Gradual Write-Barrier Insertion into a Ruby Interpreter

Koichi Sasada
Cookpad Inc.

ISMM 2019, June 23, 2019
Summary

• Now Ruby interpreter (2.6, 2018) employed advanced GCs.
  • Generational GC from Ruby 2.1 (2013)
  • Incremental GC from Ruby 2.2 (2014)
  • Ruby 2.0 and before used naïve “M&S GC” algorithm

• Write barriers (WBs) were issue to introduce these GCs.
  • To keep compatibility, we are not able to introduce WBs for 3rd party C-extension libraries.

• Proposal: New concept: “WB unprotected object”
  • Giving up WB insertion completely, but mark “WB unprotected”
  • Invented at 2013 for Ruby 2.1.
  • We can introduced advanced GCs with keeping compatibility.

• Our approach allows Gradual WB development.
Background
Ruby language

• Ruby is Object-Oriented programming language
  • Developed by Yukihiro Matsumoto (1993~)
  • Developed actively.
    • Koichi is one of the Ruby committers working on VM, GC, Concurrency management and so on.

• Ruby on Rails web-application framework is used widely, in world-wide.

• Several Ruby interpreters are available.
  • “ruby” command written in C (target of this research)
  • JRuby, Truffle Ruby written in Java
  • mruby written in C, for embedded systems
Background

Ruby (Ruby on Rails) is used seriously.

- One of our service
  - 72 countries, 29 languages
  - Around 96 million monthly unique users
  - is written in Ruby language

→ Performance of Ruby has huge impact, at least on our business

(2019/03/31)
Background
GC before Ruby 2.1 (~2013)

• Mark and Sweep GC
  • M&S GC stops application long time.
  • This was one of reason why “Ruby is SLOW”.

• Conservative marking
  • Allows to write C implementation without special macros.
    • ex) Free to update references in C assignments.
      // New reference from an Array object to obj
      RARRAY_PTR(ary)[10] = obj;
  • Ruby supports C-extension libraries with this technique.
    • 3rd party can extend Ruby with C-extensions.
    • There are many C-extensions to support Ruby’s eco-system.
  • Moving is not allowed (mostly is acceptable)
Generational GC

- **GenGC** is well-known technique.
  - **Faster** than full GC because collecting only young objects.
- **GenGC** requires write-barriers
  - To detect “Old” to “Young” reference, **write-barriers (WBs)** should be introduced.
  - “**Completeness**” is required.
    - 1 oversight cause fatal error.
Problem: Inserting WBs

• Issue: Development cost
  • Practically, it is very difficult task to introduce WBs into Ruby code (250K lines in C) at once

• Issue: Compatibility
  • We need to re-write C code with WBs if needed.
  • We can not modify 3rd party C-extension libraries.
  • Drop old libs? vs. Give up GenGC?
    • If we need to rewrite all C-extensions, the update should be very difficult for existing Ruby users.
    • Make a new interpreter natively support good GC?
Background and Problem

- Ruby 2.0 (2013) needed Generational GC for speed.
- However, inserting write-barriers into C source code *completely is difficult* for huge ruby’s source code and *impossible* for 3rd party C-extensions.

Trade off between Speed and Compatibility
Proposal: WB unprotected objects

- **Introduce** **WB protected and unprotected attribute** for all objects
  - WB protected objects (**WBp**) can detect new reference creation from them. Unprotected objects (**WBunp**) can not.
  - GC algorithm need to care about WB unprotected objects.
- **Increase WB protected objects gradually.**
  - When we insert WBs into class **K** data structure, then all instance of class **K** are WB protected objects.
  - We can priorities WB insertion development
    → Flexible development
    - **Frequently used** data types (**Array, Hash, ⋯**) have high priority.
    - Scalar data types (**String, ⋯**) also have high priority because it is easy.
WB unprotect operation

- \( WBp \) can become \( WBunp \) by **WB unprotect operation**
  - If C code acquire internal data structure such as Array memory block, the Array object becomes \( WBunp \) because unexpected reference can be created by C code.

ex)

```c
// RARRAY_PTR() macro makes “ary” unprotect.
ptr = RARRAY_PTR(ary);
// This line creates new ref: ary→obj
// which GC can not detect.
ptr[10] = obj;
```
Generational marking

• Basic algorithm
  • Two generations: Young and Old
  • Objects have age 0~3 and age 3 is an old object.
  • Only generational marking (not generational sweeping)
  • Minor GC and Major (full) GC

NOTE: See our paper to refer complete algorithm
Generational marking with WB unprotected objects

• Additional Rule for $WB_{unp}$
  1. $WB_{unp}$ can not promote.
  2. If old objects refer to a $WB_{unp}$, then the $WB_{unp}$ is remembered **until next major GC** because $WB_{unp}$ can refer young objects.
  3. If Old objects become $WB_{unp}$ by $WB_{unp}Op$, it will be remembered.
Incremental marking with WB unprotected objects.

- At the end of normal incremental marking (3 color algorithm), mark all living (black) $WB_{unp}$ at once (not incremental).
  - This phase can introduce long pause time.
  - $O(n)$, $n$ is the number of living $WB_{unp}$.

NOTE: See our paper to refer complete algorithm
Implementation technique

Bitmap

• We introduce bitmap to represent $WBunp$.
• With this bitmap and marking bitmap, we can easily list “living $WBunp$” for incremental GC.

<table>
<thead>
<tr>
<th>$WBunp$ bit</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marking bit</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Living $WB$ unprotected object
Evaluation Measurements

• Several measurements
  • Microbenchmark
  • Application benchmark
    • RDoc
    • Ruby on Rails web application

• Environment
  • Intel(R) Core(TM) i7-6700 CPU, 64GB of memory, Ubuntu 18.04.2, gcc 7.3.0
  • ruby 2.7.0dev (2019-03-08 trunk 67194) x86_64-linux
Evaluation
Microbenchmark

```ruby
def make_linked_list n
  list = []
n.times{
    list = [list]
    # $prob is percentage
    # of WB unprotected objs.
    if rand(100) < $prob
      list.wb_unprotect
    end
  }
  list
end
```

# Create a long linked list
huge_list = make_linked_list(10_000_000)

# Create 100 M empty arrays
to invoke minor GC
100_000_000.times { [ ] }

We can control the ratio of WB unprotected arrays.
Evaluation Microbenchmark

Increasing $WBunp$ slows down the application.

Percentage of WB unprotected objects

(rightmost datapoints is "disabled" results with $\text{prob} == 0$)

Lower is better
Application benchmark
RDoc

• RDoc is document generation system
  • Reading ruby/c source code and generate formatted reference.
  • Source is ruby’s source code.

<table>
<thead>
<tr>
<th></th>
<th>Total time (s)</th>
<th>GC time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disabled</td>
<td>30.46</td>
<td>10.20</td>
</tr>
<tr>
<td>Enabled</td>
<td>22.57</td>
<td>1.63</td>
</tr>
</tbody>
</table>

The ratio of WBunp objects is 2%.
There are only few remembered WB unprotected objects
Application benchmark
RDoc (sampling per 1 GC)
Application benchmark
Discourse (Ruby on Rails web app) (response time percentile in milliseconds)

<table>
<thead>
<tr>
<th>Page</th>
<th>Disable</th>
<th>50%</th>
<th>75%</th>
<th>90%</th>
<th>99%</th>
</tr>
</thead>
<tbody>
<tr>
<td>categories</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enable</td>
<td>36</td>
<td>44</td>
<td>148</td>
<td>159</td>
<td></td>
</tr>
<tr>
<td>Enabled</td>
<td>35</td>
<td>36</td>
<td>52</td>
<td>87</td>
<td></td>
</tr>
<tr>
<td>home</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Disable</td>
<td>39</td>
<td>148</td>
<td>161</td>
<td>164</td>
<td></td>
</tr>
<tr>
<td>Enabled</td>
<td>39</td>
<td>42</td>
<td>56</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>categories_admin</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Disable</td>
<td>64</td>
<td>179</td>
<td>186</td>
<td>194</td>
<td></td>
</tr>
<tr>
<td>Enabled</td>
<td>63</td>
<td>70</td>
<td>82</td>
<td>147</td>
<td></td>
</tr>
<tr>
<td>home_admin</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Disable</td>
<td>71</td>
<td>180</td>
<td>186</td>
<td>194</td>
<td></td>
</tr>
<tr>
<td>Enabled</td>
<td>67</td>
<td>80</td>
<td>86</td>
<td>157</td>
<td></td>
</tr>
</tbody>
</table>

Generational GC improve performance (~90%)
Incremental GC is effective, but not enough (99%).
## Evaluation

### Gradual WB development

<table>
<thead>
<tr>
<th>WB implementation history</th>
<th>WB protected classes</th>
</tr>
</thead>
</table>
| Ruby 2.1 (2013)           | **Container types:** Array, Hash, Struct, Object (User defined classes), Class  
**Scalar types:** String, Range, Regexp, RubyVM::ISeq (bytecode) |
| Ruby 2.4 (2016)           | Proc (closure class), Env (local variables) (postponed to impl. them at 2013 **because it was difficult task**) |
| Ruby 2.5 (2017)           | Dir, Binding, Thread::Queue, Thread::SizedQueue and Thread::ConditionVariable |

- We can **give up** difficult WB insertions
  - Some kind of “Class” objects has complex relations and I can not remove a bug
  
→ Make them *WBunp* with WB unprotect operation
Related work

• TruffleRuby introduce special wrappers to support C-extension library [7]
  • Issue: We need two GCs
• Special C-preprocessor to auto-WB insertion [5]
  • Issue: False positive. Difficult to maintain.
• Using hardware memory protection to detect writing [3]
  • Issue: Portability problem (difficult to maintain)
• Scan all old spaces [1]
  • Issue: Scanning cost
Summary

- Now Ruby interpreter (2.6, 2018) employed **advanced GCs**.
  - **Generational GC** from Ruby 2.1 (2013)
  - **Incremental GC** from Ruby 2.2 (2014)
  - Ruby 2.0 and before used naïve “M&S GC” algorithm
- **Write barriers** (WBs) were issue to introduce these GCs.
  - To keep compatibility, we are not able to introduce WBs for 3rd party C-extension libraries.
- Proposal: New concept: “WB unprotected object”
  - Giving up WB insertion completely, but mark “WB unprotected”
  - Invented at 2013 for Ruby 2.1.
  - We can introduced advanced GCs with **keeping compatibility**.
- Our approach allows **Gradual WB development**.
Message to researchers

• Ruby interpreter is used by many people and the performance is still an issue.
• We (other Ruby committers and Cookpad) can help your research on Ruby.
• Please contact us if you have interest: ko1@atdot.net

Thank you for your attention!