Memory management tuning in Ruby

Koichi Sasada

<ko1@heroku.com>
Summary of this talk

• Introduction of new versions
  • Ruby 2.1 (2.1.1 was released)
  • Ruby 2.2 (currently working on)

• Basic of Ruby’s memory management (GC)

• GC tuning parameters
  • “What” and “How” we can tune by GC parameters
Who am I?

• Koichi Sasada a.k.a. ko1
• From Japan
• 笹田 (family name) 耕一 (given name) in Kanji character
  • “Ichī” (Kanji character “一”) means “1” or first
  • This naming rule represents I’m the first son of my parents
  • Ko”ichi” → ko1
Who am I?

• CRuby/MRI committer
  • Virtual machine (YARV) from Ruby 1.9
  • YARV development since 2004/1/1
  • Recently, improving GC performance

• Matz team at Heroku, Inc.
  • Full-time CRuby developer
  • Working in Japan

• Director of Ruby Association
• Foundation to encourage Ruby developments and communities
  • Chairman is Matz
  • Located at Matsue-city, Shimane, Japan

• Activities
  • Maintenance of Ruby (Cruby) interpreter
    • Now, it is for Ruby 1.9.3
    • Ruby 2.0.0 in the future?
  • Events, especially RubyWorld Conference
  • Ruby Prize
  • Grant project. We have selected **3 proposals** in 2013
    • Win32Utils Support, Conductor, Smalruby - smalruby-editor
    • We will make this grant 2014!!
• **Donation** for Ruby developments and communities
You should know about Heroku!!

Heroku supports Ruby development

- Many talents for Ruby, and also other languages
- Heroku employs 3 *Ruby interpreter core developers*
  - Matz
  - Nobu
  - Ko1 (me)
- We name our group “Matz team”

This talk is also sponsored by Heroku!
“Matz team” in Heroku
Matz team in Heroku in Japan

Matz @ Shimane
Title collector

Nobu @ Tochigi
Patch monster

ko1 @ Tokyo
EDD developer

Memory management tuning in Ruby, RubyConfPH 2014 by K.Sasada
<ko1@heroku.com>
Matz team at Heroku

Hierarchy

Matz @ Shimane
Title collector

ko1 @ Tokyo
EDD developer

Communication
with Skype

[Not stupid boss]

Nobu @ Tochigi
Patch monster

Memory management tuning in Ruby,
RubyConfPH 2014 by K.Sasada
<ko1@heroku.com>
Matz
Title collector

• He has so many (job) title
  • Chairman - Ruby Association
  • Fellow - NaCl
  • Chief architect, Ruby - Heroku
  • Research institute fellow – Rakuten
  • Chairman – NPO mruby Forum
  • Senior researcher – Kadokawa Ascii Research Lab
  • Visiting professor – Shimane University
  • Honorable citizen (living) – Matsue city
  • Honorable member – Nihon Ruby no Kai
  • ...

• This margin is too narrow to contain
Message from Matz

“I am awfully sorry for not being here.
But I love you.
Maybe next time!”
Nobu
Patch monster

• Great patch creator
Nobu
Patch monster

COMMIT RATIO IN LAST 5 YEARS

nobu 29%

akr 12%
svn 9%
naruse 8%
usa 4%
ko1 3%
dsbrain 3%
kosaki 2%
tenderlove 2%
tszak 2%
kou 2%
kmarcandre 2%
matz 2%
tenderlove 2%
shyouhei 1%
smour 2%
marcandre 2%
znak 2%
kou 2%
kmarcandre 2%

Memory management tuning in Ruby,
RubyConfPH 2014 by K.Sasada
<ko1@heroku.com>
Ko1
EDD developer

Commit number of ko1 (last 3 years)

EDD: Event Driven Development

<ko1@heroku.com>
Mission of Matz team

• **Improve quality of next version of CRuby**
  • Matz decides a spec finally
  • Nobu fixed huge number of bugs
  • Ko1 improves the performance

Current target is Ruby 2.2!!
Now, Ruby 2.1 is old version for us.
Ruby 2.1
Current stable

http://www.flickr.com/photos/loginesta/5266114104
Ruby 2.1

- **Ruby 2.1.0** was released at **2013/12/25**
  - New features
  - Performance improvements
- **Ruby 2.1.1** was released at **2014/02/24**
  - Includes many bug fixes found after 2.1.0 release
  - Introduce a new GC tuning parameter to change generational GC behavior (introduce it later)
Ruby 2.1 the biggest change
Version policy

• Change the versioning policy
  • Drop “patch level” in the version
  • Teeny represents patch level
    • Release new teeny versions about every 3 month
    • Teeny upgrades keep compatibility
  • Minor upgrades can break backward compatibility
    • We make an effort to keep compatibility
      (recently. Remember Ruby 1.9 😊)
Ruby 2.1 New syntax

• New syntaxes
  • Required keyword parameter
  • Rational number literal
  • Complex number literal
  • `def` returns symbol of method name

http://www.flickr.com/photos/rooreynolds/4133549889
Ruby 2.1 Syntax

Required keyword parameter

• Keyword argument (from Ruby 2.0.0)
  • def foo(a: 1, b: 2); end
  • `a` and `b` are optional parameters
  • OK: foo(); foo(a: 1); foo(a: 1, b: 2); foo(b: 2)

• Required keyword argument from 2.1
  • def foo(a: 1, b: )
  • `a` is optional, but `b` is required parameter
  • OK: foo(a: 1, b: 2); foo(b: 2)
  • NG: foo(); foo(a: 1)
Ruby 2.1 Syntax
Rational number literals

• To represent $\frac{1}{2}$, in Ruby “Rational(1, 2)”
  → Too long!!
• Introduce “r” suffix
  $\frac{1}{2} \rightarrow 1/2r$
• “[digits]r” represents “Rational([digits], 1)”
• $\frac{1}{2} \rightarrow 1/2r$
  • $1/2r$ # => 1/Rational(2, 1)
  • 1/Rational(2, 1) # => Rational(1/2)

Memory management tuning in Ruby,
RubyConfPH 2014 by K.Sasada
<ko1@heroku.com>
Ruby 2.1 Syntax
Complex number literals

- We already have “Integer#i” method to make imaginary number like “1+2.i”
- We already introduced “r” suffix for Rational → No reason to prohibit “i” suffix!!
- [digits]i represents “Complex(0, [digits])”
- 1+2i #=> 1+Complex(0, 2)
- 1+Complex(0, 2) #=> Complex(1, 2)

- You can mix “r” and “i” suffix
Ruby 2.1 Syntax

Return value of `def` syntax

• Return value of method definition
  • Method definition syntax returns symbol of defined method name
  • `def foo; ...; end` #=> :foo

• Method modifier methods
  • Example:
    • private def foo; ...; end
    • public static void def main(args); ...; end
Ruby 2.1 Runtime new features

• String#scrub
• Process.clock_gettime
• Binding#local_variable_get/set
• Bignum now uses GMP (if available)
• Extending ObjectSpace
Ruby 2.1 Runtime new features
Object tracing

• ObjectSpace. trace_object_allocations
  • Trace object allocation and record allocation-site
    • Record filename, line number, creator method’s id and class
  • Usage:
    ObjectSpace.trace_object_allocations{ # record only in the block
      o = Object.new
      file = ObjectSpace.allocation_sourcefile(o) #=> __FILE__
      line = ObjectSpace.allocation_sourceline(o) #=> __LINE__ -2
    }

Performance improvements

• Optimize “string literal”.freeze
• Sophisticated inline method cache
• Introducing Generational GC: RGenGC
RGenGC: Generational GC for Ruby

• RGenGC: Restricted Generational GC
  • Generational GC (minor/major GC uses M&S)
  • **Dramatically speedup for GC-bottleneck applications**
  • New generational GC algorithm allows mixing “Write-barrier protected objects” and “WB unprotected objects”
    → **No (mostly) compatibility issue** with C-exts

• Inserting WBs gradually
  • We can concentrate WB insertion efforts for major objects and major methods
  • Now, most of objects (such as Array, Hash, String, etc.) are WB protected
    • Array, Hash, Object, String objects are very popular in Ruby
    • Array objects using `RARRAY_PTR()` change to WB unprotected objects (called as Shady objects), so existing codes still works.
RGenGC
Performance evaluation (RDoc)

About x15 speedup!

* Disabled lazy sweep to measure correctly.
RGenGC
Performance evaluation (RDoc)

* 12% improvements compare with w/ and w/o RGenGC
* Disabled lazy sweep to measure correctly.
Ruby 2.2
Next version

http://www.flickr.com/photos/adafruit/8483990604
Schedule of Ruby 2.2

• Not published officially
• Schedule draft is available by Naruse-san
Ruby 2.2 schedule

2013/12
Ruby 2.1.0

2014/12/25
Ruby 2.2.0

We are here!

Rubyconf.PH
3/28, 29

RDRC
6/26, 27

Rubyconf.tw
4/25, 26

RubyKaigi
9/18, 19, 20

RubyConf
11/17, 18, 19

Events are important for
EDD (Event Driven Development) Developers
Ruby 2.2 (rough) schedule

- **2013/12**
  - Ruby 2.1.0

- **2014/12/25**
  - Ruby 2.2.0

- **Sep/2014**
  - Preview 1
  - Big feature freeze

- **Nov/2014**
  - Preview 2
  - Feature freeze

- **Dec/2014**
  - Critical Bug fix only

- **Release candidate**

---

Memory management tuning in Ruby,
RubyConfPH 2014 by K.Sasada
<ko1@heroku.com>
2.2 big features (planned)

• New syntax: not available now
• New method: not available now
• Internal
  • GC
    • Symbol GC (merged recently)
    • 2age promotion strategy for RGenGC
    • Incremental GC to reduce major GC pause time
• VM
  • More sophisticated method cache
Symbol GC

• Symbols remain forever → Security issue
  • “n.times{|i| i.to_s.to_sym}”
    creates “n” symbols and they are never collected
• Symbol GC: Collect dynamically created symbols
Garbage collection
The automatic memory management

Today’s main subject
From basic to advanced topics
Automatic memory management
Basic concept

- “Object.new” allocate a new object
  - “foo” (string literal) also allocate a new object
  - Everything are objects in Ruby!
- We don’t need to “de-allocate” objects manually
Automatic memory management
Basic concept

- Garbage collector recycled “unused” objects automatically
1st question

How to collect “unused” objects?
How to collect “unused” objects?

• Using (well-known) GC algorithm
  • Mark and sweep algorithm (from the first version of Ruby)
  • Generational GC algorithm (from Ruby 2.1)
Mark & Sweep algorithm

1. Mark reachable objects from root objects

2. Sweep unmarked objects (collection and de-allocation)

Collect unreachable objects
Generational GC (GenGC)

- Weak generational hypothesis:
  “Most objects die young”

→ Concentrate reclamation effort only on the young objects

Memory management tuning in Ruby, RubyConfPH 2014 by K.Sasada
<ko1@heroku.com>
Generational hypothesis

Object lifetime in RDoc
(How many GCs surviving?)

95% of objects dead by the first GC
Generational hypothesis

Object lifetime in RDoc
(How many GCs survive?)

Some type of objects (like Class) has long lifetime

Memory management tuning in Ruby,
RubyConfPH 2014 by K.Sasada
<ko1@heroku.com>
Generational GC (GenGC)

• Separate young generation and old generation
  • Create objects as young generation
  • Promote to old generation after surviving $n$-th GC
  • In CRuby, $n == 1$ (after 1 GC, objects become old)
• Usually, GC on young space (minor GC)
• GC on both spaces if no memory (major/full GC)
Generational GC (GenGC)

• Minor GC and Major GC can use different GC algorithm
  • Popular combination is:
    Minor GC: Copy GC, Major GC: M&S
  • On the CRuby, we choose:
    Minor GC: M&S, Major GC: M&S
  • Because of CRuby’s restriction (we can’t use moving algorithm)
GenGC [Minor M&S GC] (1/2)

- Mark reachable objects from root objects.
  - Mark and promote to old generation
  - Stop traversing after old objects
    → Reduce mark overhead
- Sweep not (marked or old) objects
- Can’t collect Some unreachable objects

Don’t collect old object even if it is unreachable.
GenGC [Minor M&S GC] (2/2)

• Mark reachable objects from root objects.
  • Mark and promote to old generation
  • Stop traversing after old objects
  \[\rightarrow\] Reduce mark overhead

• Sweep not (marked or old) objects

• Can’t collect Some unreachable objects

Don’t collect old object even if it is unreachable.

Root objects

2\textsuperscript{nd} MinorGC

Memory management tuning in Ruby, RubyConfPH 2014 by K.Sasada
<ko1@heroku.com>
GenGC [Major M&S GC]

• Normal M&S
• Mark reachable objects from root objects
  • Mark and **promote to old gen**
• Sweep unmarked objects
  • **Sweep all unreachable (unused) objects**
NOTE: Generational GC details

• Skip details of generational GC
  • Remember set
  • Write barrier
  • RGenGC techniques

• See my previous slides for details
  • [http://www.atdot.net/~ko1/activities/#idx4](http://www.atdot.net/~ko1/activities/#idx4)
2\textsuperscript{nd} question

“When” should we collect objects?
“When” collect objects?

1. Object space is full
2. Exceed limit of Malloc’ed memory size
3. User specified timing (GC.start, etc)

• (1) and (3) is easy to understand
• (2) needs more explanation
Exceed limit of Malloc’ed memory size

- When many memories are allocated by “malloc()”
- Introduce two variables
  - a counter “malloc_increase”
  - a threshold value “malloc_limit” (16MB)
- Rule
  - (1) Increase “malloc_increase” by malloc’ed size
  - (2) “malloc_increase” is reset at every GC time
    → “malloc_increase” represents “how many memory allocated (by malloc()) without GC”
- If “malloc_increase” > “malloc_limit”, then invoke GC to recycle malloc’ed objects
Exceed limit of Malloc’ed memory size

Invoke GC

Memory management tuning in Ruby,
RubyConfPH 2014 by K.Sasada
<ko1@heroku.com>
3\textsuperscript{rd} question

“What happen” when no space after GC?
What happen when no space after GC?

• Terminology
  • Total slots: total prepared object places
  • Living objects: Used objects

• GC detects “No Space” just after sweeping
  if [# of Total slots] * 0.7 < [# of Living objects]

• Allocate new space expand current space x1.8
What happen when no space after GC?

![Diagram showing the relationship between the number of total slots and the number of living objects after sweeping, with a condition that the number of living objects exceeds 0.7 times the number of total slots.](image-url)
Trade-off
Speed-Memory Trade-off

Performance vs. Memory usage

- Many GCs slow application performance
- Few GC increase memory consumption

http://www.flickr.com/photos/mcerasoli/6484117955/
Speed-Memory Trade-off

- Usually no problem
- On big production application, this can be an issue
Speed-Memory Trade-off

• Solution 1: Use big memory machine
Speed-Memory Trade-off

• Solution 1: Use big memory machine
  • Recent price of memory is very cheap
  • Heroku provides “PX: Performance dyno” (6GB)

Heroku XL: Focusing on Large Scale Apps

https://blog.heroku.com/archives/2014/2/3/heroku-xl
Speed-Memory Trade-off

• Solution 2: Find out good points
  • Choose good “GC tuning parameters”
GC tuning parameters
GC tuning parameters

• There are several GC tuning parameters
  • Specified by environment variables
    • Use like that: $ RUBY_GC_INIT_SLOTS=10000 ruby script.rb
  • Affect only launched time
GC tuning parameters

• How many GC parameters now?
  • Please raise your hand if you think it is:
    ① 3
    ② 7
    ③ 10
    ④ 11
    ⑤ 13
GC tuning parameters

• How many GC parameters now?
  • Please raise your hand if you think it is:
    ① 3 (ruby 1.9)
    ② 7
    ③ 10 (ruby 2.1.0)
    ④ 11 (ruby 2.1.1) ← Now!!
    ⑤ 13
GC tuning parameters (Ruby 2.1.1)

1. RUBY_GC_HEAP_INIT_SLOTS
2. RUBY_GC_HEAP_FREE_SLOTS
3. RUBY_GC_HEAP_GROWTH_FACTOR (new from 2.1)
4. RUBY_GC_HEAP_GROWTH_MAX_SLOTS (new from 2.1)
5. RUBY_GC_HEAP_OLDOBJECT_LIMIT_FACTOR (new from 2.1.1)
6. RUBY_GC_MALLOC_LIMIT
7. RUBY_GC_MALLOC_LIMIT_MAX (new from 2.1)
8. RUBY_GC_MALLOC_LIMIT_GROWTH_FACTOR (new from 2.1)
9. RUBY_GC_OLDMALLOC_LIMIT (new from 2.1)
10. RUBY_GC_OLDMALLOC_LIMIT_MAX (new from 2.1)
11. RUBY_GC_OLDMALLOC_LIMIT_GROWTH_FACTOR (new from 2.1)

• Obsolete
  • RUBY_FREE_MIN -> RUBY_GC_HEAP_FREE_SLOTS (from 2.1)
  • RUBY_HEAP_MIN_SLOTS -> RUBY_GC_HEAP_INIT_SLOTS (from 2.1)
GC_HEAP_INIT/FREE_SLOTS

• RUBY_GC_HEAP_INIT_SLOTS (default: 10000)
  • How many slots prepared at initialize

• RUBY_GC_HEAP_FREE_SLOTS (default: 4096)
  • At least how many slots are available after GC
  • free_min = max(RUBY_GC_HEAP_FREE_SLOTS, total_slots * 0.3)
RUBY_GC_HEAP_GROWTH_FACTOR (new from 2.1)

- RUBY_GC_HEAP_GROWTH_FACTOR (default: 1.8)
  - Growth factor of expanding object space
  - Grow object space exponentially to reduce GC time

```
# of living objects
# of total slots
free_min

total_slots = total_slots * RUBY_GC_HEAP_GROWTH_FACTOR
```

Memory management tuning in Ruby, RubyConfPH 2014 by K.Sasada
<k01@heroku.com>
GC_HEAP_GROWTH_MAX_SLOTS (new from Ruby 2.1)

• GC_HEAP_GROWTH_MAX_SLOTS (default: 0)
  • Stop exponential expanding, start linear expanding
  • The value “0” remove this cap
RUBY_GC_HEAP_OLDOBJECT_LIMIT_FACTOR (from Ruby 2.1.1)

- **RUBY_GC_HEAP_OLDOBJECT_LIMIT_FACTOR**
  - Default value: 2.0
  - Tuning major (full) GC frequency
    - Bigger value: rare, Smaller value: frequent
    - < 1.0: Every GC will be major (full) GC

Memory management tuning in Ruby, RubyConfPH 2014 by K.Sasada
<ko1@heroku.com>
RUBY_GC_MALLOC_LIMIT(…)

• RUBY_GC_MALLOC_LIMIT (default: 16MB)
  • Initial value of “malloc_limit”
  • Tuning GC frequency
    • Bigger: rare → High throughput, but consumes memory
    • Smaller: frequent → Low throughput, small memory

• RUBY_GC_MALLOC_LIMIT_MAX (default: 32MB)
  • Maximum value of “malloc_limit”

• RUBY_GC_MALLOC_LIMIT_GROWTH_FACTOR (default: 1.4)
  • Growth ratio of “malloc_limit”
RUBY_GC_MALLOC_LIMIT(...)

RUBY_GC_MALLOC_LIMIT_MAX (= 32MB)

RUBY_GC_MALLOC_LIMIT (= 16MB)

\[\text{malloc Limit} = \text{malloc Limit} \times \text{RUBY_GC_MALLOC_LIMIT_GROWTH\_FACTOR} (= 1.4)\]
RUBY_GC_OLDMALLOC_LIMIT(...)

• RUBY_GC_OLDMALLOC_LIMIT (default: 16MB)
• RUBY_GC_OLDMALLOC_LIMIT_MAX (default: 128MB)
• RUBY_GC_OLDMALLOC_LIMIT_GROWTH_FACTOR (default: 1.2)

• Similar to RUBY_GC_MALLOC_LIMIT(...), but parameter for major (full) GC timing
4th question

How to use tuning parameters?
How to use tuning parameters?

1. Profile your application
2. Try GC parameters (environment variables)
Profile memory management
GC.stat (MRI specific)

• “GC.stat” returns statistics information about GC
  • Counts
    • :count=>2,    # GC count
    • :minor_gc_count=>2,  # minor GC count
    • :major_gc_count=>0,    # major GC count
  • Current slot information
    • :heap_live_slot=>6836, #=> # of live objects
    • :heap_free_slot=>519, #=> # of freed objects
    • :heap_final_slot=>0,   #=> # of waiting finalizer objects
    • total_slots = heap_live_slot + heap_free_slot + heap_final_slot
  • Statistics
    • :total_allocated_object=>7674,    # total allocated objects
    • :total_freed_object=>838,           # total freed objects
    • Current living objects = total_allocated_object - total_freed_object

Memory management tuning in Ruby,
RubyConfPH 2014 by K.Sasada
<ko1@heroku.com>
Profile memory management
GC.latest_gc_info (MRI specific)

• “GC.latest_gc_info” returns details of latest GC
  • :gc_by=>:newobj    # why GC invoked?
    • newobj: no slots available
    • malloc: malloc_increase > malloc_limit
  • :major_by=>nil    # why major GC invoked?
  • :have_finalizer=>false    # have finalizer?
  • :immediate_sweep=>false    # immediate sweep?
Profile memory management
“gc_tracer” gem (MRI 2.1.0 later!!)

- GC::Tracer.start_logging(filename)
  - Save all GC.stat/GC.latest_gc_info results at every GC events into specified file
- GC events:
  - Start
  - End marking
  - End sweeping
Profile memory management “gc_tracer” gem

- Run your application with gc_tracer
- Plot with Excel!

http://www.flickr.com/photos/microsoftsweden/5394685465
Profile memory management “gc_tracer” gem

Memory management tuning in Ruby, RubyConfPH 2014 by K.Sasada
<ko1@heroku.com>
Profile memory management
“gc_tracer” gem

Memory management tuning in Ruby,
RubyConfPH 2014 by K.Sasada
<ko1@heroku.com>
Profile memory management
“gc_tracer” gem

Memory management tuning in Ruby,
RubyConfPH 2014 by K.Sasada
<ko1@heroku.com>
Profile memory management
“gc_tracer” gem

ruby 2.2 dev/RUBY_GC_HEAP_OLDOBJECT_FACTOR=2.0 (default)
Profile memory management “gc_tracer” gem

Ruby 2.2dev w/ RUBY_GC_HEAP_OLDOBJECT_FACTOR=1.3

Memory management tuning in Ruby,
RubyConfPH 2014 by K.Sasada
<ko1@heroku.com>
Try GC parameters

• General concept
  
  Speed <-> Memory trade-off

• You have huge memory

  → Increase parameters to improve performance

  • RUBY_GC_HEAP_INIT_SLOTS (initial slots)
  • RUBY_GC_HEAP_FREE_SLOTS (prepared free slots after GC)
  • RUBY_GC_MALLOC_LIMIT (reduce GC frequency)
Try GC parameters

- You have small memory

  **Reduce parameters to reduce memory usage**

  - IaaS, PaaS environments (ex: Heroku 1X dyno (512MB))
  - RUBY_GC_HEAP_GROWTH_FACTOR (heap expanding factor)
  - RUBY_GC_HEAP_OLDOBJECT_LIMIT_FACTOR (for more full GC)
    - If you have memory usage trouble when migrating from 2.0 to 2.1, please try to reduce this variable

Or you can try

Heroku 2X dyno (1GB) / PX dyno (6GB)!!
Try GC parameters

• There is no silver bullet
  • No one answer for all applications
  • You should not believe other applications settings easily

• Try and try and try!

http://www.flickr.com/photos/rowanbank/8483526808
See also

• Excellent blog articles by @tmm1
  • http://tmm1.net/

• Demystifying the Ruby GC by Sam Saffron
  • http://samsaffron.com/archive/2013/11/22/demystifying-the-ruby-gc

• Why I am excited about Ruby 2.1? by Sam Saffron
  • https://speakerdeck.com/samsaffron/why-ruby-2-dot-1-excites-me
  • http://vimeo.com/89491942
Summary of this talk

• New versions
  • Ruby 2.1 (released)
  • Ruby 2.2 (currently working on)
• Basic of Ruby’s memory management (GC)
• GC tuning parameters
  • “What” and “How” we can tune by GC parameters
Thank you for your attention
Q&A?

Koichi Sasada
<ko1@heroku.com>