Benchmark Classification for GC Tuning

Olivier Flückiger

March 12, 2018

Abstract

For evaluating the performance of a garbage collector (GC) a balanced set of benchmarks is key. To provide meaningful guidance, we must understand what kind of workload a certain benchmark simulates. Since tuning parameters heavily influence GC performance, we need to be careful, to evaluate each kind of workload in isolation. We propose to develop a methodology for automatically clustering benchmarks by their interaction with GC tuning.

1 Introduction

A typical GC implementation has several tuneable parameters, such as heap grow rate, various thresholds for triggering collection cycles, and so on. Picking an optimal set of parameters is a multi-dimensional optimization problem, and the resulting default parameters are always a compromise.

To better understand GC tuning, we propose to develop a methodology for classifying benchmarks according to their behavior under varying GC parameters. This allows us to produce a reduced benchmark suite, which yields a reasonable approximation of the full suite, under varying GC parameters. The intuition is, that many benchmarks will have similar allocation behavior. Including more than one benchmark of the same kind is redundant, when we are concerned with stressing the GC.

There are several applications of such a classification. For efficient testing, we want to know the minimally required classes of benchmarks. For a fair and unbiased comparison, we want to ensure, that we covered all facets of a GC algorithm. Also, we can provide more insightful evaluations, if we report performance numbers for each class. Finally, if we know which classes of programs exist, we can detect them and adjust tuning parameters at runtime, overcoming the one size fits all nature of ahead-of-time GC tuning.

2 Project Proposal

For this project we will focus on one specific instance. We will use a standard R benchmark suite called *rbench*[1] and execute the contained benchmarks with the latest

[1] https://github.com/rbenchmark/benchmarks

version of the reference R bytecode interpreter [R Core Team, 2017]. As a preparatory step we identify and expose all external and internal tuning parameters of the R GC.

For data collection we we execute each benchmark in the suite for a broad selection of tuning parameters. For each run we keep track of execution time and maximum working set size (WSS). We say that a run is optimal if it is within $n\%$ (for a yet to be determined n) of the best run and sub-optimal otherwise. This classification allows us to terminate processes early, if they exceed the current threshold for sub-optimality, and allows for fast data collection. This will generate a table with the following columns, where P_n denotes the n-th tuning parameter, and Time and WSS can be ∞ for aborted runs.

$$\text{Program}, \quad P_1, \quad P_2, \quad \ldots, \quad P_n, \quad \text{Time}, \quad \text{WSS}$$

From those results our aim is to identify classes of similarly behaved programs. For example, if all runs are classified optimal, the program does not depend on the GC parameters at all.

We expect the optimal runs to form a mostly closed region around the best run. But GC's are highly complex, so we expect outliers and noise, or even completely separate islands of optimality. Hence, we will use a support vector machine (SVM) with a rbf kernel as a model to carve out the areas of optimal parameters. Finally, we use k-means (or a similar algorithm) to cluster programs with similar SVM models.

To evaluate the results we study a recent GC parameter tuning in the official R interpreter. SVN commit 67660[2] introduces a GC tuning, with the following commit message: *"Increased R_NGrowIncrFrac and R_VGrowIncrFrac to 0.2 to reduce GC frequency when the heap needs to grow"*. If our hypothesis holds, the median speedup of this commit on the full *rbench* must be similar to the median speedup for the reduced suite.

3 Status

So far we have assessed the status of the *rbench* benchmark suite. We have discovered and fixed several issues. Also, we have adjusted the running times of the individual benchmarks to more reasonable defaults.

Then, we have run the *rbench* suite against R interpreters built from svn commits 67659 and 67660 (the commits before and after the GC tuning mentioned above). We have verified that we the tuning is visible in the benchmark results. The change speeds up roughly 10% of the benchmarks by 5-25% (50% in one case), while using 10-40% more RAM in about 20% of them.

4 Related Work

Various authors document efforts to specialize GC parameters to the workload. There are approaches trying to optimize parameters for average performance [Brecht et al., 2006], for application specific performance [Lengauer and Mössenböck, 2014, Singer et al., 2007], as well as on-line methods, for tuning at runtime [Cheng et al., 1998, Singer et al., 2011].

[2]https://github.com/wch/r-source/commit/15712025cae094879f92d6999ec9d7ee76ccddbf

References

[Brecht et al., 2006] Brecht, T., Arjomandi, E., Li, C., and Pham, H. (2006). Controlling garbage collection and heap growth to reduce the execution time of java applications. *ACM Trans. Program. Lang. Syst.*, 28(5):908–941.

[Cheng et al., 1998] Cheng, P., Harper, R., and Lee, P. (1998). Generational stack collection and profile-driven pretenuring. In *Proceedings of the ACM SIGPLAN 1998 Conference on Programming Language Design and Implementation*, PLDI '98, pages 162–173, New York, NY, USA. ACM.

[Lengauer and Mössenböck, 2014] Lengauer, P. and Mössenböck, H. (2014). The taming of the shrew: Increasing performance by automatic parameter tuning for java garbage collectors. In *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering*, ICPE '14, pages 111–122, New York, NY, USA. ACM.

[R Core Team, 2017] R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

[Singer et al., 2007] Singer, J., Brown, G., Watson, I., and Cavazos, J. (2007). Intelligent selection of application-specific garbage collectors. In *Proceedings of the 6th International Symposium on Memory Management*, ISMM '07, pages 91–102, New York, NY, USA. ACM.

[Singer et al., 2011] Singer, J., Kovoor, G., Brown, G., and Luján, M. (2011). Garbage collection auto-tuning for java mapreduce on multi-cores. In *Proceedings of the International Symposium on Memory Management*, ISMM '11, pages 109–118, New York, NY, USA. ACM.