Garbage Collection Independent Study Reflection

Alexander Kyte

What did I do?

I spent the start of this independent study covering the text. I took copious notes on the many chapters, and augmented my understanding through examination of the algorithms as applied in a number of production-quality runtimes.

These initial weeks contained two assignments as well. The first one had me design a simple mark-sweep collector inside of a teaching platform made in Racket. The second one had me take this platform beyond the initial design and create a multi-survivor-space generational garbage collector. Both required an understanding of the collection algorithms.

Due to Racket's weaker typing system, a number of bugs had to be diagnosed at runtime through examining the way that the collectors misbehaved and tracking this back to collector behavior. I believe that this was a good immersion into black-box debugging of runtimes. Debugging collectors is inherently difficult because it tends to take a while to observe the program going awry. Inserting print statements or breakpoints would be too noisy, as the allocation function is called very frequently.

Following the completion of the textbook, I proceeded to read the influential white papers published recently while selecting the platform for my closer analysis. I chose the OpenJDK HotSpot JVM and focused on the new concurrent Shenandoah collector that they are implementing. This was contrasted by a study of another JVM, the Jikes RVM, which is used as a reference runtime in many academic papers to experimentally observe the performance of a new algorithm.

At this point I was ahead of schedule, I had to find a project which was not entirely out of the time scope but still offered potential to be technically interesting. During this process, I looked into the research on heap sizing in order to discuss ways in which MLton might better manage the resizing of it's heap. After a few ideas, I settled on a project.

A previous student, Russell Harmon, had used the idea of using debugging information for runtime introspection of C programs. I saw the potential to use this to collect type information for a C garbage collector. There is currently a collector, called the Boehm collector, which provides conservative collection for C. This collector is used by many fledgeling languages, but has the failure point of not being able to move data. This leads to an inability to shrink the heap, and high heap fragmentation rates. Long-lived, quickly-allocating programs with very variably-sized objects are a pathological worst case which is not uncommon to encounter.

Researching the Boehm collector, Russell's thesis, and the criticisms of current conservative collectors, I sought to write an extension to the Boehm collector. This reading was interleaved with implementation of a root-scanning infrastructure which used the DWARF debugging symbols format. After nearing completion for this, I came to realized that Boehm made many optimizations of their mark-sweep collector that would be awkward to disable for compaction. Also, their support for tracking types involved a separate data structure external to the heap which appeared to add a cost to scanning linear with respect to the number of typed objects. A nearly-fully-typed heap would thus perform poorly on the Boehm collector.

I have recently began work on an original partially conservative collector. We know the types of most variables, but C's void * doesn't tell us what is being pointed to. If this pointed-to object contains a byte which might be a reference into the heap, then we must pin what that points to. It's a novel problem, to allocate data without knowing if it will suddenly become non-moving. This necessitated the mark-compact collection strategy which I am currently working on. I hoped to make this incremental, since mark-compact algorithms allow us to determine the future location of all live data at the time of marking.

I would estimate that I've spent on average between 5 and 7 hours per week on goals part of this independent study.

What did I learn besides GC?

I believe that this independent study has given me skills which I will make use of for the rest of my academic and professional career.

I have become familiar with the structure of white papers and I have reached a high level of literacy in the memory-management field. I can, and do, select papers from ACM and from conferences to read without having to worry about a lack of comprehension. I am familiar enough with the names and contributions of important papers in the subject that I can recognize and name the use of patterns in novel management strategies. My class blog has notes on papers ranging from use of GPUs for garbage collection all the way through the use of control theory for heap sizing. I can understand benchmarks and evaluate new techniques based on their many different kinds of costs and benefits.

Lastly, I have become better at navigating large language codebases. My time spent looking at the application of algorithms in the go and python runtimes, in MLton, in Jikes, and in HotSpot has made me much more comfortable navigating and modifying large codebases.

What do I have to show for it?

These skill became indispensable during the recent hiring season. I interviewed at a number of places, and recently accepted an offer working on the Mono(C#) runtime at Xamarin this Spring and Summer. During my interviews with Google, Mozilla, and Xamarin(all programming language internships) I was extensively pressed to make alterations to major

collector frameworks to show an understanding of the systems. I was repeatedly told that I was incredibly fast at the tasks. Since I had never previously looked into the runtimes I was asked to work on, I will credit this to my recently-earned familiarity with the general structure of a garbage collector/runtime.

I also have a prolific class blog hosted at akyte.blogspot.com, which has received around 600 views. This blog contains my notes for my textbook readings and whitepapers. It contains a fairly significant amount of text, and will continue to be a good resource for more general GC algorithm reference.

Lastly, I have my two toy Racket collectors to look back on as well as the collector which I am currently working on. I hope to invest the time to bring my current collector to maturity, with the hope that it may be used as a replacement for the Boehm collector and that it will be an option for those language projects unable or unwilling to write their own collectors.

Schedule

Week	Summary
1	Readings of Garbage Collection Handbook up through Chapter 4.
2	Readings of Garbage Collection Handbook up through Chapter 8.
3	Work on simple mark-sweep GC in Racket
4	Readings of Garbage Collection Handbook up through Chapter 12.
5	Start Generational GC, readings of GC Handbook up through Chapter 19.
6	Study of HotSpot JVM's collector, along with Jikes RVM (papers, source, presentations, etc)
7	Selected Readings, search for project. Focus on heap sizing
8	Selected Readings(ISMM, POPL, new/exciting papers)

9	Present project proposals, chose DWARF-based root scanner for BDW collector
10	Selected Readings(Conservative collectors, BDW collector)
11	Selected Readings, Work on root scanner
12	Selected Readings, Work on root scanner
13	Work on root scanner
14	Work on root scanner
15	Selected Readings, Decision to write custom Mark/Compact collector/not use BDW to best capitalize on precise roots
16	Readings on more advanced, concurrent mark-compact algorithms. Architecting of collector.

References

Text:

R. Jones, A. Hosking, and E. Moss. The Garbage Collection Handbook: The Art of Automatic Memory Management. Chapman & Hall/CRC, 1st edition, 2011.

Papers:

David Detlefs, Christine Flood, Steve Heller, and Tony Printezis. 2004. Garbage-first garbage collection. InProceedings of the 4th international symposium on Memory management (ISMM '04). ACM, New York, NY, USA, 37-48. DOI=10.1145/1029873.1029879 http://doi.acm.org/10.1145/1029873.1029879

David R. White, Jeremy Singer, Jonathan M. Aitken, and Richard E. Jones. 2013. Control theory for principled heap sizing. SIGPLAN Not. 48, 11 (June 2013), 27-38. DOI=10.1145/2555670.2466481 http://doi.acm.org/10.1145/2555670.2466481

David Terei, Alex Aiken, and Jan Vitek. 2014. M3: high-performance memory management from off-the-shelf components. In Proceedings of the 2014 international symposium on Memory management (ISMM '14). ACM, New York, NY, USA, 3-13. DOI=10.1145/2602988.2602995 http://doi.acm.org/10.1145/2602988.2602995

David F. Bacon, Perry Cheng, and Sunil Shukla. 2012. And then there were none: a stall-free real-time garbage collector for reconfigurable hardware. In Proceedings of the 33rd ACM SIGPLAN conference on Programming Language Design and Implementation (PLDI '12). ACM, New York, NY, USA, 23-34. DOI=10.1145/2254064.2254068 http://doi.acm.org/10.1145/2254064.2254068

Emery D. Berger and Benjamin G. Zorn. 2006. DieHard: probabilistic memory safety for unsafe languages. InProceedings of the 2006 ACM SIGPLAN conference on Programming language design and implementation(PLDI '06). ACM, New York, NY, USA, 158-168. DOI=10.1145/1133981.1134000 http://doi.acm.org/10.1145/1133981.1134000

E. P. Wentworth. 1990. Pitfalls of conservation garbage collection. Softw. Pract. Exper. 20, 7 (July 1990), 719-727. DOI=10.1002/spe.4380200707 http://dx.doi.org/10.1002/spe.4380200707

Filip Pizlo, Lukasz Ziarek, Petr Maj, Antony L. Hosking, Ethan Blanton, and Jan Vitek. 2010. Schism: fragmentation-tolerant real-time garbage collection. In Proceedings of the 2010 ACM SIGPLAN conference on Programming language design and implementation (PLDI '10). ACM, New York, NY, USA, 146-159. DOI=10.1145/1806596.1806615 http://doi.acm.org/10.1145/1806596.1806615

Grosser, T., Zheng, H., A, R., Simbürger, A., Größsslinger, A. and Pouchet, L.-N. 2011. Polly - Polyhedral optimization in LLVM. First International Workshop on Polyhedral Compilation Techniques (IMPACT'11) (Chamonix, France, Apr. 2011).
KC Sivaramakrishnan, Lukasz Ziarek, and Suresh Jagannathan. 2012. Eliminating read barriers through procrastination and cleanliness. SIGPLAN Not. 47, 11 (June 2012), 49-60. DOI=10.1145/2426642.2259005 http://doi.acm.org/10.1145/2426642.2259005

Hans-Juergen Boehm. 1993. Space efficient conservative garbage collection. In Proceedings of the ACM SIGPLAN 1993 conference on Programming language design and implementation (PLDI '93), Robert Cartwright (Ed.). ACM, New York, NY, USA, 197-206. DOI=10.1145/155090.155109 http://doi.acm.org/10.1145/155090.155109

Martin Maas, Philip Reames, Jeffrey Morlan, Krste Asanović, Anthony D. Joseph, and John Kubiatowicz. 2012. GPUs as an opportunity for offloading garbage collection. In Proceedings of the 2012 international symposium on Memory Management (ISMM '12). ACM, New York, NY, USA, 25-36. DOI=10.1145/2258996.2259002 http://doi.acm.org/10.1145/2258996.2259002

Nicolas Geoffray, Gaël Thomas, Julia Lawall, Gilles Muller, and Bertil Folliot. 2010. VMKit: a substrate for managed runtime environments. SIGPLAN Not. 45, 7 (March 2010), 51-62. DOI=10.1145/1837854.1736006 http://doi.acm.org/10.1145/1837854.1736006

Y.C. Tay and X.R. Zong. 2010. A page fault equation for dynamic heap sizing. In Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering (WOSP/SIPEW '10). ACM, New York, NY, USA, 201-206. DOI=10.1145/1712605.1712636 http://doi.acm.org/10.1145/1712605.1712636

Resources

Blog/Notes: akyte-gc.blogspot.com
Project: https://github.com/alexanderkyte/Bug-Collector