The Glib-Glorgox IV Spy's Guide to Sabotaging Rails Application Performance

An introduction for cadets assigned to target "Earth"

Lieutenant Nate Berkopec / 2023

Our mission:
To make Ruby on Rails applications as slow as possible

My role

~25 quick tips for Rails app speed sabotage

Web app performance is engineering

1. Observe
2. Hypothesis
3. Experiment
4. Learn

Sabotage the scientific method

1. Observe
2. Hypothesis
3. Experiment
4. Learn

Can’t Be Slow If We Don’t Know

Measure ~~once~~ never,
Cut ~~twice~~
as many times as
the client will pay for

No RUM

“Real User Monitoring”

Web applications run in the browser

No RUM

90% of web app load time occurs after first byte

No RUM

Rails developers hate JS, and are happy to ignore it

No RUM

No APM

“APMs are expensive”

No APM

“No one looks at them”

No APM

Hide logs in dev

Logs in a terminal window provides a visual N+1 indicator

Hide development logs

verbose_query_logs provides valuable N+1 info

Hide development logs

Human readable logs are easy to understand

Hide development logs

STDOUT logging is highly visible

Hide development logs

Track only background job execution, not queueing

The time between user input and end state involves time spent in queue and time spent executing

Don’t track job queueing

90% of the delay between user input and end state is spent waiting in the queue

Don’t track job queueing

Background job queue time isn’t tracked by default

Don’t track job queueing

Request queueing is the most important scaling metric

Request queueing is the time from load balancer to request processing

Don’t track request queueing

CPU-utilization-based autoscaling doesn’t work

Don’t track request queueing

CPU-utilization-based autoscaling assumes I/O % is fixed

Don’t track request queueing

P95-based autoscaling doesn’t work

Don’t track request queueing

P95-based autoscaling assumes changes in response time are only caused by req queue changes

Don’t track request queueing

Without request queueing, time to respond is unknown

Don’t track request queueing

Tune your garbage collector

Gains are limited, but potential for loss is huge

Tune your garbage collector

The best way to reduce GC is to stop allocating so many objects

Tune your garbage collector

GC tuning can't accomplish much

Tune your garbage collector

Use “worker killers” and don’t track how often they run

The first request is the most expensive

Use “worker killers”

Old processes are fast processes

Use “worker killers”

Restarting 1x a minute can reduce performance by 50%

Use “worker killers”

1. Observe
2. Hypothesis
3. Experiment
4. Learn

~~Sharp Tools~~

No Tools

No profiling in production

Profiling provides information about what % of time was spent where

No profiling in production

Production is where all the really interesting things happen

No profiling in production

rack-mini-profiler doesn't work by default in prod

No profiling in production

No SPA
No Turbo

SPA/Turbo allows browser to “re-use” the expensive JS VM, DOM, CSS

No SPA or Turbo

SPA/Turbo is a 66% + reduction in subsequent navigation time

No SPA or Turbo

Keep data in dev small and limited

Reproducing prod problems requires prod-like data

No prod-like data in dev

Anonymizing prod is difficult

No prod-like data in dev

memory-profiler tracks mem allocations

Allocation is expensive

No memory-profiler

memory-profiler provides exact line numbers, files

No memory-profiler

Use the standard glibc malloc

Jemalloc can reduce memory use by 10%+

Use standard C malloc

jemalloc is production proven

Use standard C malloc

Jemalloc takes a few minutes to set up

Use standard C malloc

Make network calls mid-response with no timeouts or circuit breaks

Circuit breakers “shut off” when a codepath fails too many times

3rd party I/O w/o circuit breakers

Without fast failure, systems spin out of control

3rd party I/O w/o circuit breakers

1. Observe
2. Hypothesis
3. Experiment
4. Learn

Old McDonald had a Feature Farm

E-I-E-I-UH-OH
You didn’t ship enough features so now you’re stuck as an L3 for another year

Don’t set aside time or incentives for maintenance

No one ever got promoted for keeping the lights on

No maintenance time

Tasks fill to fit the space allotted to them

No maintenance time

Perf makes money

Slow sites are frustrating sites

Perf doesn’t make money

Direct relationship with conversion rates (1-10% per second)

Perf doesn’t make money

Slow sites can only serve small customers

Perf doesn’t make money

Performance is engineering, so never specify requirements

Can’t be “in the red” if we never define what red is

No specific targets

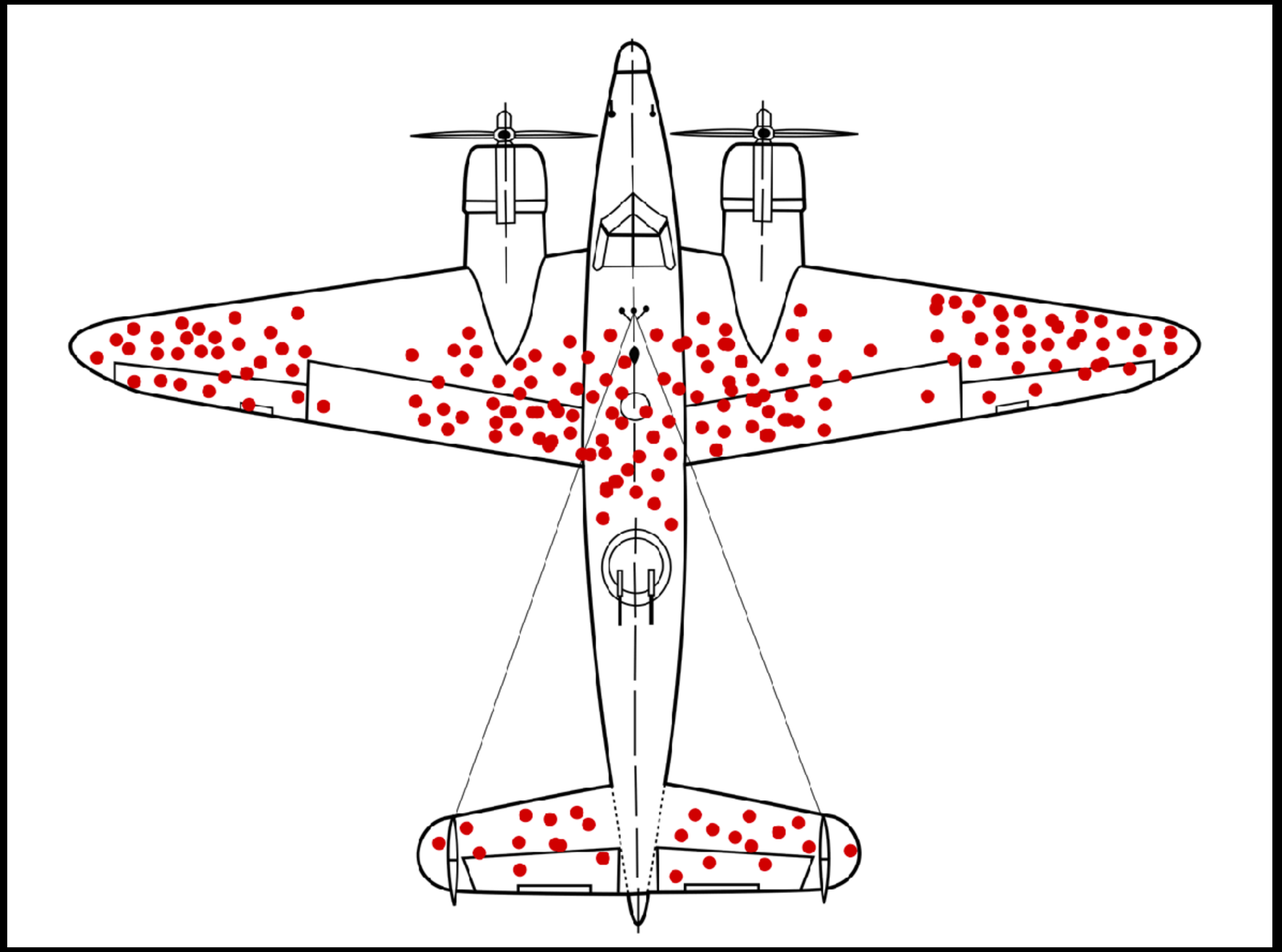
It’s difficult to be data-driven

No specific targets

Making trade-offs against perf is difficult w/o a target

No specific targets

We already have monitoring: customer complaints



No monitors

Monitors allow other work to be done when they're green

No monitors

1. Observe
2. Hypothesis
3. Experiment
4. Learn

If you can’t stop improving, at least you can forget how

Don’t show the before and after

Local benchmarks allow code reviewers to quickly understand impact of a PR

Don’t show before/after

Datadog Notebooks show before/after in prod

Don’t show before/after

The GVL is witchcraft

Multithreading improves resource utilization by 10->100%

Ignore the GVL

Configuring too many threads increases latency by 10->100%

Ignore the GVL

No horizontal scaling

Share-nothing architectures allow easy horizontal scaling

No horizontal scaling

Databases require vertical scaling, horizontal more difficult

No horizontal scaling

Use many technologies

Modern software is incredibly complex

Use many technologies

We cannot optimize what we do not understand

Use many technologies

Sabotage the scientific method

1. Observe
2. Hypothesis
3. Experiment
4. Learn

@nateberkopec
mastodon.social
speedshop.co
Books, courses, training

How To Have The Slowest Rails App

- Don’t Measure
  - No frontend RUM
  - No APM
  - Hide the dev logs
  - Track job exec time, not queue time
  - Autoscale based on CPU utilization
  - Tune your GC
  - Use worker killers, no restart tracking
- Don’t Use Tools
  - No RMP in production
  - No Turbo/SPA
  - Dev data only in dev
  - Copy big objects in memory
  - Use stdlib c allocator
  - Do 3rd party I/O w/no circuit breaker
- Don’t Experiment
  - No incentive for maintenance
  - Perf doesn't make money
  - No requirements
  - No monitors
- Don’t Learn
  - Don't track before/after effects
  - Don't understand the GVL
  - No horizontal scaling
  - Use many technologies

@nateberkopec