Package 'mpifarm'

June 28, 2016

Type Package

Title Farm Out a Sequence of Jobs to a Cluster of Slaves using MPI

Version 1.20-1

Date 2015-07-09

Author Aaron A. King <kingaa@umich.edu>

Maintainer Aaron A. King <kingaa@umich.edu>

Description Parallel job-farming.

Depends R(>= 3.0.0)

Imports Rmpi

License GPL(>= 2)

Collate mpifarm.R farmer.R

R topics documented:

`mpi.farmer` *Farm out jobs across a cluster using MPI*

Description

Farm out a procedure (an arbitrary block of R-code) to a cluster of slaves, with different values of the variables.

Usage

```
mpi.farmer(..., jobs, common, main, post,
           chunk = 1, blocking = FALSE,
           checkpoint.file = NULL, checkpoint = 0,
           max.backup = 20,
           stop.condition = TRUE, info = TRUE,
           verbose = getOption("verbose"))
```

Arguments

...	named arguments other than those described below will be placed into an environment. This codes jobs, common, and post will be evaluated in this environment.
jobs	job setup code. An arbitrary block of R code that creates a list of jobs. Each element of jobs should be a named list, defining some or all of the variables referred to by the code in main. If a checkpoint file exists, it will be loaded and jobs will not be executed.
common	common variables setup code. An arbitrary block of R code that creates a list of variables common to all the jobs. If a checkpoint file exists, it will be loaded and common will not be executed.
main	main computation code, for execution in parallel. An arbitrary block of R code that will be executed for each element in joblist. At each execution, this code will be evaluated in an environment consisting of the corresponding element of joblist plus the common list. That is, when the k-th job is executed, the variables referred to in main will be sought for in joblist[[k]] and, if they are not found there, in common.
post	post-processing code. An arbitrary block of R code that will be processed after the main computations are all finished. When this code is executed, a list named results will be present and will contain the results of evaluating main on each element in the list created by jobs.
chunk	optional integer; the chunk size to be used. By default, chunk=1 and each job in joblist is sent individually to a slave and returned when finished. If chunk>1, chunks of this size are sent and processed sequentially. In other words, chunk adjusts the granularity of the parallel computation.
blocking	If TRUE, blocking MPI calls will be used. If FALSE non-blocking MPI calls are used for greater evenness and efficiency in the scheduling.
stop.condition	For this option to have an effect, main must return a named list, call it X, and stop.condition must evaluate to TRUE or FALSE in the context of X. In this case, after each individual job completes, stop.condition will be evaluated in the context of that job's returned list. If it evaluates to TRUE, the job is deemed to be finished. If it evaluates to FALSE, the job is deemed to be unfinished and main to be evaluated again; the context for this next evaluation will be X. This allows one to complete the calcuation in several steps, which in turn allows for more effective load-balancing. If the return-value of main is not a list, this option has no effect.
info	If info=TRUE, information on the progress of each slave will be printed.
checkpoint	optional integer specifying the granularity of checkpointing. That is, the checkpoint file will be saved once every checkpoint jobs from joblist are completed. If checkpoint<1, no checkpointing will be performed.
checkpoint.file	optional filename. if checkpoint>0 then once every checkpoint jobs are finished, the current state of the computation will be saved to a binary-format file of the name given in checkpoint.file. If checkpoint>0 and the file named in checkpoint.file exists, it will be loaded and computations will be resumed from that point. Old checkpoint files will be backed up.

max.backup	positive integer; maximum number of backup checkpoint files that will be created.
verbose	logical; if TRUE, information will be printed both by the master and the slaves.

Details

mpi.farmer will execute the code in main repeatedly in environments defined by the entries of the list created by jobs and the variables in the list created by common. If **Rmpi** slaves have been spawned, the jobs will be farmed out to them according to a load-balancing algorithm; if no slaves are running, or **Rmpi** has not been loaded, the jobs will be executed serially. The post argument allows an arbitrary post-processing step to be performed after all the jobs are completed. mpi.farmer also implements checkpointing.

Value

If post is not set, mpi.farmer returns a list with one entry for each of the elements in the list created by jobs. If post is set, mpi.farmer returns whatever post does.

Iterated parallel computation

The basic idea of mpi.farmer is that a list of jobs (created by jobs), together with a set of variable common across jobs (created by common) defines a set of contexts in which main will be evaluated. Because it is possible for the evaluation of main to result in the creation of a new context, it is possible to iterate this process. Specifically, if, acting on job X, main returns a named list Y and if stop.condition, evaluated in the context of Y, is FALSE, then X is replaced by Y and Y is then returned to the stack of jobs to be evaluated. This recycling will continue until stop.condition evaluates to TRUE.

Checkpointing

If the file named in checkpoint.file exists, mpi.farmer will not execute jobs or common but will instead load the checkpoint file and resume the computations.

Stochastic simulations and parallel computation

For many of the applications envisioned, the jobs the slaves are assigned involve stochastic simulations. Because of the way that R initializes its pseudorandom number generators (RNGs), it is easy to make the mistake of failing to initialize the RNGs on different slaves to different states. If one fails to do this (and doesn't use a sophisticated parallel RNG like SPRNG) then it is possible that the random numbers generated on different slaves will be correlated or even identical. For this reason, it is a good idea to set the seed of the RNG as part of the block of code main. Storing the state of the RNG before doing so is often desirable, but this can be frustrating if the RNG has not been initialized. mpi.farmer checks to see if .Random.seed exists and, if it does not, initializes the RNG with a call to runif. Thus, the user is guaranteed that the RNG has been initialized on each slave.

Interrupt behavior

A user interrupt to `mpi.farmer` results in an attempt to terminate the slaves cleanly. This may take some time, since each slave has to finish the job it is currently working on before it becomes receptive to messages from the master. A user interrupt issued during the abort process will leave some finished jobs in the MPI queue and therefore compromise the integrity of future parallel computations. For this reason, when it is necessary to abort `mpi.farmer` and not possible to allow it to terminate cleanly, it is recommended that the slaves be closed (via `mpi.close.Rslaves`) and restarted before further parallel computations are attempted.

Author(s)

Aaron A. King

Examples

```
  ## Not run:
library(Rmpi)

mpi.spawn.Rslaves(nslaves=5)

set.seed(87544545L)

## simulate some order statistics
mpi.farmer(
           n=1000,
           stdev=1,
           ntries=10,
           common={
             list(n=ntries,sd=stdev)
           },
           jobs={
             seeds <- as.integer(ceiling(runif(n=n,min=0,max=2^31-1)))
             lapply(seeds,function(s)list(seed=s))
           },
           main={
             save.seed <- .Random.seed
             set.seed(seed)
             x <- sort(rnorm(n=n,mean=0,sd=sd))
             .Random.seed <<- save.seed
             list(seed=seed,x=x)
           },
           post={
             res <- lapply(results,function(y)y$x)
             res <- do.call(rbind,res)
             as.data.frame(res)
           },
           chunk=10
           ) -> results

## do the same, with checkpointing
mpi.farmer(
```

```
            checkpoint=100,
            checkpoint.file="farmer.rda",
            n=1000,
            stdev=1,
            ntries=10,
            common={
              list(n=ntries,sd=stdev)
            },
            jobs={
              seeds <- as.integer(ceiling(runif(n=n,min=0,max=2^31-1)))
              lapply(seeds,function(s)list(seed=s))
            },
            main={
              save.seed <- .Random.seed
              set.seed(seed)
              x <- sort(rnorm(n=n,mean=0,sd=sd))
              .Random.seed <<- save.seed
              list(seed=seed,x=x)
            },
            post={
              res <- lapply(results,function(y)y$x)
              res <- do.call(rbind,res)
              as.data.frame(res)
            },
            chunk=10
            ) -> results

mpi.close.Rslaves()

## End(Not run)
```

Index