

Pruebas de carga



Curso de escalabilidad v2, día 2

Programa

Uso de herramientas

Herramientas a medida

Variabilidad y percentiles

Leyes de Pareto y Amdahl

Incertidumbre y error

Uso de herramientas



Apache ab

Parte del paquete Apache

```
$ ab -n 1000 -c 10 http://service.pinchito.es:3000/a
```

Opciones:

- `-n [N]`: número total de peticiones
- `-c [C]`: concurrencia, número de hilos
- `-t [T]`: tiempo de prueba

wrk

Escrito por [Will Glozer](#)

Diseño multi-hilo

```
$ wrk -t10 -c400 -d30s http://service.pinchito.es:3000/a
```

Opciones:

- `-t [T]`: usa T hilos (threads)
- `-c [C]`: número de conexiones abiertas
- `-d [D]`: duración de la prueba (con unidades)

wrk2

Forqueado por [Gil Tene](#)

Añade una nueva opción --rate o -R

```
$ wrk -t10 -c400 -d30s --rate 2000 http://service.pinchito.es:3000/a
```

Opciones:

- --rate [R]: tasa de peticiones por segundo
- --latency: muestra latencia con percentiles

Modos de carga

Modo "amable": espera a que termine una petición
antes de lanzar otra

Simula sistemas batch

Modo "agresivo": lanza R peticiones por segundo
incluso aunque no hayan terminado las anteriores

Simula sistemas online

Un servicio mal diseñado aguantará mucho menos
en el modo agresivo que en el amable

modo amable

req								
-----	-----	-----	-----	-----	-----	-----	-----	-----

req							
-----	-----	-----	-----	-----	-----	-----	-----

modo agresivo

req								
-----	-----	-----	-----	-----	-----	-----	-----	-----

req								
-----	-----	-----	-----	-----	-----	-----	-----	-----

The diagram consists of several overlapping rectangular boxes, each containing the word "req". The boxes are arranged in a staggered, overlapping pattern across three horizontal rows. The top row has two boxes. The middle row has four boxes. The bottom row has five boxes. All boxes are identical in size and shape.

req	req			
req	req	req	req	
req	req	req	req	req

loadtest

Autor principal: [vuestro humilde servidor](#)

Contribuciones de 36 devs

Modo agresivo: --rps

```
$ loadtest -n 1000 -c 10 --rps 2000 --keepalive http://service.pinchito.es:3000/a
```

Opciones:

- -n N: número total de peticiones
- -c C: concurrencia
- -t T: tiempo en segundos
- --rps R: peticiones sostenidas por segundo

autocannon

Autor: [Matteo Collina](#), miembro del TSC de Node.js

Inspirado por wrk, wrk2

Modo "agresivo": -R

```
$ autocannon -c 400 -d 10 -R 2000 http://service.pinchito.es:3000/a
```

Opciones:

-c C: conexiones

-d s: duración en segundos

-R R: peticiones por segundo

Ejercicio: Pruebas de carga

Crea una instancia en AWS EC2

Región eu-west-3, Europe (Paris)

Tipo t2-small (o superior)

Imagen **pinchito-loadtest-2020-11-08**

Lanza pruebas contra <http://service.pinchito.es:3000/a>

Usa ab, wrk2, loadtest, autocannon

Comprueba si la frecuencia "natural" es la misma



Ejercicio +

Lanza pruebas de carga a tasa de RPS constante

```
$ autocannon -R [R] ...
$ loadtest --rps [R] ...
```

Usa una tasa algo por encima y por debajo de la frecuencia natural

Comprueba si los resultados son consistentes

Si varían, ¿por qué lo hacen?



Ejercicio +

Ahora prueba contra <http://service.pinchito.es:3000/d>

¿Cambian los resultados?

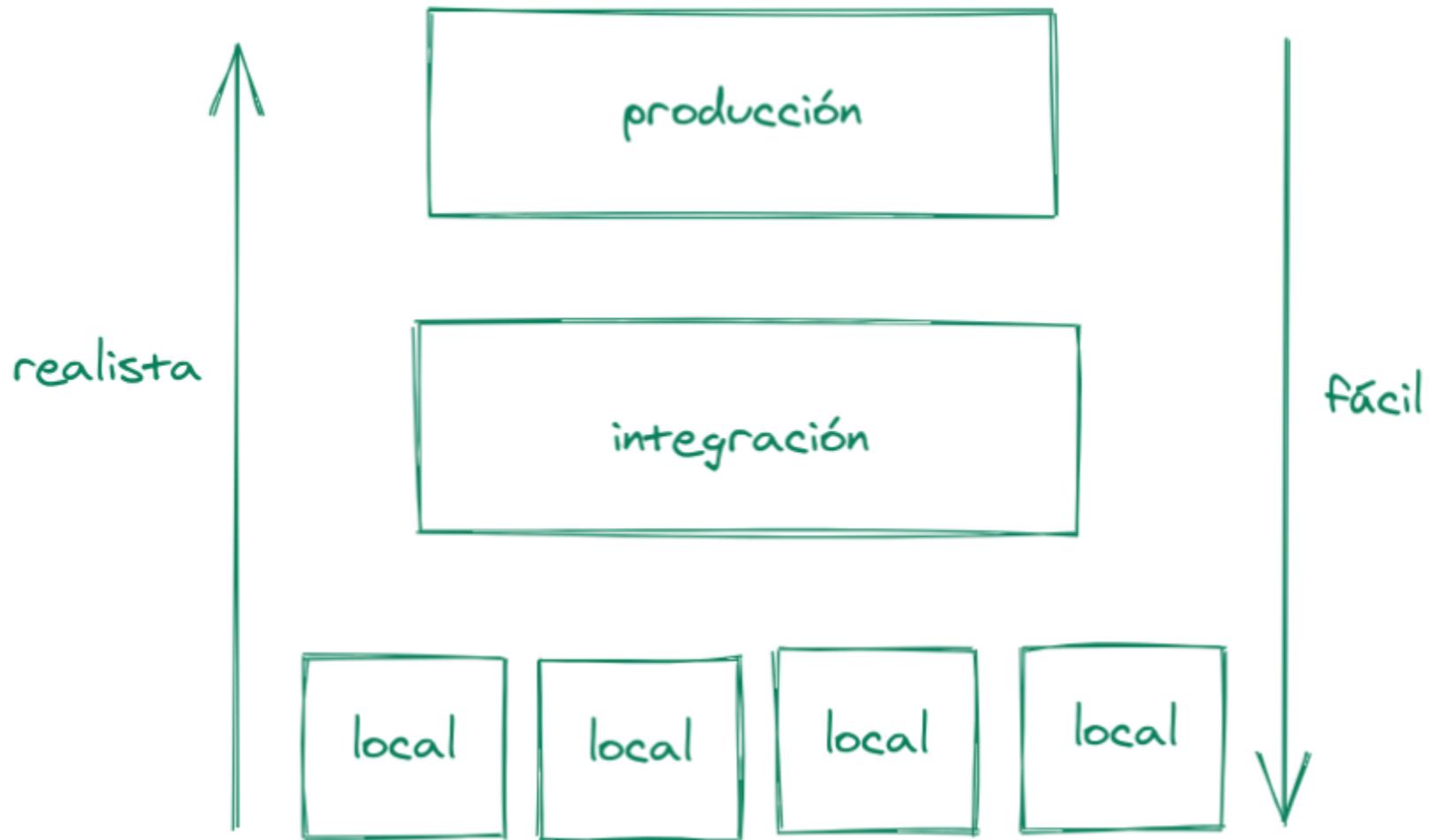
¿Por qué?



You broke it!



Entornos de prueba



Herramientas a medida



¡Crea tu propio loadtester!

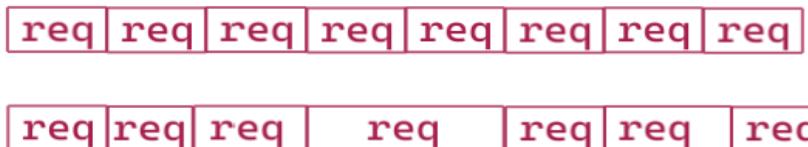
Lanza peticiones concurrentes http

Recoge las respuestas:

- 200: OK
- 204: sin respuesta
- 301: moved permanently
- 302: moved
- 4xx: bad request
- 5xx: server error

Modo amable: cada hilo envía peticiones seguidas

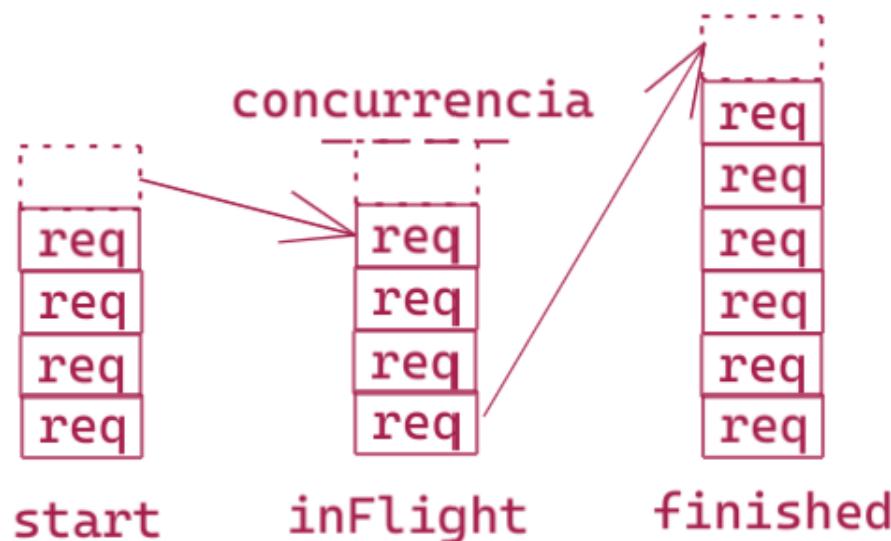
modo amable



Proyecto lambda-loadtester

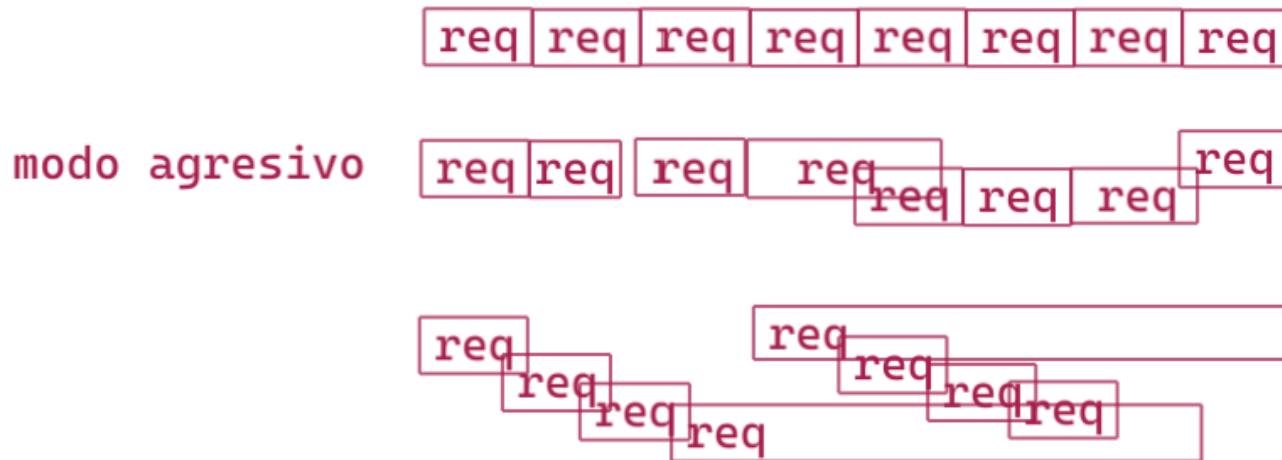
Autor: vuestro humilde servidor

Pruebas de AWS Lambda
Acumula promesas en un array
Lanza las promesas con la concurrencia pedida



Tasa de RPS fija

Modo agresivo: R peticiones por segundo
Una petición cada $1000/R$ milisegundos



Se puede usar por ejemplo setInterval() en Node.js

¿Qué pasa cuando $1000/R$ es una fracción?

Ejercicio: Diseña un temporizador

Modo "agresivo" de peticiones fijas

Primero intenta con una tasa redonda: 10 rps

Estudia cómo poner un intervalo

¿Qué pasa si una petición tarda demasiado?

¿Qué pasa cuando hay un retraso del sistema?



Ejercicio +

Ahora intenta que funcione con tasas fraccionales, e.g. 400 rps

¿Cómo gestionas fracciones de milisegundo?



Ejercicio +

Ahora piensa que tienes que lanzar más de 1000 rps

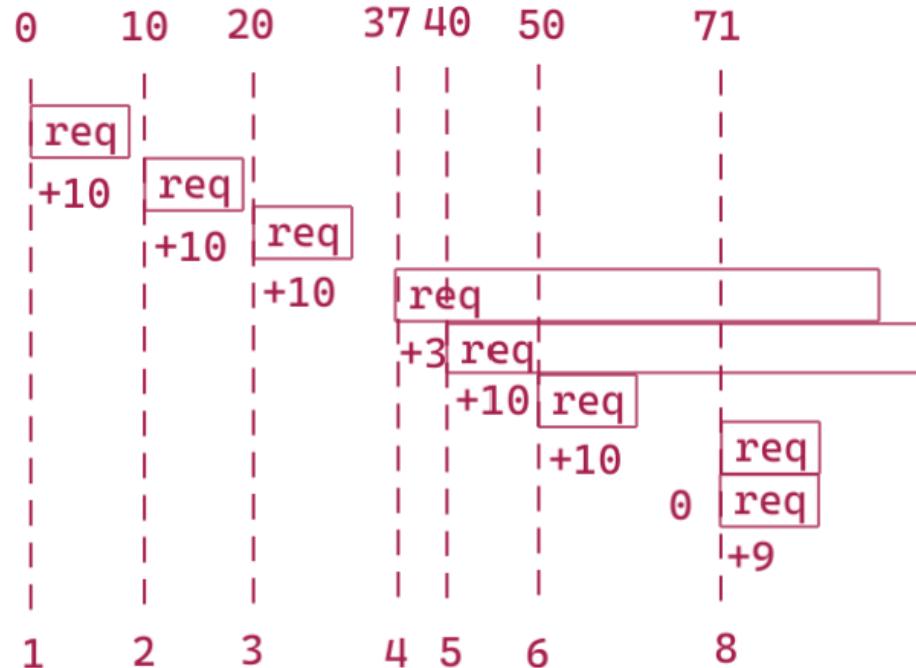
¿Se puede garantizar que la tasa sea ± constante?

¿Se puede garantizar al menos que el retraso no se acumule?



Ejercicio +

timer preciso 10 ms



hrtimer en loadtest



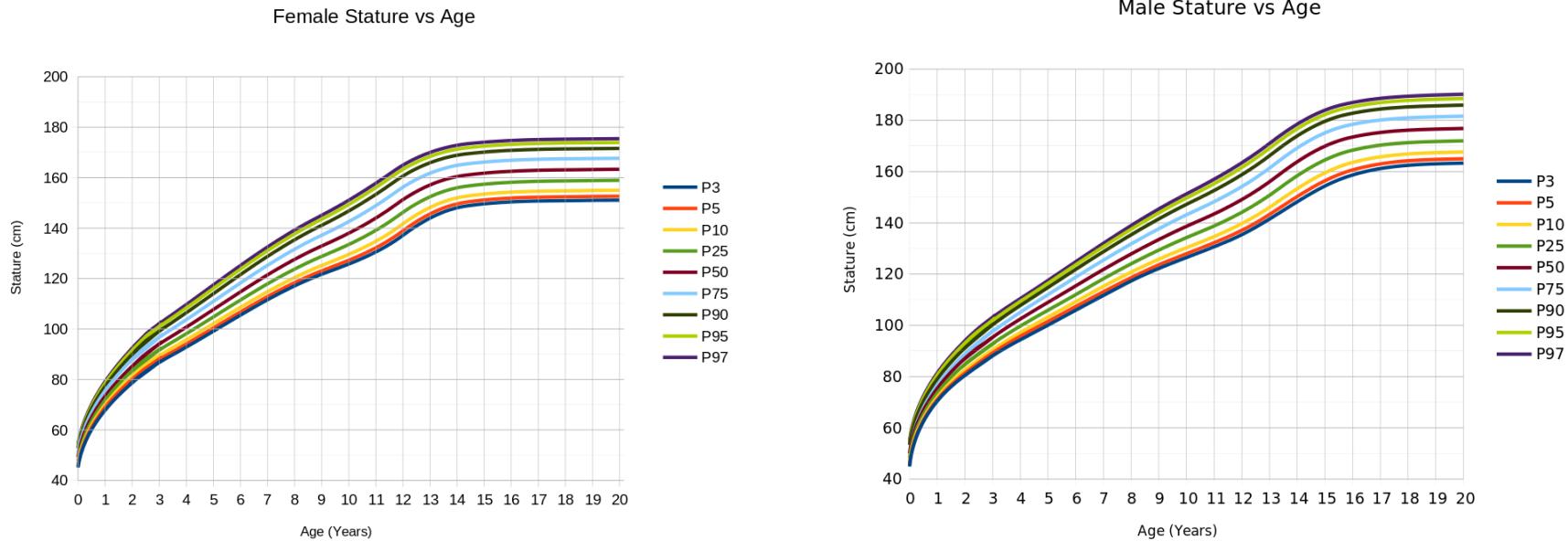
A prueba de balas!



Variabilidad y percentiles



Ejemplo: altura



Distribución de alturas en USA, [fuente](#)

Fuente inagotable de preocupación entre padres novatos

Distribución de alturas

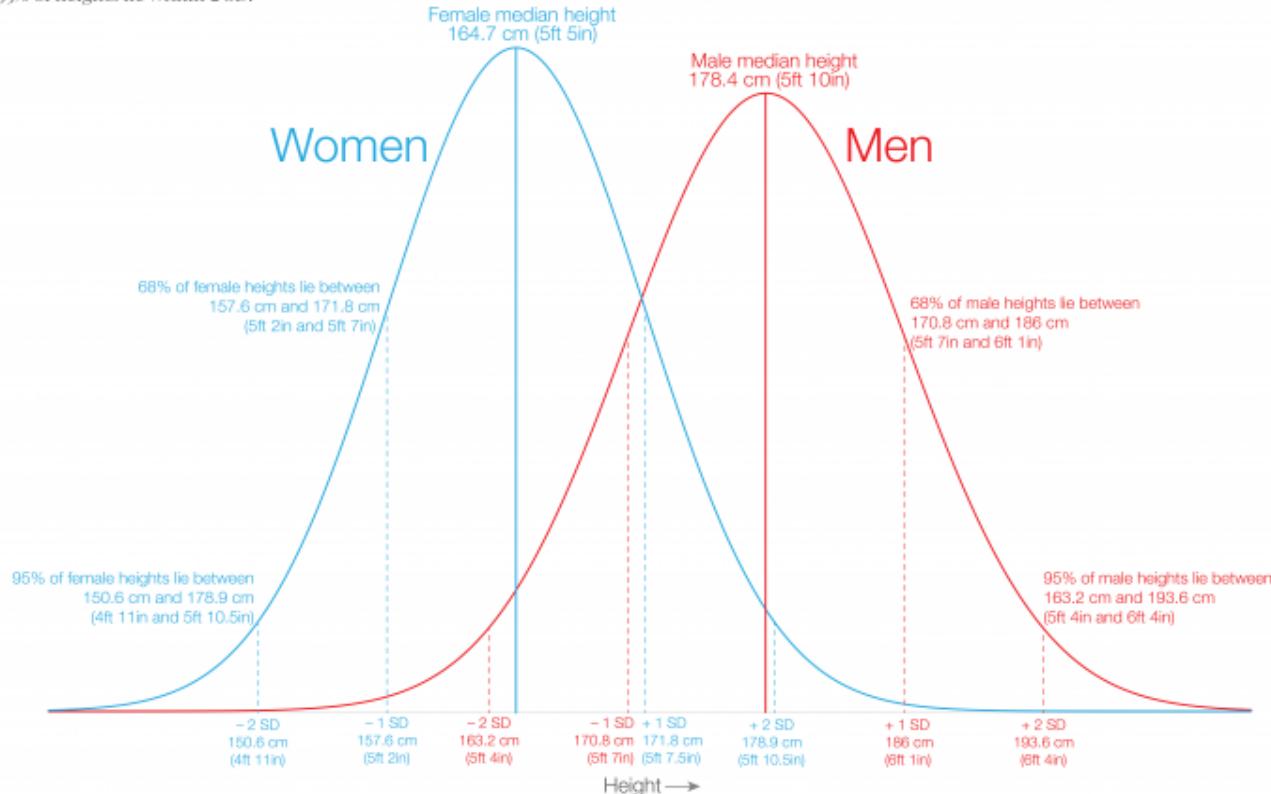
The distribution of male and female heights

The distribution of adult heights for men and women based on large cohort studies across 20 countries in North America, Europe, East Asia and Australia. Shown is the sample-weighted distribution across all cohorts born between 1980 and 1994 (so reaching the age of 18 between 2008 and 2012).

Our World
in Data

Since human heights within a population typically form a normal distribution:

- 68% of heights lie within 1 standard deviation (SD) of the median height;
- 95% of heights lie within 2 SD.



Note: this distribution of heights is not globally representative since it does not include all world regions due to data availability.

Data source: Jelenkovic et al. (2016). Genetic and environmental influences on height from infancy to early adulthood: An individual-based pooled analysis of 45 twin cohorts.

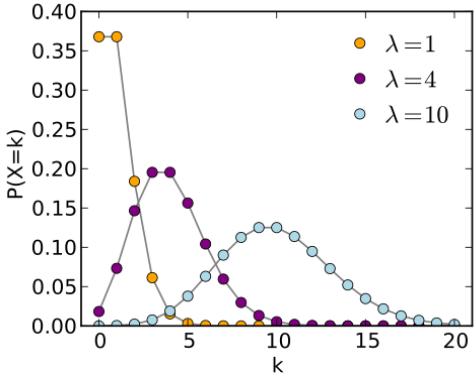
This is a visualization from OurWorldInData.org, where you find data and research on how the world is changing.

Licensed under CC-BY by the author Cameron Appel.

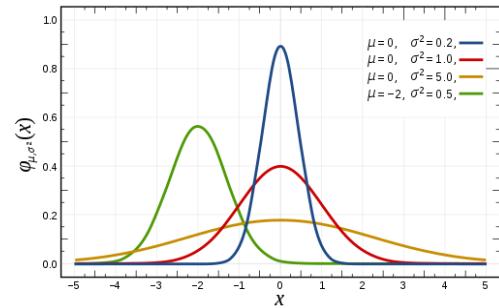
Distribución "normal"

Distribuciones importantes

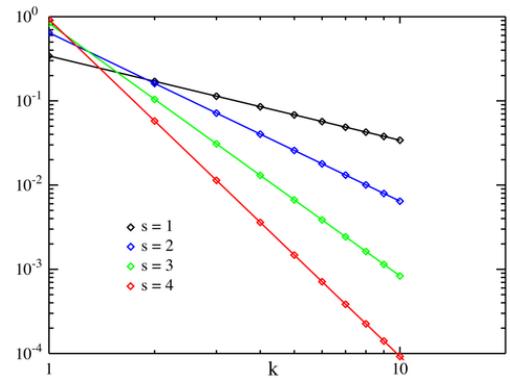
Gaussiana (normal)



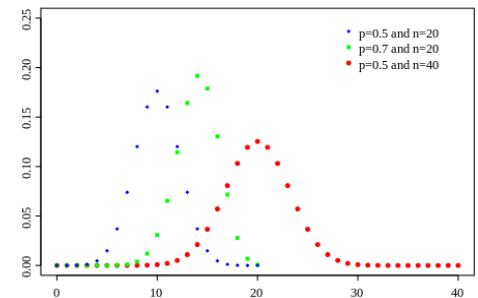
Poisson: llegada aleatoria



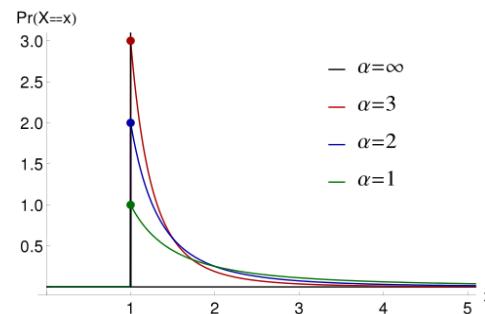
Binomial: monedas aleatorias



Zipf: frecuencia de palabras



Pareto: riqueza



Percentiles y servicio

La media da una idea sesgada de la experiencia

Los percentiles cuentan una historia más completa

Si hablamos del tiempo por petición:

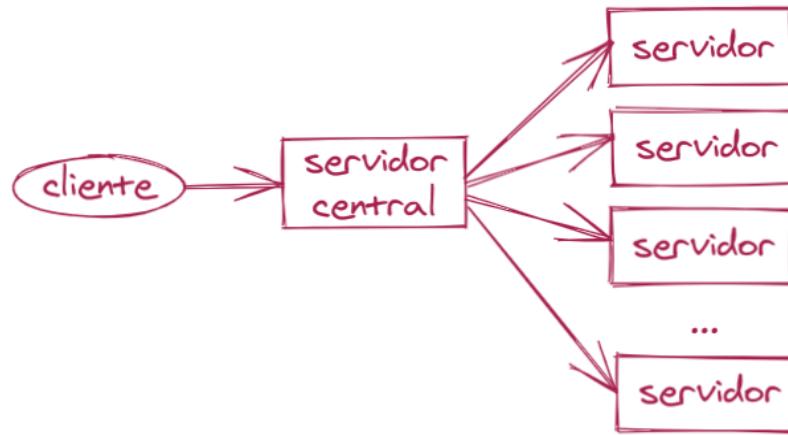
- Percentil 50: la mitad de las peticiones están por encima
- Percentil 90: 1 de cada 10 peticiones está por encima
- Percentil 99: 1 de cada 100 peticiones por encima

Mal percentil 99, a 100 peticiones por segundo:

Estamos cometiendo 1 pifia cada segundo

Ejercicio: Tiempo total

Una consulta utiliza 10 servidores **en paralelo**



El percentil 50 es 50 ms

El percentil 90 es de 200 ms

Estima una cota mínima para el tiempo medio por consulta



Ejercicio +

Simula una distribución de Pareto

$$T = \frac{x_m}{U^{1/\alpha}}$$

$$x_m = 28 \text{ ms}$$

$$\alpha = 1.16$$

Como función U puedes usar `Math.random()`



Ejercicio +

Simula 100k muestras de Pareto

Calcula la media, mínimo y máximo

Calcula los percentiles 5, 50, 90, 95, 99, 99.9

Nota para calcular percentiles:

- Ordena el array (¡como números!)
- Para el percentil 50%, vete a la mitad del array
- Para el percentil 90% vete a la posición 90k
- ...



Ejercicio +

Ahora simula peticiones con 10 llamadas a servidores en paralelo
= el máximo de 10 muestras de Pareto

Calcula mínimo, media, percentiles 50 y 90

¿Son iguales que antes?



Ejercicio +

Por fin, simula peticiones con 10 llamadas a servidores en serie



Será el resultado de sumar 10 muestras de Pareto

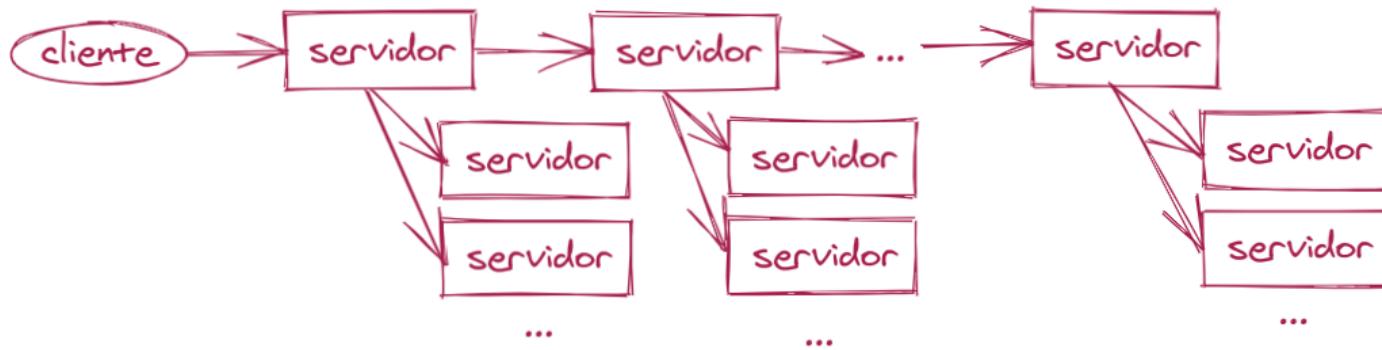
Calcula el percentil 50

¿Es 10 veces el valor de antes?



Ejercicio +

Para notísima: simula una petición a 10 servidores en serie, cada una de las cuales hace 10 en paralelo



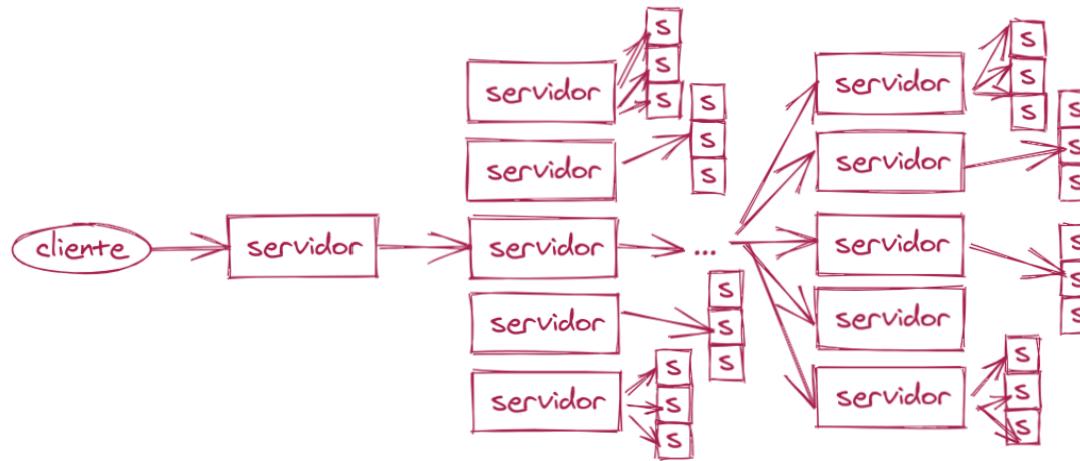
¿Cuáles son media, mínima, percentil 50?



Ejercicio +

Esta última parte **no es ciencia ficción**

En 2009 una búsqueda en Google usaba 1000 servidores



Tiempo total 200 ms

¿Se te ocurre alguna forma de mejorar la respuesta?



Timed out!



VICELAND

Paquete pareto-simulator

Instala pareto-simulator:

```
$ npm i -g pareto-simulator
```

Ahora prueba con unos cuantos comandos:

```
$ pareto --xm 28
$ pareto --xm 28 --parallel 10
$ pareto --xm 28 --series 10
$ pareto --xm 28 --series 10 --parallel 10
$ pareto --xm 1 -n 1000 --parallel 30 --series 30 --timeout 10 --linear
```

¿A qué distribuciones te recuerdan?

Ley de Pareto



Regla del 80/20

O Principio de Pareto

Un 20% de las causas genera el 80% de los efectos

Aplica a un amplio espectro de fenómenos:

- Distribución de la propiedad de las tierras
- El 20% de los clientes generan el 80% de las ventas
- Distribución de las ciudades por población

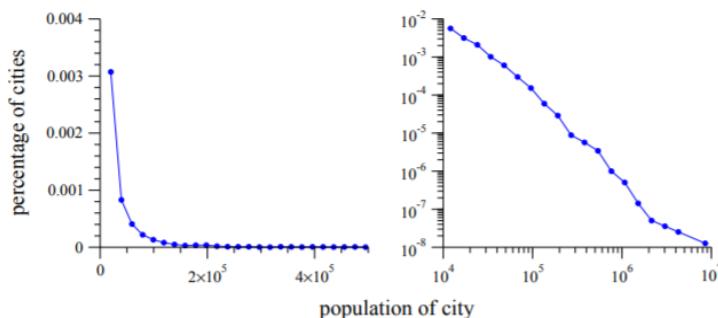


FIG. 2 Left: histogram of the populations of all US cities with population of 10 000 or more. Right: another histogram of the same data, but plotted on logarithmic scales. The approximate straight-line form of the histogram in the right panel implies that the distribution follows a power law. Data from the 2000 US Census.

Trabajo de optimización

Toca el duro trabajo de optimizar un servicio

Consume demasiado/a:

- CPU
- memoria
- descriptores de fichero
- búffers de entrada/salida
- ...

¿Por dónde empezamos?

Localizamos los 🔥 hot spots 🔥

¡Qué suerte!

Los 🔥hot spots🔥 siguen el principio de Pareto

Un 20% del código ocupa el 80% del tiempo de proceso

La ley aplica recursivamente...

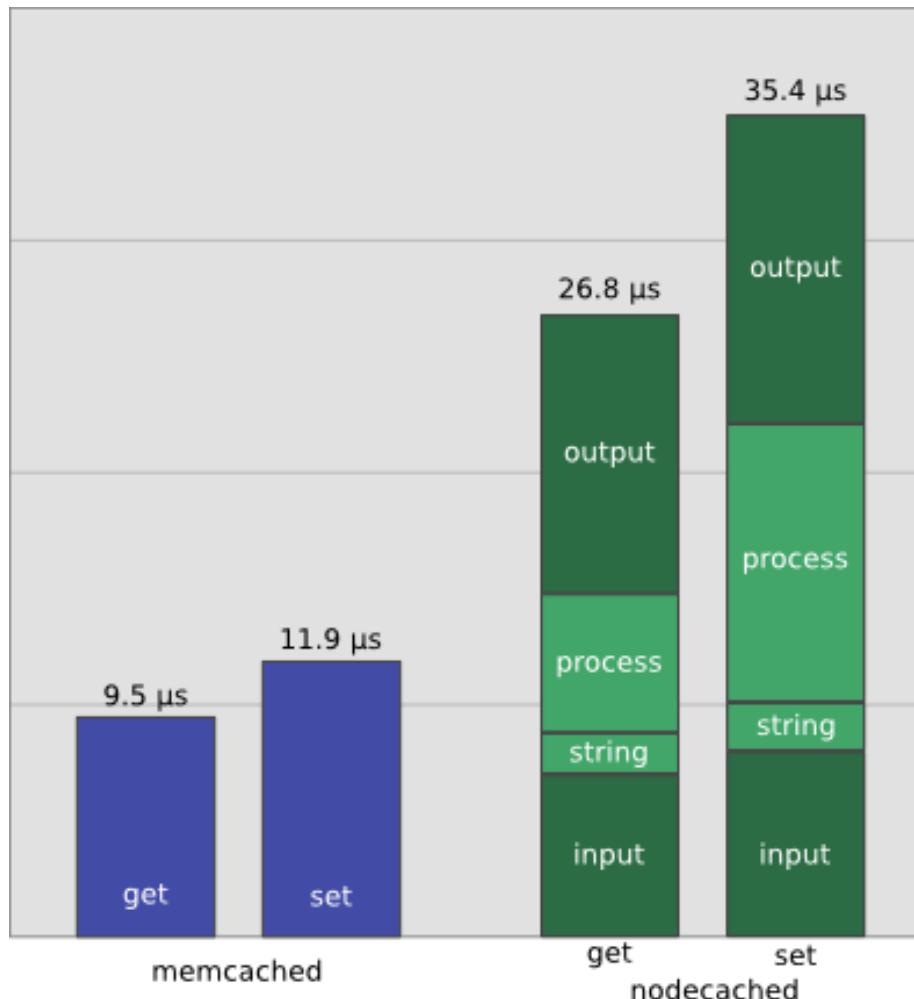
Un 4% del código ocupa el 64% del tiempo de proceso

¿Vale la pena?

HOW LONG CAN YOU WORK ON MAKING A ROUTINE TASK MORE
EFFICIENT BEFORE YOU'RE SPENDING MORE TIME THAN YOU SAVE?
(ACROSS FIVE YEARS)

		HOW OFTEN YOU DO THE TASK					
		50/DAY	5/DAY	DAILY	WEEKLY	MONTHLY	YEARLY
HOW MUCH TIME YOU SHAVE OFF	1 SECOND	1 DAY	2 HOURS	30 MINUTES	4 MINUTES	1 MINUTE	5 SECONDS
	5 SECONDS	5 DAYS	12 HOURS	2 HOURS	21 MINUTES	5 MINUTES	25 SECONDS
	30 SECONDS	4 WEEKS	3 DAYS	12 HOURS	2 HOURS	30 MINUTES	2 MINUTES
	1 MINUTE	8 WEEKS	6 DAYS	1 DAY	4 HOURS	1 HOUR	5 MINUTES
	5 MINUTES	9 MONTHS	4 WEEKS	6 DAYS	21 HOURS	5 HOURS	25 MINUTES
	30 MINUTES	6 MONTHS	5 WEEKS	5 DAYS	1 DAY	2 HOURS	
	1 HOUR	10 MONTHS	2 MONTHS	10 DAYS	2 DAYS	5 HOURS	
	6 HOURS			2 MONTHS	2 WEEKS	1 DAY	
	1 DAY				8 WEEKS	5 DAYS	

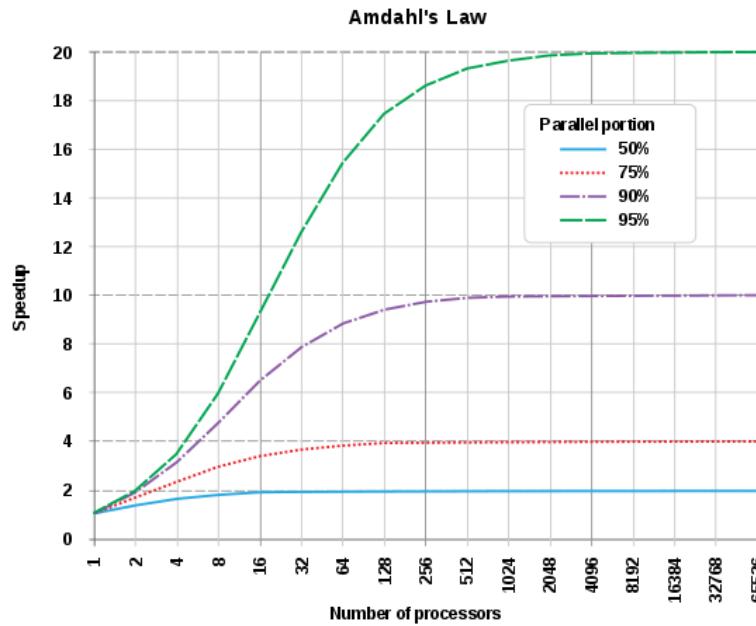
Ley de Amdahl



Fuente

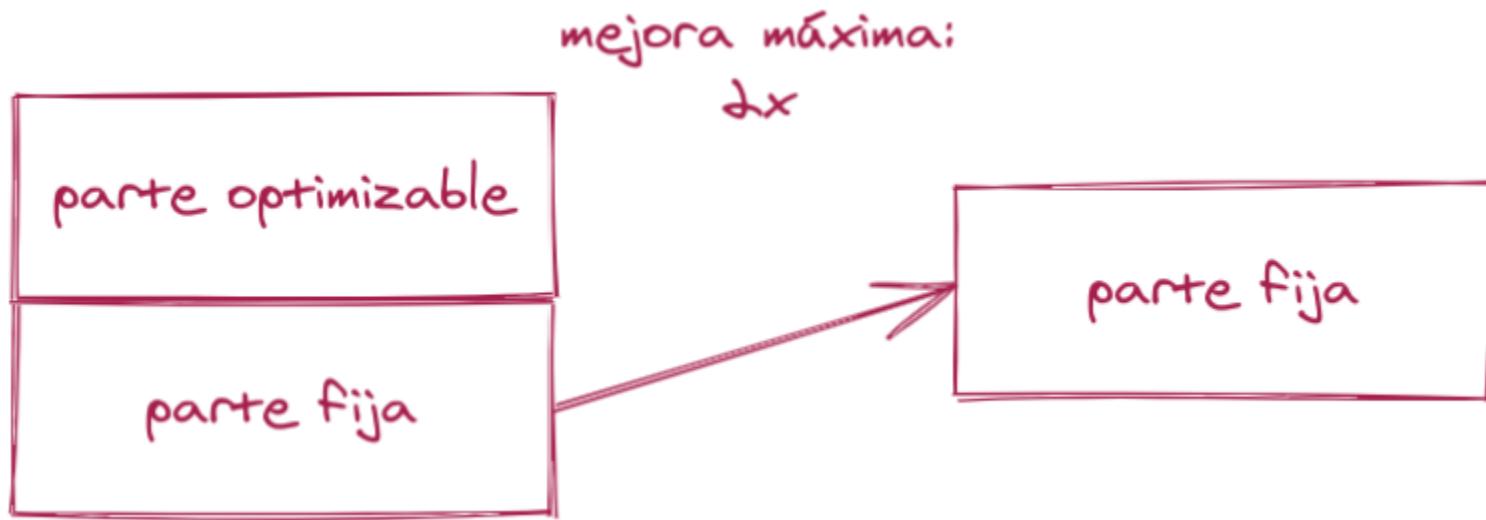
Una ley en apariencia complicada

$$S_{\text{latency}}(s) = \frac{1}{(1 - p) + \frac{p}{s}}$$



Wikipedia

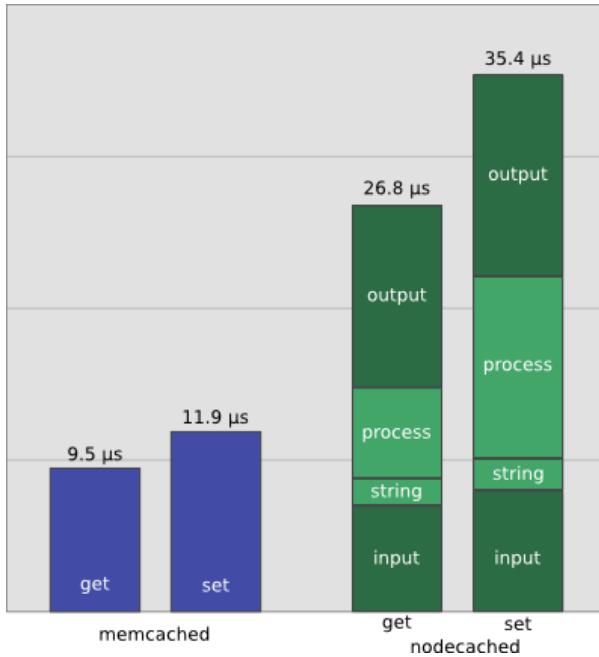
Más fácil



Ejercicio: Límites de optimización

Petición get de nodecached: 37250 rps

Desglose de tiempos:



¿Cuánto podemos acelerar el servidor nodecached?



Ejercicio +

Nos centramos en la operación get:

- 2 µs: conversión a string
- 6 µs: procesamiento interno
- 7 µs: input
- 12 µs: output

Suponemos que optimizamos string y procesamiento (0µs)

¿Cuánto es el máximo teórico de peticiones por segundo?



Ejercicio +

Fórmula: $R \text{ [rps]} = 1000000 / t \text{ [\mu s]}$

Ejercicio de imaginación:

¿Qué estrategias podemos seguir para optimizar más?



Acelerando!

ELMOTOR

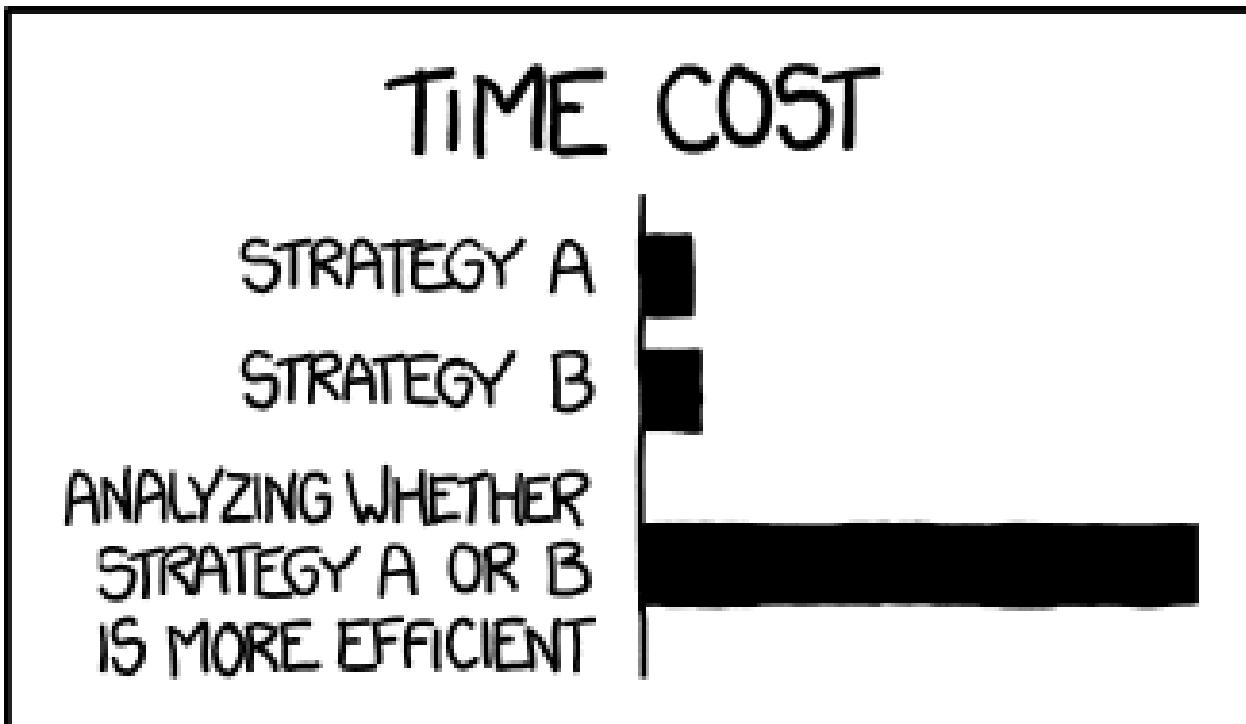


Ley de retornos decrecientes

Según avanzamos en la optimización el retorno de inversión disminuye

Ley poco rigurosa (pero útil)

Cuidado con la micro-optimización



THE REASON I AM SO INEFFICIENT

Incertidumbre y error



Error sistemático o aleatorio

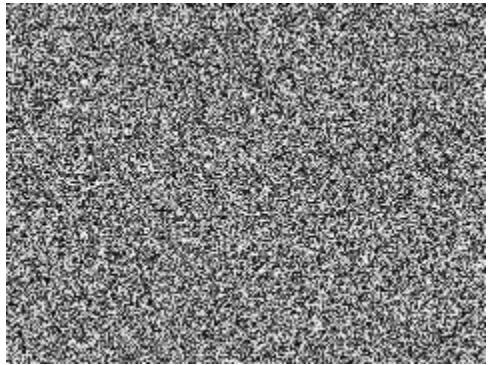


Error sistemático

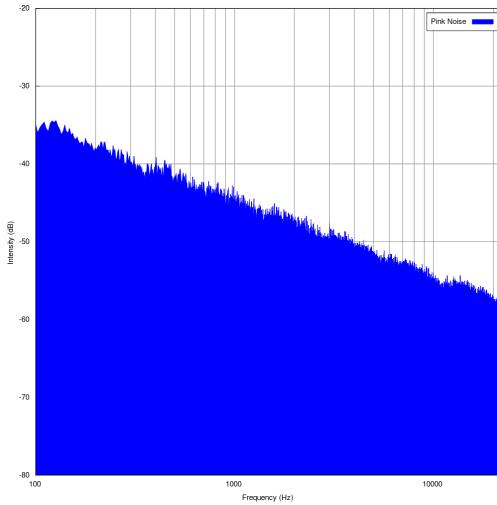


Error aleatorio

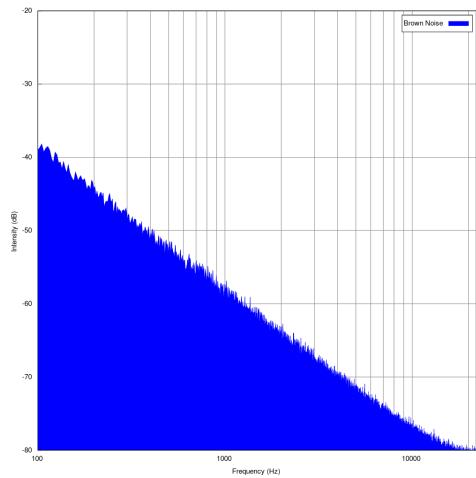
Tipos de aleatoriedad



Ruido blanco



Ruido rosa



Ruido marrón

Ejercicio: Distribución real

Queremos medir la distribución de tiempos

Peticiones al servicio <http://service.pinchito.es:3000/a>



Ejercicio +

Bájate a local el proyecto loadtest

(Lo tienes preparado en la imagen pinchito-loadtest)

Trúcalo para que pinte el tiempo de cada petición

(También en la imagen)

Lanza las pruebas:

```
node bin/loadtest.js http://service.pinchito.es:3000/a -n 15000 -c 100 --rps 300
```

Usa rps por encima y por debajo del valor sin --rps (e.g. 300 y 500)



Ejercicio +

Saca el resultado a fichero y extrae los valores numéricos

\$ comando | grep -v INFO > service-times-300.csv

Dibuja un histograma con los valores

Dibuja un histograma log-log

¿Qué te dicen las gráficas?



Awesome!



Ejercicio +

Mis resultados

Perfilado de código



I JUST WANNA GO FAST!

Perfilado con microprofiler

Paquete `microprofiler`

Permite ver en qué se va el tiempo de proceso

Es una capa fina sobre `process.hrtime()`

Se instrumentan secciones de código:

```
const start = microprofiler.start()
```

Empieza a medir

```
microprofiler.measureFrom(start, 'label', 10000)
```

Mide el tiempo entre `start()` y `measureFrom()`

Pinta un resumen para `label` cada 10000 llamadas

Perfilado nativo

Node.js incluye también un [profiler](#) majo:

```
$ node --prof ...
```

Esto genera un fichero como `isolate-0x440c2f0-28473-v8.log`

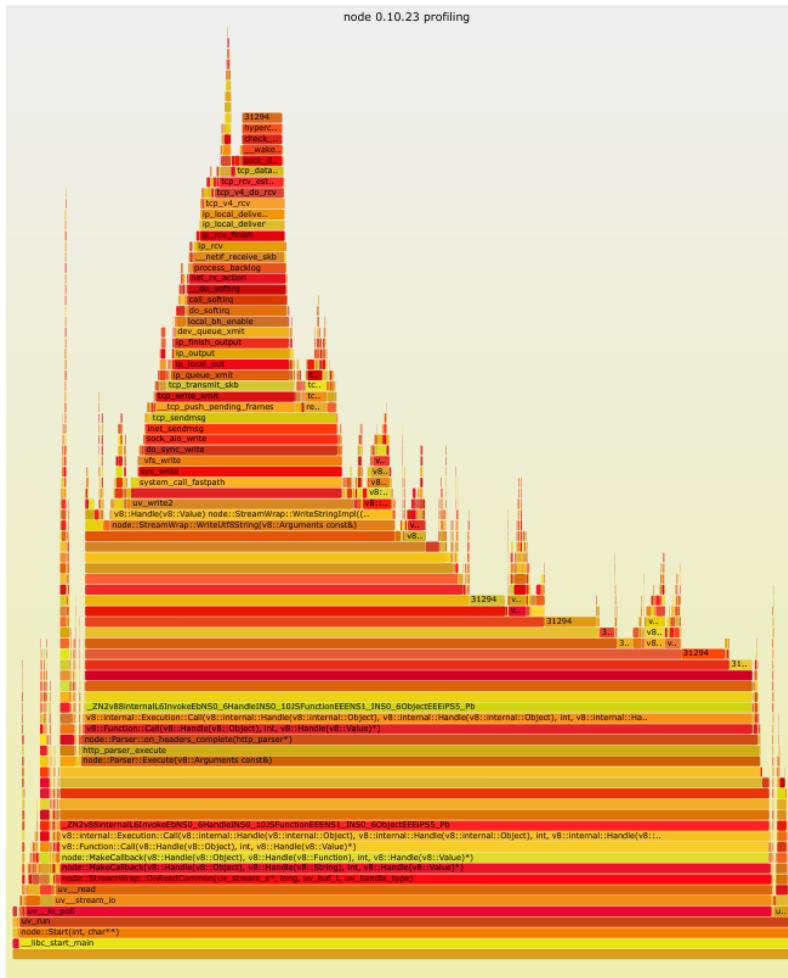
Se puede interpretar con el comando

```
node --prof-process isolate-0x440c2f0-28473-v8.log
```

La salida tiene tres partes:

- [Summary]: Resumen de tiempos
- [JavaScript], [C++ entry points]: Secciones desglosadas
- [Bottom up (heavy) profile]: Perfil jerárquico

Flamegraphs



Fuente

Paquetes de flamegraphs

Paquete [0x](#)

Node.js y 0x

Node.js in Flames

Demo en directo

Ejercicio: Ahorrando microsegundos

Queremos optimizar el código de [pareto-simulator](#)

¿Dónde se nos va el tiempo?

Vamos a ver dos enfoques alternativos



Ejercicio +

Clona el paquete pareto-simulator

```
$ git clone https://github.com/alexfernandez/pareto-simulator.git  
$ cd pareto-simulator
```

Instala el paquete [microprofiler](#)

```
$ npm i microprofiler
```

Léete las instrucciones



Ejercicio +

Instrumenta el código:

```
const microprofiler = require('microprofiler');
...
computeSamples() {
    for (let i = 0; i < options.number; i++) {
        const start = microprofiler.start()
        const sample = this.computeSample()
        microprofiler.measureFrom(start, 'sample', 10000)
        this.samples.push(sample)
        this.sum += sample
        if (sample > this.max) this.max = sample
        if (sample < this.min) this.min = sample
        microprofiler.measureFrom(start, 'stats', 10000)
    }
}
```

Ejecuta y mira los resultados:

```
$ node index.js --xm 1 -n 1000 --parallel 30 --series 30 --timeout 10 --linear
```



Ejercicio +

Ahora instrumenta la función `computeSample()`

Mira a ver si hay alguna sorpresa

Intenta buscar alguna optimización...

Y vuelve a medir



Ejercicio+

En este último paso vamos a ejecutar el profiler de Node.js:

```
$ node --prof index.js --xm 28 -n 10000000
```

Y ahora interpretamos la salida:

```
$ node --prof-process isolate-0x...-v8.log
```

¿Te llama algo la atención?

¿Se te ocurre cómo optimizarlo?

¿Es este profiler igual de ágil que microprofiler?



Good job!



Bibliografía

pinchito.es: Optimizando sockets con node.js

pinchito.es: node.js: ¿rápido como el rayo?

pinchito.es: Pruebas de carga

Node.js: Flame Graphs

Netflix Tech Blog: Node.js in Flames

Node.js: Easy profiling for Node.js Applications