Ruby master - Feature #15626

Manual Compaction for MRI's GC (`GC.compact`)

02/27/2019 10:13 PM - tenderlovemaking (Aaron Patterson)

Status:	Closed	
Priority:	Normal	
Assignee:		
Target version:		

Description

Hi,

I've attached a patch that implements a compactor for MRI. I'll try to describe the patch in detail below, but the biggest user facing changes are:

1. A new method GC.compact that can be called from Ruby
2. C extensions have a new "after compaction" callback they can register
3. Introduction of new marking functions that allow objects to be marked, but not "pinned" to the same location
4. A function to get the new location of an object (to be used from the "after compaction" callback)

Compactor for RGenGC

This document describes the compactor implementation for RGenGC. This patch adds a new method to Ruby, GC.compact which will compact the objects in the heap. What follows is a description of the algorithm, along with achievements and technical challenges.

Steps for Compaction

These are the high level steps taken in GC.compact are as follows:

1. Full GC
2. Move objects
3. Update references
4. Full GC

Step One, Full GC

Not all objects in Ruby's heap can move. In particular, C extensions may hold references to VALUE pointers. If the VALUE pointers move, the C extension may not be able to find them and will get a SEGV or possibly wrong data.

To prevent this, a new "pinning" table was introduced. Any object marked via rb_gc_mark is "pinned" and is not allowed to move. C extensions must mark their references in order to keep them alive. Since C extensions use rb_gc_mark to mark a reference, we know not to move that reference. In order to ensure all references get marked, we perform a full GC.

New functions, rb_gc_mark_no_pin, etc were introduced and used internally. These functions keeps the object alive but without pinning the reference.

Summary: Perform a full GC so that objects marked with rb_gc_mark do not move.

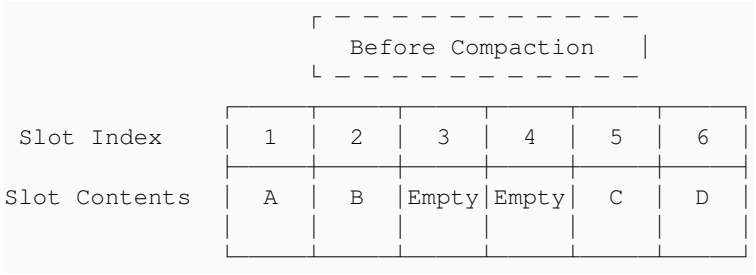
Step Two, Move Objects

This compactor uses a "two finger" algorithm which was introduced in "The Programming Language LISP: Its Operation and Applications" (page 228)[1]. Two pointers point at each side of the heap, if one slot is empty, and the other is moveable, it swaps places and leaves a T_MOVED object that contains a forwarding address.

In Ruby, the algorithm looks something like this:

```
def compact
  heap = [ ... ] # many slots
  left  = 0
  right = heap.length - 1

  while left < right
    left_slot = heap[left]
    right_slot = heap[right]

    if is_empty?(left_slot) && !is_empty?(right_slot) && can_move?(right_slot)
      swap(left, right)
      heap[right] = T_MOVED.new(left) # leave forwarding address
    end

    while !is_empty?(heap[left])
      left += 1
    end

    while is_empty?(heap[right]) || !can_move?(heap[right])
      right -= 1
    end
  end
end
```

If we have a heap with 6 slots, before compaction it might look something like this:



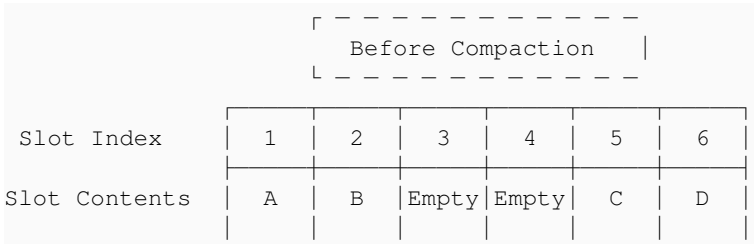
After compaction, but before reference update, it will look like this:

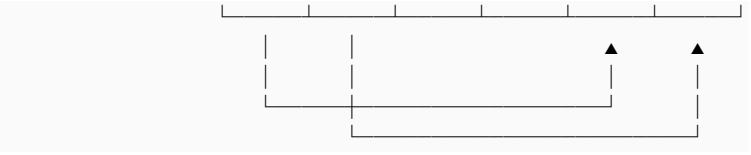
Slots 5 and 6 contain T_MOVED objects that point to the new locations of D and C objects.

Step Three, Update References

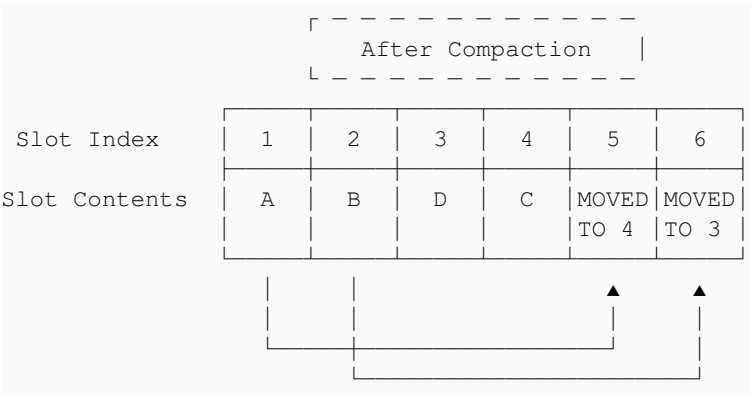
After objects have moved, references are updated. This is a matter of updating any references to use the forwarding address rather than the original location.

For example, if object A has a reference to C, and B a reference to D:





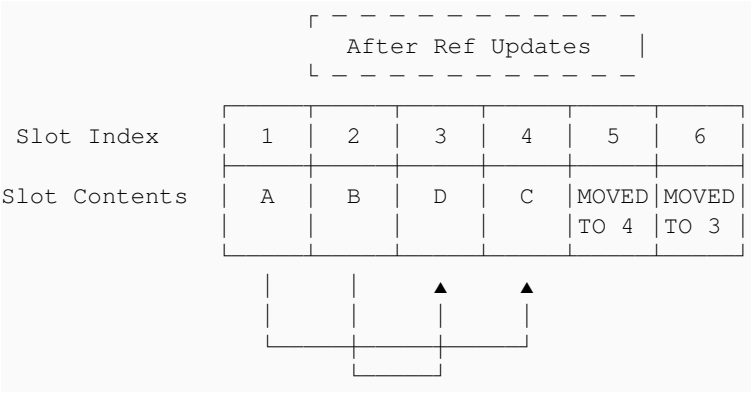
After compaction, but before updating references, the heap will look like this:



The update references step looks at each object in the heap, checks to see if it references any moved slots, then updates the reference to use the new location. In Ruby, it looks like this:

```
def update_references
  heap.each do |slot|
    next if is_empty?(slot) || is_moved?(slot)

    slot.references.each_with_index do |child, i|
      if is_moved?(child)
        slot.set_reference(i, child.new_location)
      end
    end
  end
end
```

After reference updates, the heap will look like this:



Step Four, Full GC

After references have been updated, a full GC is performed. This converts the T_MOVED slots back in to T_EMPTY slots. The GC automatically takes care of this because no object should reference a T_MOVED slot.

Non Moving Objects

Non moving objects are:

1. Objects on the stack
2. Objects marked with rb_gc_mark

3. Hash key objects

Objects on the stack

The stack should not be mutated, so these are pinned.

Objects marked with rb_gc_mark

As mentioned earlier, any object marked with rb_gc_mark may not move because a C extension may hold a reference. Here is an excerpt from Yajl, a JSON parser. This is the mark function it uses:

```
static void yajl_encoder_wrapper_mark(void * wrapper) {
    yajl_encoder_wrapper * w = wrapper;
    if (w) {
        rb_gc_mark(w->on_progress_callback);
        rb_gc_mark(w->terminator);
    }
}


obj = Data_Make_Struct(klass, yajl_encoder_wrapper, yajl_encoder_wrapper_mark,
yajl_encoder_wrapper_free, wrapper);
```

The values in this mark function w->on_progress_callback and w->terminator will not move since they are pinned by rb_gc_mark.

Hash key objects

Certain hash keys will not move. Most objects use their address as the hash value. If the object moves, the hash value could change, so hash keys are not allowed to move.

There is an exception for string objects. String objects calculate their hash based on the bytes in the string. Since the bytes in the string do not change after move, they are allowed to move. Most hash keys in large programs are strings, so only allowing string keys to move seemed enough.

Special Considerations

Object ID

Object#object_id bases its value from the address of the object. For example:

```
x = Object.new
id = x.object_id
GC.compact        # `x` moves
id == x.object_id # `id` should be same as `x.object_id`
```

We expect the object id to remain the same regardless of compaction. To deal with object_id, two hashes were introduced. One hash contains a map of the object to the *seen* object id. The second map contains all seen object ids. The maps are updated any time object_id is called, an object moves, or an object is freed.

object_id implementation in Ruby:

```
$obj_to_id = {}
$id_to_obj = {}

class Object
  def object_id
    if $obj_to_id.key?(address)
      # Second call to object_id on this object
      return $obj_to_id[address]
    else
```

```
      # First call to object_id on this object

      addr = self.address

      loop do
        if $seen_ids.key?(addr)
          # Resolve conflict
          addr += sizeof(RVALUE)
        else
          $obj_to_id[self.address] = addr
          $id_to_obj[addr]         = self
          return addr
        end
      end
    end
  end

  private

  def address
    # returns address in memory of object
  end
end
```

During compaction:

```
def compact
  heap = [ ... ] # many slots

  while left < right
    dest_slot = heap[left]
    source_slot = heap[right]

    if moving?(source_slot)
      if $obj_to_id.key?(source_slot.address)
        id = $obj_to_id.delete(source_slot.address)
        $obj_to_id[dest_slot.address] = id
        $id_to_obj[id]                = dest_slot.address
      end

      # etc
    end
  end
end
```

During Free:

```
def free(obj)
  if $obj_to_id.key?(obj.address)
    $seen_ids.delete($obj_to_id.delete(obj.address))
  end
end
```

ObjectSpace._id2ref

When generating an object id, in the case of a collision, the address is incremented by 40. This enables _id2ref to determine there is a VALUE pointer and round trip the object correctly.

Compaction Support for C extensions

Any object marked with rb_gc_mark will be "pinned" and cannot move. C extensions mark objects via rb_gc_mark, so this ensures that pointers in C stay valid.

However, some C extensions may want to support reference movement.

Reference Movement in C extensions

In order to maintain backwards compatibility, if a C extension holds a reference to a VALUE, the VALUE should not move. Going forward, C extensions can support moving VALUEs. To support moving VALUEs, a C extension should change rb_gc_mark to rb_gc_mark_no_pin, then implement a new callback that is called during compaction that gives the extension an opportunity to update its own references. A new function rb_gc_new_location will return the new location for a particular VALUE.

Here is an example for autoload from variable.c. A new compaction callback is registered, autoload_i_compact:

```
static const rb_data_type_t autoload_data_i_type = {
    "autoload_i",
    {autoload_i_mark, autoload_i_free, autoload_i_memsize, autoload_i_compact},
    0, 0, RUBY_TYPED_FREE_IMMEDIATELY
};
```

The mark function changes to *mark* references, but *not* pin them:

```
static void
autoload_i_mark(void *ptr)
{
    struct autoload_data_i *p = ptr;

    rb_gc_mark_no_pin(p->feature);

    /* allow GC to free us if no modules refer to this via autoload_const.ad */
    if (list_empty(&p->constants)) {
        rb_hash_delete(autoload_featuremap, p->feature);
    }
}
```

After the heap is compacted, any C extensions that registered a "compaction" callback will be called and have a chance to update internal references. The autoload compaction function is like this:

```
static void
autoload_i_compact(void *ptr)
{
    struct autoload_data_i *p = ptr;
    p->feature = rb_gc_new_location(p->feature);
```

The compaction callback asks for the new location for the p->feature reference. rb_gc_new_location will either return the current value of p->feature (in the case the VALUE did not move), or the new location.

Reference Verification and Testing

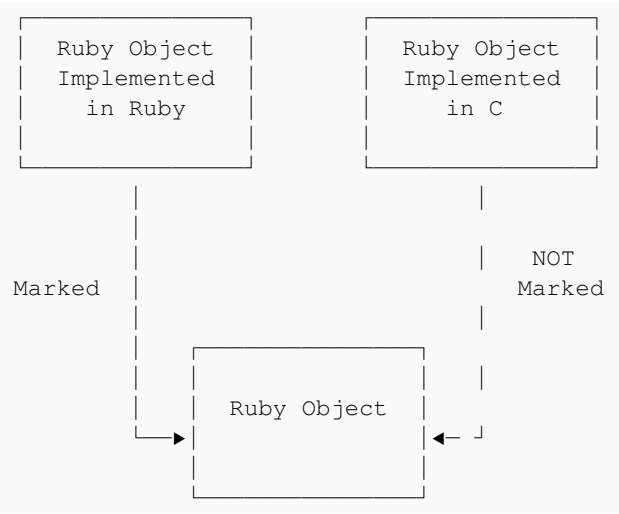
After compaction, objects that have moved are changed to T_MOVED objects with forwarding addresses. Once all references are updated, no object should point to a T_MOVED slot. We can say that before and after compaction, no object should ever point at a T_MOVED slot. So if a T_MOVED object is ever pushed on to the mark queue, there is a bug. push_mark_stack has a check for T_MOVED objects, and will crash if a T_MOVED object is ever pushed on to the mark stack.

A method GC.verify_compaction_references was added that doubles the available heap size, then compacts the heap. Since the heap has doubled in size, any object that can move will move. Any references to T_MOVED objects will be caught during the GC phase after compaction.

Known Issue

Safety for C extensions depends entirely on C extensions marking their

references. If a C extension does not mark a reference, the compactor assumes that the object is safe to move. This can cause an error in the following situation, when there is an object graph like this:



If the C extension contains a reference to an object, but expects the object not to move because a Ruby object contains a reference, then the target Ruby object *may* move and the reference in the C extension will be wrong. I like to call these "ghost references" because the GC cannot see them but they will come back to haunt you.

The solution to this is either:

1. Change the C extension to rb_gc_mark the object
2. Remove the reference from the C extension

One example of this situation was fixed here:

https://github.com/msgpack/msgpack-ruby/issues/133

Summary

Backwards compatibility with C extensions is maintained by "pinning" any references marked with rb_gc_mark. A full mark *must* be done before compacting to discover all pinned references.

A new callback for C extensions is introduced so that C extensions can support compaction. C extensions that wish to support compaction must use the new callback, rb_gc_mark_no_pin, and rb_gc_new_location.

C extensions that maintain pointers but do not mark those pointers may SEGV. We can use GC.verify_compaction_references to discover these issues.

---

1. See also "The Garbage Collection Handbook" page 32 ↩

Associated revisions

Revision 3ef4db15 - 04/09/2019 08:32 PM - tenderlove

Adding GC.compact and compacting GC support.

This commit adds the new method GC.compact and compacting GC support.
Please see this issue for caveats:

https://bugs.ruby-lang.org/issues/15626

[Feature #15626]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@67479 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 67479 - 04/09/2019 08:32 PM - tenderlovemaking (Aaron Patterson)

Adding GC.compact and compacting GC support.

This commit adds the new method GC.compact and compacting GC support.
Please see this issue for caveats:

https://bugs.ruby-lang.org/issues/15626

[Feature #15626]

Revision 3c55b643 - 04/17/2019 03:17 AM - tenderlove

Adding GC.compact and compacting GC support.

This commit adds the new method GC.compact and compacting GC support.
Please see this issue for caveats:

https://bugs.ruby-lang.org/issues/15626

[Feature #15626]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@67576 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 67576 - 04/17/2019 03:17 AM - tenderlovemaking (Aaron Patterson)

Adding GC.compact and compacting GC support.

This commit adds the new method GC.compact and compacting GC support.
Please see this issue for caveats:

https://bugs.ruby-lang.org/issues/15626

[Feature #15626]

History

#1 - 02/27/2019 10:26 PM - tenderlovemaking (Aaron Patterson)

- File after_compact-1.png added

- File before_compact-1.png added

- File after_compact-0.png added

- File before_compact-0.png added

I've tested the compactor with a Rails application, and I want to share my results.

I added the middleware below:

```
# This file is used by Rack-based servers to start the application.

require_relative 'config/environment'
require 'objspace'

$COUNT = 0
class CompactHeap
  def initialize app
    @app = app
  end

  def call env
    x = @app.call env
    GC.start
    File.open("before_compact-#{$COUNT}.txt", 'w') do |f|
      ObjectSpace.dump_all(output: f)
    end
    GC.compact
    File.open("after_compact-#{$COUNT}.txt", 'w') do |f|
      ObjectSpace.dump_all(output: f)
    end
    $COUNT += 1
    x
  end
```

```
end

use CompactHeap
run Rails.application
```

The above middleware will:

1. Run the web request
2. Run the GC
3. Dump the heap
4. Compact the heap
5. Dump the heap

This gives us an idea what the heap looks like after a request, but before and after compaction. I've attached visualizations of the heap for two requests. Each visualization is a PNG file that represents the heap. Each column of the image represents a page in the heap. Columns are two pixels wide. Each square in the column represents a slot. Slots are 2px by 2px squares. Red squares are slots that have an unmovable (pinned) object. Black squares are slots that have a moveable object. Other squares are empty slots.

The first image "before-compact-0.png" is before a compaction has ever occurred. You can see there are many unfilled pages. Of the 582 pages, 98 are full, and 484 have empty slots. After compaction "after-compact-0.png", there are 576 pages, 411 full, and 165 have empty slots.

Subsequent requests have similar looking heaps to "after-compact-0" (as expected).

The program I used to generate these visualizations can be found here:

https://gist.githubusercontent.com/tenderlove/578cfb4ce677465d0a8ceb4362928a89/raw/41798db55154b235059f1414a8bb18cd955d1ace/heapviz.rb

I'll publish the Rails application I used to generate these graphs and add a link here. :)

#2 - 02/27/2019 10:57 PM - tenderlovemaking (Aaron Patterson)

- File 0001-Compacting-GC-for-MRI.patch added

Adding updated patch.

#3 - 02/27/2019 10:57 PM - tenderlovemaking (Aaron Patterson)

- File deleted (0001-Compacting-GC-for-MRI.patch)

#4 - 02/27/2019 11:37 PM - tenderlovemaking (Aaron Patterson)

- File 0001-GC-Compaction-for-MRI.patch added

#5 - 02/27/2019 11:38 PM - tenderlovemaking (Aaron Patterson)

- File deleted (0001-Compacting-GC-for-MRI.patch)

#6 - 02/27/2019 11:43 PM - tenderlovemaking (Aaron Patterson)

- File 0001-GC-Compaction-for-MRI.patch added

#7 - 02/27/2019 11:44 PM - tenderlovemaking (Aaron Patterson)

- File deleted (0001-GC-Compaction-for-MRI.patch)

#8 - 02/27/2019 11:59 PM - tad (Tadashi Saito)

Really interesting, but what are the benefits for users who writes Ruby?

Did the Rails app get faster enough? Or you have some other motivations?

#9 - 02/28/2019 04:25 AM - duerst (Martin Dürst)

Agree, really interesting. Several comments/questions:

- Why a full CG at the end? Wouldn't it be much cheaper to just collect the T_MOVED objects (these objects could contain an internal pointer each, chaining them together).
- Hash key objects: You say most hash keys are strings, and they can be moved, but what about Symbols?

- I had a look at the .png files. I see the distinction between red squares and other squares very well, but I can't see any distinction between movable and empty slots, everything that's not red is black.

#10 - 02/28/2019 06:18 AM - ko1 (Koichi Sasada)

Great post. I think there are no big changes we discussed before.

Points:

- I'm not sure we can accept the "Known Issue". GC.verify_compaction_references can't discover this issue, right? It can introduce difficult bugs to remove... I have no good idea to discover it. "Time" (not time class, of course) can solve it, but...
- Can we introduce it just after every major (full) gc?

Minor points:

- Could you use gist (or something useful) to show your benchmark results with figures?

#11 - 03/11/2019 08:12 AM - matz (Yukihiro Matsumoto)

The idea of GC.compact sounds great. I understand the issue but it seems to be acceptable.

Matz.

#12 - 03/11/2019 09:00 AM - naruse (Yui NARUSE)

I want to introduce this feature in Ruby 2.7.0-preview1.
Therefore could you write an item for the release note?

The article should be like "JIT [Experimental]" in 2.6.0.
https://www.ruby-lang.org/en/news/2018/12/25/ruby-2-6-0-released/

The description should explain

- What is the merit of this Compaction GC. (the audience of this sentence is News media)
- The technical detail (appeal) of this feature. (the audience of this sentences is Ruby developer)
- About the known issue and how to debug and workaround
  - If you hit SEGV, comment out GC.compact and try; if it works it's because of Compaction GC
  - check extension libraries and if you can identify it use rb_gc_mark or something
  - Add more debug mode for example just do compaction but not unmap the old page; set guard page or write garbage data.

#13 - 03/18/2019 08:51 PM - nateberkopec (Nate Berkopec)

> Can we introduce it just after every major (full) gc?

I agree. If the necessary steps are Full GC -> Update -> Move -> Full GC, then it would be faster to just compact automatically after a full GC which is already occurring. I understand this could aggravate the "known issue" however by making compaction mandatory rather than optional.

Aaron, regarding the API: how do you envision this will be called "in the real world"? Unicorn::OOBGC, a similar project and API, was responsible for a lot of wasted CPU cycles because it ran after every request. Either you or Sam blogged about this at some point (I can't remember and can't find it), but that approach caused something like an extra 10% CPU utilization on a big Rails app.

> This converts the T_MOVED slots back in to T_EMPTY slots. The GC automatically takes care of this because no object should reference a T_MOVED slot.

Is this strictly necessary? I think the fact that GC.compact requires 2 full GC cycles makes it rather expensive. What if we could just leave "T_MOVED" slots around until the next full GC cycle?

If we could take Koichi's suggestion and also "live with" T_MOVED slots living in the heap until the next full GC, then GC.compact does *not* add any full GC cycles to an application, and the only added time cost is the cost of updating and moving.

#14 - 03/19/2019 09:33 AM - PikachuEXE (Pikachu Leung)

nateberkopec (Nate Berkopec) wrote:

> > Can we introduce it just after every major (full) gc?

> I agree. If the necessary steps are Full GC -> Update -> Move -> Full GC, then it would be faster to just compact automatically after a full GC which is already occurring. I understand this could aggravate the "known issue" however by making compaction mandatory rather than optional.
>
> Aaron, regarding the API: how do you envision this will be called "in the real world"? Unicorn::OOBGC, a similar project and API, was responsible for a lot of wasted CPU cycles because it ran after every request. Either you or Sam blogged about this at some point (I can't remember and can't find it), but that approach caused something like an extra 10% CPU utilization on a big Rails app.
>
> > This converts the T_MOVED slots back in to T_EMPTY slots. The GC automatically takes care of this because no object should reference a T_MOVED slot.
>
> Is this strictly necessary? I think the fact that GC.compact requires 2 full GC cycles makes it rather expensive. What if we could just leave "T_MOVED" slots around until the next full GC cycle?
>
> If we could take Koichi's suggestion and also "live with" T_MOVED slots living in the heap until the next full GC, then GC.compact does *not* add any full GC cycles to an application, and the only added time cost is the cost of updating and moving.

As an experimental feature shouldn't it be disabled by default and only enabled via something like env variable?

But I do agree that given a full GC is always required, might as well run (2) and (3) after an existing full GC, leave (4) for next full GC

How do I write my own patch and test it myself?

#15 - 03/19/2019 03:25 PM - headius (Charles Nutter)

We have already discussed in https://bugs.ruby-lang.org/issues/15408 the deprecation and eventual removal of _id2ref, so perhaps we can move these things together.

The additional hashing logic to maintain the Object-to-ID maps will add significant overhead to objects in any application using object_id for e.g. logging, debugging, or other possibly unsavory purposes.

Better to just move forward with EOL for _id2ref and go with just simple monotonic object IDs! :-D

https://bugs.ruby-lang.org/issues/15408

#16 - 03/20/2019 09:41 AM - PikachuEXE (Pikachu Leung)

Made some changes for "auto compaction"
I am a newbie in C programming
This is just for discussion, not tested nor compiled yet

https://github.com/ruby/ruby/compare/trunk...PikachuEXE:feature/gc-compact/pika

#17 - 03/21/2019 05:37 AM - PikachuEXE (Pikachu Leung)

```
To prevent this, a new "pinning" table was introduced. Any object marked via
rb_gc_mark is "pinned" and is not allowed to move. C extensions must mark
their references in order to keep them alive. Since C extensions use
rb_gc_mark to mark a reference, we know not to move that reference. In order
to ensure all references get marked, we perform a full GC.
```

I wrote a draft to allow myself to understand the GC flow a bit better
But I don't really understand which step contains code for "ensure all references get marked"

```
(Most of method calls that seems to be related to profiling logging are ignored)

## `rb_gc`
- `garbage_collect`
- `gc_finalize_deferred` (digged until finalize_list, something like killing zombies?)

## `garbage_collect`
- `gc_rest`
- `gc_start`

## `gc_rest`
- `gc_verify_internal_consistency` (when `RGENGC_CHECK_MODE >= 2`)
- `gc_marks_rest` (when `is_incremental_marking(objspace)`)
- `gc_sweep_rest` (when `is_lazy_sweeping(heap_eden)`)

## `gc_start`
- `gc_verify_internal_consistency` (when `RGENGC_CHECK_MODE >= 2`)
- `gc_prof_setup_new_record` (I skipped)
- `gc_reset_malloc_info` (I skipped)
- `rb_transient_heap_start_marking` (I skipped)
```

```
- `gc_marks` (main work)

## `gc_marks`
- `gc_marks_rest` (when `USE_RGENGC` and `!is_incremental_marking(objspace)`)
- `gc_marks_start` + `gc_marks_rest` (when not `USE_RGENGC`)

## `gc_marks_start`
- So many stuff in the middle I don't understand
- `gc_mark_roots` (I read but don't understand)

## `gc_marks_rest`
- `gc_mark_stacked_objects_incremental` + `gc_marks_finish` (if `is_incremental_marking(objspace)`)
- `gc_mark_stacked_objects_all` + `gc_marks_finish` (unless `is_incremental_marking(objspace)`)
- `gc_sweep`

## `gc_sweep`
- Auto Compaction (Added in my branch, when auto compaction allowed, not sure if this is the right place)
- `gc_sweep_start` + `gc_sweep_rest` (if `immediate_sweep`)
- `gc_sweep_start` + `gc_sweep_step` (unless `immediate_sweep`)
- `gc_heap_prepare_minimum_pages` (Prepare enough heap pages for ???)

## `gc_sweep_start`
- `gc_sweep_start_heap` with logging (I read but don't understand)

## `gc_sweep_step`
- Mostly implementation I don't understand
- `gc_sweep_finish`

## `gc_sweep_finish`
- `heap_pages_free_unused_pages`
- `heap_allocatable_pages_set` if `heap_allocatable_pages < heap_tomb->total_pages`
- `gc_verify_internal_consistency` if `heap_allocatable_pages < heap_tomb->total_pages`
```

#18 - 03/22/2019 09:21 AM - PikachuEXE (Pikachu Leung)

- File ruby_2019-03-22-171839-1_PikachuEXEs-MBP-2018.crash added

- File ruby_2019-03-22-171839_PikachuEXEs-MBP-2018.crash added

I got segfault with this patch
I added 1 extra change to fix compile error on MacOS
https://github.com/ruby/ruby/compare/trunk...PikachuEXE:feature/gc-compact/15626

Installed via
rvm install ruby-head --reconfigure --repo https://github.com/PikachuEXE/ruby.git --branch feature/gc-compact/15626 -n ruby_27_gc_comapct

ruby-head has no such issue
Using passenger open source 6.0.2 for testing

Edit: Forgot to mention, I did not call GC.compact

#19 - 03/22/2019 08:51 PM - noahgibbs (Noah Gibbs)

One problem with automatically compacting on full GC: compaction appears to require two full GCs to do its thing. It might be possible to integrate this into Ruby's normal cycle. But I don't think this is designed for that as-is.

Aaron has commented on this in his talks - usually he calls GC.compact once after giving everything a chance to "settle" (running the app for awhile.)

You could call it periodically, but I think this is a heavier-weight operation than a normal full GC.

#20 - 03/23/2019 12:57 AM - duerst (Martin Dürst)

noahgibbs (Noah Gibbs) wrote:

> Aaron has commented on this in his talks -

Any pointers?

> usually he calls GC.compact once after giving everything a chance to "settle" (running the app for awhile.)

#21 - 03/23/2019 09:50 PM - tenderlovemaking (Aaron Patterson)

naruse (Yui NARUSE):

> I want to introduce this feature in Ruby 2.7.0-preview1.
> Therefore could you write an item for the release note?

Yes. I need to fix a bug with THEAP integration and I'll commit this along with NEWS.

ko1 (Koichi Sasada):

> I'm not sure we can accept the "Known Issue". GC.verify_compaction_references can't discover this issue, right? It can introduce difficult bugs to remove... I have no good idea to discover it. "Time" (not time class, of course) can solve it, but...

Neither do I. Doubling the heap size then compacting should give highest probability to find these references (I think). Unfortunately the way it finds them is by crashing.

> Can we introduce it just after every major (full) gc?

I think it's a good idea, but maybe only with GC.stress = true? I don't have a good idea about the performance (wall clock time), but I'm worried about adding it automatically without doing some performance improvements.

> Could you use gist (or something useful) to show your benchmark results with figures?

Yes, I'll make a gist containing benchmarks and figures. I'll post them here before committing.

@martin:

> Why a full CG at the end? Wouldn't it be much cheaper to just collect the T_MOVED objects (these objects could contain an internal pointer each, chaining them together).

Mainly because I'm lazy. :D

We could definitely do that, I just wanted to prove that we could do compaction in the first place. I have a lot of ideas for performance improvements.

> Hash key objects: You say most hash keys are strings, and they can be moved, but what about Symbols?

Symbols probably can be moved. My approach was conservative: assume nothing can move then slowly introduce things I know to be movable. We can probably do Symbols too, I just didn't bother because the results were so good with just strings.

> I had a look at the .png files. I see the distinction between red squares and other squares very well, but I can't see any distinction between movable and empty slots, everything that's not red is black.

Sorry, I used translucent pixels for the "empty" locations. If you view the png on a white background it should look right, but I'll regenerate the pngs with a white background instead of translucent.

@nate:

> Aaron, regarding the API: how do you envision this will be called "in the real world"?

Initially, just GC.compact before fork (is how we've used it). Once I address performance issues it should just be automatic; it should happen after the mark phase on a full GC (as a mark-compact GC).

@Pikachu:

> But I don't really understand which step contains code for "ensure all references get marked"

Full GC marks everything in the mark table. Mark table bits aren't cleared until the next GC.

> I got segfault with this patch
> I added 1 extra change to fix compile error on MacOS

I'll take a look at the core file, but can you give more info so I can try locally?

> Edit: Forgot to mention, I did not call GC.compact

Hmmm. My patch should have no impact without calling GC.compact.

@Noah:

> Aaron has commented on this in his talks - usually he calls GC.compact once after giving everything a chance to "settle" (running the app for awhile.)

We call it after loading as much code as possible, and right before forking. This way we have as many heap pages packed with probably long lived objects before forking (to maximize CoW).

#22 - 03/25/2019 01:54 AM - noahgibbs (Noah Gibbs)

duerst (Martin Dürst): looks like he's updated his approach a bit since I last saw the talk (quite awhile ago!) But here's a 2017 version: https://www.youtube.com/watch?v=ptG5HtmvQx8

At around 19:45 he's talking about compacting before fork.

When I saw an older talk, he was talking about (I believe) running a large Rails app for awhile and then just showing that moving around long-lived blocks of memory reduced fragmentation. But I may remember wrong.

#23 - 03/25/2019 02:52 AM - PikachuEXE (Pikachu Leung)

> I'll take a look at the core file, but can you give more info so I can try locally?

I just start my current project with passenger installed
I will create a sample project to reproduce this issue
Takes time though (stealing work time to do this, though this is also legit work >:)

Any info I can provide without creating the sample project?

#24 - 03/26/2019 03:00 AM - noahgibbs (Noah Gibbs)

I'm trying to run this and check speed and stability with Rails Ruby Bench. I'm having an odd bug when I try to use GC.compact. But it may be my fault so I won't post it (yet).

However, I *do* notice a speed regression when running w/ your patch from recent head-of-master (SHA c92c0a593593da2eb1ff94d83d80f71e7ae5343c before patching.)

I'm seeing unpatched performance of 187.4 req/sec with variance 3.8. But with your patch, I'm seeing 178.8 req/sec with variance 4.7. That's close enough it *could* be measurement error if I was unlucky, and I'll try running it again. But I think this may currently reduce performance even without calling GC.compact.

Due to the way my benchmark works, increased memory usage can also manifest as reduced speed -- I don't have an easy way to tease those apart with RRB. But one or the other seems likely.

On the plus side, I did *not* see any segfaults, and RRB is a pretty good stability torture-test for crashes. I ran this with 30 batches of 30,000 HTTP requests per Ruby version (unpatched vs patched.)

#25 - 03/26/2019 06:12 AM - noahgibbs (Noah Gibbs)

Okay. Second benchmark run gives essentially the same results:

unpatched: 187.9 reqs/sec, variance 4.1
patched: 179.2 reqs/sec, variance 4.8

So yeah, I'm seeing a performance regression with the patched code with *no* call to GC.compact.

If I run GC.compact in an initializer in Discourse, I wind up getting "no such method" errors, though I seem to get different ones run-to-run. For instance:

```
bundler: failed to load command: ./start.rb (./start.rb)
NoMethodError: undefined method `create_id' for #<EmailHelper:0x0000555c88717e30>
  /home/ubuntu/mri-gccompact/lib/ruby/2.7.0/json/common.rb:156:in `initialize'
  /home/ubuntu/mri-gccompact/lib/ruby/2.7.0/json/common.rb:156:in `new'
  /home/ubuntu/mri-gccompact/lib/ruby/2.7.0/json/common.rb:156:in `parse'
  /home/ubuntu/.rvm/gems/ext-mri-march-gccompact/gems/sprockets-3.7.1/lib/sprockets/manifest.rb:318:in `json_d
ecode'
  /home/ubuntu/.rvm/gems/ext-mri-march-gccompact/gems/sprockets-3.7.1/lib/sprockets/manifest.rb:73:in `initial
ize'
  /home/ubuntu/.rvm/gems/ext-mri-march-gccompact/gems/sprockets-rails-3.2.0/lib/sprockets/railtie.rb:201:in `n
ew'
```

```
  /home/ubuntu/.rvm/gems/ext-mri-march-gccompact/gems/sprockets-rails-3.2.0/lib/sprockets/railtie.rb:201:in `b
uild_manifest'
  /home/ubuntu/.rvm/gems/ext-mri-march-gccompact/gems/sprockets-rails-3.2.0/lib/sprockets/railtie.rb:214:in `b
lock in <class:Railtie>'
  /home/ubuntu/.rvm/gems/ext-mri-march-gccompact/gems/activesupport-4.2.8/lib/active_support/lazy_load_hooks.r
b:36:in `execute_hook'
  /home/ubuntu/.rvm/gems/ext-mri-march-gccompact/gems/activesupport-4.2.8/lib/active_support/lazy_load_hooks.r
b:45:in `block in run_load_hooks'
  /home/ubuntu/.rvm/gems/ext-mri-march-gccompact/gems/activesupport-4.2.8/lib/active_support/lazy_load_hooks.r
b:44:in `each'
  /home/ubuntu/.rvm/gems/ext-mri-march-gccompact/gems/activesupport-4.2.8/lib/active_support/lazy_load_hooks.r
b:44:in `run_load_hooks'
  /home/ubuntu/.rvm/gems/ext-mri-march-gccompact/gems/railties-4.2.8/lib/rails/application/finisher.rb:62:in `
block in <module:Finisher>'
  /home/ubuntu/.rvm/gems/ext-mri-march-gccompact/gems/railties-4.2.8/lib/rails/initializable.rb:30:in `instanc
e_exec'
  /home/ubuntu/.rvm/gems/ext-mri-march-gccompact/gems/railties-4.2.8/lib/rails/initializable.rb:30:in `run'
  /home/ubuntu/.rvm/gems/ext-mri-march-gccompact/gems/railties-4.2.8/lib/rails/initializable.rb:55:in `block i
n run_initializers'
  /home/ubuntu/mri-gccompact/lib/ruby/2.7.0/tsort.rb:228:in `block in tsort_each'
  /home/ubuntu/mri-gccompact/lib/ruby/2.7.0/tsort.rb:350:in `block (2 levels) in each_strongly_connected_compo
nent'
  /home/ubuntu/mri-gccompact/lib/ruby/2.7.0/tsort.rb:431:in `each_strongly_connected_component_from'
  /home/ubuntu/mri-gccompact/lib/ruby/2.7.0/tsort.rb:349:in `block in each_strongly_connected_component'
  /home/ubuntu/mri-gccompact/lib/ruby/2.7.0/tsort.rb:347:in `each'
  /home/ubuntu/mri-gccompact/lib/ruby/2.7.0/tsort.rb:347:in `call'
  /home/ubuntu/mri-gccompact/lib/ruby/2.7.0/tsort.rb:347:in `each_strongly_connected_component'
  /home/ubuntu/mri-gccompact/lib/ruby/2.7.0/tsort.rb:226:in `tsort_each'
  /home/ubuntu/mri-gccompact/lib/ruby/2.7.0/tsort.rb:205:in `tsort_each'
  /home/ubuntu/.rvm/gems/ext-mri-march-gccompact/gems/railties-4.2.8/lib/rails/initializable.rb:54:in `run_ini
tializers'
  /home/ubuntu/.rvm/gems/ext-mri-march-gccompact/gems/railties-4.2.8/lib/rails/application.rb:352:in `initiali
ze!'
  /home/ubuntu/.rvm/gems/ext-mri-march-gccompact/gems/railties-4.2.8/lib/rails/railtie.rb:194:in `public_send'
  /home/ubuntu/.rvm/gems/ext-mri-march-gccompact/gems/railties-4.2.8/lib/rails/railtie.rb:194:in `method_missi
ng'
  /home/ubuntu/rails_ruby_bench/work/discourse/config/environment.rb:5:in `<top (required)>'
  /home/ubuntu/rails_ruby_bench/start.rb:13:in `require'
  /home/ubuntu/rails_ruby_bench/start.rb:13:in `<top (required)>'
```

The only difference between this run and the (error-free) patched version speed run above is that I add a single initializer in discourse/config/initializers/900-gc-compact.rb, consisting of only the text "GC.compact".

However, the stack trace is *not* always the same - I got a different "no-such-method"-type error on the previous run.

Here's another stack trace from running the exact same setup a second time in a row -- let me know if you'd like me to send information, or set you up with an account on a VM to look at it.

```
NoMethodError: undefined method `create_id' for ".avi.jst.coffee.erb":String
  /home/ubuntu/mri-gccompact/lib/ruby/2.7.0/json/common.rb:156:in `initialize'
  /home/ubuntu/mri-gccompact/lib/ruby/2.7.0/json/common.rb:156:in `new'
  /home/ubuntu/mri-gccompact/lib/ruby/2.7.0/json/common.rb:156:in `parse'
  /home/ubuntu/.rvm/gems/ext-mri-march-gccompact/gems/sprockets-3.7.1/lib/sprockets/manifest.rb:318:in `json_d
ecode'
  /home/ubuntu/.rvm/gems/ext-mri-march-gccompact/gems/sprockets-3.7.1/lib/sprockets/manifest.rb:73:in `initial
ize'
  /home/ubuntu/.rvm/gems/ext-mri-march-gccompact/gems/sprockets-rails-3.2.0/lib/sprockets/railtie.rb:201:in `n
ew'
  /home/ubuntu/.rvm/gems/ext-mri-march-gccompact/gems/sprockets-rails-3.2.0/lib/sprockets/railtie.rb:201:in `b
uild_manifest'
  /home/ubuntu/.rvm/gems/ext-mri-march-gccompact/gems/sprockets-rails-3.2.0/lib/sprockets/railtie.rb:214:in `b
lock in <class:Railtie>'
  /home/ubuntu/.rvm/gems/ext-mri-march-gccompact/gems/activesupport-4.2.8/lib/active_support/lazy_load_hooks.r
b:36:in `execute_hook'
  /home/ubuntu/.rvm/gems/ext-mri-march-gccompact/gems/activesupport-4.2.8/lib/active_support/lazy_load_hooks.r
b:45:in `block in run_load_hooks'
  /home/ubuntu/.rvm/gems/ext-mri-march-gccompact/gems/activesupport-4.2.8/lib/active_support/lazy_load_hooks.r
b:44:in `each'
  /home/ubuntu/.rvm/gems/ext-mri-march-gccompact/gems/activesupport-4.2.8/lib/active_support/lazy_load_hooks.r
b:44:in `run_load_hooks'
  /home/ubuntu/.rvm/gems/ext-mri-march-gccompact/gems/railties-4.2.8/lib/rails/application/finisher.rb:62:in `
block in <module:Finisher>'
  /home/ubuntu/.rvm/gems/ext-mri-march-gccompact/gems/railties-4.2.8/lib/rails/initializable.rb:30:in `instanc
e_exec'
  /home/ubuntu/.rvm/gems/ext-mri-march-gccompact/gems/railties-4.2.8/lib/rails/initializable.rb:30:in `run'
```

```
  /home/ubuntu/.rvm/gems/ext-mri-march-gccompact/gems/railties-4.2.8/lib/rails/initializable.rb:55:in `block i
n run_initializers'
  /home/ubuntu/mri-gccompact/lib/ruby/2.7.0/tsort.rb:228:in `block in tsort_each'
  /home/ubuntu/mri-gccompact/lib/ruby/2.7.0/tsort.rb:350:in `block (2 levels) in each_strongly_connected_compo
nent'
  /home/ubuntu/mri-gccompact/lib/ruby/2.7.0/tsort.rb:431:in `each_strongly_connected_component_from'
  /home/ubuntu/mri-gccompact/lib/ruby/2.7.0/tsort.rb:349:in `block in each_strongly_connected_component'
  /home/ubuntu/mri-gccompact/lib/ruby/2.7.0/tsort.rb:347:in `each'
  /home/ubuntu/mri-gccompact/lib/ruby/2.7.0/tsort.rb:347:in `call'
  /home/ubuntu/mri-gccompact/lib/ruby/2.7.0/tsort.rb:347:in `each_strongly_connected_component'
  /home/ubuntu/mri-gccompact/lib/ruby/2.7.0/tsort.rb:226:in `tsort_each'
  /home/ubuntu/mri-gccompact/lib/ruby/2.7.0/tsort.rb:205:in `tsort_each'
  /home/ubuntu/.rvm/gems/ext-mri-march-gccompact/gems/railties-4.2.8/lib/rails/initializable.rb:54:in `run_ini
tializers'
  /home/ubuntu/.rvm/gems/ext-mri-march-gccompact/gems/railties-4.2.8/lib/rails/application.rb:352:in `initiali
ze!'
  /home/ubuntu/.rvm/gems/ext-mri-march-gccompact/gems/railties-4.2.8/lib/rails/railtie.rb:194:in `public_send'
  /home/ubuntu/.rvm/gems/ext-mri-march-gccompact/gems/railties-4.2.8/lib/rails/railtie.rb:194:in `method_missi
ng'
  /home/ubuntu/rails_ruby_bench/work/discourse/config/environment.rb:5:in `<top (required)>'
  /home/ubuntu/rails_ruby_bench/start.rb:13:in `require'
  /home/ubuntu/rails_ruby_bench/start.rb:13:in `<top (required)>'
```

#26 - 04/03/2019 12:37 AM - tenderlovemaking (Aaron Patterson)

@noah I think I have these errors fixed on my branch (including the performance regression). I slowed down every call to free an object because they may need to be removed from the global object_id table. Should be fixed here though:
https://github.com/ruby/ruby/commit/82f3d8b6cc4e491f13ec29f67daa2bb4ae0f1dbd (GC benchmarks are the same speed on trunk vs my branch)

I'll try to merge this this week, but I'm traveling tomorrow so I may commit this on Monday. I'm really sorry for the delay.

#27 - 04/03/2019 01:17 AM - tenderlovemaking (Aaron Patterson)

Sorry, I meant noahgibbs (Noah Gibbs)

#28 - 04/03/2019 05:50 AM - PikachuEXE (Pikachu Leung)

tenderlovemaking (Aaron Patterson)

I try your patch again and no error this time (without the middleware to call GC.compact)
Maybe I forgot to restart server after switching to branch without middleware on my last test

However with middleware (I have restarted server after switching branch this time, yup)
I can see the ruby processes running at 100% after requests finished and it does not happen on app boot

However since I am still using old ruby-head without your commit
https://github.com/ruby/ruby/commit/82f3d8b6cc4e491f13ec29f67daa2bb4ae0f1dbd

What's the best way to install the latest ruby-head with for latest version of this change?

#29 - 04/09/2019 12:56 AM - tenderlovemaking (Aaron Patterson)

Hi,

Here are some statistics I gathered. I'll present a few more at RubyKaigi using our application.

https://gist.github.com/tenderlove/99112e9fcc85d9c6c7d9d0ea40063fc6

#30 - 04/09/2019 08:32 PM - tenderlovemaking (Aaron Patterson)

- Status changed from Open to Closed

Applied in changeset trunk|r67479.

Adding GC.compact and compacting GC support.

This commit adds the new method GC.compact and compacting GC support.
Please see this issue for caveats:

https://bugs.ruby-lang.org/issues/15626

[Feature #15626]

#31 - 04/15/2019 04:27 AM - noahgibbs (Noah Gibbs)

Tested again w/ merged version of this patch. My "before" and "after" timings are within the margin of measurement error. I no longer see a performance regression. Thanks!

Files

before_compact-0.png	111 KB	02/27/2019	tenderlovemaking (Aaron Patterson)
after_compact-0.png	80.6 KB	02/27/2019	tenderlovemaking (Aaron Patterson)
before_compact-1.png	81.2 KB	02/27/2019	tenderlovemaking (Aaron Patterson)
after_compact-1.png	79.4 KB	02/27/2019	tenderlovemaking (Aaron Patterson)
0001-GC-Compaction-for-MRI.patch	77.4 KB	02/27/2019	tenderlovemaking (Aaron Patterson)
ruby_2019-03-22-171839_PikachuEXEs-MBP-2018.crash	71.4 KB	03/22/2019	PikachuEXE (Pikachu Leung)
ruby_2019-03-22-171839-1_PikachuEXEs-MBP-2018.crash	71.4 KB	03/22/2019	PikachuEXE (Pikachu Leung)