Exploring Memory in Ruby

Building a Compacting GC
Aaron Patterson
@tenderlove
It's a wig!
Ruby Core && Rails Core
* Note: English speakers please ask me about this slide, it cannot be translated 💘💙💜💗💚❤💓💛💚💗
git push -f
Two Cats
Mark / Compact GC
Exploring Memory in Ruby

- Copy on Write
- Building a Compacting GC (in MRI)
- Memory Inspection Tools
Low Level
New People:
What is "Copy on Write"?
What is "Compaction"?
I can do this!
Experienced People:
Algorithms
Implementation Details
Copy on Write Optimization
What is CoW?
require 'objspace'

str = "x" * 9000
p ObjectSpace.memsize_of(str) # => 9041

str2 = str.dup
p ObjectSpace.memsize_of(str2) # => 40

str2[1] = 'l'

p ObjectSpace.memsize_of(str2) # => 9041

Initial String

No Copy

Copied "on write"
require 'objsmsgce'
array = ["x"] * 9000
p ObjectSpace.memsize_of(array) # => 72040
array2 = array.dup
p ObjectSpace.memsize_of(array2) # => 40
array2[1] = 'l'
array2[1] = 'l'
p ObjectSpace.memsize_of(array2) # => 72040

Initial Array

No Copy

Copied "on write"
require 'objspace'

hash = ('a'..'zzz').each_with_object({}) { |k,h| h[k] = :hello }

p ObjectSpace.memsize_of(hash) #=> 917600

hash2 = hash.dup

p ObjectSpace.memsize_of(hash2) #=> 917600
No Observable Difference
Operating System
`fork`
string = "x" * 90000
p PARENT_PID: $$
going
child_pid = fork do
  p CHILD_PID: $$
going
  string[1] = 'y'
going
end

Process.waitpid child_pid

---

Ruby Fork

Initial String

No Copy

Copied "on write"
OS Memory Copy

Parent Process

Child Process

xyxx
"CoW Page Fault"
Why is CoW Important?
Unicorn is a forking webserver
Unicorn Parent

Unicorn Child

Unicorn Child

Unicorn Child
Reduce Boot Time
Decreases Memory Usage
This is how it works today.
Reducing Page Faults
What causes page faults?
Mutating Share Memory
Garbage Collector
Object Allocation
Object Allocation

Ruby Objects

Parent Process Memory

Child Process Memory

Filled

Empty
How can we reduce this space?
GC compaction
Compact Before Fork

Ruby Objects

Parent Process Memory

Page 1

Page 2

Empty
Filled
Compact Before Fork

Ruby Objects

Parent Process Memory

Page 1

Child Process Memory

Page 2

Empty
Filled
GC Compaction
What is "compaction"?
Compaction

Ruby Objects

Parent Process Memory

Page 1

Page 2

Empty
Filled
Why compact?
Reduce Memory Usage
"Impossible"
Compaction Algorithms
Two Finger Compaction
Disadvantages

• It’s slow

• Objects move to random places
Advantage

• It’s EASY!
Algorithm

Object Movement

Reference Updating
Object Movement

Address: 1 2 3 4 5 6 7 8 9 a b
Heap: Free Free Free Obj Free 5 3 2 Free Free 1

Free Pointer
Done!
Scan Pointer
Reference Updating

Before Compaction

Ruby

```ruby
a = { c: 'd' }
```
Reference Updating

After Compaction

Address
1     2     3     4     5     6     7     8     9     a     b

Heap
Obj   Obj   Obj   Obj   Obj   5     3     2     Free    Free    1

Ruby

```ruby
a = { c: 'd' }
```
Reference Updating

After Compaction

Ruby

```
a = { c: 'd' }
```
Reference Updating

After Compaction

Address

1 2 3 4 5 6 7 8 9 a b

<table>
<thead>
<tr>
<th>Heap</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>Obj</th>
<th>Obj</th>
<th>Obj</th>
<th>Obj</th>
<th>Free</th>
<th>Free</th>
<th>Free</th>
<th>Free</th>
<th>Free</th>
<th>Free</th>
</tr>
</thead>
</table>

Ruby

\[
a = \{ \text{c: 'd'} \}
\]
Done!!
Implementation Details
Code:

https://github.com/github/ruby/tree/gc-compact
GC.compact
Usage

# Parent unicorn process
load_all_of_rails_and_dependencies
load_all_of_application

GC.compact

N.times do
  fork do
    # Child worker processes
    # handle requests
  end
end
Changes to gc.c
```
gc_move
```

<table>
<thead>
<tr>
<th>Address</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap</td>
<td>Free</td>
<td>Obj</td>
<td>1</td>
</tr>
</tbody>
</table>
`T_MOVED`
`gc_compact_heap`

<table>
<thead>
<tr>
<th>Address</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap</td>
<td>Free</td>
<td>Obj</td>
<td>Obj</td>
</tr>
</tbody>
</table>

💡💡💡

👉 👈

👉 👈

👉 👈
`gc_update_object_references`
Reference Update Helpers

- gc_ref_update_array
- gc_ref_update_object
- hash_foreach_replace
- gc_ref_update_method_entry
- ..... etc
`pinned_bits[];`
What objects can move?
What can move?

- Everthing
Finding References
class Foo
  def initialize obj
    @bar = obj
  end
end

class Bar
end

bar = Bar.new
foo = Foo.new(bar)
class Bar
end

bar = Bar.new
foo = Foo.new(bar)
C References

class Bar
end

bar = Bar.new
foo = Foo.new(bar)

rb_gc_mark(
  T_MOVED
)
class Bar
end

bar = Bar.new
foo = Foo.new(bar)

rb_gc_mark(Bar)

Cannot update
rb_gc_mark

1. Mark the object
2. Pin the object in `pinned_bits` table
1. Full GC (so objects get pinned)
2. Compact objects
3. Update references
What can move?

- Everthing
- Except objects marked with `rb_gc_mark`
Movement Problems
Hash Tables
Hashing

\[
\text{hash_key( Object )} = \text{memory address}
\]
Fix: cache hash key
What can move?

- Everthing
- Except objects marked with `rb_gc_mark`
- and hash keys
Dual References
Dual References

Foo \rightarrow C \rightarrow T\_MOVED \rightarrow Baz
Dual References
Fix: Call rb_gc_mark, or use only Ruby

https://github.com/msgpack/msgpack-ruby/pull/135
What can move?

• Everything

• Except objects marked with `rb_gc_mark`

• and hash keys

• and dual referenced objects
Global Variables
Global Variables (in C)

VALUE cFoo;

void Init_foo() {
    cFoo = rb_define_class("Foo", rb_cObject);
}

Fix: use heuristics to pin objects
What can move?

- Everthing
- Except objects marked with `rb_gc_mark`
- and hash keys
- and dual referenced objects
- and objects created with `rb_define_class`
String Literals
def foo
  puts "hello world"
end
Updating bytecode is hard
What can move?

- Everything
- Except objects marked with `rb_gc_mark`
- and hash keys
- and dual referenced objects
- and objects created with `rb_define_class`
- and string literals
It seems like nothing can move
Most can be fixed
46% can move!
Inspecting Memory
ObjectSpace.dump_all
require "objspace"

File.open("out.json", "w") { |f|
  ObjectSpace.dump_all(output: f)
}
Measuring Rails Boot

$ RAILS_ENV=production \
bin/rails r \
'require "objspace"; GC.compact; File.open("out.json", "w") { |f| 
  ObjectSpace.dump_all(output: f)
}'}
Output

{"address": "0x7fcc6e01a198", "type": "OBJECT", "class": "0x7fcc6c93d420", "ivars": 3,
"references": ["0x7fcc6e01bed0"], "memsize": 40, "flags": {"wb_protected": true, "old": true,
"uncollectible": true, "marked": true}}
```json
{
    "address": "0x7fcc6e01a198",
    "type": "OBJECT",
    "class": "0x7fcc6c93d420",
    "ivars": 3,
    "references": [
        "0x7fcc6e01bed0"
    ],
    "memsize": 40,
    "flags": {
        "wb_protected": true,
        "old": true,
        "uncollectible": true,
        "marked": true
    }
}
```
Address = Location
Heap Fragmentation

- Object
- Empty
Heap Fragmentation
Heap Fragmentation

- Pinned
- Moves
- Empty
Heap Fragmentation

- Pinned
- Moves
- Empty
https://github.com/tenderlove/heap-utils
Inspecting CoW Memory
/proc/{PID}/smaps
Address Range

55a92679a000-55a926b53000 rw-p 00000000 00:00 0 [heap]

Size: 3812 kB
Rss: 3620 kB
Pss: 3620 kB
Shared_Clean: 0 kB
Shared_Dirty: 0 kB
Private_Clean: 0 kB
Private_Dirty: 3620 kB
Referenced: 3620 kB
Anonymous: 3620 kB
AnonHugePages: 0 kB
Shared_Hugetlb: 0 kB
Private_Hugetlb: 0 kB
Swap: 0 kB
SwapPss: 0 kB
KernelPageSize: 4 kB
MMUPageSize: 4 kB
Locked: 0 kB

RSS & PSS
RSS

Shared_Clean + Shared_Dirty + Private_Clean + Private_Dirty
PSS

\[
\frac{\text{Shared}_\text{Dirty}}{\text{Number of Processes}} + \text{Shared}_\text{Clean} + \text{Private}_\text{Clean} + \text{Private}_\text{Dirty}
\]
## RSS vs PSS

<table>
<thead>
<tr>
<th></th>
<th>RSS</th>
<th>PSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unicorn Parent</td>
<td>3620 kB</td>
<td>1840 kB</td>
</tr>
<tr>
<td>Unicorn Child</td>
<td>3620 kB</td>
<td>1840 kB</td>
</tr>
</tbody>
</table>

Total Usage is 3620 kB **not** 7240 kB
Copying Memory

```ruby
x = "x" * 9000
p PID: $$
gets

child_pid = fork do
  puts "forked"
  9000.times do |i|
    puts("I: #{i}") || gets if i % 1000 == 0
    x[i] = 109.chr
  end
  puts "done"
  gets
end

Process.waitpid child_pid
```
Shared_Dirty, PSS, RSS

The graph shows the memory usage in Kb for Shared_Dirty, PSS, and RSS as a function of the number of writes. The memory usage decreases with the number of writes for Shared_Dirty and PSS, while RSS remains relatively stable.
Compaction Impact

```ruby
p PID: $$
arry = []
GC.start

gets

GC.compact if ENV["COMPACT"]
child_pid = fork do
  pages = GC.stat(:heap_allocated_pages)
  while pages == GC.stat(:heap_allocated_pages)
    arry << Object.new
  end
  puts "done"
  gets
end

Process.waitpid child_pid
```
No Compaction PSS: 2684Kb
Compaction PSS: 2530Kb
Compaction Savings: 154Kb
Conclusion
Compaction Savings: Unknown
Use `ObjectSpace`
/proc/{PID}/smaps
Why compact?
"Impossible"
Question Your Assumptions
We’ve entered grass
Thank you!