Profiling

Zheng Wu

11/27/2017

1.preperation

```
library(ggplot2)
library(profr)
library(magrittr)
library(profvis)
```

2.Introduction

We've all encountered the situations where the r script runs slower than usual, especially when calculating a large amount of data, processing a long table, or generating a complex plot. Even though for most of the cases we have experienced so far, it only took maybe one more second for your Rstudio to get to the result, it is always beneficial to have a script written in better ways so that it takes less time. This will become a more practical and essential problem when we work on a considerably larger dataset in our potential workplace. Therefore we always want to identify the bottleneck and improve the code to make it run faster. Profiling then comes into play, which provides a powerful tool to find out the performance of the code. With profiling, we can do more than simply understanding the code with intuition, but with a more empirical statistics about the code.

Profiling in short is the performance evaluation for the codes, which provides information for the programmer to improve the codes. It is a practice that is useful for not limited to R language. According to the demostration on wikipedia, "Profiling" in software

> is a form of dynamic program analysis that measures, for example, the space (memory) or time complexity of a program, the usage of particular instructions, or the frequency and duration of function calls. Most commonly, profiling information serves to aid program optimization.

For example, in one of the post I reviewed during research, the author shared how he reduced the running time from 28 seconds to 2 seconds, just by changing one line of the code, from

```
x = data.frame(a = variable1, b = variable2)
```

to

```
x = c(variable1, variable2)
```

The change was significant because this line of code was called multiple times during the execution of a function.

Profiling is achieved by instrumenting either the program source code or its binary executable form using a tool, named *profiler* (similar to the *tester* file we did in homework four "unit testing".) In general, profiles may use a variety of different techniques, such as event-based, statistical, instrumented, and simulation methods.

In this post, I will share with you some basics about conducting profiling with R scripts and some generic ways to diagnose time or resource consuming parts. The first part will be a quick look at a profiler application and its output. The second part will be some introduction to alternative packages. The third part would be some breif things to pay attention when profilinf. Last but not lease, the fourth part is some information about optimising the codes.

I hope you will find this post interesting and useful.

3.Rprof

Profiling can be utilized through base functions or many profiling packages. The most basic profiling function is `Rprof`. Here's an example of how it works:

```
Rprof(tmp <- tempfile())

waste <- function(x) {
  temp <- rep(0, length(x))
  for (i in 1:length(x)) {
    temp[i] = sample(1:10, size = 1, replace = TRUE)
  }
  temp <- sort(temp)
  return(temp[1])
  print(temp[1])
}
replicate(n = 1000, waste(4))
```

```
##    [1]  9  1  8 10  6  8  2  5  8  4  1  2  8  9  8 10  8  3 10  2  2  2  2
##   [24]  8  4  1  1  7  9  2  9  8  6  5  4  1  8  5  6  6  9  5  7  3  7  3
##   [47]  8  9  3  8  7 10  6  4  8 10  5  8  6  6  6  7 10 10  5  9  3 10  9
##   [70]  2  2  1  3  7  1  8  9  7  2  3  9  3 10  8  5  2  2  7  2  7  3  2
##   [93]  3  5  4  2  1  3  2  9  6  3  4  9  6  4  7  9  3  2  2  3  7  2  4
##  [116]  8  6  1  4  5  4  2  1  5  8  9 10  3  3  5  9  7  8  7  1 10  2  2
##  [139]  1  8 10 10  7  7  2  3  4  3  7 10  3  6  3  4  5  2  5  3  8 10 10
##  [162]  5  3  4  9  6  4  3  6  4  4  1  5  9  2  8  1  5  6  2  9  7  1  4
##  [185]  1  2  1  4  5  7  9  2  7 10  8  7  6  2  1  5  7  6  1  9  7 10  2
##  [208]  5  5  4  9  6  5  4  5  2  2  8  9  4  6  1  8  3 10  8  1  9  4  6
##  [231]  8  6  3  5  2 10  5  4  3  6  6  2 10 10 10  4  2  1  8  2  6  1  2
##  [254]  1  4  5  2  2  6  8  7  7  8  6  8  6  7  7  4  5  6  5  2 10  9  9
##  [277]  1  1  9  7  1  5  3  5  5  6  8  8  6  3  5  7  1  5  3  9 10  9  3
##  [300]  9  8  5  6  4  1 10  4  9  3  5 10  1  1  1  8  9  9  7  2  8  5  5
##  [323]  2  3  1  5  3  6  9  5  8  8  1  4  2  5  3  6  4  6  9  8  1  2  5
##  [346]  6  5  9 10  4  4  8  9  7  7  2  4  5  1  2  6  2  2  5  6  6  1  5
##  [369]  9  6  7  1  1 10  2  7  1  8 10  8  6  3  5  7  9  9  4  3  2  8 10
##  [392]  7  8 10  4  4  1  2  7  8  4  9  8  7  8  9  6  8  4  1 10  8  8 10
##  [415]  4  6  5 10 10 10  4  2  3  9  5  3  9  2  3  8  9  1  7  4  6  8  4
##  [438]  4  8  5  7  9  4  8  6  1  4  7  1  7  8  7  2  8 10  6  2  5  8 10
##  [461]  6  9  2  4  9  8  2  2  2  5  5  9 10 10  3  8 10  2 10  6  5 10  8
##  [484]  4  4  1  1  9  4  6  9  8 10  2  8  6  5  4  9  3  5  2  4  7 10  2
##  [507]  9  7  7  4  8  8  8  6  3  6 10  9  8  1  8  3  6  7  9  3  2  7  1
##  [530]  3  4  1  2  8  5  7  5  8  5  7  4  2  6  2  7  6  4  2  8 10  1 10
##  [553]  1  2  3  5  2  8  5 10  6  7  3  1  6 10  4 10  9  5  6  8  4  3  3
##  [576]  2  6  1  5  1  3  4  3  7  5  3  8  9  3  7  1  1  5  2  4  7  8  7
##  [599]  3  1  5  5  9  5  2  6  3  8  6  2  9  3  9  9  9  7  7  8  2  3  6
##  [622]  5  3  9  2  3  9  6  1  8  2  6  9  6  5  2  4  4  3  5  9  7  8  7
##  [645]  7  8  8  4  1  7  3  1  3  7  3  1  6  4  9  1  6  2  7  8  5  1  1
##  [668]  1  8  7 10  2  7  6  8  2  7  7  2  2  9  8  3  2  4  4  8  6  5  4
##  [691]  9  4  4  9  2  6  6  4  5  2  7  2  6  4  1  4  3  5  1 10  6  9  2
##  [714]  3  3  3  3  4  1  9  2  7  8 10  9  9 10  3  4  8  8  4  2  5  2 10
##  [737]  9  2 10  8  1  5  4 10  2  6  5 10  1  3  7  4  8  3  7  1  6  6  5
##  [760]  8  1  9  8  6  9  1  6  9  8  8  6  2  4 10  6 10  7  4 10  7  8  7
##  [783]  6  7  9 10  7  8 10  7  2  8  1  9  9  8 10  6  4  3  5  8  1  8 10
##  [806]  6  8  1  8  7  6  1  7 10  1  5  2  9  9  7  4 10  9  5  3  5  2 10
##  [829]  7  3  7 10  9  6  5  5  7  1  9  9  1  5  8  3 10  9  3  2  3  5  8
##  [852]  7  4  1 10  3  2  3  4  7 10  9  5  3  1  5  1  3  7  3  5  8  6  4
##  [875]  8 10  9  4  6  1  9  2  9  7 10  4  3  4  7  9  8  3  8  1  6  3  6
##  [898]  6  5  3  6  1  1  9 10  6  9  1  4  9  6  2  1  6  1  6  2  2  6  6
##  [921] 10  6  1  9  1 10  5  5  5  5  7  2  4  6 10  8  3  6  2 10  9  4 10
##  [944] 10  9 10  1  5  9  5  4  2  9  1  4 10 10 10  1  8  4  6  1 10  2 10
##  [967]  3  4  2  6  3 10  9  9  8  9  2  8  4  7  4 10  1 10  5  1 10  6  4
##  [990]  3  6  6  3  8  7  3  7  5  4  5
```

```
Rprof(NULL)
summaryRprof(tmp)
```

```
## $by.self
##              self.time self.pct total.time total.pct
## "sample.int"      0.02       50       0.02        50
## "sort"            0.02       50       0.02        50
##
## $by.total
##                       total.time total.pct self.time self.pct
## "block_exec"                0.04       100      0.00        0
## "call_block"                0.04       100      0.00        0
## "eval"                      0.04       100      0.00        0
## "evaluate_call"             0.04       100      0.00        0
## "evaluate"                  0.04       100      0.00        0
## "FUN"                       0.04       100      0.00        0
## "handle"                    0.04       100      0.00        0
## "in_dir"                    0.04       100      0.00        0
## "knitr::knit"               0.04       100      0.00        0
## "lapply"                    0.04       100      0.00        0
## "process_file"              0.04       100      0.00        0
## "process_group.block"       0.04       100      0.00        0
## "process_group"             0.04       100      0.00        0
## "replicate"                 0.04       100      0.00        0
## "rmarkdown::render"         0.04       100      0.00        0
## "sapply"                    0.04       100      0.00        0
## "timing_fn"                 0.04       100      0.00        0
## "waste"                     0.04       100      0.00        0
## "withCallingHandlers"       0.04       100      0.00        0
## "withVisible"               0.04       100      0.00        0
## "sample.int"                0.02        50      0.02       50
## "sort"                      0.02        50      0.02       50
## "sample"                    0.02        50      0.00        0
##
## $sample.interval
## [1] 0.02
##
## $sampling.time
## [1] 0.04
```

In this example, `Rprof` outputs a summary of the profiling which contains four parts of information:

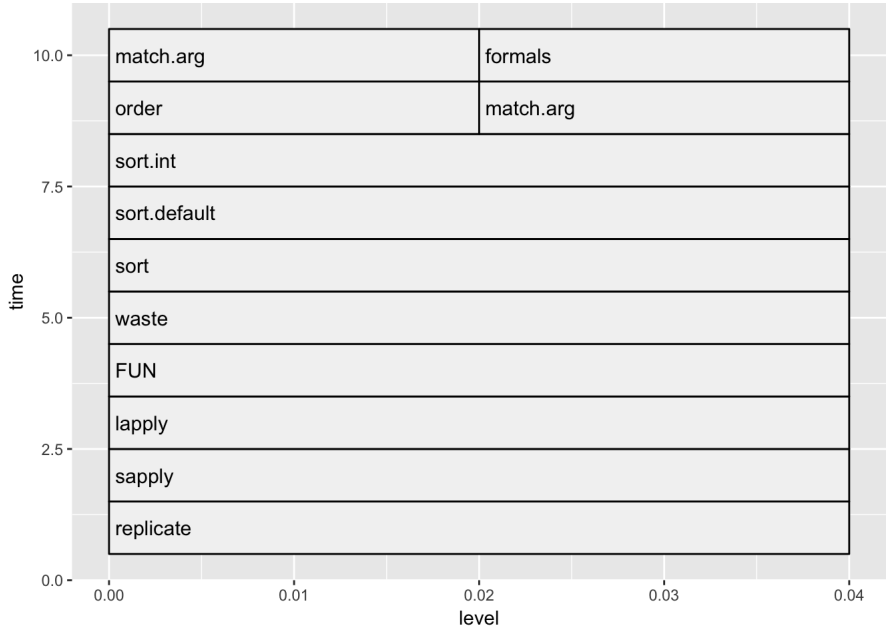
- by.self
- by.total
- sample.interval, that is the period of testing
- sampling.time, that is the total time spent running the function

The machanism of `Rprof` is that it test every testing period to see which execution it is running. By doing this, we get an idea of how much time is spent on each function. The time interval in the example above is by default 0.02 second. When the function is too short to fit in the standard time interval while you still want to see its performance, what you can do is `summaryRprof(replicate(n = 1000, waste(4)))`. This repeats the function by many times and therefore you can calculate the average time of each run.

4.Alternative Packages

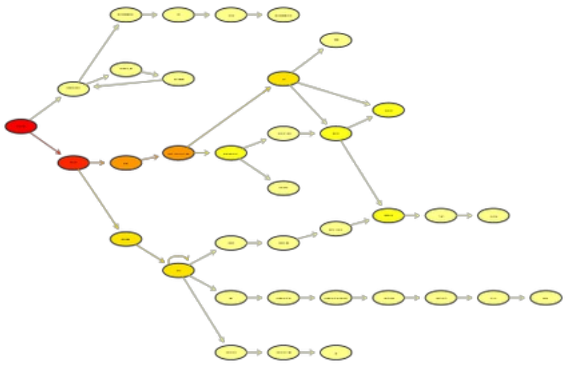
There are many alternative packages that help profile the functions. For example, `profr` profiles with visualized summary output:

```
ggplot(profr(replicate(n = 1000, waste(4))))
```



This function corperates with `ggplot2` and produce a chart of the summary. For more information, you can refer to the documentation of profr

Another alternative visualizing package is `proftools`. It does a little bit more than `profr` by producing a node diagram of the execution of the function. Here's an example.



You can find more information about `proftools` in its documentation: proftools

The third package I want to introduce is `profvis`. You may have guessed that this function does something similar to what `ggvis` does, well, you are right. `profvis` produces an interactive summary of profiling where you can see the time flow on function executions. Here's an example:

```
profvis({
  waste <- function(x) {
  temp <- rep(0, length(x))
  for (i in 1:length(x)) {
    temp[i] = sample(1:10, size = 1, replace = TRUE)
  }
  temp <- sort(temp)
  return(temp[1])
  print(temp[1])
}
  replicate(n = 1000, waste(4))
})
```

In the output, you can see an interactive interface on which you can hovor your mouse and see more operations. `profvis` is very powerful in transverting the time into a time cordinate so that it is more intuitive and easy to find the bottleneck.

5.Tricks

In this section, I'll introduce some tricks or places to notice when doing profiling that I found online or during my own experience.

create a profiling file

It is useful to create a seperate profiling file like "profile_functions.R" to do the profiling, instead of trying to accomplish everything in the original "function.R" that is to be profiled. This allows you to keep everything arranged, and you won't accidentally forget to delete profiling part in the original file. Similarly, you may want to put the outputs together and keep a track of the performance of each version of your function, so that you can always refer to or roll back a version.

clean up memory

It is stated in a blog I reviewed that the existing memory may have a side effect on the result you get when profiling. Therefore, it is beneficial to clean the memory every time before profiling to get a consistent measurement.

Memory usage is also a factor that affects performance. You may want to keep this in mind when you are writing a function that will allocate a large amount of memory.

6.After Profiling

The ultimate goal of profiling is to optimize the performance of the functions. Since it's not the main topic of this post, here I will only briefly talk about some points that may make a difference on your codes.

think about complexity

Complexity is the concept that evalutes the function's time spent with an input of size n. This is an important aspect of algorithm that everyone should know if they are to become a professional programmer. Here's a link to complexity's definition

Complexity requires skills and familarity with programming. It is an important step to take in advancing your programming ability.

avoid unnecesary loops

Reduce the amount of unnecessary loops, and make sure not to include actions that are not needed to be repeated in the loops. When it is executed by a large number of times, a minor difference can be huge.

function selection

This is somewhat trivial but good to know. Some functions are just not working as well as others, because of their nature or specialization. For example, `vapply()` works faster than `sapply()` because it specifies the output type.

Also, make sure to specify some attributes for the functions so that it will do less work if you give more information. For example, specify the known `levels` when using `factor()`.

A guide

You can find more about optimizing codes in advanced R written by Hadley Wickham, where he shared many useful tips for us to learn.

7.Conclusion

Profiling is especially important when you are dealing with a large number of data, or are writing a reproducible code that is gonna be used by many. This may look unnecessary for now, but will definitely make your life easier when it becomes a major part of your work.

Profiling and optimizing is an attitude of challenging and advancing our skills endlessly. It should always be a passionate pursuit to write the best codes. I hope you find this post useful, and thank you for your time. Good luck with finals!

8.Reference

- https://en.wikipedia.org/wiki/Profiling_(computer_programming)
- https://www.r-bloggers.com/profiling-r-code/
- https://support.rstudio.com/hc/en-us/articles/218221837-Profiling-with-RStudio
- http://adv-r.had.co.nz/Profiling.html
- https://blog.rstudio.com/2016/05/23/profiling-with-rstudio-and-profvis/
- https://rdatamining.wordpress.com/2012/08/01/examples-of-profiling-r-code/
- https://rstudio.github.io/profvis/
- http://dirk.eddelbuettel.com/papers/rFinanceHPC.pdf
- http://adv-r.had.co.nz/Profiling.html#be-lazy