How Do We Look for a Memory Leak in Rails Apps

by Tito Pandu Brahmanto
KMK-0619

2016-07-26

- Alert pada Vidio instance memory yang penuh. Hal ini terjadi setelah deploy di hari sebelumnya dan dalam kurun waktu 24 jam. Penggunaan memory naik terus menerus hingga lebih dari 4 GB untuk proses Ruby.

- Investigasi awal dilakukan melalui mini-profiler, pada investigasi ini, kami hanya menemukan besarnya penggunaan memory oleh Thumbor.

2016-08-03

- Dari hasil research, kami melihat ada beberapa gem pada Gemfile.lock yang berpotensi memory leak, kami mengupgrade beberapa gem tersebut, namun hasilnya… sama saja, tetap **leak**

- https://github.com/ASoftCo/leaky-gems

2016-08-12

- Optimize Thumbor dilakukan berdasarkan hasil di mini-profiler.

2016-08-15

- Update dalli, melihat penggunaan memory yang selalu berada di tempat pertama di beberapa request.

2016-08-25

- Update Ruby ke 2.3.1

2016-08-29

- Mencoba hanya membuat url Thumbor satu kali dalam Desktop Picture

2016-09-08

- Revert commit yang membuat memory leak, sejak deploy revert ini, penggunaan memory vidio kembali normal

Bagaimana cara menemukannya?

- Kata Tommy, “use scientific method”

- Reproduce environment production di mesin development menggunakan Vagrant (database, thumbor setting, tunnel ke lateral production, Ruby version)

- Record dan replay request di production menggunakan GOR - https://goreplay.org

- Revert **commit saat deploy, satu persatu**

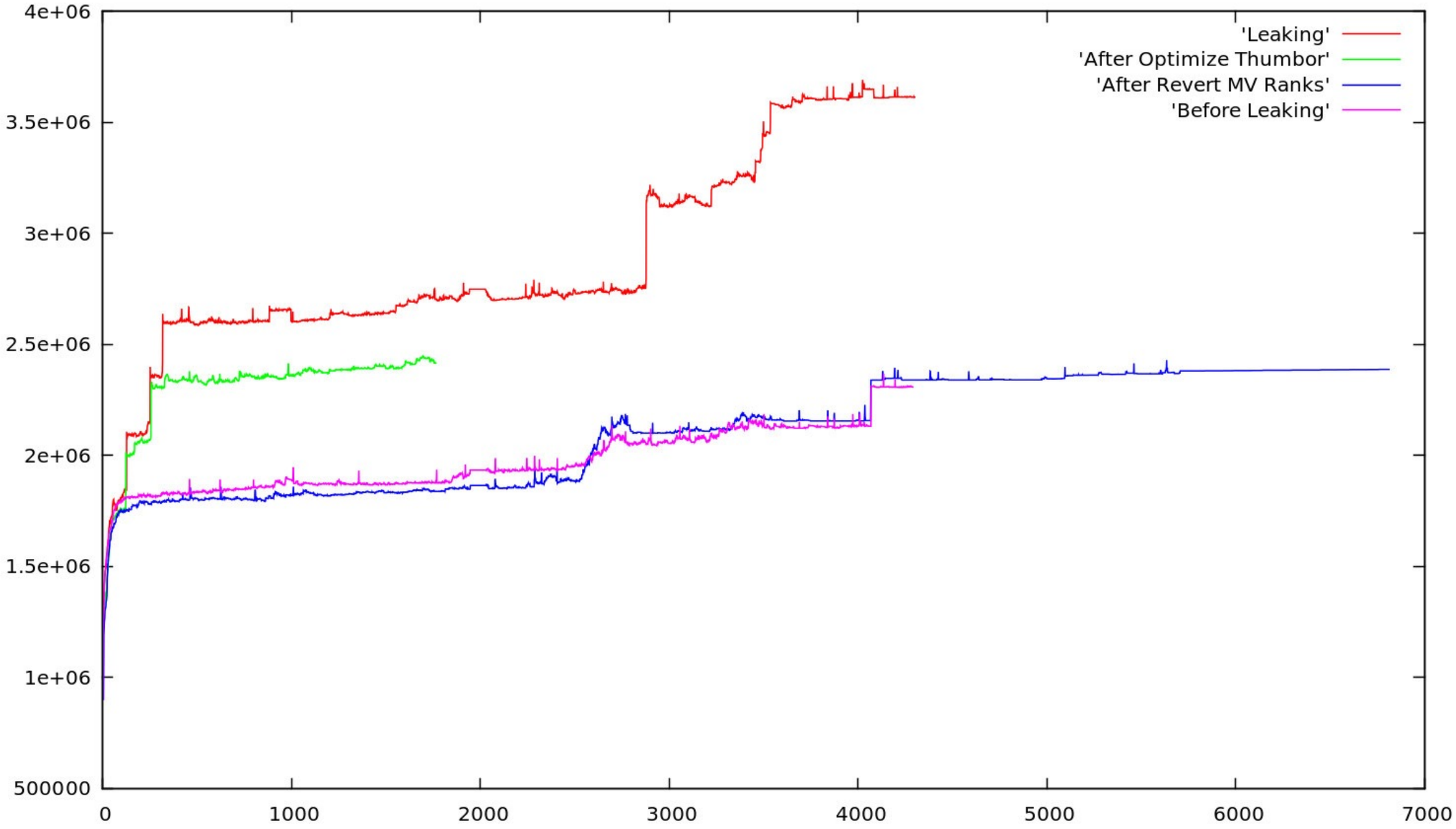
- Record memory usage dari Puma

- Buat grafiknya

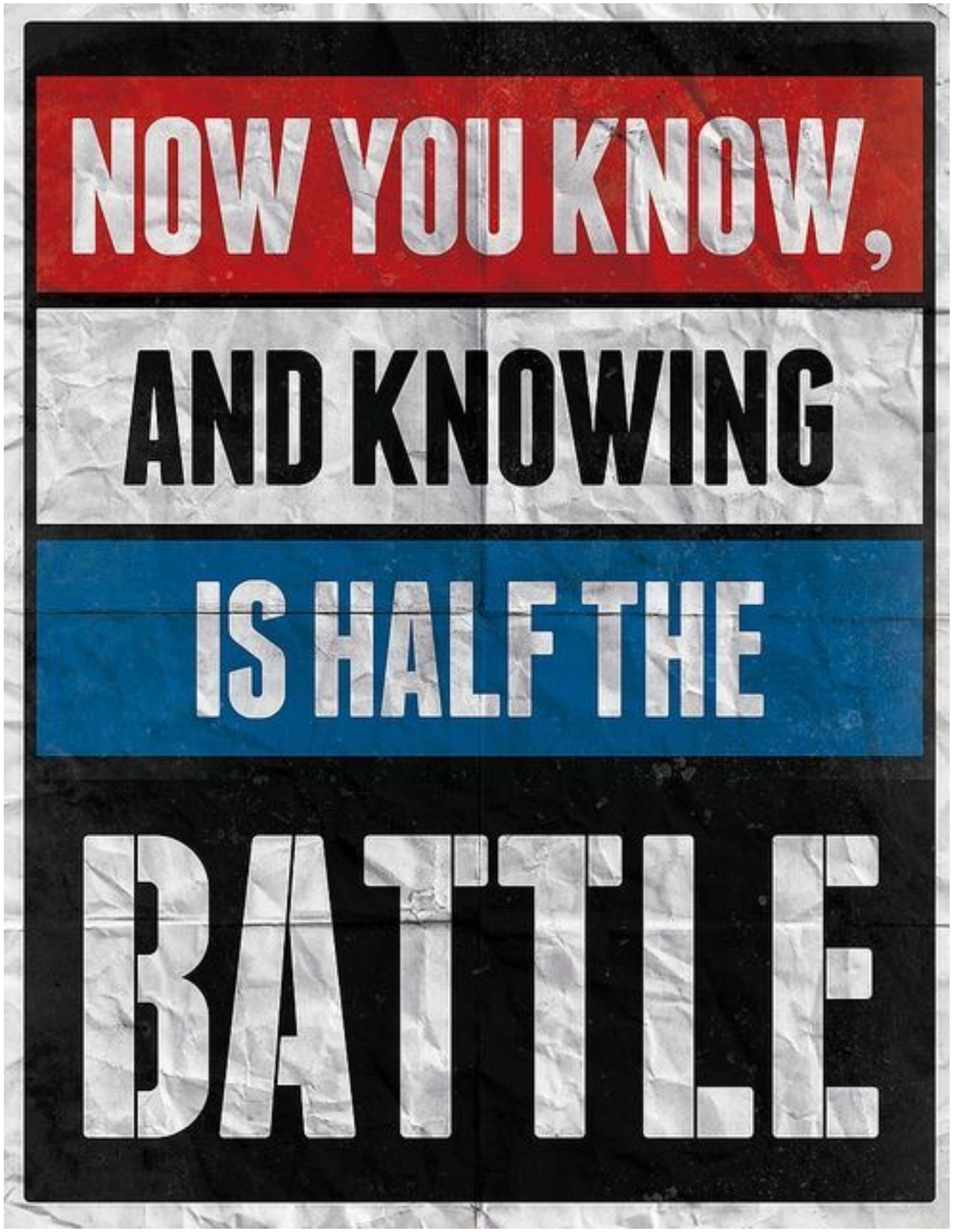
Puma RSS Monitor

```
#!/bin/bash

if [ $# -ne 1 ]; then
  echo "Usage: puma_rss_monitor file_output"
else
  echo "start monitoring..."
  while true
  do
    cat /tmp/puma.vidio.pid | xargs ps -o pid h --ppid | xargs ps -o rss h | paste -s -d+ | bc >> $1
    sleep 10
  done
fi
```


'Leaking'
'After Optimize Thumbor'
'After Revert MV Ranks'
'Before Leaking'
4e+06
3.5e+06
3e+06
2.5e+06
2e+06
1.5e+06
1e+06
500000
0
1000
2000
3000
4000
5000
6000
7000

Grafik


NOW YOU KNOW,
AND KNOWING
IS HALF THE
BATTLE

2016-09-22

- Mengembalikan commit yang di-revert sebelumnya dan merubah 1 line yang bermasalah.

Grand Commit

```
diff --git a/app/controllers/ranks_controller.rb b/app/controllers/ranks_controller.rb
index b3feae6..c5a5214 100644
--- a/app/controllers/ranks_controller.rb
+++ b/app/controllers/ranks_controller.rb
@@ -28,7 +28,7 @@ class RanksController < ApplicationController
   def video_ranks
     cache_key = "ranks-controller/show/#{params[:id]}/#{params[:type]}/#{params[:time]}"
     Rails.cache.fetch(cache_key, expires_in: 12.hours, race_condition_ttl: 60) do
-      if @category.try(:videos).present?
+      if @category.try(:videos)
         @category.videos.send(type, time)
       else
         if type == "ranking" && time == :last_7_days
```

Bagaimana cara menemukannya?

- Kata Tommy, “use scientific method”
- Reproduce environment production di mesin development menggunakan Vagrant (database, thumbor setting, tunnel ke lateral production, Ruby version)
- Record dan replay request di production menggunakan GOR - https://goreplay.org
- ~~Revert **commit saat deploy, satu persatu**~~
- Revert **line by line di commit yang bermasalah**
- Record memory usage dari Puma
- Bikin grafiknya Pek

Memory Profiler

With Present (LEAK!!):

Total allocated:
382.610.941 bytes
(3.048.127 objects)

Total retained:
327.607.064 bytes
(2.731.695 objects)

Without Present (Safe):

Total allocated:
1.264.030 bytes
(8.947 objects)

Total retained:
217.725 bytes
(527 objects)

Mengapa begitu?

ActiveRecord::Relation#present?

Object#present?

! ActiveRecord::Relation#blank?

to_a.blank?

https://github.com/rails/rails/blob/4-2-stable/activerecord/lib/active_record/relation.rb#L621-L623

Lesson Learned

- Jangan pernah pakai **:present?** untuk **ActiveRecord::Relation**, jika tidak menggunakan limit
- “Use scientific method”, by Tommy
  - Pakai environment yang sama
  - Rubah satu bagian
  - Record data-nya
  - Repeat
- Tidak ada effort yang sia-sia, dengan kita meng-optimasi code Vidio, meskipun tidak bisa menghilangkan memory leak ini, namun hal tersebut justru menghilangkan memory leak yang terjadi saat upgrade Ruby 2.3.1 (yang saat itu di-revert kembali menjadi 2.1.5).

Thank you

No Demo