Parallel processing with the RNetLogo Package

Jan C. Thiele
Department of
Ecoinformatics, Biometrics
and Forest Growth
University of Göttingen
Germany

Abstract

RNetLogo is a flexible interface for NetLogo to R. It opens various possiblities to connect agent-based models with advanced statistics. It opens the possiblity to use R as the starting point to design systematic experiments with agent-based models and perform parameter fittings and sensitivity analysis. Therefore, it can be necessary to perform repeated simulations which could be parallelized. Here, I present how such a parallelization could be done for the RNetLogo package. The things presented here could be used to run multiple simulations in parallel on a single computer with multiple processors or to spread multiple simulation to several processors in computer clusters/grids. Using the parallel package has an positive side effect: It enables you to start more than one NetLogo instance with GUI in parallel, which is not possible with parallelization.

Keywords: NetLogo, R, agent based modelling, abm, individual based modelling, ibm, parallelization.

1. Motivation

Since modern computers mostly have more than one processor and agent-based simulations are often complex and time comsuming it is desireable to spread repeated simulations, for example for parameter fitting or sensitivity analysis, to multiple processors in parallel. Here, I will present one way of how it is possible to spread multiple NetLogo simulations controlled from R via the RNetLogo package to multiple processors.

2. Parallelization in R

R itself is not able to make use of multiple processors of a computer. But there are several R packages available, which enable the use to spread repeated processes to multiple processors. There is a CRAN Task View called "High-Performance and Parallel Computing with R" at `http://cran.r-project.org/web/views/HighPerformanceComputing.html`. Since R version 2.14.0 there is the **parallel** package included in every standard R installation. In the following I will present how to use this **parallel** package in conjunction with RNetLogo. Therefore, to follow the examples it requires that you have at R version >= 2.14.0 installed. There is a pdf file coming with the parallel packing giving a short introduction into the usage of the package and the plattform specific differences. You should always start by reading this document. A last note, before we start: The commands presented in the following have been tested on Windows and Linux operation systems only. If you have experiences with Mac OS please let me know.

3. Parallelize a simple process

The basic concept of the **parallel** package is to parallelize an apply (or lapply, sapply etc.) operation. This means, that the process you want to parallelize has to be process which is applied to an array, matrix, list or whatever.

Let us start with a simple example without using RNetLogo. First, we define a simple function which calculates the square of an input number.

```
R> testfun1 <- function(x) {
+    return(x*x)
+  }
```

We could apply this simple function to a vector of values using `sapply` like this:

```
R> my.v1 <- 1:10
R> print(my.v1)
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

```
R> my.v1.quad <- sapply(my.v1, testfun1)
R> print(my.v1.quad)
```

```
 [1]   1   4   9  16  25  36  49  64  81 100
```

The result is a vector with the quadratic values of the input vector, i.e. the function was applied sequentially to each element of the input vector.

One way to use the **parallel** package is to use the parallel version of the `sapply` function which is called `parSapply`.

But before we can use this function, we have to make/register a cluster, as you know from the manual of the **parallel** package. Therefore, we could, for example, detect the number of cores of our local computer and start a local cluster with this number of processors, as shown here:

```
R> # load the parallel package
R> library(parallel)
R> # detect the number of cores available
R> processors <- detectCores()
R> # create cluster
R> cl <- makeCluster(processors)
```

Then, we can run our simple function on this cluster. At the end, we should always stop the cluster.

```
R> # call parallel sapply
R> my.v1.quad.par <- parSapply(cl, my.v1, testfun1)
R> print(my.v1.quad.par)

 [1]   1   4   9  16  25  36  49  64  81 100

R> # stop cluster
R> stopCluster(cl)
```

4. Parallelize RNetLogo

As you know from the RNetLogo manual, it requires an initialization using the `NLStart` and (maybe) `NLLoadModel` function. To parallelize (R)NetLogo we need this initialization to be done for every processor, because they are independent from each other (which is a very important property, because, for example, random processes in parallel simulations should not beeing influenced by each other).

Therefore, it makes sence to put the initialization, the simulation and the quiting process into seperate functions. These functions could look like the following (if you want to test these, don't forget to adapt the paths appropriate):

```
R> # the initialization function
R> prepro <- function(dummy, gui, nl.path, model.path) {
+    library(RNetLogo)
+    NLStart(nl.path, nl.version=5, gui=gui)
+    NLLoadModel(model.path)
+  }
R> # the simulation function
R> simfun <- function(x) {
+    NLCommand("print ",x)
+    NLCommand("set density", x)
+    NLCommand("setup")
+    NLCommand("go")
+    NLCommand("print count turtles")
+    ret <- data.frame(x, NLReport("count turtles"))
+    names(ret) <- c("x","turtles")
+    return(ret)
+  }
R> # the quit function
R> postpro <- function(x) {
+    NLQuit()
+  }
```

4.1. With Graphical User Interface (GUI)

Now, we have to start the cluster, run the initialization function in each processor, which will open so many NetLogo windows as we have processors.

Note, that this is also a nice way to run multiple NetLogo GUI instances in parallel, what is not possible within one R session without this parallelization (see documentation of the RNetLogo package for detail).

```
R> # load the parallel package, if not already done
R> require(parallel)
R> # detect the number of cores available
R> processors <- detectCores()
R> # create cluster
R> cl <- makeCluster(processors)
R> # set variables for the start up process
R> # adapt path appropriate
R> gui <- TRUE
R> nl.path <- "C:/Program Files/NetLogo 5.0"
R> model.path <- "models/Sample Models/Earth Science/Fire.nlogo"
R> # load NetLogo in each processor/core
R> invisible(parLapply(cl, 1:processors, prepro, gui=gui,
+                       nl.path=nl.path, model.path=model.path))
```

After the initialization is done in all processors, we can run the simulation. Here, we will use the Fire model from NetLogo's Model Library. We will vary the density value from 1 to 20, i.e. we will run 20 independent simulations each with a different density value.

```
R> # create a vector with 20 density values
R> density <- 1:20
R> print(density)

 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20

R> # run a simulation for each density value
R> # by calling parallel sapply
R> result.par <- parSapply(cl, density, simfun)
R> print(data.frame(t(result.par)))

    x turtles
1   1     254
2   2     254
3   3     255
4   4     265
5   5     260
6   6     263
7   7     261
8   8     276
9   9     268
10 10     273
11 11     270
12 12     277
13 13     284
```

```
14 14      293
15 15      298
16 16      294
17 17      292
18 18      304
19 19      302
20 20      300
```

At the end, we should stop all NetLogo instances and the cluster.

```
R> # Quit NetLogo in each processor/core
R> invisible(parLapply(cl, 1:processors, postpro))
R> # stop cluster
R> stopCluster(cl)
```

4.2. Headless

The same is possible with the headless mode, i.e. with the GUI. We just have to set the variable `gui` to `FALSE`.

It could look like this:

```
R> # run in headless mode
R> gui <- FALSE
R> # create cluster
R> cl <- makeCluster(processors)
R> # load NetLogo in each processor/core
R> invisible(parLapply(cl, 1:processors, prepro, gui=gui,
+                      nl.path=nl.path, model.path=model.path))
R> # create a vector with 20 density values
R> density <- 1:20
R> print(density)

 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20

R> # run a simulation for each density value
R> # by calling parallel sapply
R> result.par <- parSapply(cl, density, simfun)
R> print(data.frame(t(result.par)))

   x turtles
1  1     255
2  2     255
3  3     258
4  4     257
5  5     261
6  6     266
7  7     263
```

```
8    8       280
9    9       274
10  10       278
11  11       288
12  12       282
13  13       283
14  14       283
15  15       281
16  16       290
17  17       302
18  18       297
19  19       304
20  20       300
```

```
R> # Quit NetLogo in each processor/core
R> invisible(parLapply(cl, 1:processors, postpro))
R> # stop cluster
R> stopCluster(cl)
```

5. Conclusion

We have seen one way of how it is possible to spread repeated and independent simulations to multiple processors using the **parallel** package. Therefore, RNetLogo can be efficiently used to perform parameter fittings and sensitivity analyses where large number of repeated simultions are required.

Affiliation:

Jan C. Thiele
Department of Ecoinformatics, Biometrics and Forest Growth
University of Göttingen
Büsgenweg 4
37077 Göttingen, Germany
E-mail: jthiele@gwdg.de
URL: http://www.uni-goettingen.de/en/72779.html