Data Science with R
Parallel Execution

Graham.Williams@togaware.com

13th July 2013

Visit http://onepager.togaware.com/ for more OnePageR's.

R supports several levels of parallel execution, starting with executing code on multiple cores, and going up to executing code in massively parallel Hadoop platforms. Since R Version 2.14.0 parallel has provided support for parallel computation through forking (c.f. multicore) and sockets (c.f. snow).

To illustrate parallel computation we will build rpart decision trees in parallel.

The required packages for this module include:

```
library(parallel)
library(rpart)
```

As we work through this module, new R commands will be introduced. Be sure to review the command's documentation and understand what the command does. You can ask for help using the `?` command as in:

```
?read.csv
```

We can obtain documentation on a particular package using the *help=* option of `library()`:

```
library(help=rattle)
```

This module is intended to be hands on. To learn effectively, you are encouraged to have R running (e.g., RStudio) and to run all the commands as they appear here. Check that you get the same output, and you understand the output. Try some variations. Explore.

1 Weather Data

We will model the **weatherAUS** dataset. We choose this dataset since it is reasonably large, and takes quite a few seconds to build a decision tree.

We have a CSV version of the dataset available in the local data folder.

```
dir(path="data", pattern="*.csv")

## [1] "heart.csv"      "stroke.csv"     "weatherAUS.csv"
```

The data is directly read into a data frame.

```
ds <- read.csv(file="data/weatherAUS.csv")
```

As always, we first check the contents of the dataset to ensure everything looks okay:

```
dim(ds)

## [1] 66672    24

head(ds)

##         Date Location MinTemp MaxTemp Rainfall Evaporation Sunshine
## 1 2008-12-01   Albury    13.4    22.9      0.6          NA       NA
## 2 2008-12-02   Albury     7.4    25.1      0.0          NA       NA
## 3 2008-12-03   Albury    12.9    25.7      0.0          NA       NA
## 4 2008-12-04   Albury     9.2    28.0      0.0          NA       NA
....

tail(ds)

##             Date Location MinTemp MaxTemp Rainfall Evaporation Sunshine
## 66667 2012-11-24   Darwin    25.5    34.1      0.0         5.0      6.2
## 66668 2012-11-25   Darwin    24.4    35.7      0.2         4.8     11.7
## 66669 2012-11-26   Darwin    25.0    35.4      0.0         7.4     11.7
## 66670 2012-11-27   Darwin    26.5    35.9      0.0         8.0     10.3
....

str(ds)

## 'data.frame': 66672 obs. of  24 variables:
##  $ Date         : Factor w/ 1826 levels "2007-11-01","2007-11-02",..: 397 ...
##  $ Location     : Factor w/ 46 levels "Adelaide","Albany",..: 3 3 3 3 3 3 ...
##  $ MinTemp      : num  13.4 7.4 12.9 9.2 17.5 14.6 14.3 7.7 9.7 13.1 ...
##  $ MaxTemp      : num  22.9 25.1 25.7 28 32.3 29.7 25 26.7 31.9 30.1 ...
....

summary(ds)

##          Date            Location        MinTemp         MaxTemp
##  2009-01-01:   46   Canberra: 1826   Min.   :-8.5   Min.   :-3.1
##  2009-01-02:   46   Sydney  : 1734   1st Qu.: 7.3   1st Qu.:17.6
##  2009-01-03:   46   Adelaide: 1583   Median :11.7   Median :22.1
##  2009-01-04:   46   Brisbane: 1583   Mean   :11.9   Mean   :22.7
....
```

2 Prepare for Modelling

Following the template presented in the Models module, we continue with setting up some fo the modelling parameters.

```
target <- "RainTomorrow"
risk   <- "RISK_MM"
dsname <- "weather"
```

```
ds[target] <- as.factor(ds[[target]])
summary(ds[target])

##  RainTomorrow
##  No  :50187
##  Yes :15259
##  NA's: 1226
....
```

```
vars    <- colnames(ds)
ignore  <- vars[c(1, 2, if (exists("risk")) which(risk==vars))]
vars    <- setdiff(vars, ignore)
(inputs <- setdiff(vars, target))

##  [1] "MinTemp"       "MaxTemp"       "Rainfall"      "Evaporation"
##  [5] "Sunshine"      "WindGustDir"   "WindGustSpeed" "WindDir9am"
##  [9] "WindDir3pm"    "WindSpeed9am"  "WindSpeed3pm"  "Humidity9am"
## [13] "Humidity3pm"   "Pressure9am"   "Pressure3pm"   "Cloud9am"
....

nobs    <- nrow(ds)
dim(ds[vars])

## [1] 66672    21
```

```
(form <- formula(paste(target, "~ .")))

## RainTomorrow ~ .
```

```
(seed <- sample(1:1000000, 1))

## [1] 954262

set.seed(seed)
```

```
length(train <- sample(nobs, 0.7*nobs))

## [1] 46670

length(test  <- setdiff(seq_len(nobs), train))

## [1] 20002
```

3 Build a Model

An exercise in the Functions module developed `wsrpart()` to build one or more rpart (Therneau and Atkinson, 2013) decision trees based on a random subset of the data and a weighted random subset of the variables. We can use this function here, and will do so to distribute computation first over multiple cores and then over multiple servers.

Each time `wsrpart()` builds a decision tree it selects a different random training dataset and a different random choice of variables to use in the tree building. The processing for each call to the function to build the decision tree (`rpart()`) will be distributed across multiple cores or servers.

Building a single model returns the model and other information.

```
set.seed(42)
system.time(model <- wsrpart(form, ds[train, vars], ntrees=1))

##    user  system elapsed
##   0.268   0.004   2.851

model[[1]]$model

## n=45860 (810 observations deleted due to missingness)
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
....

model[[1]]$vars

##  [1] "Rainfall"    "Humidity3pm" "Cloud3pm"    "Evaporation" "MaxTemp"
##  [6] "Pressure3pm" "Cloud9am"    "Temp3pm"     "Pressure9am" "RainToday"
## [11] "MinTemp"

model[[1]]$accuracy

## NULL
```

4 Build a Second Model

We can call it again to obtain another model:

```
set.seed(84)
system.time(model <- wsrpart(form, ds[train, vars], ntrees=1))

##    user  system elapsed
##   2.732   0.056   2.676

model[[1]]$model

## n=45761 (909 observations deleted due to missingness)
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
....

model[[1]]$vars

##  [1] "Humidity9am"   "Temp3pm"       "Sunshine"      "Pressure9am"
##  [5] "Humidity3pm"   "RainToday"     "Cloud9am"      "WindGustSpeed"
##  [9] "Pressure3pm"   "WindGustDir"   "Cloud3pm"

model[[1]]$oob.error

## [1] 0.1767
```

5 Build Models in Parallel

The parallel (?) package provides functions to distribute the computation across multiple cores and servers.

We first determine the number of cores available on the computer we are processing our data on:

```
cores <- detectCores()
cores

## [1] 4
```

We can then start a parallel run of building models using `mcparallel()`. This command forks the current process to build the tree (and hence will not work on MS/Windows). Here we build one tree for each core.

```
jobs <- lapply(1:cores,
               function(x) mcparallel(wsrpart(form, ds[train,vars], ntrees=1),
                                      name=sprintf("dt%02d", x)))
```

We can inspect the first two processes:

```
jobs[1:2]

## [[1]]
## $pid
## [1] 25308
##
## $fd
## [1] 4 7
##
## $name
## [1] "dt01"
##
## attr(,"class")
## [1] "parallelJob"  "childProcess" "process"
##
## [[2]]
## $pid
## [1] 25309
##
## $fd
## [1] 5 9
##
## $name
## [1] "dt02"
##
## attr(,"class")
## [1] "parallelJob"  "childProcess" "process"
```

6 Collect Results

We now wait for the jobs to finish:

```
system.time(model <- mccollect(jobs, wait=TRUE))

##    user  system elapsed
##   3.404   0.232   6.218
```

The decision trees will then be available in the resulting list:

```
length(model)

## [1] 4

model[[1]][[1]]$model

## n=45832 (838 observations deleted due to missingness)
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 45832 10610 No (0.7685 0.2315)
##    2) Humidity3pm< 69.5 36727  5143 No (0.8600 0.1400) *
##    3) Humidity3pm>=69.5 9105  3636 Yes (0.3993 0.6007)
##      6) Humidity3pm< 82.5 5482  2496 No (0.5447 0.4553)
##       12) Rainfall< 2.3 3620  1353 No (0.6262 0.3738) *
....

model[[2]][[1]]$model

## n=45843 (827 observations deleted due to missingness)
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 45843 10530 No (0.7702 0.2298)
##    2) Humidity3pm< 70.5 37372  5297 No (0.8583 0.1417) *
##    3) Humidity3pm>=70.5 8471  3235 Yes (0.3819 0.6181)
##      6) Humidity3pm< 82.5 4976  2391 No (0.5195 0.4805)
##       12) Rainfall< 1.7 3068  1194 No (0.6108 0.3892) *
....
```

7 Exercise: Multiple Cores

Our exercise here is to write a function to build decision trees on multiple cores. We will modify the function `wsrpart` we built in the Models module. It will use multiple cores in building the forest of trees in parallel. We specify the number of trees to build (`ntrees=`) and optionally the maximum number of cores to use `parallel=`. At most, and by default, one less than the number of available cores (but at least 1) will be used. Thus, that many trees will be built in parallel at any time. More trees than cores can be specified, and as trees finish being built across the cores, further trees can start being built. We do this to allow the user to decide how best to manage the parallel execution across the cores, without swamping the server with too many processes.

The actual solution will produce the following output, showing also some timings:

```
system.time(model <- wsrpart(form, ds, ntrees=4, parallel=2))

##    user  system elapsed
##   8.552   0.284   5.967

num.trees <- cores
set.seed(42)
system.time(model <- wsrpart(form, ds, ntrees=num.trees, parallel=2))

##    user  system elapsed
##  10.004   0.296   5.211

model[[1]]$model

## n=65412 (1260 observations deleted due to missingness)
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
....
```

We might do some more timings:

```
set.seed(42)
system.time(model <- wsrpart(form, ds, ntrees=num.trees, parallel=2))

##    user  system elapsed
##   9.372   0.312   4.942

set.seed(42)
system.time(model <- wsrpart(form, ds, ntrees=num.trees, parallel=2))

##    user  system elapsed
##   7.168   0.196   4.986
```

8 Parallel Processes Through Local Sockets

Before we proceed to run parallel processes over a network of workers (remote servers) we will do the same, but have a single node cluster (the current server). We use `makeCluster()` from parallel to do this.

We begin with a simple example. Here we create a cluster of as many nodes as there are cores on the local host.

```
cl <- makeCluster(rep("localhost", cores))
cl

## socket cluster with 4 nodes on host 'localhost'
```

Now we ask each node of the cluster to do something. In this case we get the addition function with the additional argument 3. The 1:2 are the arguments passed to each node, so that node 1 gets 1 and node 2 gets 2.

```
clusterApply(cl, 1:2, get("+"), 3)

## [[1]]
## [1] 4
##
## [[2]]
....
```

Because the nodes are each on the local host we don't need to export data to the nodes. Also, each node has the current working directory set appropriately. We can use `clusterEvalQ()` to have the same expression executed on each node.

```
clusterEvalQ(cl, getwd())

## [[1]]
## [1] "/home/gjw/projects/onepager"
##
## [[2]]
....
```

We should close the cluster once we are finished with it, though this is optional since the nodes will terminate themselves when the associated socket becomes unavailable.

```
stopCluster(cl)
```

9 Build Models Through Local Sockets

Create a new cluster of as many nodes as there are cores on the local host.

```
cl <- makeCluster(rep("localhost", cores))
cl

## socket cluster with 4 nodes on host 'localhost'
```

We load rpart and rattle on each node of the cluster.

```
clusterEvalQ(cl, {library(parallel); library(rpart); library(rattle)})

## [[1]]
##  [1] "rattle"    "rpart"     "parallel"  "methods"   "stats"
##  [6] "graphics"  "grDevices" "utils"     "datasets"  "base"
##
....
```

Note that we might not have previously noticed that `library()` returns the current library search path as its value. It is returned silently and so normally we don't see the return result. With `clusterEvalQ()` the final result is displayed.

We need to load the dataset onto each node:

```
clusterExport(cl, c("ds", "form", "train", "vars"))
```

Now to build the decision trees. We need to export the definition of `wsrpart()` (and its support functions) to each node of the cluster. We can then call on the nodes to run the command:

```
clusterExport(cl, c("varWeights", "selectVars", "wsrpart"))
system.time(model <- clusterCall(cl, wsrpart, form, ds[train, vars], ntrees=4))

##    user  system elapsed
##   0.980   0.028  24.153

length(model)

## [1] 4
```

Clean up after ourselves:

```
stopCluster(cl)
```

10 Build Models Through Multiple Servers

This page under construction.

We call makeCluster() to build a cluster of servers to run our decision trees on the nodes.

```
nodes <- paste("node", 1:10, sep="")
cl <- makeCluster(nodes)
cl
```

If there are errors, particularly about ssh askpass, then chances are you can not connect to the remote nodes using ssh without a password. You will need to set up a public ssh key (using `ssh-keygen -t dsa` without a pass phrase) on node1 and copy the resulting file `.ssh/id_dsa.pub` to each of the nodes as `.ssh/authorized_keys`.

Now we build just a single decision tree on each node:

```
clusterEvalQ(cl, {library(parallel); library(rpart); library(rattle)})
clusterExport(cl,  c("varWeights", "selectVars", "wsrpart"))
clusterExport(cl, c("varWeights", "selectVars", "wsrpart"))
system.time(model <- clusterCall(cl, wsrpart, form, ds[train, vars], ntrees=4))
```

```
stopCluster(cl)
```

11 Build Models Through Multiple Servers and Multiple Cores

We call `makeCluster()` again to build a cluster of servers to run our decision trees on the nodes. We then ask each node to use `mcparallel()` to use all cores on each node, through `mcwsrpart()`.

```
nodes <- paste("node", 1:10, sep="")
cl <- makeCluster(nodes)
cl
```

```
clusterEvalQ(cl, {library(rpart); library(rattle)})
clusterExport(cl, "weatherDS")
clusterExport(cl, c("varWeights", "selectVars", "wsrpart", "mcwsrpart"))
system.time(forest <- clusterCall(cl, mcwsrpart, weatherDS, 8))
```

```
stopCluster(cl)
```

12 Exercise: Build Decision Trees on Multiple Servers

This page under construction.

Extend the function `wsrpart()` to take an argument, `parallel=`, which lists a cluster of servers on which we are to build the decision trees in parallel. On each server we use as many cores as available in building trees.

13 Installing Packages Across Servers

THIS PAGE UNDER CONSTRUCTION

```
nodes <- paste("node", 2:10, sep="")
cl <- makeCluster(nodes)
clusterEvalQ(cl,
             install.packages("rattle",
                              lib="/usr/local/lib/R/site-library",
                              repos="http://rattle.togaware.com"))
stopCluster(cl)
```

14 Further Reading

15 References

R Core Team (2013). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Therneau TM, Atkinson B (2013). *rpart: Recursive Partitioning*. R package version 4.1-1, URL http://CRAN.R-project.org/package=rpart.

Williams GJ (2009). "Rattle: A Data Mining GUI for R." *The R Journal*, **1**(2), 45–55. URL http://journal.r-project.org/archive/2009-2/RJournal_2009-2_Williams.pdf.

Williams GJ (2011). *Data Mining with Rattle and R: The art of excavating data for knowledge discovery*. Use R! Springer, New York. URL http://www.amazon.com/gp/product/1441998896/ref=as_li_qf_sp_asin_tl?ie=UTF8&tag=togaware-20&linkCode=as2&camp=217145&creative=399373&creativeASIN=1441998896.