Experimental and Theoretical Analyses of Memory Allocation Algorithms

†Diego Elias, †Rivalino Matias, †Marcia Fernandes, ‡Lucio Borges
† School of Computer Science, ‡ School of Mathematics
Federal University of Uberlandia, Brazil
diegoelias@comp.ufu.br, rivalino@fc.ufu.br, marcia@facom.ufu.br, lucio@famat.ufu.br

ABSTRACT
In this paper, we present an experimental study to compare six user-level memory allocators. In addition, we compare the experimental results with the asymptotic analyses of the evaluated algorithms. The experimental results show that parallelism affects negatively the investigated allocators. The theoretical analysis of the execution time demonstrated that all evaluated allocators show linear complexity with respect to the number of allocations.

Categories and Subject Descriptors
D.4.2 [**Operating Systems**]: Storage Management – *allocation/deallocation strategies, main memory*; D.4.8 [**Operating Systems**]: Performance – *measurements.*

General Terms
Experimentation, Measurement, and Performance.

Keywords
Memory allocators, multithreading, algorithm analysis.

1. INTRODUCTION
In computer systems engineering, memory management has a significant impact on performance. Real-world applications need to allocate and release portions of memory, many times, during their runtime [6]. The code responsible for these operations is usually part of a system library (e.g., libc), which is called user-level memory allocator (UMA) [11].

Previous works (e.g., [1], [3], [4], [6], and [9]) have evaluated different UMAs from an experimental point of view. They used real applications and benchmark tools, which sometimes make it difficult to control important factors for a more comprehensive UMA evaluation, such as varying the size of memory blocks, number of threads, number of allocations per thread, and others. We also observe that many of these previous works did not apply a rigorous statistical method to plan their experiments and analyze the results.

SAC'14, March 24-28, 2014, Gyeongju, Korea.

http://dx.doi.org/10.1145/2554850.2555149

Hence, in this work we evaluate six widely adopted memory allocator algorithms through statistically controlled experiments. We compare the performance of the allocators in terms of execution time and memory usage. Complementarily, the algorithms are also evaluated from a theoretical viewpoint by means of asymptotic analysis.

2. METHODOLOGY
This work evaluated the following user-level memory allocators: Hoard (v3.8) [4], Ptmallocv2 [8], Ptmallocv3 [8], TCMalloc (v1.5) [7], Jemalloc (v2.0.1) [5], and Miser [12]. A detailed description of each allocator may be obtained in [6]. The experiments were conducted in a test bed composed of a quad-core computer running the Linux OS (kernel 2.6.37). We created a test program that allows us to control the number of allocations (NA), size of allocations (SA), and number of threads (NT). We also control a fourth factor related to the number of processors (NP). Each thread performs NA/NT allocation requests, where every request size is a random value drawn from a Uniform distribution of the specified size interval (SA), and filled with zeros right after their allocation. After the memory usage of all allocated blocks, 50% of them are released; the first half of the allocated area. This releasing policy enables us to measure the UMA's allocation overhead, which includes the effects of internal and external heap fragmentations [11]. Table 1 summarizes the adopted factors and their respective levels. The level values used are based on previous experimental works (e.g., [2], [4], and [6]).

Table 1. Experimental Plan Summary

		Levels
Factors	Number of allocations (NA)	500 thousands, 1 million
	Size of allocations (SA)	16..64 bytes, 256..1024 bytes
	Number of threads (NT)	1, 2, 3, 4
	Number of processors (NP)	1, 2, 3, 4

To identify the principal factors and interactions that influence the response variables (execution time and memory consumption), we use the analysis of variance (ANOVA) technique [10]. In order to find out the statistically significant interactions, we applied the Tukey test [10] for multiple comparisons of the response variable averages, assuming a significance level of 5% (i.e., α=0.05).

Our experimental plan is based on a mixed 2 and 4 Level Factorial Design, which resulted in a total of 64 treatments. Each treatment is a test for a given combination of factors and levels [10]. We repeated each test execution 10 times, so in total we execute 768 tests per allocator. For each test, we averaged the ten replications' values for each response variable.

3. EXPERIMENTAL RESULT ANALYSIS

3.1 Execution Time (ET)

For the six allocators, we found that all evaluated factors presented statistically significant effects on ET. Consistently, for all allocators the lowest averages of execution time were obtained with a single processor (NP=1), and the highest averages of execution times with four processors (NP=4) as it can be observed in the Figure 1. For scenarios with one thread (NT=1) and four processors the lowest execution times are obtained with Ptmallocv2 and Ptmallocv3, regardless of the other factors. For scenarios with four threads (NT=4) and four processors, the TCMalloc and Ptmallocv2 present the lowest execution times. For all evaluated scenarios, the lowest average of execution times is observed with Ptmallocv2 followed by TCMalloc and Jemalloc.

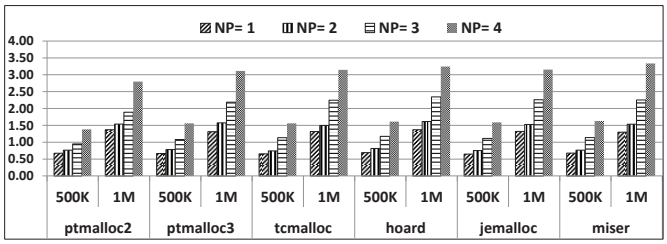

Figure 1. Allocators' execution time for NT=4 × SA=16..64.

3.2 Space Usage (SU)

In general, all factors showed significant influence on SU. In all allocators, for scenarios with small allocations (SA=16..64) the lowest average of SU was obtained with one thread and two processors (NP=2 and NT=1). We hypothesize that the influence of processors and threads on SU may be a consequence of the multiple copies of heap area across processors or threads, which is adopted in many allocators to minimize the effect of thread-contention when multiple threads are accessing the same heap. For most common scenarios (NP=1 or NP=4 combined with NT=1 or NT=4), the allocators that present the lowest average of SU are Ptmallocv3 followed by Jemalloc. In average, the same result was observed for all other test scenarios. The third best allocator regarding SU is Ptmallocv2, which shows an average SU of approximately 54.12% higher than Ptmallocv3 and Jemalloc.

4. ASYMPTOTIC ANALYSIS

In order to determine the functions that express the execution time of the evaluated allocation algorithms, the three most important factors, from the theoretical viewpoint, are the number of processors (NP), number of threads (NT), and number of allocations (NA). Given that NP is quite limited in regular computers, for this analysis we mainly considered NT and NA. All allocators' algorithms showed linear asymptotic complexity for execution time and they could be represented as O(NA + NT). However, NA usually is greater than NT, so we considered their complexity as O(NA).

5. CONCLUSIONS

In this work, we presented experimental and theoretical analyses of six memory allocation algorithms. In general, based on the experimental results and considering the common factor interactions identified for all allocators, we group them in the following clusters: (Hoard, Jemalloc), (Ptmallocv2, Ptmallocv3, TCMalloc), and (Miser), where the order does not mean importance. In terms of execution time (ET), the best results were obtained with one processor, and the worst results with four processors, indicating that for the evaluated allocators multiprocessing is not presenting benefits from the speedup viewpoint. The allocator that presented the average lowest execution time was Ptmallocv2, followed by TCMalloc, Jemalloc, Ptmallocv3, Miser, and Hoard. In terms of SU, the best allocators were Ptmallocv3 and Jemalloc, which presented the average of space used reduced by 50% compared to the other allocators; they are followed by Ptmallocv2, Miser, Hoard, and TCMalloc.

6. REFERENCES

[1] Attardi, J., and Nadgir, N. *A Comparison of Memory Allocators in Multiprocessors*, http://developers.sun.com/solaris/articles/multiproc/multiproc.html, 2003.

[2] Barootkoob, G., Sharifi, M., Khaneghah, E. M., and Mirtaheri, S. L. Parameters Affecting the Functionality of Memory Allocators, In *IEEE Conference on Communication Software and Networks* (Xi'an, May 27-29, 2011). 2011.

[3] Berger, E.D., Zorn B., and McKinley, K.S. Reconsidering Custom Memory Allocation. In *Proceedings. of the Conference on Object-Oriented Programming: Systems, Languages and Applications* (Washington, Nov. 4-8, 2002). ACM SIGPLAN Notices, New York, 2002.

[4] Berger, E.D., McKinley, K.S., Blumofe, R.D., and Wilson, P.R. *Hoard: a scalable memory allocator for multithreaded applications*, ACM SIGARCH Computer Architecture News, v.28:5, 2000, 117-128.

[5] Evans, J. A scalable concurrent malloc() implementation for FreeBSD. In *Proceedings of the The BSD Conference* (Ottawa, May 12-13, 2006). 2006.

[6] Ferreira, T. B., Matias, R. J., Macedo, A., and Araujo, L. B. An Experimental Study on Memory Allocators in Multicore and Multithreaded Applications. In *Proceedings of International Conference on Parallel and Distributed Computing, Applications and Technologies* (Gwangju, Oct. 20-22, 2011). 2011, 92-98.

[7] Ghemawat, S., and Menage, P. *TCMalloc: Thread-Caching Malloc*, http://goog-perftools.sourceforge.net/doc/tcmalloc.html

[8] Gloger, W. *Ptmalloc*, http://www.malloc.de/en/

[9] Masmano, M., Ripoll, I., and Crespo, A. A comparison of memory allocators for real-time applications. In *Proceedings of 4th Int'l workshop on Java technologies for real-time and embedded systems* (Paris, Oct. 11-13, 2006). 2006, 68-76.

[10] Montgomery, D. C. *Design and Analysis of Experiments*. John Wiley, 3rd edition, 2000.

[11] Vahalia, U. *UNIX Internals: The New Frontiers*, Prentice Hall, 1995.

[12] Tannenbaum, T. *Miser: A dynamically loadable memory allocator for multithreaded applications* http://software.intel.com/en-us/articles/miser-a-dynamically-loadable-memory-allocator-for-multi-threaded-applications