ACM *Sigplan Notices* **28**,4 (April 1993), 55-57.

'Infant Mortality'[1] and Generational Garbage Collection

Henry G. Baker

Nimble Computer Corporation, 16231 Meadow Ridge Way, Encino, CA 91436 (818) 986-1436 (818) 986-1360 (FAX)

Generation-based garbage collection has been advocated by appealing to the intuitive but vague notion that "young objects are more likely to die than old objects". The intuition is, that if a generation-based garbage collection scheme focuses its effort on scanning recently created objects, then its scanning efforts will pay off more in the form of more recovered garbage, than if it scanned older objects. In this note, we show a counterexample of a system in which "infant mortality" is as high as you please, but for which generational garbage collection is ineffective for improving the average mark/cons ratio. Other benefits, such as better locality and a smaller number of large delays, may still make generational garbage collection attractive for such a system, however.

Introduction

In the decade since generational garbage collection was first proposed in print [Lieberman and Hewitt 83], a number of papers (including some of my own) have introduced their discussions of generational garbage collection by statements such as "between 80 and 98 percent of all newly-allocated objects die within a few million instructions", or "most newly created cells die young". While this intuition is sometimes backed up by measurements which "prove" the effectiveness of a particular generational scheme, there are very few theoretical models by which we can reasonably compare one generational scheme with another, or with non-generational schemes.

We show via a counterexample that the quoted statements above are essentially meaningless, because a meaningful discussion of age-based garbage collection schemes requires the presentation of significantly more data in the form of complete object lifetime probability density functions, rather than the one or two data points usually proffered.

Generational Garbage Collection

Traditional non-generational tracing garbage collection schemes require time for a single collection which is proportional to the number of cells still in use, with an additional (usually negligible) term proportional to the total amount of storage under management. Depending upon the time between such "full" garbage collections, one may allocate a large number of cells and then recover a large number of garbage cells at the end of the collection. Thus, to a first-order approximation, the cost of this traditional garbage collection scheme per cell allocated is proportional to the "mark/cons" ratio—the more marking we perform per cell allocated, the lower the fraction of execution time will be spent marking, but the larger the total amount of space will be required to support N live cells.

Generational garbage collection schemes attempt to take advantage of the intuition that "most cells that die, die young" by separating the objects into different categories based on their age; these categories are called *generations*. If the intuition is correct, and if most tracing/marking can be focussed on the younger generations, then the categorical schemes can achieve a lower average amount of tracing/marking work per object allocated and thereby achieve a net performance advantage over a non-categorical collector [DeTreville77]. In addition to these first-order savings, one can also achieve second-order savings of increased locality in the memory hierarchy because the more localized tracing work is less likely to disrupt the residence of live objects in the faster memories/caches.

To summarize, generational garbage collection has been advocated for at least three reasons:

- generational garbage collection can reduce the mark/cons ratio in some applications
- generational GC can improve the locality of reference over a non-generational GC
- generational GC has many small pauses instead of a few large pauses

[1]Can't computer scientists come up with less grisly names for these concepts?

"Radioactive Decay" Model of Object Lifetimes

Let us consider an application whose objects die through a process analogous to the radioactive decay process found in unstable physical isotopes. (An application with these characteristics may not be prototypical of garbage-collected applications, but it does provide an ideal mathematical model which can represent one end of a spectrum.) In such a process, the probability of any particular object decaying within a particular period of time is related to the length of the period, but not to the particular particle, or to the existing age of the particle. In particular, such a process is oblivious to the age already achieved by a particle, but depends solely upon whether it has already "decayed". Such a decay process has but one parameter—its "half life" τ, which is the period of time during which half of its particles can be expected to decay, and therefore the number of particles likely to decay in any short period of time is proportional to the number of undecayed particles still left. Thus, if N_0 is the number of particles at t=0, then the expected number of particles still undecayed at any time t>0 is a simple negative exponential function $N_0 \bullet 2^{-t/\tau}$.

Now consider such an application in which new particles are continually being added/created at the same rate that particles are decaying. The average number of particles is then constant over time.[2] In this situation, the same propensity of an object to die independent of its achieved lifetime still holds. In such a radioactive-like system, one will find it true that "most objects die young", and for any *single* measurement of the form "only 2% of the objects survived 1 second", it is trivial to compute a "half life" consistent with this observation—i.e., τ=177 msec.

However, if one attempts to utilize a generational garbage collector for this application, one finds that no matter how the generations are chosen, the decay rate for each generation is proportional to the size of the generation, not to its age, and therefore, the average mark/cons ratio for each generation is the same. For such an application, a generational garbage collector cannot improve the average mark/cons ratio, and the additional overheads of generational collection may cause the total amount of work to increase relative to a non-generational collector. On the other hand, a generational garbage collector may show increased locality of reference, and/or better average response, due to second-order effects. Thus, it is very important for papers which provide measurements of garbage collection algorithms to document the mark/cons ratios, localities and other important parameters, in addition to total runtime and/or GC load.

Conclusions

Generational garbage collection can often, within a given amount of storage, 1) reduce the average mark/cons ratio, 2) improve the locality of reference, and 3) trade a small number of long pauses for many short pauses. While the locality of reference improvement and the improved response time are available across a wide range of applications, a decreased load due to a reduced mark/cons time is less intuitively realized. The statistics must be very far from equilibrium, the overheads of the generational scheme have to be very small, and the predictions of the generational scheme have to be extremely good in order to produce any mark/cons benefit relative to a non-generational collector. Furthermore, the vague appeals to intuition in the form "most objects die quickly" can be shown to apply equally well to processes in which all generations produce exactly the same mark/cons ratios.

In the newest generation of microprocessors which incorporate a significant amount of cache memory, and for which a cache "miss" may cost more than 100 instructions, the "second order" locality effects of generational garbage collection may be more important than the "first order" effects of an improved mark/cons ratio. For example, our stack-based "lazy allocation" generational scheme [Baker92] is highly integrated with the programming language implementation, and shares

---

[2]We consider the simple situation in which a new particle is added/created *exactly* when one decays. The more likely situation in which only the *average rates* are equal leads to significant deviations from the constant number of live particles and to second-order effects, although the first-order effects are the same.

most of its work with the existing programming language parameter-passing and value-returning mechanisms, and therefore avoids gratuitous non-local behavior.

Our personal belief is that the object lifetimes found in garbage collection measurements cannot be adequately modelled by traditional probability/statistics models which assume that distributions have finite "standard deviations", but will require *fractal* models [Mandelbrot83], which can have infinite second moments.

Acknowlegements

Many thanks to the participants of the International Workshop on Memory Management '92 [Bekkers92] for their interesting and inspirational discussions.

REFERENCES

Appel, A.W. "Simple Generational Garbage Collection and Fast Allocation". *Soft. Prac. & Exper. 19*,2 (Feb. 1989), 171-183.

Baker, H.G. "List Processing in Real Time on a Serial Computer". *CACM 21*,4 (April 1978), 280-294.

Baker, H.G. "CONS Should not CONS its Arguments, or, a Lazy Alloc is a Smart Alloc". ACM *Sigplan Not. 27*,3 (March 1992), 24-34.

Bekkers, Y., and Cohen, J. *eds. Memory Management: Proceedings of the International Workshop on Memory Management*, St. Malo, France. Springer LNCS 637, 1992.

Clark, D.W., and Green, C.C. "An Empirical Study of List Structure in LISP". *CACM 20*,2 (Feb. 1977),78-87.

DeTreville, John. "Reducing the Cost of Garbage Collection". Unpublished manuscript, May, 1977.

Lieberman, H., and Hewitt, C. "A Real-Time Garbage Collector Based on the Lifetimes of Objects". *CACM 26*,6 (June 1983), 419-429.

Mandelbrot, B. *The Fractal Geometry of Nature*. W.H.Freeman & Co., New York, 1983.

Moon, D. "Garbage Collection in a Large Lisp System". *ACM Symp. on Lisp and Functional Prog.*, Austin, TX, 1984, 235-246.

Unger, D. "Generation Scavenging: A non-disruptive, high performance storage reclamation algorithm". *ACM Soft. Eng. Symp. on Prac. Software Dev. Envs., Sigplan Not. 19*,6 (June 1984), 157-167.

Wilson, Paul R. "Some Issues and Strategies in Heap Management and Memory Hierarchies". *ACM Sigplan Not. 26*,3 (March 1991), 45-52.

