

Ruby - Feature #7472

Add a mechanism to remove objects from the GC cycle

11/30/2012 08:47 AM - sam.saffron (Sam Saffron)

Status:	Rejected																			
Priority:	Normal																			
Assignee:																				
Target version:	2.0.0																			
Description																				
<p>For typical rails apps there is a huge largely static object graph in memory. Requests come in, some objects are added, and the GC runs.</p> <p>The blocking nature of the GC introduces a jitter that stops all execution.</p> <p>In our typicalish Rails application, we see approximately 500k objects in the object space, each request add about 150k objects. On our production machine this means the GC stops execution for 30-40ms on a fairly beefy box:</p> <p>GC Profiler ran during this request, if it fired you will see the cost below:</p> <p>GC 457 invokes.</p> <table><tr><th>Index</th><th>Invoke Time(sec)</th><th>Use Size(byte)</th><th>Total Size(byte)</th><th>Total Object</th><th>GC Time(ms)</th></tr><tr><td>1</td><td>11.169</td><td>13134480</td><td>17014400</td><td>425360</td><td>40.002000000000120326149</td></tr><tr><td>2</td><td>11.249</td><td>13350000</td><td>17014400</td><td>425360</td><td>32.00199999999853162080</td></tr></table> <p>It is clear that introducing a full scoped generational GC is a massive undertaking, but I was thinking that certain GC apis can make application servers like thin,unicorn or puma operate way more efficiently.</p> <p>How about we add.</p> <p>GC.skip(object) # remove object from GC cycle, essentially copy the ref to a place that is not scanned</p> <p>GC.queue(object) # add an object back to the GC cycle</p> <p>GC.skipped # list all the objects that are skipped</p> <p>Careful use of these methods can reduce GC code by a very large amount as it would reduce fragmentation and logic required to determine if an object is alive or not.</p> <p>Application servers could keep snapshots of object ids on regular intervals and decide which ones to skip. Sure memory usage would rise, but GC blocking would decrease.</p> <p>Thoughts?</p>			Index	Invoke Time(sec)	Use Size(byte)	Total Size(byte)	Total Object	GC Time(ms)	1	11.169	13134480	17014400	425360	40.002000000000120326149	2	11.249	13350000	17014400	425360	32.00199999999853162080
Index	Invoke Time(sec)	Use Size(byte)	Total Size(byte)	Total Object	GC Time(ms)															
1	11.169	13134480	17014400	425360	40.002000000000120326149															
2	11.249	13350000	17014400	425360	32.00199999999853162080															

History

#1 - 11/30/2012 09:59 AM - drbrain (Eric Hodel)

sam.saffron (Sam Saffron) wrote:

How about we add.

GC.skip(object) # remove object from GC cycle, essentially copy the ref to a place that is not scanned

What do you mean by "copy"?

If you mean "move to a different memory address" this will cause crashes. If a C extension has a reference to the object and you move it you will crash when using the old reference. If the object is referenced on the stack and you dereference the old location you will crash.

#2 - 11/30/2012 10:57 AM - sam.saffron (Sam Saffron)

if you mean "move to a different memory address" this will cause crashes.

no, not at all, leave it exactly where it is, just have objectspace partitioned in 2. the refs that are "skipped" don't get scanned in the lazy sweep.

#3 - 11/30/2012 11:48 AM - Anonymous

What happens if an object in the 'permanent' objectspace references an object in the ephemeral objectspace?

Now this essentially becomes a generational GC and brings along all the implementation problems of one.

#4 - 11/30/2012 12:27 PM - sam.saffron (Sam Saffron)

@charlie looking at the code and the heap design I think there is very little cheating we could do here.

I vote to close for now.

Perhaps some mechanism for optimising the freelist could give the GC a boost:

Something like reorder freelist so it groups on heaps ordered by emptiest heap first, then the lazy sweep can sweep the heaps in allocation order.

If you allow for a large amount of free space odds are that your lazy sweep could be really fast. The vast majority of stuff allocated is very short lived.

Anyway, any work here is going to require a huge amount of experimentation, the simple api I proposed is not going to be technically feasible, especially since RVALUES can not be moved and the implications of object references causing leaks (making A permanent could mean you are making B,C,D,E permanent as a side effect) .

#5 - 12/06/2012 01:03 AM - mame (Yusuke Endoh)

- *Status changed from Open to Rejected*