Big Data Analytics

Lecture 3:

Computation and Memory Part II

Prof. Dr. Ulrich Matter
11/03/2021

Updates

Group examination: take-home exercises

- Analysis of (big) dataset in R.
- Report in R Markdown.
- Conceptual questions.
- Collaborate, hand-in, feedback via GitHub.

Group projects:

- A simple empirical research question.
- A large (>2GB) data set (of your choice).
  - Get inspired here, here, and here
- Implement analysis in R.
- Present results in 6-7 minutes.
  - R-markdown (ioslides/shiny) or R presentation.
- Q&A, Feedback

Group projects:

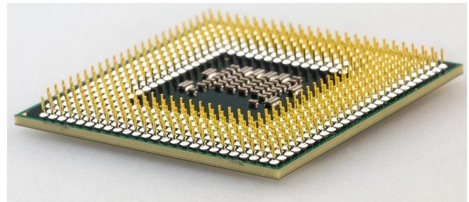
- Send short **disposition** to ulrich.matter@unisg.ch by end of March.
  - Data set (short description/link)
  - Research question
  - Idea for analysis (statistical approach)

Goals for today

1. Understand basics of how to control resource allocation in R.
2. Know the basics of parallel computing in R.
3. Know the basics of efficient memory allocation and virtual memory (in data analytics context).

Recap of Week 2

Components of a computing environment



Components of a computing environment

Why should we care?

Big Data (Analytics)

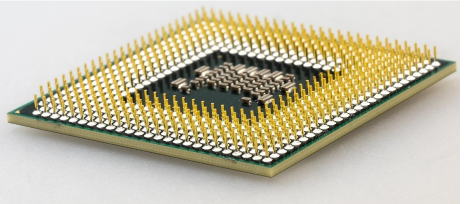
- Find an efficient (fast) statistical procedure. (Uluru vs OLS example)
- Need to understand how to **make best use of the available resources**, given a specific data analysis task.
  - CPU: Parallel processing (use all cores available)
  - RAM: Efficient memory allocation and usage
  - RAM + Mass Storage: Virtual memory, efficient swapping

Computation and Memory (Part II)

Efficient Use of Resources

1) Parallel processing: CPU/core

- A CPU on any modern computer has several **cores**.
- The OS usually assigns automatically which tasks/processes should run on which core.
- We can explicitly instruct the computer to dedicate N cores to a specific computational task: **parallel processing**.



2) Memory allocation: RAM

- Standard computation procedures happen **in-memory**: data needs to be loaded into RAM.
- Default lower-level procedures to **allocate memory** might not be optimal for large data sets.
- We can explicitly use **faster** memory allocation procedures for a specific big data task.

3) Beyond RAM: virtual memory

- What if we run out of RAM?
- The OS deals with this by using part of the hard disk as **virtual memory**.
- By explicitly instructing the computer how to use **virtual memory for specific big data tasks**, we can speed things up.

Case study: Parallel processing

We start with importing the data into R.

```
url <- "https://vincentarelbundock.github.io/Rdatasets/csv/carData/MplsStops.csv"
stopdata <- data.table::fread(url) # skipNul avoids running into encoding issues with this data set
```

Case study: Parallel processing

First, let's remove observations with missing entries (`NA`) and code our main explanatory variable and the dependent variable.

```
# remove incomplete obs
stopdata <- na.omit(stopdata)
# code dependent var
stopdata$vsearch <- 0
stopdata$vsearch[stopdata$vehicleSearch=="YES"] <- 1
# code explanatory var
stopdata$white <- 0
stopdata$white[stopdata$race=="White"] <- 1
```

Case study: Parallel processing

We specify our baseline model as follows.

```
model <- vsearch ~ white + factor(policePrecinct)
```

Case study: Parallel processing

And estimate the linear probability model via OLS (the `lm` function).

```
fit <- lm(model, stopdata)
summary(fit)
```

```
##
## Call:
## lm(formula = model, data = stopdata)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.13937 -0.06329 -0.05473 -0.04227  0.97729
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)              0.054733   0.005154  10.619  < 2e-16 ***
## white                   -0.019553   0.004465  -4.380 1.19e-05 ***
## factor(policePrecinct)2  0.008556   0.006757   1.266   0.2054
## factor(policePrecinct)3  0.003409   0.006483   0.526   0.5990
## factor(policePrecinct)4  0.084639   0.006232  13.582  < 2e-16 ***
## factor(policePrecinct)5 -0.012465   0.006371  -1.956   0.0504 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.254 on 19078 degrees of freedom
```

Case study: Parallel processing

Compute bootstrap clustered standard errors.

```r
# load packages
library(data.table)
# set the 'seed' for random numbers (makes the example reproducible)
set.seed(2)

# set number of bootstrap iterations
B <- 10
# get selection of precincts
precincts <- unique(stopdata$policePrecinct)
# container for coefficients
boot_coefs <- matrix(NA, nrow = B, ncol = 2)
# draw bootstrap samples, estimate model for each sample
for (i in 1:B) {

     # draw sample of precincts (cluster level)
     precincts_i <- sample(precincts, size = 5, replace = TRUE)
     # get observations
     bs_i <- lapply(precincts_i, function(x) stopdata[stopdata$policePrecinct==x,])
     bs_i <- rbindlist(bs_i)

     # estimate model and record coefficients
     boot_coefs[i,] <- coef(lm(model, bs_i))[1:2] # ignore FE-coefficients
}
```

Case study: Parallel processing

Finally, let's compute SE_{boot}.

```
se_boot <- apply(boot_coefs,
                 MARGIN = 2,
                 FUN = sd)
se_boot
```

```
## [1] 0.004042725 0.004689611
```

Case study: Parallel processing

Parallel implementation…

```
# install.packages("doSNOW", "parallel")
# load packages for parallel processing
library(doSNOW)
# set the 'seed' for random numbers (makes the example reproducible)
set.seed(2)

# get the number of cores available
ncores <- parallel::detectCores()
# set cores for parallel processing
ctemp <- makeCluster(ncores) #
registerDoSNOW(ctemp)


# set number of bootstrap iterations
B <- 10
# get selection of precincts
precincts <- unique(stopdata$policePrecinct)
# container for coefficients
boot_coefs <- matrix(NA, nrow = B, ncol = 2)

# bootstrapping in parallel
boot_coefs <-
     foreach(i = 1:B, .combine = rbind, .packages="data.table") %dopar% {

          # draw sample of precincts (cluster level)
```

Case study: Parallel processing

As a last step, we compute again SE_{boot}.

```
se_boot <- apply(boot_coefs,
                 MARGIN = 2,
                 FUN = sd)
se_boot
```

```
## (Intercept)       white
## 0.004989630 0.002641027
```

Case study: Memory allocation

```r
############################################################
# Big Data Statistics: Flights data import and preparation
#
# U. Matter, January 2019
############################################################

# SET UP -----------------

# fix variables
DATA_PATH <- "../data/flights.csv"

# DATA IMPORT ----------------
flights <- read.csv(DATA_PATH)

# DATA PREPARATION --------
flights <- flights[,-1:-3]
```

Case study: Memory allocation

Inspect the memory usage.

```
# SET UP -----------------

# fix variables
DATA_PATH <- "../data/flights.csv"
# load packages
library(pryr)
```

```
# check how much memory is used by R (overall)
mem_used()
```

```
## 1.08 GB
```

```
# check the change in memory due to each step

# DATA IMPORT ----------------
mem_change(flights <- read.csv(DATA_PATH))
```

```
## 611 kB
```

```
# DATA PREPARATION --------
flights <- flights[,-1:-3]
```

Case study: Memory allocation

'Collect the garbage'…

```
gc()
```

```
##             used  (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells   1063414  56.8    1759040   94.0   1759040   94.0
## Vcells 127595092 973.5  213343342 1627.7 211037840 1610.1
```

Case study: Memory allocation

Alternative approach (via memory mapping).

```
# load packages
library(data.table)

# DATA IMPORT ----------------
flights <- fread(DATA_PATH, verbose = TRUE)
```

```
##   OpenMP version (_OPENMP)       201511
##   omp_get_num_procs()            12
##   R_DATATABLE_NUM_PROCS_PERCENT  unset (default 50)
##   R_DATATABLE_NUM_THREADS        unset
##   R_DATATABLE_THROTTLE           unset (default 1024)
##   omp_get_thread_limit()         2147483647
##   omp_get_max_threads()          12
##   OMP_THREAD_LIMIT               unset
##   OMP_NUM_THREADS                unset
##   RestoreAfterFork               true
##   data.table is using 6 threads with throttle==1024. See ?setDTthreads.
## Input contains no \n. Taking this to be a filename to open
## [01] Check arguments
##   Using 6 threads (omp_get_max_threads()=12, nth=6)
##   NAstrings = [<<NA>>]
##   None of the NAstrings look like numbers.
##   show progress = 0
##   0/1 column will be read as integer
```

Case study: Memory allocation

Alternative approach (via memory mapping).

```
# SET UP -----------------

# fix variables
DATA_PATH <- "../data/flights.csv"
# load packages
library(pryr)
library(data.table)

# housekeeping
flights <- NULL
gc()
```

```
##              used  (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells   1052450  56.3    1759040   94.0   1759040   94.0
## Vcells 124429344 949.4  213343342 1627.7 211037840 1610.1
```

```
# check the change in memory due to each step

# DATA IMPORT ----------------
mem_change(flights <- fread(DATA_PATH))
```

```
## 35.8 MB
```

Insight from analyzing methods conceptually

- Methods for big data analytics come with an **‘overhead’**
  - Additional ‘preparatory’ steps.
  - Only faster than traditional methods if data set has a certain size!

Insight from analyzing methods conceptually

- Methods for big data analytics come with an **'overhead'**
  - Additional 'preparatory' steps.
  - Only faster than traditional methods if data set has a certain size!
- Examples:
  - Parallel processing: Distribute data/task, combine afterwards.
  - `fread`: Memory maps data before actually reading it into RAM.

Beyond memory

- RAM is not sufficient to handle the amount of data to be analyzed…
- **What to do?**

Beyond memory

- RAM is not sufficient to handle the amount of data to be analyzed…
- **What to do?**
- Scale up by using parts of the available Mass Storage (hard-disk) as **virtual memory**

Out-of-memory strategies

- Chunked data files on disk
- Memory-mapped files and shared memory

Out-of-memory strategies

- Chunked data files on disk: `ff`-package
- Memory-mapped files and shared memory: `bigmemory`-package

Chunking data with the `ff`-package

Preparations

```
# SET UP --------------

# install.packages(c("ff", "ffbase"))
# load packages
library(ff)
library(ffbase)
library(pryr)

# create directory for ff chunks, and assign directory to ff
system("mkdir ffdf")
options(fftempdir = "ffdf")
```

Chunking data with the `ff`-package

Import data, inspect change in RAM.

```
##              used  (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells   1086692  58.1    1759040   94.0   1759040   94.0
## Vcells 128919617 983.6  213343342 1627.7 211037840 1610.1
```

```
mem_change(
flights <-
     read.table.ffdf(file="../data/flights.csv",
                     sep=",",
                     VERBOSE=TRUE,
                     header=TRUE,
                     next.rows=100000,
                     colClasses=NA)
)
```

```
## read.table.ffdf 1..100000 (100000)  csv-read=0.371sec ffdf-write=0.048sec
## read.table.ffdf 100001..200000 (100000)  csv-read=0.387sec ffdf-write=0.035sec
## read.table.ffdf 200001..300000 (100000)  csv-read=0.387sec ffdf-write=0.03sec
## read.table.ffdf 300001..336776 (36776)  csv-read=0.152sec ffdf-write=0.016sec
##  csv-read=1.297sec  ffdf-write=0.129sec  TOTAL=1.426sec
```

```
## -30 MB
```

Chunking data with the `ff`-package

Inspect file chunks on disk and data structure in R environment.

```
# show the files in the directory keeping the chunks
list.files("ffdf")
```

```
##   [1] "clone1664b7fbd953f.ff" "clone1664b9b8cca9.ff"  "clone1e7014c0a1cd8.ff"
##   [4] "clone1e7015a4f712e.ff" "clone2aea22211d9e1.ff" "clone2aea2360c6703.ff"
##   [7] "clone2aea2566ab42d.ff" "clone2aea25e1c1f75.ff" "clone2d49618dbfbf6.ff"
##  [10] "clone2d4965ee3349a.ff" "clone2d49664b07745.ff" "clone2d49672b82b88.ff"
##  [13] "clone308112a4ca401.ff" "clone308113d044b7c.ff" "clone308113d22fb5f.ff"
##  [16] "clone3081149714ed4.ff" "clone399cd5627eb1f.ff" "clone399cd72c6506d.ff"
##  [19] "clone399cd78f6c4e6.ff" "clone399cd8b1f075.ff"  "clone3c3ef1e38eca1.ff"
##  [22] "clone3c3ef4ac46441.ff" "clone3c3ef514956e9.ff" "clone3c3efcb5fb24.ff"
##  [25] "clone3f8e9146bba78.ff" "clone3f8e94d9633f0.ff" "clone3f8e9506fcdf1.ff"
##  [28] "clone3f8e9b9a7b8.ff"   "clone432452f2dbbc3.ff" "clone4324578aefe51.ff"
##  [31] "clone4324579d1ad52.ff" "clone432457a4743f9.ff" "clone47e962c215aa7.ff"
##  [34] "clone47e9634b394d6.ff" "clone47e96595b1fd7.ff" "clone47e96a5d61e5.ff"
##  [37] "clone605a84106bec0.ff" "clone605a86e293a8f.ff" "clone605a86fc8f5a2.ff"
##  [40] "clone605a8dfeeebd.ff"  "clonee6e02b3603f7.ff"  "clonee6e065135290.ff"
##  [43] "clonee6e0e80612d.ff"   "clonee6e0f7dd64d.ff"   "ff1664b222c38f0.ff"
##  [46] "ff1664b4d23ee78.ff"    "ff1664b4d7f1e3e.ff"    "ff1e7011754e092.ff"
##  [49] "ff1e7011a76d5a6.ff"    "ff1e7017084631e.ff"    "ff2aea22c3703b9.ff"
##  [52] "ff2aea2664ee33.ff"     "ff2aea26b164ce7.ff"    "ff2d49627cd458.ff"
##  [55] "ff2d49631ca5a34.ff"    "ff2d4964237cc21.ff"    "ff30811207897c4.ff"
##  [58] "ff308115699c1a.ff"     "ff30811b3430e.ff"      "ff399cd1a2ffc0e.ff"
##  [61] "ff399cd1e963877.ff"    "ff399cd5477c29.ff"     "ff3c3ef17300293.ff"
```

Memory mapping with `bigmemory`

Preparations

```r
# SET UP ----------------

# load packages
library(bigmemory)
library(biganalytics)
```

Memory mapping with `bigmemory`

Import data, inspect change in RAM.

```r
# import the data
flights <- read.big.matrix("../data/flights.csv",
                     type="integer",
                     header=TRUE,
                     backingfile="flights.bin",
                     descriptorfile="flights.desc")
```

Memory mapping with bigmemory

Inspect the imported data.

```
summary(flights)
```

```
##                       min         max        mean          NAs
## year          2013.000000 2013.000000 2013.000000     0.000000
## month            1.000000   12.000000    6.548510     0.000000
## day              1.000000   31.000000   15.710787     0.000000
## dep_time         1.000000 2400.000000 1349.109947  8255.000000
## sched_dep_time 106.000000 2359.000000 1344.254840     0.000000
## dep_delay      -43.000000 1301.000000   12.639070  8255.000000
## arr_time         1.000000 2400.000000 1502.054999  8713.000000
## sched_arr_time   1.000000 2359.000000 1536.380220     0.000000
## arr_delay      -86.000000 1272.000000    6.895377  9430.000000
## carrier          9.000000    9.000000    9.000000 318316.000000
## flight           1.000000 8500.000000 1971.923620     0.000000
## tailnum                                           336776.000000
## origin                                            336776.000000
## dest                                              336776.000000
## air_time        20.000000  695.000000  150.686460  9430.000000
## distance        17.000000 4983.000000 1039.912604     0.000000
## hour             1.000000   23.000000   13.180247     0.000000
## minute           0.000000   59.000000   26.230100     0.000000
## time_hour     2013.000000 2014.000000 2013.000261     0.000000
```

Memory mapping with `bigmemory`

Inspect the object loaded into the R environment.

```
flights
```

```
## An object of class "big.matrix"
## Slot "address":
## <pointer: 0x558ce6187310>
```

Memory mapping with `bigmemory`

- `backingfile`: The cache for the imported file (holds the raw data on disk).
- `descriptorfile`: Metadata describing the imported data set (also on disk).

Memory mapping with **bigmemory**

Understanding the role of `backingfile` and `descriptorfile`.

First, import a large data set without a backing-file:

```
# import data and check time needed
system.time(
     flights1 <- read.big.matrix("../data/flights.csv",
                                 header = TRUE,
                                 sep = ",",
                                 type = "integer")
)
```

```
##    user  system elapsed
##   1.164   0.016   1.180
```

```
# import data and check memory used
mem_change(
     flights1 <- read.big.matrix("../data/flights.csv",
                                 header = TRUE,
                                 sep = ",",
                                 type = "integer")
)
```

```
## 736 B
```

Memory mapping with `bigmemory`

Understanding the role of `backingfile` and `descriptorfile`.

Second, import the same data set with a backing-file:

```
# import data and check time needed
system.time(
     flights2 <- read.big.matrix("../data/flights.csv",
                                 header = TRUE,
                                 sep = ",",
                                 type = "integer",
                                 backingfile = "flights2.bin",
                                 descriptorfile = "flights2.desc"
                                 )
)
```

```
##    user  system elapsed
##   1.100   0.048   1.148
```

```
# import data and check memory used
mem_change(
     flights2 <- read.big.matrix("../data/flights.csv",
                                 header = TRUE,
                                 sep = ",",
                                 type = "integer",
                                 backingfile = "flights2.bin",
```

Memory mapping with `bigmemory`

Understanding the role of `backingfile` and `descriptorfile`.

Third, re-import the same data set with a backing-file.

```
# remove the loaded file
rm(flights2)

# 'load' it via the backing-file
system.time(flights2 <- attach.big.matrix("flights2.desc"))
```

```
##    user  system elapsed
##   0.000   0.000   0.001
```

```
flights2
```

```
## An object of class "big.matrix"
## Slot "address":
## <pointer: 0x558ce5a65230>
```

References