

Package ‘doMPI’

May 2, 2017

Type Package

Title Foreach Parallel Adaptor for the Rmpi Package

Version 0.2.2

Description Provides a parallel backend for the `%dopar%` function using the Rmpi package.

Depends R (>= 2.14.0), foreach(>= 1.3.0), iterators(>= 1.0.0), Rmpi(>= 0.5-7)

Imports parallel, compiler, utils

Suggests randomForest, itertools

License GPL-2

NeedsCompilation no

Author Steve Weston [cre, aut, cph]

Maintainer Steve Weston <stephen.b.weston@gmail.com>

Repository CRAN

Date/Publication 2017-05-01 22:13:18 UTC

R topics documented:

doMPI-package	2
cluster	3
dompiWorkerLoop	4
exportDoMPI	5
getDoMpiCluster	5
openMPIcluster	6
registerDoMPI	7
setRngDoMPI	7
sinkWorkerOutput	8
startMPIcluster	9

Index	11
--------------	-----------

Description

The doMPI package provides a parallel backend for the foreach package. It is similar to the doSNOW package, but uses Rmpi directly. This allows it to do more, and execute more efficiently. It can also make use of the multicore package to execute tasks across multiple cores on the worker nodes. This is can give very good performance on a computer cluster with multicore processors.

Details

There are several backend-specific options that can be specified when using doMPI. They are specified to foreach as a list using the `.options.mpi` argument. The currently supported options are:

<code>chunkSize</code>	Number of tasks to send at a time to the cluster workers
<code>info</code>	Display extra information, particularly about exported variables
<code>initEnvir</code>	A function to be called on each worker before executing any tasks
<code>initArgs</code>	List of extra arguments to pass to the <code>initEnvir</code> function
<code>initEnvirMaster</code>	A function called on the master at the same time as <code>initEnvir</code>
<code>initArgsMaster</code>	List of extra arguments to pass to the <code>initEnvirMaster</code> function
<code>finalEnvir</code>	A function to be called on each worker after executing all tasks
<code>finalArgs</code>	List of extra arguments to pass to the <code>finalEnvir</code> function
<code>profile</code>	Display profiling information from the master's point of view
<code>bcastThreshold</code>	Used to decide whether to piggy-back or broadcast job data
<code>forcePiggyback</code>	Always piggy-back job environment with first task to each worker
<code>nocompile</code>	Don't compile the R expression
<code>seed</code>	Starting seed for tasks

The `chunkSize` option is particularly important, since it can be much more efficient to send more than one task at a time to the workers, particularly when the tasks execute quickly. Also, it can allow the workers to execute those tasks in parallel using the `mclapply` function from the multicore package. The default value is 1.

The `info` option is used to print general information that is specific to the doMPI backend. This includes information on what variables are exported, for example. The default value is FALSE.

The `initEnvir` option is useful for preparing the workers to execute the subsequent tasks. The execution environment is passed as the first argument to this function. That allows you to define new variables in the environment, for example. If `initArgs` is defined, the contents of the list will be passed as arguments to the `initEnvir` function after the environment object.

The `initEnvirMaster` option is useful if you want to send data from the master to the workers explicitly, perhaps using `mpi.bcast`. This avoids object serialization, which could improve performance for large matrices, for example. The `initArgsMaster` option works like `initArgs`, however, it is probably less useful, since the `initEnvirMaster` function runs locally, and can access variables via lexical scoping.

The `finalEnvir` option is useful for “finalizing” the execution environment. It works pretty much the same as the `initEnvir` function, getting extra arguments from a list specified with the `finalArgs` option.

The `profile` option is used to print profiling information at the end of the `%dopar%` execution. It basically lists the time spent sending tasks to the workers and retrieving results from them. The default value is `FALSE`.

The `bcastThreshold` option is used to decide whether to piggy-back the job data, or broadcast it. The job data is serialized, and if it is smaller than `bcastThreshold`, it is piggy-backed, otherwise, it is broadcast. Note that if you want to force piggy-backing, you should use the `forcePiggyback`, rather than setting `bcastThreshold` to a very large value. That avoids serializing the job data twice, which can be time consuming.

The `forcePiggyback` option is used to force the job data to be “piggy-backed” with the first task to each of the workers. If the value is `FALSE`, the data may still be piggy-backed, but it is not guaranteed. In general, the job data is only piggy-backed if it is relatively small. The default value is `FALSE`.

The `nocompile` option is used to disable compilation of the R expression in the body of the `foreach` loop. The default value is `FALSE`.

The `seed` option is used for achieving reproducible results. If set to a single numeric value, such as 27, it is converted to a value that can be passed to the `nextRNGSubStream` function from the **parallel** package. This value is assigned to the global `.Random.seed` variable on some cluster worker when it executes the first task (or task chunk). The `nextRNGSubStream` function is used to generate subsequent values that are assigned to `.Random.seed` when executing subsequent tasks. Thus, RNG substreams are associated with tasks, rather than workers. This is necessary for reproducible results, since the **doMPI** package uses load balancing techniques that can result in different tasks being executed by different workers on different runs of the same `foreach` loop. The default value of the `seed` option is `NULL`.

Additional documentation is available on the following functions:

<code>startMPIcluster</code>	Create and start an MPI cluster object
<code>registerDoMPI</code>	Register a cluster object to be used with <code>%dopar%</code>
<code>closeCluster</code>	Shutdown and close a cluster object
<code>clusterSize</code>	Return the number of workers associated with a cluster object
<code>setRngDoMPI</code>	Initialize parallel random number generation on a cluster

For a complete list of functions with individual help pages, use `library(help="doMPI")`. Use the command `vignette("doMPI")` to view the vignette entitled “Introduction to doMPI”. Also, there are a number of doMPI example scripts in the `examples` directory of the doMPI installation.

Description

`clusterSize` returns the number of workers in a cluster. `closeCluster` shuts down and cleans up a cluster.

Usage

```
clusterSize(cl, ...)  
closeCluster(cl, ...)
```

Arguments

cl	The cluster object.
...	Currently unused.

dompiWorkerLoop	<i>Create and start an MPI cluster</i>
-----------------	--

Description

The `dompiWorkerLoop` function is used from a cluster worker to run the worker loop in order to execute worker tasks. This is intended to be used from a doMPI script that is executed in “non-spawn” mode.

Usage

```
dompiWorkerLoop(cl, cores=1, verbose=FALSE)
```

Arguments

cl	a <code>dompicluster</code> object created with <code>startMPIcluster</code> .
cores	Maximum number of cores for workers to use. Defaults to 1.
verbose	Indicates if verbose messages should be enabled. Defaults to FALSE.

Examples

```
## Not run:  
cl <- openMPIcluster()  
dompiWorkerLoop(cl)  
  
## End(Not run)
```

exportDoMPI	<i>Export variables to doMPI cluster</i>
-------------	--

Description

The exportDoMPI function exports variables to a doMPI cluster.

Usage

```
exportDoMPI(cl, varlist, envir=.GlobalEnv)
```

Arguments

cl	The doMPI cluster.
varlist	Vector of variable names.
envir	Environment to get variables from.

Examples

```
## Not run:  
cl <- startMPIcluster(count=2)  
f <- function() 'foo'  
g <- function() f()  
exportDoMPI(cl, c('f', 'g'))  
  
## End(Not run)
```

getDoMpiCluster	<i>Get the registered doMPI cluster object</i>
-----------------	--

Description

The getDoMpiCluster function is used to get the cluster object that was registered using the registerDoMPI function. This can be useful when you want to get the communicator object for performing MPI operations in a foreach program.

Usage

```
getDoMpiCluster()
```

openMPIcluster *Create an MPI cluster object*

Description

The openMPIcluster function is used to create an MPI cluster object in a cluster worker. It is never executed by the master process. Unlike startMPIcluster, it does not actually launch workers. It simply creates an MPI cluster object, which is passed to the workerLoop function. It is used internally in spawn mode, but it also needs to be used in doMPI scripts that are started in non-spawn mode.

Usage

```
openMPIcluster(bcast=TRUE, comm=0, workerid=mpi.comm.rank(comm), verbose=FALSE,
               mtag=10, wtag=11)
```

Arguments

bcast	Indicates if a true MPI broadcast should be used to send shared “job” data to the workers. If FALSE is specified, the data is sent by separate messages to each worker, which is sometimes faster than using a broadcast. So this option really controls whether to do a real or an emulated broadcast. Defaults to TRUE.
comm	The MPI communicator number. This should always be 0 when called from non-spawn mode. Defaults to 0.
workerid	The rank of the worker calling openMPIcluster.
verbose	Indicates if verbose messages should be enabled. Defaults to FALSE.
mtag	Tag to use for messages sent to the master. Do not use this option unless you know what you’re doing, or your program will very likely hang. Defaults to 10.
wtag	Tag to use for messages sent to the workers. Do not use this option unless you know what you’re doing, or your program will very likely hang. Defaults to 11.

Note

Make sure that openMPIcluster is called consistently with startMPIcluster, otherwise your program will hang. In particular, make sure that bcast is set the same, and that comm is 0, which is the default value.

Examples

```
## Not run:
# make an MPI cluster object with emulated broadcast:
cl <- openMPIcluster(bcast=FALSE)

## End(Not run)
```

registerDoMPI	<i>registerDoMPI</i>
---------------	----------------------

Description

The registerDoMPI function is used to register doMPI with the foreach package. Specifically, you register a particular cluster object which will be used when executing the %dopar% operator. The cluster object is created using startMPIcluster.

Usage

```
registerDoMPI(cl)
```

Arguments

cl The cluster object to use for parallel execution.

See Also

[startMPIcluster](#)

Examples

```
## Not run:
# start and register an MPI cluster with two workers:
cl <- startMPIcluster(2)
registerDoMPI(cl)

## End(Not run)
```

setRngDoMPI	<i>Setup parallel RNG on a doMPI cluster</i>
-------------	--

Description

The setRngDoMPI function initializes the workers of a doMPI cluster to use parallel random number generation. To do this, it uses the "L'Ecuyer-CMRG" RNG support provided by the **base** and **parallel** packages. Specifically, the nextRNGStream function is used to assign each worker in the cluster to a different stream of random numbers.

This function follows the outline presented in section 6 of the vignette for the **parallel** package written by R-Core.

Note that the goal of setRngDoMPI is to insure that the cluster workers each generate different streams of random numbers, not to insure repeatable results. For repeatable results, use the doMPI-specific seed option via the **foreach** .options.mpi argument. See [doMPI-package](#) for more information.

Usage

```
setRngDoMPI(c1, seed=NULL)
```

Arguments

<code>c1</code>	The doMPI cluster to initialize.
<code>seed</code>	Used to seed the random number generators on the cluster workers if not NULL. Note that the use of <code>seed</code> does not guarantee repeatable results because the tasks are not guaranteed to be repeatably executed by the same cluster workers.

See Also

[doMPI-package](#), [startMPIcluster](#), [nextRNGStream](#), [RNG](#)

Examples

```
## Not run:  
c1 <- startMPIcluster(count=2)  
setRngDoMPI(c1, seed=1234)  
  
## End(Not run)
```

<code>sinkWorkerOutput</code>	<i>Redirect worker output to a file</i>
-------------------------------	---

Description

The `sinkWorkerOutput` function is used to redirect worker output to a file. It is intended to be used from a doMPI script that is executed in “non-spawn” mode.

Usage

```
sinkWorkerOutput(outfile)
```

Arguments

<code>outfile</code>	a character string naming the file to write to.
----------------------	---

Examples

```
## Not run:  
sinkWorkerOutput(sprintf('worker_  
  
## End(Not run)
```

startMPIcluster	<i>Create and start an MPI cluster</i>
-----------------	--

Description

The startMPIcluster function is used to start an MPI cluster.

Usage

```
startMPIcluster(count, verbose=FALSE, workdir=getwd(), logdir=workdir,
               maxcores=1, includemaster=TRUE, bcast=TRUE,
               comm=if (mpi.comm.size(0) > 1) 0 else 3,
               intercomm=comm + 1, mtag=10, wtag=11,
               defaultopts=list())
```

Arguments

count	Number of workers to spawn. If you start your script using mpirun, then you don't really need to use the count argument, because startMPIcluster will try to do something reasonable. To be more specific, if comm is 0, then it will set count to <code>mpi.comm.size(0) - 1</code> . In fact, it is an error to set count to any other value. If comm is greater than 0, it will determine the number of processes to spawn by calling <code>mpi.universe.size()</code> . If that value is greater than one, then count is set to one less. If that value is equal to one, then count is arbitrarily set to 2. Note that if you've started the script without mpirun, then <code>mpi.universe.size()</code> will always return 1, so count will default to 2.
verbose	Indicates if verbose messages should be enabled. Defaults to FALSE.
workdir	Working directory of the cluster workers. Defaults to the master's working directory.
logdir	Directory to put the worker log files. Defaults to workdir.
maxcores	Maximum number of cores for workers to use. Defaults to 1.
includemaster	Indicates if the master process should be counted as a load on the CPU. This will effect how many cores will be used on the local machine by mclapply, if a worker process is started on the local machine. Defaults to TRUE.
bcast	Indicates if a true MPI broadcast should be used to send shared "job" data to the workers. If FALSE is specified, the data is sent by separate messages to each worker, which is sometimes faster than using a broadcast. So this option really controls whether to do a real or an emulated broadcast. Defaults to TRUE.
comm	Communicator number to use. A value of 0 means to use non-spawn mode, which means the cluster workers are started using mpirun/ortrun with more than one worker. A value of 1 or more forces spawn mode. Multiple clusters can be started by using different values for comm and intercomm. It defaults to 0 if <code>mpi.comm.size(0) > 1</code> , otherwise 3.
intercomm	Inter-communicator number. Defaults to comm + 1.

mtag	Tag to use for messages sent to the master. Do not use this option unless you know what you're doing, or your program will very likely hang. Defaults to 10.
wtag	Tag to use for messages sent to the workers. Do not use this option unless you know what you're doing, or your program will very likely hang. Defaults to 11.
defaultopts	A list containing default values to use for some of the .options.mpi options. These options include: chunkSize, info, profile, bcstThreshold, forcePiggyback, nocompile, and seed.

Note

The startMPIcluster function will return an MPI cluster object of different classes, depending on the bcst option. This is because broadcasting is implemented as a method on the MPI cluster object, and that method is implemented differently in the different classes.

Also note that the bcst option has no effect if the backend-specific forcePiggyback option is used with foreach, since “piggy-backing” is an alternative way to send the job data to the workers in separate messages.

So there are currently three ways that the job data can be sent to the workers: piggy-backed with the first task to each worker, broadcast, or sent in separate messages. Which method is best will presumably depend on your hardware and your MPI implementation.

Examples

```
## Not run:
# start and register an MPI cluster with two workers in verbose mode:
cl <- startMPIcluster(count=2, verbose=TRUE)
registerDoMPI(cl)
# and shut it down
closeCluster(cl)

# set the working directory to /tmp:
cl <- startMPIcluster(count=2, workdir='/tmp')
registerDoMPI(cl)
# and shut it down
closeCluster(cl)

## End(Not run)
```

Index

*Topic **package**

doMPI-package, 2

*Topic **utilities**

cluster, 3

dmpiWorkerLoop, 4

exportDoMPI, 5

getDoMpiCluster, 5

openMPIcluster, 6

registerDoMPI, 7

setRngDoMPI, 7

sinkWorkerOutput, 8

startMPIcluster, 9

closeCluster (cluster), 3

cluster, 3

clusterSize (cluster), 3

doMPI (doMPI-package), 2

doMPI-package, 2

dmpiWorkerLoop, 4

exportDoMPI, 5

getDoMpiCluster, 5

nextRNGStream, 8

openMPIcluster, 6

PRNG (setRngDoMPI), 7

registerDoMPI, 7

RNG, 8

setRngDoMPI, 7

sinkWorkerOutput, 8

startMPIcluster, 7, 8, 9