Profiling

ROBINE Thomas

8 juin 2020

I. Introduction

We do the profiling of the simple case vignette and we want to know which code points take the most time and check that they match the way the code was thought. Here we test the optimization of a parameter of a USM. We carry out these tests with a repetition number of 2 and a number of assessments of 250 for two reasons. The first being the execution time and the second, the fact that the IDE Rstudio has trouble loading the results of the profiler with too large repetitions and evaluations. Finally, these tests are performed in two configurations, a first sequential configuration and a second using parallelisation with a CPU.

So, we profile our code using the profiling tool from RStudio and we obtaine a flame graph that we analyzed. Here is the listing of all the calls made after the "model_function" call, that we know their origin. There are many calls that we don't know from what part they are called from.

II. Calls' path and source

do.call : SticsOnR/R/stics_wrapper.R#118

do.call -> exists_sticks_exe : SticsOnR/R/stics_wrapper.R#642

do.call -> check_sticks_exe : SticsOnR/R/stics_wrapper.R#642

do.call -> check_sticks_exe -> suppressWarnings : SticsOnR/R/stics_exe_utilities.R#288

do.call -> check_sticks_exe -> suppressWarnings -> run_system_cmd : SticsOnR/R/stics_exe_utilities.R#288

do.call -> check_sticks_exe -> suppressWarnings -> run_system_cmd -> system2 : SticsOnR/R/run_system_cmd.R#19

check_sticks_exe : SticsOnR/R/stics_wrapper.R#131

check_sticks_exe -> suppressWarnings : SticsOnR/R/stics_exe_utilities.R#288

check_sticks_exe -> suppressWarnings -> run_system_cmd : SticsOnR/R/stics_exe_utilities.R#288

check_sticks_exe -> suppressWarnings -> run_system_cmd -> system2 : SticsOnR/R/run_system_cmd.R#19

parallel::makeCluster : SticsOnR/R/stics_wrapper.R#140

parallel::clusterCall : SticsOnR/R/stics_wrapper.R#143

%dopar% : SticsOnR/R/stics_wrapper.R#237

III. Parallel's runs with 1 USM

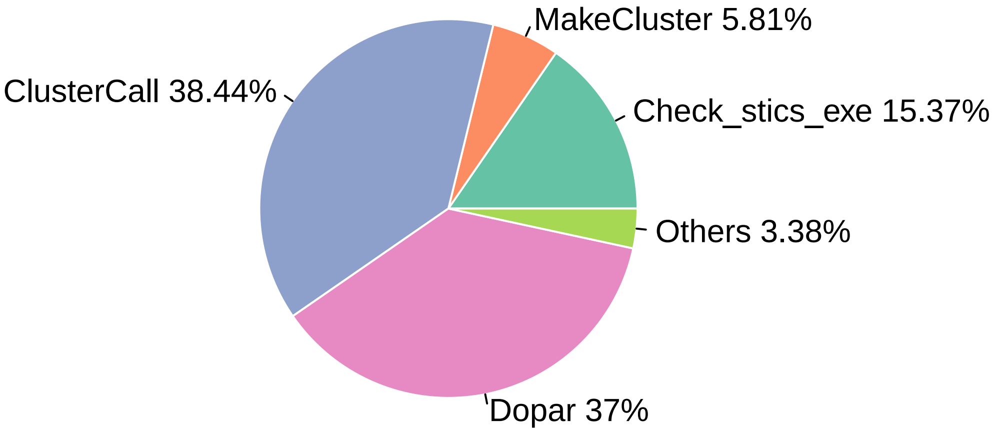
Call	Run time (ms)			Global run time (ms)			Mean rate (%)
	Min	Max	Mean	Min	Max	Mean	
prog	-	-	-	203 600	212 620	207 493	-
model_function	-	-	-	194 930	200 880	197 613	95.24
-> do.call	170	180	173	15 430	16 150	15 723	7.58
-> -> exists_stics_exe	10	10	10	620	860	707	0.35
-> -> check_stics_exe	160	160	160	31 810	32 020	31 900	15.37
-> -> -> suppressWarnings	160	160	160	31 770	32 110	31 900	15.37
-> -> -> -> run_system_cmd	160	160	160	31 760	31 990	31 853	15.35
-> -> -> -> -> system2	160	160	160	34 680	35 830	34 973	16.86
-> check_stics_exe	160	210	180	31 810	32 020	31 900	15.37
-> -> suppressWarnings	160	210	180	31 770	32 110	31 900	15.37
-> -> -> run_system_cmd	160	210	180	31 760	31 990	31 853	15.35
-> -> -> -> system2	160	210	180	34 680	35 380	34 973	16.86
-> parallel::makeCluster	120	130	123	11 880	12 290	12 056	5.81
-> parallel::clusterCall	790	830	810	78 210	80 670	79 750	38.44
-> %dopar%	650	890	750	72 960	81 600	76 790	37

Table 1. Thread method's profile for 1 USM

The "Average run time" column represents the time spent in the call. For example, the time spent in the "do.call -> check_sticks_exe" call is 160 ms whereas in the "check_sticks_exe" call without "do.call", it is 180 ms.

The column "Average global run time" represents the total time spent in the function throughout the program. This is why some functions have an "Average global run time" higher than the function that called it. In this case, the called function was called elsewhere in the program without going through the previous function.

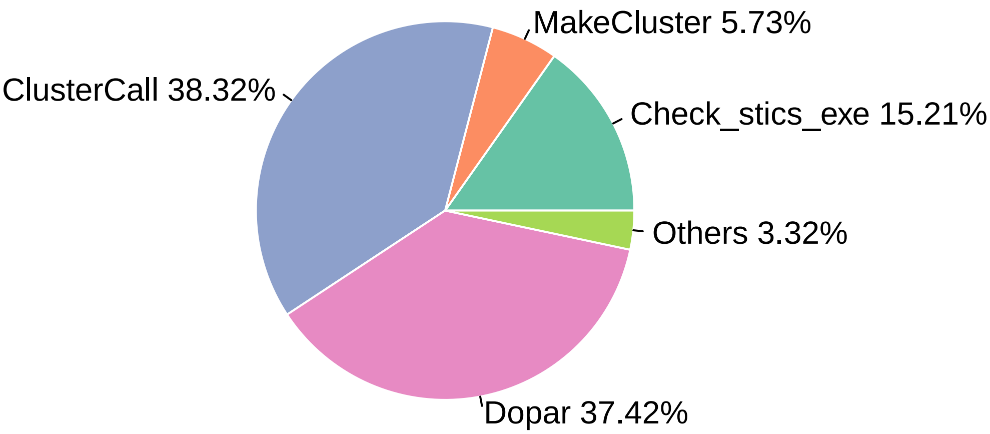
By looking at the calls, we can group them into four parts. The first includes the call to Stics (System2), then we have the parts concerning the calls to the cluster and finally the %dopar% part.



IV. Sequential's runs with 1 USM

Call	Run time (ms)			Global run time (ms)			Mean rate (%)
	Min	Max	Mean	Min	Max	Mean	
prog	-	-	-	206 950	212 170	210 166	-
model_function	-	-	-	196 130	201 730	199 810	95.07
-> do.call	140	150	143	15 830	16 090	15 770	7.50
-> -> exists_stics_exe	10	10	10	790	840	810	0.39
-> -> check_stics_exe	130	140	133	31 510	32 250	31 966	15.21
-> -> -> suppressWarnings	130	140	133	31 500	32 250	31 946	15.20
-> -> -> -> run_system_cmd	130	140	133	31 490	32 220	31 933	15.19
-> -> -> -> -> system2	130	140	133	34 750	35 090	34 940	16.62
-> check_stics_exe	160	180	166	31 510	32 250	31 966	15.21
-> -> suppressWarnings	160	180	166	31 500	32 250	31 946	15.20
-> -> -> run_system_cmd	160	180	166	31 490	32 220	31 933	15.19
-> -> -> -> system2	160	180	166	34 750	35 090	34 940	16.62
-> parallel::makeCluster	110	130	120	11 680	12 590	12 053	5.73
-> parallel::clusterCall	760	820	790	77 230	83 940	80 546	38.32
-> %dopar%	630	800	740	76 610	79 970	78 650	37.42

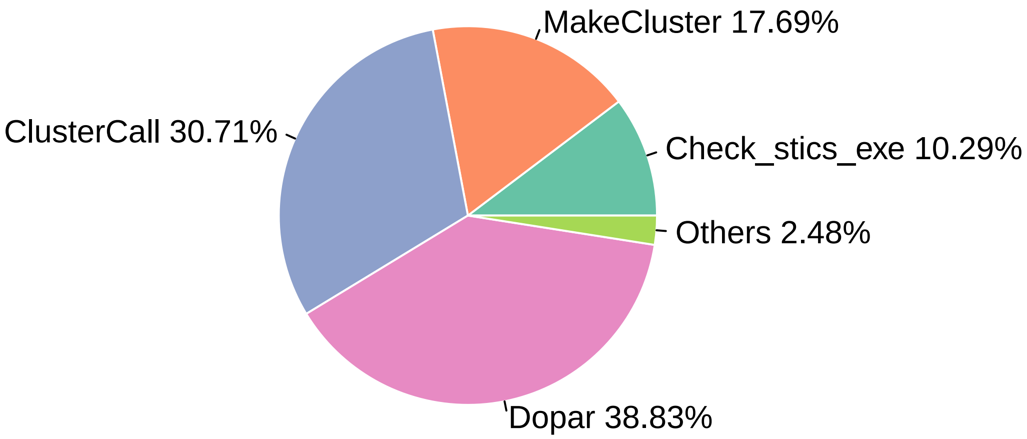
Table 1. Sequential method's profile for 1 USM



V. Parallel's runs with 7 usms

Call	Run time (ms)			Global run time (ms)			Mean rate (%)
	Min	Max	Mean	Min	Max	Mean	
prog	-	-	-	415 290	425 370	421 367	-
model_function	-	-	-	407 040	414 030	409 520	97.19
-> do.call	80	200	147	20 290	22 060	22 253	5.28
-> -> exists_stics_exe	10	20	13	460	1410	1047	0.25
-> -> check_stics_exe	70	180	130	42 960	44 070	43 347	10.29
-> -> -> suppressWarnings	70	180	123	42 330	43 320	42 690	10.13
-> -> -> -> run_system_cmd	70	180	123	42 290	43 320	42 673	10.13
-> -> -> -> -> system2	70	180	123	44 820	46 970	45 677	10.84
-> check_stics_exe	90	180	143	42 960	44 070	43 347	10.29
-> -> suppressWarnings	90	180	143	42 330	43 320	42 690	10.13
-> -> -> run_system_cmd	90	180	143	42 290	43 320	42 673	10.13
-> -> -> -> system2	90	180	143	44 820	46 970	45 677	10.84
-> parallel::makeCluster	540	590	567	74 000	75 270	74 537	17.69
-> parallel::clusterCall	960	980	967	128 320	131 150	129 387	30.71
-> %dopar%	1 150	1 220	1 183	163 280	164 090	163 630	38.83

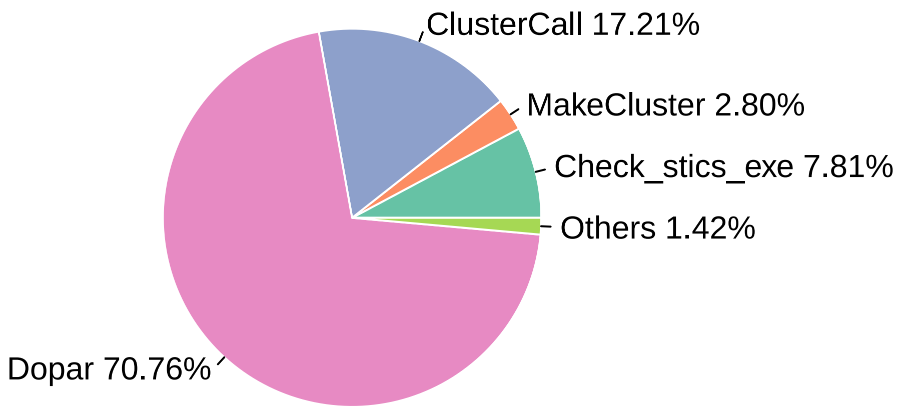
Table 1. Thread method's profile for 7 USMs



VI. Sequential's runs with 7 usms

Call	Run time (ms)			Global run time (ms)			Mean rate (%)
	Min	Max	Mean	Min	Max	Mean	
prog	-	-	-	499 960	534 460	512 723	-
model_function	-	-	-	492 120	525 760	504 023	98.30
-> do.call	150	160	153	20 080	21 180	20 533	4
-> -> exists_stics_exe	10	10	10	1 020	1 110	1070	0.21
-> -> check_stics_exe	130	150	140	37 980	43 460	40 040	7.81
-> -> -> suppressWarnings	130	150	140	37 980	43 460	40 037	7.81
-> -> -> -> run_system_cmd	130	150	140	37 980	43 440	40 027	7.81
-> -> -> -> -> system2	130	150	140	40 290	46 180	42 703	8.33
-> check_stics_exe	140	170	157	37 980	43 460	40 040	7.81
-> -> suppressWarnings	140	170	157	37 980	43 460	40 037	7.81
-> -> -> run_system_cmd	140	170	157	37 980	43 440	40 027	7.81
-> -> -> -> system2	140	170	157	40 290	46 180	42 703	8.33
-> parallel::makeCluster	100	110	107	13 820	15 240	14 350	2.80
-> parallel::clusterCall	610	700	660	85 200	93 580	88 230	17.21
-> %dopar%	2 690	2 770	2 723	356 130	374 740	362 797	70.76

Table 1. Sequential method's profile for 7 USMs



VII. Conclusion

We can see several things. First, when we optimize multiple Usms, the dopar part works as expected by taking a bigger part of the program sequentially than in parallel. On the contrary, we can see that the dopar part occupies only one third of the execution time of the program whereas it should take as much as what it takes in the case with 7 usms with the sequential method. So, we can save time on cluster functions by storing their results in global variables for reuse. We can also make conditions, but it would significantly make the code heavier. Finally, we can store the result of the check_stics command to avoid having to perform it sequentially throughout the program.