Ruby master - Feature #15667

Introduce malloc_trim(0) in full gc cycles

03/14/2019 08:34 PM - sam.saffron (Sam Saffron)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		

Description

Per Hongli's excellent article it looks like malloc_trim can help tremendously with memory bloat issues.

https://www.joyfulbikeshedding.com/blog/2019-03-14-what-causes-ruby-memory-bloat.html#a-magic-trick-trimming

I would like to get this patch tested side-by-side at Discourse, GitHub and Shopify. If it looks good I think this is both a great candidate for 2.7 and and 2.4,2.5,2.6 backports.

Will coordinate with Shopify and GitHub to see if we can get numbers posted here, I will run tests on a live Discourse instance over the next week and report numbers here.

Koichi, what are your thoughts, to me this looks like an incredibly safe patch, the amount of work added to major GCs is tiny compared to the potential benefit, walking all pages is a very cheap operation.

Related issues:

Related to Ruby master - Feature #14759: [PATCH] set M_ARENA_MAX for glibc ma...	**Open**

History

#1 - 03/14/2019 11:50 PM - sam.saffron (Sam Saffron)

Relevant code in glibc

https://github.com/bminor/glibc/blob/c2d8f0b704c2b828bcd8d517a2376c0240c73c09/malloc/malloc.c#L4448-L4539

https://github.com/bminor/glibc/blob/c2d8f0b704c2b828bcd8d517a2376c0240c73c09/malloc/malloc.c#L4805

#2 - 03/15/2019 07:04 AM - duerst (Martin Dürst)

- Related to Feature #14759: [PATCH] set M_ARENA_MAX for glibc malloc added

#3 - 03/19/2019 05:42 PM - carlos@redhat.com (Carlos O'Donell)

As a maintainer of the quoted glibc code I'd be really interested in the results of this work. Please share when ready.

#4 - 03/20/2019 01:40 AM - mame (Yusuke Endoh)

- File ruby_gc_malloc_trim.patch added

I created a patch.

> I would like to get this patch tested side-by-side at Discourse, GitHub and Shopify.

Could you test the patch attached?

#5 - 03/25/2019 12:52 AM - sam.saffron (Sam Saffron)

mame (Yusuke Endoh) / carlos (Carlos Sánchez), absolutely, I just need a few more days here, mounting this kind of test is not trivial even with docker containers.

#6 - 03/28/2019 01:52 AM - sam.saffron (Sam Saffron)

- File Screenshot_2019-03-28 Grafana - Compare Discourse Perf.png added

mame (Yusuke Endoh) / carlos (Carlos Sánchez) attached is a screenshot of side by side testing on live traffic patterns

containers run multiple ruby processes (a few unicorn workers and a sidekiq worker)

standard14 = Ruby 2.6.2 + jemalloc
standard14_a = Ruby 2.6.2 + glibc malloc
standard14_b = Ruby 2.6.2 + glibc malloc + patch

My conclusions from these graphs:

- Memory is clearly down with the patch
- 99th percentile performance is slightly impacted
- cpu is very slightly higher
- jemalloc still fairs better than glibc even after the patch

I think I would support a slightly amended patch that only does the trim once every say 10 minutes (maybe even in a background thread), happy to test that out as well.

That said... selfishly for Discourse this does not matter that much we will still stick with jemalloc cause memory is better and performance is better under jemalloc.

For the wider ruby community though a safer default is very appealing.

#7 - 03/29/2019 04:30 AM - bluz71 (Dennis B)

Thanks Sam, a very nice set of results.

Notice that 99th percentile Topic list was faster with the patch, whilst slower with Topic view. So I'm not sure we can say that the patch will always be slower on the worst runs.

Query, what is the version of jemalloc that you are using? One of the interesting observations in #14759 is the variance between jemalloc versions (say 3.6.0 vs 5.1.0).

#8 - 03/31/2019 01:52 AM - tessi (Philipp Tessenow)

FYI: For easier testing this idea, I just pushed a small gem to rubygems malloc_trim (https://github.com/tessi/malloc_trim). This gives access to malloc_trim to ruby land to let us play with it without the need to re-compile ruby and/or deploying a custom patched ruby.

With MallocTrim.enable_trimming, there is also a built in way to run malloc_trim(0) after every GC MARK (the most relevant internal event I found to hook into). It probably calls malloc_trim somewhat too often -- I'd be happy for suggestions to find a better hook.

In any way, doing a manual MallocTrim.trim after a GC.start (e.g. between two rails requests) is still possible with this gem.

#9 - 04/01/2019 04:05 AM - sam.saffron (Sam Saffron)

- File crash.png added

tessi (Philipp Tessenow)

My tests were with 3.6.0, I will do a side by side now that I have all the infrastructure of 5 vs 3.6 and even tcmalloc.

Nice to see that gem! Looking at my graphs I think best bang would just be to spin a thread that does trimming every 10-30 minutes or something, especially if you can release the GVL prior to calling it (provided this thing is thread safe, which carlos (Carlos Sánchez) should know)

#10 - 09/19/2019 08:07 AM - dosadnizub (Borna Novak)

I'd just like to point out that calling malloc_trim(0) according to article explaining the feature - https://www.joyfulbikeshedding.com/blog/2019-03-14-what-causes-ruby-memory-bloat.html - will not fix memory fragmentation - it'll just make the "used memory" number go down - the graph on https://www.joyfulbikeshedding.com/images/2019/os_heap_visualization_after_trim-b3e36496.png still means that an attempt to grab 15 pages of memory (~16kb) - by ruby or any other process - will fail even though there's 8MB of memory "free" causing out-of-memory errors on systems that are not even in swap.

There's good reason why modern software typically doesn't release memory back to the system but rather reuses the pages it has allocated previously - or am I missing something important?

Files

ruby_gc_malloc_trim.patch	1011 Bytes	03/20/2019	mame (Yusuke Endoh)
Screenshot_2019-03-28 Grafana - Compare Discourse Perf.png	530 KB	03/28/2019	sam.saffron (Sam Saffron)
crash.png	85.9 KB	04/01/2019	sam.saffron (Sam Saffron)