')
registerDoRedis('jobs')
foreach(j=1:10,.combine=sum,.multicombine=TRUE) %dopar%
          4*sum((runif(1000000)^2 + runif(1000000)^2)<1)/10000000
removeQueue('jobs')

## End(Not run)
```

removeQueue *Remove doRedis work queues.*

Description

Use the removeQueue function to delete one or more doRedis work queues.

Usage

```
removeQueue(queue)
```

Arguments

queue	A character work queue name or vector or list of queue names.

Details

This action will terminate the worker loops running on any corresponding back-end workers. Upon termination the workers will clean up any ancillary Redis keys.

Value

TRUE is returned if the queues were successfully deleted, FALSE and probably an error condition otherwise.

Note

All doRedis functions require network access to a Redis server (not included with the doRedis package).

Author(s)

B. W. Lewis <blewis@illposed.net>

See Also

registerDoRedis

Examples

```
## Not run:
# The example assumes that a Redis server is running on the local host
# and standard port.

# 1. Open one or more 'worker' R sessions and run:
require('doRedis')
redisWorker('jobs')

# We use the name 'jobs' to identify a work queue.
```

```
# 2. Open another R session acting as a 'master' and run this simple
#    sampling approximation of pi:
require('doRedis')
registerDoRedis('jobs')
foreach(j=1:10,.combine=sum,.multicombine=TRUE) %dopar%
          4*sum((runif(1000000)^2 + runif(1000000)^2)<1)/10000000
removeQueue('jobs')

## End(Not run)
```

---

setChunkSize *setChunkSize*

Description

Set the default granularity of distributed tasks.

Usage

```
setChunkSize(value = 1)
```

Arguments

value	The new default chunk size.

Details

The setChunkSize function lets users set the default number of jobs that are doled out to each worker process. The doRedis package doles out jobs one at a time by default. Setting the default chunk size larger for shorter-running jobs can substantially improve performance. Setting this value too high can negatively impact load-balancing across workers, however.

This value is overriden by setting the 'chunkSize' option in the foreach loop (see the examples).

Value

Nothing is returned.

Note

All doRedis functions require network access to a Redis server (not included with the doRedis package).

Author(s)

B. W. Lewis <blewis@illposed.net>

Examples

```
## Not run:
require('doRedis')
setChunkSize(10)

## Override the default value in a loop as shown in the following example:
foreach(j=1:1000, .options.redis=list(chunkSize=100))

## End(Not run)
```

`setExport` *setExport*

Description

Manually add symbol names to the worker environment export list.

Usage

```
setExport(names = c())
```

Arguments

`names`	A vector of symbol names to export.

Details

The setExport function lets users manually declare symbol names of corresponding objects that should be exported to workers.

The `foreach` function includes a similar `.export` parameter.

We provide this supplemental export option for users without direct access to the `foreach` function, for example, when `foreach` is used within a package.

Value

Nothing is returned.

Author(s)

B. W. Lewis <blewis@illposed.net>

Examples

```
## Not run:
require("doRedis")
registerDoRedis("work queue")
startLocalWorkers(n=1, queue="work queue")

f <- function() pi

foreach(1)
# Returns the error:
# Error in eval(call("f")) : task 1 failed - could not find function "f"

# Manuall export the symbol f:
setExport("f")
foreach(1)
# Ok then.
#[[1]]
#[1] 3.141593
removeQueue("work queue")

## End(Not run)
```

`setGetTask` *setGetTask*

Description

Define a function used by workers to pull tasks.

Usage

```
setGetTask(fn = default_getTask)
```

Arguments

`fn`	A task pulling function, see details below.

Details

The `doRedis` package organizes work into a collections of tasks called a job. One job may contain several uniquely identified task collections, and jobs themselves are uniquely identified. The task collections are labeled by the `taskLabel` function.

A `doRedis` task is specifically a list of two elements: `task_id` and `args`, that specify the unique ID of the task collection, and the `foreach` loop expression arguments, respectively.

As of version 1.1.0 of the `doRedis` package, task collections are placed in a Redis `hash` table identified by the job ID. The `getTask` function is used by the R worker processes to pull tasks from this hash table. The `getTask` function must take at least two arguments, `queue` and `job_id` that

specify the job queue and job ID, respectively. The function should work with Redis to obtain and return a task collection, or return NULL if no tasks are available.

The getTask function almost always defines a short Lua scipt run on the Redis server.

The default getTask function removes tasks in the order that they appear in the hash table. Custom getTask functions can be defined often in association with custom taskLabel functions. The custom functions can instruct Redis to perferentially dole out tasks based on network distance to data, for example.

Value

Nothing is returned.

Note

All doRedis functions require network access to a Redis server (not included with the doRedis package).

Author(s)

B. W. Lewis <blewis@illposed.net>

Examples

```
## Not run:
require('doRedis')

# The default getTask function defines a Redis Lua script that returns the first
# listed task or NULL:sk <-  function(queue, job_id, ...)

getTask <- function(queue, job_id, ...)
{
  key <- sprintf("
  redisEval("local x=redis.call('hkeys',KEYS[1])[1];
             if x==nil then return nil end;
             local ans=redis.call('hget',KEYS[1],x);
             redis.call('hdel',KEYS[1],x);i
             return ans",key)
}
setGetTask(getTask)

## End(Not run)
```

---

setTag *setTag*

Description

Set a worker 'tag' reported to the getTask function.

Usage

```
setTag(label)
```

Arguments

label	A character label.

Details

Workers report their tags to the getTask function. They are ignored by this function by default. Use custom tags in combination with custom Redis server-side Lua scripts to define custom task-pulling behavior.

Value

Nothing is returned.

Author(s)

B. W. Lewis <blewis@illposed.net>

---

setTaskLabel *setTaskLabel*

Description

Define a function that uniquely labels collections of tasks.

Usage

```
setTaskLabel(fn = I)
```

Arguments

fn	A task labeling function, see details below.

Details

The doRedis package organizes work into a collections of tasks called a job. One job may contain several uniquely identified task collections, and jobs themselves are uniquely identified. The task collections are labeled by the taskLabel function.

The taskLabel function must take exactly one argument, a single integer value. Its output must be coerceable into a character string and the function must be injective.

The default getTask function removes tasks in the order that they appear in the hash table. Custom getTask functions can be defined often in association with custom taskLabel functions. The custom functions can instruct Redis to perferentially dole out tasks based on network distance to data, for example.

Value

Nothing is returned.

Note

All doRedis functions require network access to a Redis server (not included with the doRedis package).

Author(s)

B. W. Lewis <blewis@illposed.net>

Examples

```
## Not run:
require('doRedis')
# The default getTask function defines a Redis Lua script that returns the first
# listed task or NULL:sk <-  function(queue, job_id, ...)
getTask <- function(queue, job_id, ...)
{
  key <- sprintf("
  redisEval("local x=redis.call('hkeys',KEYS[1])[1];
             if x==nil then return nil end;
             local ans=redis.call('hget',KEYS[1],x);
             redis.call('hdel',KEYS[1],x);i
             return ans",key)
}

setTaskLabel(fn = I)

## End(Not run)
```

---

startLocalWorkers *startLocalWorkers*

Description

Start background R worker processes on the local system.

Usage

```
startLocalWorkers(n, queue, host = "localhost",
                  port = 6379, iter = Inf,
                  timeout = 30, log = stdout(),
                  Rbin = paste(R.home(component="bin"),"R",sep="/"),
                  password=NULL)
```

Arguments

n	The number of workers to start.
queue	A (character) work queue name, or a list or character vector of queue names.
host	The Redis server host name or (character) I. P. address.
port	The Redis server port number.
iter	The maximum number of jobs to execute before exiting the worker loop (defaults to infinity).
timeout	The worker loop terminates if the work queue is deleted after the specified timeout interval.
log	Log messages to the specified destination (defaults to stderr()).
Rbin	The full path to the command-line R program.
password	The Redis server password.

Details

Use `startLocalWorkers` to start one or more doRedis R worker processes in the background. The worker processes are started on the local system using the `redisWorker` function.

Running workers self-terminate when their work queues are deleted with the `removeQueue` function.

Value

Nothing is returned. Set the `log` parameter to `stdout()` to see log messages printed on standard output of the invoking R session.

Note

All doRedis functions require network access to a Redis server (not included with the doRedis package).

Author(s)

B. W. Lewis <blewis@illposed.net>

See Also

`registerDoRedis` `redisWorker`

Examples

```
## Not run:
require('doRedis')
registerDoRedis('jobs')
startLocalWorkers(n=2, queue='jobs')
print(getDoParWorkers())
foreach(j=1:10,.combine=sum,.multicombine=TRUE) %dopar%
          4*sum((runif(1000000)^2 + runif(1000000)^2)<1)/10000000
removeQueue('jobs')
```

```
## End(Not run)
```

Index