Speedup Ruby Interpreter

Koichi Sasada
<ko1@heroku.com>
Today’s talk

• Ruby 2.1 and Ruby 2.2
• How to speed up Ruby interpreter?
  • Evaluator
  • Threading
  • Object management / Garbage collection
Koichi Sasada as a Japanese

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

• **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**
The Ruby Association was founded to further development of the programming language Ruby. The goals of the Ruby Association are to improve relationship between Ruby-related projects, communities and businesses, and to address issues connected with using Ruby in an enterprise environment.

Quoted from http://www.ruby.or.jp/en/
• Foundation to encourage Ruby dev. and communities

• Activities
  • Ruby programmer certification program
  • Grant project. We have selected 3 proposals in 2013
  • Ruby Prize
    • To recognize the efforts of “New members” to the Ruby community
  • Maintenance of Ruby (Cruby) interpreter
    • Now, it is for Ruby 2.0.0
  • Events, especially RubyWorld Conference
  • **Donation** for Ruby developments and communities
Ruby Association Certified Ruby Programmer

The Ruby Association Certified Ruby Programmer examinations are intended for engineers who design, develop, and/or operate Ruby-based systems, consultants who make Ruby-based system proposals, and instructors who teach Ruby. Those who are certified are recognized for their skills as Ruby engineers and as having high levels of Ruby-based system development capabilities. Those who pass the examination are certified by the Ruby Association as a Ruby Association Certified Ruby Programmer.

Registration of Ruby Association Certified Programmer (Prometric Site)

Overview and purposes of certification examinations

The overall purpose of the certification program is to:

1. Set a standard by which goals can be set for studying and teaching Ruby
2. Set a standard by which Ruby engineers can measure and prove their skill level
3. Set a decision-making standard for companies and other entities seeking to hire Ruby engineers or outsource development projects

The certification examinations are linked to the different sets of qualifications required for certification as a Ruby Association Certified Ruby Programmer, and there is a certification examination that corresponds to each set of qualifications. The Ruby Association will issue a certificate to those who pass the examinations.
The Ruby Prize Award 2014 now accepting nominations

It has been decided to hold the Ruby Prize2014, to recognize the efforts of New members to the Ruby community.

This "Ruby Prize" will hold meetings by the executive committee comprised of three parties, which is Ruby Association, Nihon Ruby no Kai and Matsue city.

Ruby Prize Award Winner and nominees will receive an award at the RubyWorld Conference 2014, to be held in Matsue, Shimane Prefecture November 13th & 14th

It should be noted the winner of the Ruby Prize will also be awarded sub-prize money of 1million yen!

See last year's Ruby Prize 2013

Ruby Prize winner Tomoyuki Chikanaga and finale nominees are celebrated at the RubyWorld conference.

Congrats!
プログラミング言語「Ruby」は、2013年2月にその開発から20年を迎えるとともに、5年ぶりのメジャーバージョンとしてRuby2.0がリリースされ、Rubyは新たな時代へと突入し、様々な場面での利用が拡がっています。

今年で6回目となるRubyWorld Conferenceを通じて、新しい普及の段階に突入しつつあるRubyが、多様な現実世界にどのように適合し、浸透していくのか、そのような普及過程と成長を考える機会を皆様に提供いたします。

Speedup Ruby interpreter, Koichi Sasada,
DeccanRubyConf2014
You should know about Heroku!!
Check [https://www.heroku.com/features](https://www.heroku.com/features)

Build and Run Your Apps, Your Way.

Heroku supports Ruby, Node.js, Python, Java, and PHP so you can use the languages you already know to build and deploy apps on Heroku. Learn more about our language support or sign up now.
• Heroku, Inc.  [http://www.heroku.com](http://www.heroku.com)

• Heroku supports OSSs / 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”
“Matz team”

- Matz @ Shimane
  - Title collector
- Nobu @ Tochigi
  - Patch monster
- ko1 @ Tokyo
  - EDD developer

Speedup Ruby interpreter, Koichi Sasada,
DeccanRubyConf2014
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
Nobu
Patch monster

- Great patch creator
Nobu is Great Patch Monster

COMMIT RATIO IN LAST 5 YEARS

- nobu 29%
- akr 12%
- svn 9%
- naruse 8%
- usa 4%
- ko1 4%
- drbrain 3%
- kosaki 3%
- tenderlove 2%
- matz 2%
- zzak 2%
- kazu 2%
- ko1 4%
- mame 2%
- usa 4%
- naruse 8%
- svn 9%

Speedup Ruby interpreter, Koichi Sasada, DeccanRubyConf2014
Ko1
EDD developer

Commit number of ko1 (last 3 years)

EDD: Event Driven Development
“Mission of Matz team”

Improve quality of next version of CRuby
“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

• Next version of CRuby is “Ruby 2.2.0”
Ruby 2.1
Current stable

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

• **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 2.1.2** was released at **2014/05/09**
  • Solves critical bugs (OpenSSL and so on)
Ruby 2.1 the biggest change

Version policy

• Change the versioning policy
  • Drop “patch level” in the version
  • **Major version**: Big language changes (or anniversary)
  • **Minor version**: minor language changes (or annually)
  • **Teeny version**: fixing bugs with compatibility
    • Release new teeny versions about every 3 month
    • Teeny upgrades keep compatibility
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

```ruby
def foo(a:1, b:)
  ...
end

foo(a: 1, b: 2)  # OK
foo()            # NG
foo(a: 1)        # NG
```

Speedup Ruby interpreter, Koichi Sasada,
DeccanRubyConf2014
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

1/2r #=> Rational(1, 2)
Ruby 2.1 Syntax
Rational number literals

• To represent \( \frac{1}{2} \), in Ruby “Rational(1, 2)”
  \( \rightarrow \) 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 \( \rightarrow \) 1/Rational(2, 1)
  • 1/Rational(2, 1) \( \rightarrow \) Rational(1/2)
Ruby 2.1 Syntax
Complex number literals

1+2i #=> Complex(1, 2)
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

Speedup Ruby interpreter, Koichi Sasada,
DeccanRubyConf2014
Ruby 2.1 Syntax
Return value of `def` syntax

def foo()
  ...
end
 #=> :foo
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

Speedup Ruby interpreter, Koichi Sasada,
DeccanRubyConf2014
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.

Speedup Ruby interpreter, Koichi Sasada,
DeccanRubyConf2014
Ruby 2.2
Next version

Speedup Ruby interpreter, Koichi Sasada,
DeccanRubyConf2014

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!

Events are important for

**EDD** (Event Driven Development) Developers

Speedup Ruby interpreter, Koichi Sasada,
DeccanRubyConf2014
Ruby 2.2 (rough) schedule

2013/12
Ruby 2.1.0

We are here!

26\textsuperscript{th}, Jul/2014
Dev. Meeting
Feature proposal

Aug/2014
Dev. Meeting
Feature proposal

Sep/2014
Preview 1
Big feature freeze

Nov/2014
Preview 2
Feature freeze

Dec/2014
Release candidate

2014/12/25
Ruby 2.2.0

Speedup Ruby interpreter, Koichi Sasada,
DeccanRubyConf2014
2.2 big features (planned)

• New syntax: not available now
• New method: no notable methods available now
• Libraries:
  • Minitest and test/unit will be removed (provided by bundled gem)
2.2 internal changes

- Internal
  - C APIs
    - Hide internal structures for Hash, Struct and so on
    - Remove obsolete APIs
  - GC
    - Symbol GC (merged recently)
    - 2age promotion strategy for RGenGC
    - Incremental GC to reduce major GC pause time
- VM
  - More sophisticated method cache
Ruby 2.2 internals
Symbol GC

1_000_000.times{|i| i.to_s.to_sym}
p Symbol.all_symbols.size

# Ruby 2.1
#=> 1,002,376

# Ruby 2.2 (dev)
#=> 25,412

Speedup Ruby interpreter, Koichi Sasada, DeccanRubyConf2014
Ruby 2.2 internals
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
Break

http://www.flickr.com/photos/donkeyhotey/8422065722

Speedup Ruby interpreter, Koichi Sasada,
DeccanRubyConf2014
Speedup Ruby Interpreter

How do we speed up Ruby interpreter?
Software consists of many components
Ruby’s components

- **Ruby script**
- **Parse**
- **Compile**
- **Ruby Bytecode**
- **Interpret on RubyVM**
- **Bundled Libraries**
- **Gem Libraries**
- **Threaded classes and methods** (Array, String, ...)
- **Evaluator**
- **Threading**
- **Object management (GC)**

Speedup Ruby interpreter, Koichi Sasada, DeccanRubyConf2014
Working for core components

• Core components I’m working for:
  • Evaluator (10 years)
  • Thread management (10 years)
  • Memory management (few years)
History of Ruby interpreter

1993 2/24
Birth of Ruby
(in Matz’ computer)

1995/12
Ruby 0.95
1st release

1996/12
Ruby 1.0

1998/12
Ruby 1.2

1999/12
Ruby 1.4

2000/6
Ruby 1.6

2003/8
Ruby 1.8

2009/1
Ruby 1.9.0

2013/2
Ruby 2.0

2013/3
RGenGC

2013/12
Ruby 2.1.0

2004/1
YARV development

Speedup Ruby interpreter, Koichi Sasada,
DeccanRubyConf2014
Introduce our effort (especially my contributions) to speedup Ruby interpreter
Evaluator

Ruby script

Parse

Compile

Bundled Libraries

Gem Libraries

Embedded classes and methods (Array, String, ...)

Threading

Evaluator

Object management (GC)

Interpret on RubyVM

Speedup Ruby interpreter, Koichi Sasada, DeccanRubyConf2014
Evaluator

• Named YARV: Yet another RubyVM
  • Start until 10 years ago (2004/01/01)
  • Simple stack machine architecture
  • Execute each bytecode instructions one by one

• Apply many known optimization techniques
Evaluator
Compile Ruby to AST

Ruby Program

a = b + c

Abstract Syntax Tree

Parse

a = Method Dispatch(:+)

b  c
Evaluator
Compile AST to Bytecode

Abstract Syntax Tree

a =

Method Dispatch(:+)

b

c

VM Instructions

getlocal b
getlocal c
send +
setlocal a

Tree data

Sequential data

Speedup Ruby interpreter, Koichi Sasada,
DeccanRubyConf2014
Evaluator
Execution as stack machine

Ruby Program
\[ a = b + c \]

YARV Instructions
- getlocal \( b \)
- getlocal \( c \)
- send \( + \)
- setlocal \( a \)

VM Stack
- \( b+c \)
- \( b \)
- \( c \)

Speedup Ruby interpreter, Koichi Sasada, DeccanRubyConf2014
Evaluator
Optimizations

• Apply many techniques to improve performance
  • Peephole optimizations
  • Specialized instructions
  • Stack frame layout
  • Efficient exception handling
  • Efficient block representation
  • Direct threading
  • Stack caching
  • Instructions and operands unifications
  • ...

Speedup Ruby interpreter, Koichi Sasada,
DeccanRubyConf2014
Speedup Ruby interpreter, Koichi Sasada, DeccanRubyConf2014
Evaluator
Optimizations

• Analysis usage
  And optimize for frequent cases
• Example: Exception handling
  • Exceptions occur *EXCEPTIONAL* so optimize for no-exception control flow
Performance evaluation compare with Ruby 1.8

Higher is good

Speedup Ruby interpreter, Koichi Sasada,
DeccanRubyConf2014
Main components

- Evaluator
- Thread management
- Memory management
Interpret on RubyVM

Threading

Ruby script

Parse

Compile

Parse

Bundled Libraries

Gem Libraries

Embedded classes and methods (Array, String, ...)

Threaded

Evaluator

Object management (GC)

Speedup Ruby interpreter, Koichi Sasada,
DeccanRubyConf2014
Threading

• Using native threads for each Ruby threads
• Parallel ruby execution is prohibited by GVL
  • You can free GVL if you write a code carefully in C level and run it in parallel
Threading
Ruby 1.8 and before

CPU 1
Thread 1
Thread 2
OS Thread 1

CPU 2
IDLE

Speedup Ruby interpreter, Koichi Sasada,
DeccanRubyConf2014
Threading
Layered view

Ruby

Native Thread
System S/W

Thread Scheduler

Processor(s)

S/W

H/W

Speedup Ruby interpreter, Koichi Sasada,
DeccanRubyConf2014
Threading
Ruby 1.9 and later

Native threads with Giant VM Lock

CPU 1
Thread 1
OS Thread 1
Lock
Thread 2
OS Thread 2
CPU 2
IDLE

Speedup Ruby interpreter, Koichi Sasada,
DeccanRubyConf2014
Threading
Layered view

Ruby

Native Thread
System S/W

Thread Scheduler

H/W

Processor(s)

S/W

Speedup Ruby interpreter, Koichi Sasada, DeccanRubyConf2014
Threading

Why GVL?

• To protect Ruby users from nightmare debugging
  • Shared parallel threading can make non deterministic bugs which is too hard to debug

• Disadvantage
  • CRITICAL ISSUE: No parallel programming in Ruby
  • Need another programming model for parallel
    • Current *SHARED EVERYTHING* model is not match
    • Correct isolation level for each parallel execution units
Object management (GC)

Interpret on RubyVM

- Ruby script
- Parse
- Compile
- Ruby Bytecode
- Threading
- Evaluator
- Embedded classes and methods (Array, String, ...)
- Bundled Libraries
- Gem Libraries
- Object management (GC)

Speedup Ruby interpreter, Koichi Sasada, DeccanRubyConf2014
Object and memory management

• “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
Garbage collection

The automatic memory management

![Garbage Collector](http://www.flickr.com/photos/circasassy/6817999189/)

---

*Fig. 109. — A Garbage Collector.*

http://www.flickr.com/photos/circasassy/6817999189/
Automatic memory management

Basic concept

• Garbage collector recycled “unused” objects automatically
Mark & Sweep algorithm

1. Mark reachable objects from root objects
2. Sweep \textit{unmarked} objects (collection and de-allocation)

Root objects

- marked
- free

Collect unreachable objects

Speedup Ruby interpreter, Koichi Sasada, DeccanRubyConf2014
Generational GC (GenGC)

- Weak generational hypothesis:
  
  “Most objects die young”

→ Concentrate reclamation effort only on the young objects

http://www.flickr.com/photos/ell-r-brown/5026593710
Generational hypothesis

Object lifetime in RDoc
(How many GCs surviving?)

95% of objects dead by the first GC

Speedup Ruby interpreter, Koichi Sasada,
DeccanRubyConf2014
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)
    • $n == 2$ from Ruby 2.2 (plan)

• Usually, GC on young space (minor GC)

• GC on both spaces if no memory (major/full GC)
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.
• 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 [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**

Root objects:

- Traverse new
- Traverse old
- Collect old

- Traverse new/free
- Traverse new/free
- Collect new/free

Speedup Ruby interpreter, Koichi Sasada,
DeccanRubyConf2014
RGenGC
Performance evaluation (RDoc)

About x15 speedup!

* Disabled lazy sweep to measure correctly.

Speedup Ruby interpreter, Koichi Sasada,
DeccanRubyConf2014
RGenGC
Performance evaluation (RDoc)

* 12% improvements compare with w/ and w/o RGenGC
* Disabled lazy sweep to measure correctly.
Speedup Ruby Interpreter

How to speed up Ruby interpreter?
DO EVERYTHING!
NO SILVER BULLET!

- **Ruby script**
- **Parse**
- **Compile**
- **Ruby Bytecode**
- **Bundled Libraries**
- **Gem Libraries**
- **Embedded classes and methods** (Array, String, ...)
- **Threading**
- **Evaluator**
- **Object management (GC)**

Interpret on RubyVM

DO EVERYTHING!
NO SILVER BULLET!
DO EVERYTHING!
NO SILVER BULLET!

We did.
We are doing.
We will do!!

Only continuous effort improves software quality.

Speedup Ruby interpreter, Koichi Sasada,
DeccanRubyConf2014
Future work: Many many many!!

• Evaluator
  • JIT compilation
  • More drastic optimizations

• Threading
  • Parallel execution model (not a thread?)

• Object management and GC
  • Incremental GC
  • Compaction GC
  • Lightweight object allocation
Summary

• Ruby 2.1 and Ruby 2.2
• How to speed up Ruby interpreter?
  • Evaluator
  • Threading
  • Object management / Garbage collection

One answers is:

#=> Continue software development
Thank you for your attention Q&A?

With slowly/clearly English, thank you.

Koichi Sasada
<ko1@heroku.com>