DOI: http://dx.doi.org/10.26483/ijarcs.v8i7.4348

ISSN No. 0976-5697

Volume 8, No. 7, July – August 2017

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

R PROFILING: STRATEGIES FORPERFORMANCE IMPROVEMENTS IN R APPLICATION

Devharsh Trivedi
Magnetic Resonance Imaging (R&D)
Philips Innovation Campus
Bangalore, India

Abstract: This paper is in continuation with my efforts on improving performance of R application. In my previous paper titled "A Study on CRAN R and MRAN R Interpreters", I have demonstrated performance improvements by Microsoft R over core R out of the box. Microsoft R uses Intel MKL library to make R applications multithreaded. In this paper, I have highlighted some APIs using which performance will increase manifolds. This replacement requires little to no overhead in code change thus very effective to use by legacy apps.

Keywords: Performance improvements in R, Performance optimization in R, R Profiling

I. INTRODUCTION:

R is an interpreted language; hence, it is bound to be slow when compared to other compiled languages like C/C++. Mostly Data Science and Machine Learning community uses R, often dealing with huge amount of data, thus performance becomes critical. Aim of this paper is to improve sluggish performance of R. In my previous paper [1], I introduced Microsoft R [2]which using multithreading boosts performance. In this paper, tactics are presented to gain performance improvement out-of-the-box. How switching APIs like read_csv and fread over read.csv, fastPOSIXct over POSIXct, and data types like character vs factor, data.table vs data.frame can improve performance is shown in this paper.

II. SYSTEM SPECS:

Processor: Intel i5 @2.30 GHz
Cores: 2 physical, 4 logical
RAM: 8 GB
OS: Windows 7 64-bit
R: Microsoft R Open 3.4.0
RStudio: Version 1.0.143
CRAN mirror: snapshot taken on 01-May-2017
Intel MKL [3]: Enabled, using all physical cores (2)

III. PROFILING:

You can use built-in method system.time() for measuring execution time taken by a function or portion of the code or use microbenchmark() from microbenchmark[4]library. If you have a big application, it is better to use Rprofvis()[5]which records all calls executed during application lifecycle and generates a rich html file which helps finding memory/CPU bottlenecks. RStudio has RProfvis installed by default.

IV. READING CSV:

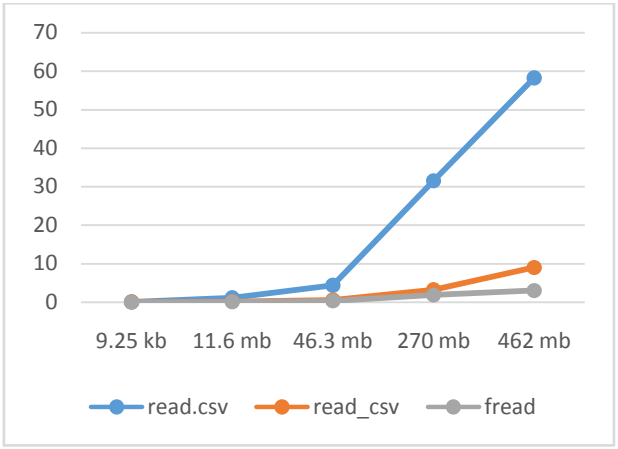

Figure 1. Time taken by different file read functions.

R applications heavily make use of CSV files, as it is a simple way of storing and sharing tabular data. It is very convenient to mold CSV data into R type like list, matrix, data.frame or data.table. Here I am comparing three popular methods for reading CSV: read.csv(), read_csv, and fread(). Read_csv() is built-in method, read_csv() requires readr[6] library, and fread() requires data.table[7] library. Following shows time comparisons of these functions for different file sizes. With fread you can get around 20 times performance improvement over read.csv().

Table I. Time taken by different file read functions

9.25 kb		read.csv	read_csv	fread
	run1	0	0.09	0
	run2	0.02	0.14	0
	run3	0.01	0.02	0
	avg	0.01	0.083333	0
11.6 mb		read.csv	read_csv	fread

	run1	1.12	0.15	0.09
	run2	1.1	0.13	0.08
	run3	1.1	0.13	0.08
	avg	1.106667	0.136667	0.083333
46.3 mb		read.csv	read_csv	fread
	run1	4.52	0.61	0.33
	run2	4.24	0.58	0.33
	run3	4.21	0.57	0.31
	avg	4.323333	0.586667	0.323333
270 mb		read.csv	read_csv	fread
	run1	31.85	3.54	1.86
	run2	31.34	3.06	1.87
	run3	31.27	3.04	1.85
	avg	31.48667	3.213333	1.86
462 mb		read.csv	read_csv	fread
	run1	59.07	9.16	3
	run2	57.89	8.97	3
	run3	57.78	8.87	3.03
	avg	58.24667	9	3.01

V. POSIXCONVERSIONS:

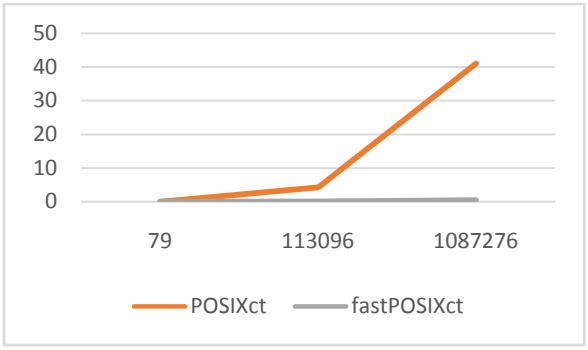

Figure 2. Time taken by POSIXct and fastPOSIXct

POSIXct is the number of seconds since the epoch – 1st January 1970. POSIXctcan be best used as a list structure of dates. If you need to convert character or factor time format to a time series object, you will convert it with as.POSIXct. But this call is very costly, use fastPOSIXct() instead from fasttime[8] library.fastPOSIXct is extremely fast compared to POSIXct as it uses pure string parsing without overhead of additional system callsas in the case of POSIXct. Following shows comparisons of time taken for POSIXct and fastPOSIXct for different numbers of records. You can achieve around 85 times performance improvement.

Table II. Time taken by POSIXct and fastPOSIXct

79 rows		POSIXct	fastPOSIXct
	run1	0	0
	run2	0	0
	run3	0	0
	avg	0	0

113096 rows		POSIXct	fastPOSIXct
	run1	4.21	0.05
	run2	4.23	0.07
	run3	4.23	0.05
	avg	4.22333	0.05666667
1087276 rows		POSIXct	fastPOSIXct
	run1	40.91	0.46
	run2	40.93	0.45
	run3	41.14	0.46
	avg	40.9933	0.45666667

VI. CHARACTER VECTORS:

Another trick is to use character vector instead of factor. Using character with fastPOSIXct() and table(unlist()) instead of factor may up the performance a few notch. Following table shows a marginal performance gain by character over factor for fastPOSIXct. Vector size was 599476 records.

Table III. Character vs factor parsing in fastPOSIXct

fastPOSIXct		character	factor
	run1	0.25	0.3
	run2	0.25	0.28
	run3	0.25	0.28
	avg	0.25	0.287

VII.DATA.TABLES:

Data.tables may improve performance over list/data.frames when using with merge() or format(). You should avoid using format() in all cases. If you have to use merge to join two frames convert it to tables, which will increase performance. Following table shows comparison:

Table IV. Comparison of data.frame and data.table

merge		data.frame	data.table
	run1	71.45	8.91
	run2	69.69	8.8
	run3	68.66	8.83
	avg	69.93333333	8.846666667

VIII. STORING RESULT:

After performing operation on a variable, saving result in different variable is faster than reusing the same variable on which operation was performed.

Table V. Time taken when result is saved in the same variable on which operation is performed v/s result saved in new variable

operation 1		same variable	different variable
	run1	1.65	0.29
	run2	1.56	0.28
	avg	1.605	0.285
operation 2		same variable	different variable
	run1	0.33	0.08
	run2	0.3	0.07
	avg	0.315	0.075

IX. OTHER OBSERVATIONS:[9][10]

1. Gsub() is slower than strptime().
2. POSIXct(strptime()) is faster than POSIXct().
3. Library dplyr is faster than plyr.
4. Stringr is faster than base-R implementation.
5. Don't use loops (scalars) instead use lapply() or other vectorization methods like apply(), mapply(), sapply(), replicate() etc.
6. Using C-compiled code with R improves performance. Use Rcpp to compile code with C or FORTRAN.
7. Keep inner loops bigger than outer loops.

X. CONCLUSION

It is observed that performance of R applications can be significantly improved by executing it in multicore environment and switching to performance oriented APIs and data types. The study was done using profilers like Rprofvis and system.time. You can opt for other profilers. One such profiler is GUIProfiler. There is an excellent case study done on it which can be find in research article by Angel Rubio and Fernando de Villar[11].

This research can be extended by yielding results for other observations mentioned in section IX.

XI. ACKNOWLEDGEMENT:

I would like to thank Mr. Manish Kumar (manish.kumar@philips.com) for guiding me during the course of this research.

XII. REFERENCES:

[1] Devharsh Trivedi. "A Study on CRAN R and MRAN R Interpreters." International Journal for Scientific Research and Development 4.2 (2016): 992-994.
[2] Microsoft R Open: The Enhanced R Distribution URL https://mran.microsoft.com/open/
[3] The Benefits of Multithreaded Performance with Microsoft R Open URL https://mran.microsoft.com/documents/rro/multithread/
[4] Package 'microbenchmark' URL https://cran.r-project.org/web/packages/microbenchmark/microbenchmark.pdf
[5] Exploring profiles in RStudio URL https://rstudio.github.io/profvis/rstudio.html
[6] Package 'readr' URL https://cran.r-project.org/web/packages/readr/readr.pdf
[7] Package 'data.table' URL https://cran.r-project.org/web/packages/data.table/data.table.pdf
[8] Package 'fasttime' URL https://cran.r-project.org/web/packages/fasttime/fasttime.pdf
[9] A Guide to Speeding Up R Code URL https://www.r-bloggers.com/faster-higher-stonger-a-guide-to-speeding-up-r-code-for-busy-people/
[10] Strategies to Speedup R Code URL https://datascienceplus.com/strategies-to-speedup-r-code/
[11] Angel Rubio, Fernando de Villar. Code Profiling in R: A Review of Existing Methods and an Introduction to Package GUIProfiler. The R Journal, 7(2):275-287, Dec. 2015.