Approaches to Adaptive Tenuring in GenShen

DRAFT

The performance of generational collectors is quite sensitive to the choice of which objects are tenured and which are kept in the young generation. In the early 80's, David Ungar at UC Berkeley and, independently, Henry Lieberman & Carl Hewitt at MIT proposed both the generational hypothesis underlying generational collection, as well as an adaptive tenuring mechanism. HotSpot as well as many older and later runtimes have used generational garbage collection, with the age of objects as the discriminating feature to adaptively make tenuring decisions. There have been subsequent attempts to use other features (e.g. object types, object connectivity, etc., most notably from IBM and Microsoft, and from McKinley, Blackburn, et al.) but none have really matched the elegance, simplicity, and performance of age-based generational collection. Darko Stefanovic at UMass Amherst studied several programs and applications in 1999 and concluded that age-based generational collection is indeed a powerful technique. He found that it captured several properties commonly true of most programs written in object-oriented languages such as Java: pointer mutation rates are highest in young objects, pointer affinity is greatest amongst age cohorts, and most very young objects aren't mortal but slightly older ones are with mortality falling rapidly thereafter. These observations have been independently rediscovered over time by anyone working with object oriented applications and language runtimes, but trying to ignore this generational, age-based nature of object demographics.

What we can conclude from experience and the studies mentioned above is that object lifetime distribution typically shows a peak or mode close to creation time, and further but much smaller modes at larger lifetimes, with the actual shape of the remainder of the distribution past the first peak typically dependent on the specifics of the application and its workload characteristics.

Estimating the knee in the age mortality curve

Given this, typical adaptations of tenuring age must try and accommodate the volume under the first peak, provided its volume isn't very large, in the young generation and relegate the remainder to collection by the older generation collector. Our attempt is to identify the flattening of the probability distribution function (PDF) of mortality beyond the first mode and to do it as close to that knee (or the bend in the hockey stick) as possible. We must also be prepared for the distribution to be somewhat non-stationary, so that the knee shifts around a bit as the application or its environment changes over time. Looking at the cumulative distribution function (CDF) for that PDF, we are looking to capture the flattening of the CDF towards the right after its initial sharp rise. Intuitively, we can think of PDF as the probability that an arbitrary object dies *at* a certain time-age. The CDF then is the probability that an object has died *by* a certain time-age. Alternatively, when looking at population statistics of an age-cohort, one can think of CDF as representing the *fraction of the cohort* that have died *by* a certain age. The *complement of this fraction* then represents the fraction of the cohort that is still *surviving past* a certain age. Since GC can measure the population of survivors and estimate mortality following the identification of survivors as a cohort ages, this will be used by GC to determine the point at which it makes sense to tenure the survivors in a cohort based on measured mortality rates. Note that we are always talking about a cohort, in other words, we are aiming to do a *longitudinal analysis* of each cohort of objects born in an epoch together and determine an optimal tenuring threshold for that cohort. This recognizes the cost of gathering census data, which can only happen at each GC, to keep costs of such censuses low. The emphasis on cohorts also recognizes that object demographics change over time with each subsequent cohort, so that the optimal tenuring threshold must also adjust in response to demographic and environmental changes.

Mechanisms & Policies

As is traditional, we record the age of each object in its header and increment its age at each epoch where it survives after each minor collection. Then, if we track the mortality rate of each age cohort as it ages, we can tell when it peaks and when it drops below some threshold after which it is more beneficial to tenure it rather than continue copying the objects in that cohort in the young generation; let us call this the *cohort tenuring age.* We can thus define a tenuring age for each cohort. At

any time, we will have in the young generation several different age cohorts. In the case of Generational Shenandoah, these may or may not be clustered in a few regions, although members of the age 0 cohort are usually to be found mostly in Eden regions. Survivor regions will often have several different age cohorts. In the case of GenZGC, on the other hand, zPages are used to segregate objects by both size and age, so object headers do not have to record their age, but rather zPages encode the age of objects on that page. We will revisit this later in this document.

By now, the most natural tenuring threshold computation algorithm will have suggested itself:

- Track the population of each cohort longitudinally at each inter-GC epoch.
- The drop in the population of the cohort at each census tells us its mortality across that age-epoch.
- We expect from the generational hypothesis that the mortality is highest at early age-epochs, and then drops over time.The speed and shape of that drop depends often on the application/service and ambient conditions. Tracking the mortality rate and capturing the point at which the mortality rate falls below some threshold would yield a reasonable criterion for tenuring that cohort.
- Cohorts of course come one after the other and, in the above formulation, each would have its own tenuring age. We recognize that this may lead to situations where a younger age cohort has reached tenuring age at a specific point in time while an older cohort may not have. We expect this to be rare, but it's possible when the service or its environment change rapidly or go through phased behavior. In this case, we will err in the direction of *not tenuring the younger cohort until the older one is ready for tenure*. (The astute reader will recognize the similarity to old-fashioned bureaucratic promotion policies, where the objective is to save money by delaying promotions.) The idea is that we want to err in favor of increasing minor collector copying cost and increasing memory usage density in the old generation. We want to collect the old generation quite infrequently and we do not want premature promotions to tie up memory with objects that may soon be dead. We can call this the "memory-cost-primal strategy". Alternatively, it is possible to use a "cpu-cost-primal strategy" where we are willing to increase memory usage in favor of reducing the cost of minor collections, but in our experience, premature promotion of objects typically increases both memory cost as well as cpu cost, through a process of nepotism where younger promoted objects keep older objects alive and cause increases in promotion rates. In our implementation, we could make this a non-default option (to be implemented) in the event that users want to play with the alternative strategy. In the default memory-cost-primal strategy, once the older cohort is ready for tenure, all contiguous younger cohorts already at tenuring age will also become eligible, so the tenuring age will drop by a few units. However, we typically expect the changes in tenuring age to be smoother in practice from one epoch to the next, even under this policy. (A per-cohort tenuring policy as envisaged in the alternative implementation, makes for a slightly more expensive per object tenuring decision.)
- Of course, once the tenuring age for a younger cohort is larger than that of an older cohort, the tenuring age will continue to increase until the former cohort reaches its "natural" tenuring age.

We still need to determine if in practice there are sudden and large variations in tenuring threshold as determined by the default memory-cost-primal strategy in the face of sudden variations in load or load spikes. Assuming that our expectation of small slow changes holds in practice, we can simplify the work done by minor GC by computing a single adaptive tenuring threshold to use uniformly for all cohorts during a specific young collection, thus simplifying the tenuring algorithm, rather than have a tenuring age vector indexed by cohort, which would make the promotion loop unnecessarily heavy by having to look up a tenuring age indexed by the age cohort of the object being evacuated. We shall share some measurements of this in a subsequent section, using benchmarks such as SpecJBB, Renaissance, and Extremem.

Some scenarios

To think through how we might take multiple cohort tenuring ages and condense them into a single optimal age, let us consider the following three temporal dynamics:

1. **Rising (spiking) load**: In this case, we typically will expect an increase in the time processing this load as well as a reduction in the time between minor GCs (assuming a fixed Eden to simplify our reasoning). This will cause an

increase in the survivor rates (or conversely a reduction in the mortality of this cohort compared with older cohorts). A reasonable reaction in this case would be to increase the tenuring age for this cohort, while leaving the tenuring age of older cohorts the same. This requires us to temporarily increase the size of the younger generation's survivor space and to copy more objects during minor GCs. In other words, this increase in mortality should result in an increase in tenuring age down the line and an increase in survivor size until this cohort ages out.

2. **Steady load**: In this case, without loss of generality, we can assume that all cohorts have the same tenuring age, and no changes are needed either in tenuring age or in the size of Eden's survivor size.
3. **Falling (dropping after a spike) load**: This is the converse of (1) above, where objects are potentially processed faster. This assumption should be taken with the appropriate dose of salt, since in a microservices architecture of servers communicating across the network, it is possible that there is a floor in the processing latency or a ceiling in the processing throughput (see (2) above). However, because there is a drop in the allocation rate, we expect the time between collections to increase, which will result in an increase in mortality rates (or conversely a decrease in survivor rates). We might potentially want to reduce our tenuring age in those circumstances because we wouldn't want to unnecessarily copy longer-lived objects in young collections. However, this is less of an issue because we are copying fewer objects anyway so copying a few older age objects may not affect performance too adversely. These objects, even if they were to reach tenuring age sooner than older cohorts of the type described in (1), would be held longer in the survivor space before they are promoted.

Now most traditional generational collectors, including Ungar's generational scavenger, compute the tenuring threshold not based on object mortality rates as we are proposing above, but rather based on the size of the survivor space. In other words, assume that a different perhaps independent mechanism is driving the sizing of the survivor space which is a given fixed value for the minor garbage collector every time it runs. In this case, the adaptive age computation will keep all those objects in the younger generation that will fit in the survivor space without risking overflow in the face of a sudden spike in allocation in the next cycle. Headroom is maintained in the form of a fractional padding, designating the size & duration of the load spike, to accommodate the population surviving such spikes. Should that happen, we may still be able to increase the size of the survivor space, when that is possible (e.g. in the case of G1), or drop the tenuring age so that younger objects are never prematurely promoted before their older counterparts. Because we do not distinguish objects in the same cohort, we want an entire cohort to tenure together or to be aged in the survivor space together. For survivor spikes that exceed this headroom, if we are using a stop-trace-and-copy collector such as Parallel GC's young collector, which lacks G1's survivor space elasticity, we will inevitably prematurely promote some quite young objects through a process of *tenuring inversion* because we ran out of space in a fixed size survivor space with no space left to copy some younger objects, while older objects that also may not have reached tenuring age might have been kept in the survivor space: we are left with older objects in the survivor space with younger ones having to be promoted, when the reverse would have been more desirable. In the case of GenShen, however, we know *exactly* what objects are live at the end of the marking step and their ages, so we can determine what the tenuring age should be to completely avoid overflow. Our use of "completely avoid" should be taken with the appropriate grain of salt — overflow and tenuring inversion could still happen in a heap constrained situation because of fragmentation from bin-packing and parallel allocation. Thus, the counterpart to Ungar's adaptive tenuring algorithm for Generation Shenandoah is to compute the tenuring age after the marking is complete and we know what is surviving, and we know what our survivor space budget is. However, this merely pushes the question of tenuring age to what the survivor space budget should be. In the absence of memory pressure, this can be based on object survivor rates, tenuring an entire cohort at its ideal tenuring age as discussed previously. However, when there is memory pressure, so that survivor space size cannot be increased, we could drop the tenuring threshold so as to reduce the chances of overflow and tenuring inversions.

Complications in conducting a concurrent census

Above, we have somewhat optimistically and incorrectly stated that:

> *In the case of GenShen, however, we know exactly what objects are live at the end of the marking step* ***and their ages****, ...*

This is not true today because GenShen marking is concurrent and would need to take the age census during marking. There is also the question of the size of the cohort that is live because of SATB, having been allocated after marking

started, but this is easy to infer from TAMS information — anything above TAMS is part of the age 0 cohort of that epoch. While finding the age of a live object is normally straightforward for stop-world collectors, monitor locking can throw a spanner into the mix as it can concurrently change the header word that holds the object age for concurrent marking generational collectors such as GenShen and GenZGC. As we mentioned in passing above, GenZGC has cleverly managed to avoid this issue altogether by keeping cohorts segregated by age associating age with these segregated zPages, rather than maintaining it individually in each object. GenShen does something similar, assigning ages to regions, although ages aren't *strictly* monotonic by epoch, presumably because of being frugal with using the free suffix of a partially allocated region when evacuation starts. (This is discussed further in the section on "Regional Matters" below.) However, currently, GenShen does also use the age in the object's header to determine if it should be tenured. It does so by obtaining the age during the evacuation phase after the evacuating thread (which is either a GC worker thread or a mutator) has claimed the exclusive rights to evacuate an object. At this point, all other threads and, a fortiori, the mutators have been locked out from modifying the header word of the claimed object and the evacuating thread can now determine the age either from the header if the object isn't locked or from the displaced header if the object is locked.

For simplicity, we shall for now stay with this arrangement, although in the future we may try to see if one may safely and efficiently extract the age during marking time. At the current time, then, the tenuring age will be based on an age census conducted at *evacuation time of the previous cycle* and may not reflect the reality of cohort populations at the current marking cycle. This however is not fatal — after all, all the existing (stop-world generational) GCs in HotSpot have relied on such census data that is one cycle old to drive tenuring decisions in the current GC cycle. Furthermore, with GenShen, we can simply keep regions or objects that have not reached tenuring age in the young generation or, if they have reached tenuring age, simply promote them in place. In other words, the tenuring age can be divorced from size calculations for the short-term, although knowing current object demographics and tenuring age could help with size budgeting decisions.

On the sufficiency of a myopic history

It is clear that if our only criterion for determining the tenuring threshold for a cohort were whether its mortality rate had fallen below a specific threshold, then all we need to do is keep two population vectors for each of the current epoch and the most recent previous epoch, and compute the mortality rate by taking the ratio of the populations of a cohort at the newer vs the older vector. However, we'd like to keep the older vectors simply to try and understand the population dynamics and explore if better tenuring strategies might be possible in the future as we gain more experience. With tenuring age having 16 values, we would need to keep track of 16 such vectors at most to have a good idea of the history of all the currently live cohorts. However, none but the two most recent are used in our current calculation of the tenuring age. We might decide at some point that we need even older population vectors, perhaps in the form of decaying averages or standard deviations, but will defer such investigations to the future.

Putting it all together

- Start with an initial value of tenuring threshold set at the user-specified value. A value of 7 is as good as any, but we might determine in our algorithm that a lower tenuring age is appropriate even before the initial tenuring age is ever hit. The user should be able to specify both a minimum and a maximum tenuring threshold within which to clamp the adaptively computed value so as to place guardrails in specific circumstances to avoid unstable or undesirable dynamics or oscillation.
- During the concurrent evacuation phase of the young collection, record object ages into a population vector *P* for each age cohort *c* of age *a*, *0 <= a <= 15*.
- Take the ratio of the population of cohort *c* of age *a* for this epoch e against the previous epoch e-*1* to determine the survival rate *S(c) = P(a, e)/P(a-1, e-1)*. We know that *P(a,e) < P(a-1, e-1),* so *S < 1.* The mortality rate *M* is its complement *M(c) = 1 - S(c)*. We now have a mortality rate vector that we can implicitly index by age *a,* for simplicity via identifying *c* with its cohort's age *a* during any specific epoch, and writing *M(a) = 1 - P(a, e)/P(a-1,e-1).* The default value of entries in the vector where the population vector has no entry (or the value 0) in the previous epoch is *0* (the minimum possible mortality rate, indicating eligibility for promotion).
- Walking over the mortality rate vector *M* in decreasing order of age from *a = 15* through *0,* we will then identify the

largest age *a** for which *M(a*) < R,* where *R* is a parameter to the algorithm (at this time) denoting the mortality rate below which we will consider the cohort population to be stable and long-lived, and thus eligible for tenuring. Other tenuring criteria are possible, as mentioned above and in the associated SIM ticket by Kelvin Nilsen, and will be investigated in the future.

- The value *a** identified above at the end of the evacuation phase in epoch *e* then is used to make tenuring decisions at the evacuation in the next epoch *e+1.*
- To the extent possible, we will factor the implementation in such a manner as to allow further future experimentation along at least three aspects:
    - Decoupling the census from the tenuring calculation, thus allowing the census to potentially move to an earlier phase, such as marking.
    - Allow the plugging in of various different adaptive tenuring policies for the purposes of experimentation and research, and
    - Within each adaptive tenuring algorithm exposing enough knobs to allow experimentation with different parameters (such as *R* for the algorithm above).
    - The recording and logging of cohort population dynamics decoupled from the adaptive algorithm so as to allow data-driven tuning and research.

Dark matter delusion & actuarial anomalies

As described above, in its current implementation, Generational Shenandoah increments object ages during the evacuation phase when the evacuating thread gains exclusive access to do so. However, this means that surviving objects that are not in the evacuation set don't get counted in the census, effectively hiding a section of the population which, borrowing a term from physics, we shall call "dark matter". Note that dark matter might occur in any age cohort, but is most likely to be the very youngest. Such dark matter may not appear in a collection, and then may reappear in a subsequent collection when it becomes visible during evacuation. If the dark matter population of a cohort is always zero, the census is accurate and we will find that the population of a particular cohort will be a monotonically (although not necessarily strictly) decreasing function of the epoch ordinal. However, if a fraction of the cohort were to arbitrarily vanish in one epoch and reappear in another, one cannot determine an accurate mortality rate in such censuses. Thus, algorithms based on mortality rate can behave erratically as we try and draw conclusions from such inaccurate censuses. It is important therefore to work out a method that isn't subject to such errors, or to eliminate, or reduce the impact of, inaccurate censuses. If collector efficiency forces us to live with dark matter, one possibility might be to apportion the dark matter population (which should be easy to determine by adding the the live object size of young regions not included in the census) to each age cohort in proportion to their imputed populations in the previous epoch. We'll have more to share on this matter following ongoing investigations.

Regional Matters

GenShen, like Shenandoah and G1 (and in the spirit of ParallelGC's "dense prefix"), does not collect regions that may have little garbage as the cost of evacuation might, in such cases, exceed the yield in free memory. As mentioned above, this causes some noise in cohort mortality figures. Such dense regions might contain a combination of younger and older objects mixed together, but for the purposes of promotion in place, GenShen pessimistically identifies the region age with the *minimum possible age* of objects that might *potentially* be in such a region. Regions might then become eligible for promotion in place whenever all of the objects in the region have exceeded the tenuring threshold. Other choices are possible, but the current policy makes good sense because it avoids premature promotion of too young objects. The policy instead defers promotion of older objects that may otherwise have been tenured if considered individually.

This brings up two issues:

1. One question naturally arises whether objects in such regions should or should not contribute to the mortality statistics of their cohort, since they are effectively outside of the usual individual object promotion criteria. This is a

somewhat subtle issue, but we believe it would be better, for the purposes of understanding cohort mortality, to include these objects in the census if possible. As we noted in the previous section, there may be a few objects older than the current tenuring threshold which our algorithm already ignores. If, on the other hand, there are many such older objects, then they would contribute a higher marking cost to minor collections, but we do not pay any evacuation or reference update costs. As noted earlier, if age census were part of the marking cycle, then one would be able to get an accurate census of such objects in dense regions as well.

2. A more difficult issue is that of promotion of *some* objects exceeding the tenuring age, while leaving other objects from the same or older cohorts in the dense region because of identifying the age of the region with that of its youngest potential surviving member. The general belief in classic Java GC circles has been that younger objects are more likely to refer to older objects than vice-versa, and if this belief were true, this could lead to more pointers from the old generation (from individually promoted younger objects) to older objects in the younger generation that have not been tenured. This in turn contributes to these older not yet tenured objects being kept alive until at least the next old generation cycle. The region density in this case might be self-fulfilling. The classic Train algorithm promotes such objects into the referring train (i.e. the old generation here). In our case, the region will either eventually become sparse if most objects died soon thereafter, allowing the older objects to be considered for evacuation, or the region remains dense and is then promoted in place after a suitable number of minor collections that age the regions not included in the collection set. In either case, the objects eventually move to the older generation, but we have meanwhile paid the marking costs for these objects. Can we do better by assigning an age to the regions that comports more closely with the age of (the majority volume of) objects in that region? It is impractical to keep a population vector for each region, but can one compute one on the fly during the liveness computation of regions to do a better job here? The simpler alternatives are computing a maximum age or a minimum age, and assigning that to the region. The minimum age would be closest to the spirit of the current implementation and would be reluctant to prematurely promote objects. The maximum age would be the opposite in spirit and would be eager to prematurely promote some objects, and would seem to be the wrong approach. The majority age would be somewhere in between, and would be closest to the spirit of the tenuring algorithm while accepting that some objects may prematurely promote. Our thinking is that we tread cautiously and use the minimum age which would more precisely implement the current intent.

Based on (2) above, we will consider using the minimum age of objects in a region to determine its age. This will be possible only when we run the census during the marking phase.

Census noise: locked object headers or collection set exclusions

One natural question with respect to locked objects, where the header containing the age has been displaced, is whether one should ignore those objects entirely, or count such objects and attribute their total mass proportionately to all existing cohorts. Either of these might seem to be reasonable policies that might be implemented. Whether the noise in the census from locked objects during marking is higher than the noise from collection set exclusion during evacuation is a question we'll likely have to empirically evaluate across a range of applications to see what might make sense. Recall that one advantage of marking-time census is that it might potentially inform better space budgeting and allow a more recent judgement on tenuring threshold prior to evacuation. While it's easier to just ignore locked objects entirely, proportionate assignment of the object's volume into all cohorts would seem to be better in that it might lead to less noise on mortality rates.

Interaction with Generation Sizing

Two questions to consider are:

1. Can the adaptive tenuring calculation use generation size as input? This is easy to do and indeed the classical algorithm makes essential use of only this factor. It's easy to make use of it in the same manner in our tenuring threshold computation.
2. Can demographic census help inform generation sizing? Clearly, a computed tenuring threshold (modulo collection set exclusion) determines the occupancy of the young generation, which in turn influences generation size budgeting

considerations. But, can the more refined census data help make better sizing decisions by potentially projecting ahead to compute (under suitable steady state assumptions) future requirements for the current objects and future allocations? It seems that we can reasonably do so, but perhaps the details of the current sizing criteria already, perhaps implicitly, incorporate this information in some time- and age-averaged form.

Some empirical observations

The following observations are a work in progress and have not involved extensive data collection — they are mainly based on SPECjbb performance data

- The cohort mortality statistics are usually remarkably steady. This means that myopic histories that look at just the two most recent snapshots might suffice, and tenuring threshold computation based on mortality rates may not be too sensitive to longer histories.
- Small survivor populations (low population cohorts) are noisy and inject noise into mortality rate computations; they are best ignored for the purposes of tenuring threshold computation. It probably doesn't hurt to promote such objects, but it might be fine to let them stay in the young generation as well. (We need to reason about this and do an empirical evaluation.)
- (to be measured) age distributions in dense and sparse regions
- (to be measured) the effect of ignoring or proportionately dividing locked object populations into all extant cohorts

Performance Measurement Experiments

1. SPECJBB

2. EXTREMEM

3. RENAISSANCE

References

1. Ungar
2. Hewitt & Lieberman
3. Stefanovic
4. McKinley & Blackburn