Ruby master - Misc #15402

Shrinking excess retained memory of container types on promotion to uncollectible

12/11/2018 08:43 PM - methodmissing (Lourens Naudé)

Status:	Open	
Priority:	Normal	
Assignee:		

Description

I've been toying with the idea of the viability of attempting to reclaim over provisioned memory from buffer capacity of container objects like Array and String, effectively reducing the footprint of retained memory of such objects.

GC at the moment covers these dominant paths:

- Collection of shallow memory: unreferenced object slot with values encoded on the object
- Collection of retained memory: unreferenced object slot with off ruby object heap pointer to String buffer, Array buffer etc. (heap.aux)
- Finalization hooks like reclaiming resources for Tempfile for example

I explored in https://github.com/ruby/ruby/pull/2037 (more details and data points on the PR) a forth one:

- Shrinking over provisioned buffer capacity of Array (also applies to String and likely others) on promotion to uncollectible (Also garbage collect excess retained space from types with a first class capacity and buffer on promotion to uncollectible)

Sharing here for feedback in case anyone has ideas for a more appropriate hook, or additional precondition for such a hook. Or if excess buffer capacity can even be considered first class garbage in a GC context.

I chose Array as a proof of concept because the type already have this optimization through ary_shrink_capa through ary_make_shared and the threshold for not encoding members on the object is quite low at 3 elements. Plausible that many framework / boot specific long lived arrays are larger than that and because the growth factor on expansion is 2x, also likely a fair amount of over provisioned capacity.

Results of the changeset:

- Benchmark so_binary_trees - 26% reduction in total memory usage
- Also a very noticeable 24% difference with app_lc_fizzbuzz
- General few bytes reduction for almost all core benchmarks
- Mainline redmine after boot - 1.5% or 45kb reduction in Array retained memory size

Implementation caveats:

- Promotion to uncollectible may be a bad heuristic for shrinking buffer capacity.
- Needed to create a new rb_ary_shrink_capa function as ary_shrink_capa is private API and has several assertions (frozen and shared check) that hard fails during GC. That way shrinking responsibility and accounting remains the responsibility of array.c - the GC just calls it (same as with the memsize APIs)
- I tried running it during GC which worked fine for benchmarks like so_binary_trees but failed under GC stress and larger heaps because of TRY_WITH_GC via objspace_xrealloc, which can invoke GC
- A reasonable workaround for this was to use the postponed job API which is used by GC for object finalization, but that's one job per object space, not 1 per Array being shrinked, which may hit the 1000 item postponed job buffer for some heaps. It degrades gracefully though with fallback being the optimization simply not being applied to the excess objects in the set.
- Have no idea about the future of the postponed job API and if this is an appropriate use case
- RVALUE_PAGE_OLD_UNCOLLECTIBLE_SET only special cases Array at the moment - it's easy to support other types

Outliers to still evaluate:

- Fragmentation does not get significantly worse through reallocs for specific rare cases post GC (no data)
- The effect of objspace_malloc_increase called by objspace_xrealloc on GC frequency (I think not much given the small reduction on retained usage, but have no data to prove yet)
- How well the postponed job pattern scales to large heaps and how much of the job slots are consumed (no data)

Thoughts on exploring more types or is the pattern tainted / broken to begin with?