Ruby trunk - Feature #10917

Add GC.stat[:total_time] when GC profiling enabled

02/27/2015 11:19 PM - jasonrclark (Jason Clark)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		

Description

This patch includes a :total_time value in GC.stat as a Fixnum of microseconds equivalent to GC::Profiler.total_time. A non-zero value is only provided if GC profiling is enabled.

This avoids problems with GC::Profiler's API in the presence of multiple callers. Well-behaved clients of GC::Profiler are expected to call clear periodically to constrain memory usage of the profiling structures. However, this causes problems when multiple callers--unaware of each other--rely on the value of GC::Profiler.total_time being unaltered since their last clear.

Using a single value in GC.stat avoids this by providing a monotonically increasing count with every GC, not influenced by clear.

Considerations and Questions

- Because the individual GC times are potentially small, I tracked the total as a double and only convert to Fixnum when constructing GC.stat's return. Is there something better I should be doing there, in light of GC.stat's insistence that the stats returned be if type size_t?
- What should I do (if anything) to cope with potential overflows? If Ruby is on a platform where size_t is 32 bits, we'd hit an overflow after 1.2 hours worth of cumulative GC time.

Future Directions

Is there any possibility of moving the GC timing (not the deeper profiling data) outside GC::Profiler.enable's gating? I would love avoid clients needing to make code changes to track their GC time with my gem (newrelic_rpm).

As it stands invocation start times are held on the GC profiling structures, so it felt like a much bigger change to alter that, and I thought I'd pitch this simpler change first.

Any advice on whether that would be possible? Is timing too slow to do by default, or is the gating just an artifact of how it was implemented with GC::Profiler?

History

#1 - 03/08/2015 10:21 PM - ko1 (Koichi Sasada)

With incremental GC (incremental marking and lazy sweep) consume very short time for each step.
I'm not sure we should measure such short time. At least, there are several BAD effect (especially on virtual machine environment).

If you want to measure only marking time, I can accept. But I'm not sure is it enough.

#2 - 03/09/2015 03:43 PM - jasonrclark (Jason Clark)

That makes sense, Koichi. I wondered if timing the small steps by default would be too heavy.

By only measuring mark time, do you mean https://github.com/ruby/ruby/blob/trunk/gc.c#L5184-L5212? For my own curiosity, I'll probably see how those timings look. Even if they're not complete, they might be useful.

Also, I apologize for mixing a patch and a separate question into one issue. The patch doesn't actually add any timings--it's just a new GC.stat counter, still gated by GC::Profiler.enable. Does that seem reasonable as it avoids issues with multiple clients calling GC::Profiler#clear?

#3 - 04/24/2015 10:18 PM - jasonrclark (Jason Clark)

Any thoughts on this patch Koichi? It just adds a GC.stat counter, and doesn't move anything outside the current GC::Profiler gating.

Files

gc-total-time.patch	3.43 KB	02/27/2015	jasonrclark (Jason Clark)