Resultados de la prueba de performance de node con y sin console.log

Primer prueba: Node profiler (performance/node-profiler)

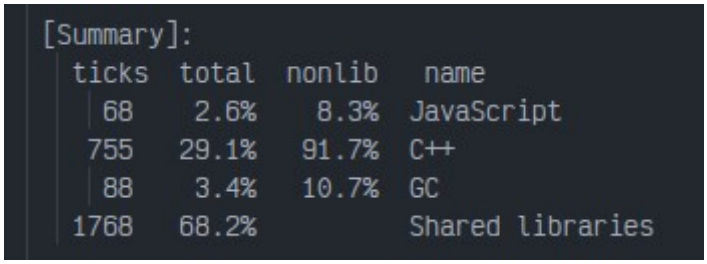
Pasos para ejecutarla con clg (console.log)

- ✗ node --prof server.js
- ✗ node --prof-process clg.log > result_prof-clg.txt

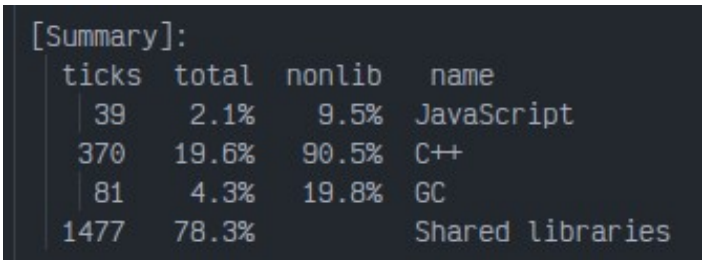
Pasos para ejecutarla sin clg (console.log)

- ✗ node --prof server.js
- ✗ node --prof-process notClg.log > result_prof-notClg.txt

Resultados:



```
[Summary]:
  ticks  total  nonlib   name
     68   2.6%    8.3%  JavaScript
    755  29.1%   91.7%  C++
     88   3.4%   10.7%  GC
   1768  68.2%          Shared libraries
```



```
[Summary]:
  ticks  total  nonlib   name
     39   2.1%    9.5%  JavaScript
    370  19.6%   90.5%  C++
     81   4.3%   19.8%  GC
   1477  78.3%          Shared libraries
```

A la izquierda se ve la prueba con el clg y a la derecha la prueba sin clg. Se puede ver que la prueba sin clg es mas performante ya que tiene menor cantidad de ticks.

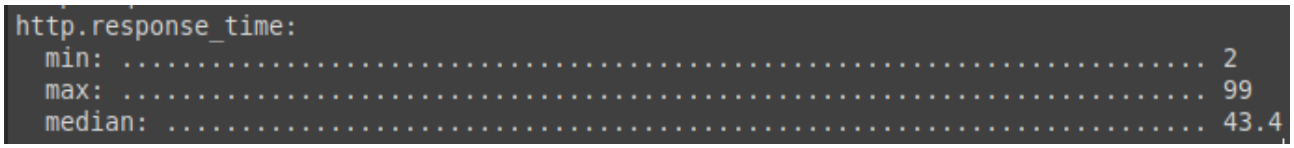
Segunda prueba: Artillery (performance/artillery)

Pasos para ejecutarla con clg (console.log)

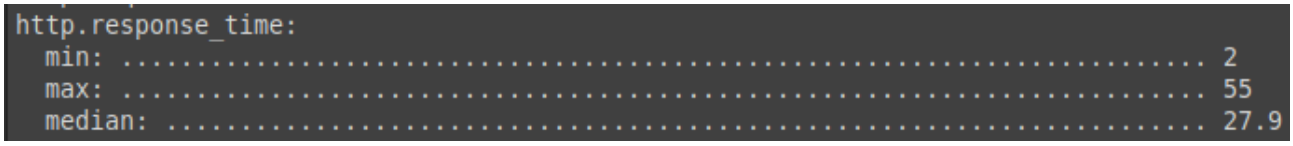
- ✗ artillery quick --count 20 -n 50 "http://localhost:8080/info" > result_clg.txt

Pasos para ejecutarla sin clg (console.log)

- ✗ artillery quick --count 20 -n 50 "http://localhost:8080/info" > result_notClg.txt



```
http.response_time:
  min: ..... 2
  max: ..... 99
  median: ..... 43.4
```



```
http.response_time:
  min: ..... 2
  max: ..... 55
  median: ..... 27.9
```

La imagen de arriba muestra la prueba con clg y la de abajo sin clg. Como podemos ver en esta prueba es mas performante el segundo caso ya que el tiempo de respuesta medio es menor.

Tercera prueba: Autocannon (performance/autocannon)

Pasos para ejecutarla con clg (console.log)

- ✗ npm start (inicio proyecto)
- ✗ node benchmark.js (desde otra consola)

Para este caso en ambas pruebas se usan los mismos comandos, pero usando clg en la primer prueba y comentándolo en la segunda. En este caso se emulan 100 conexiones concurrentes realizadas en un tiempo de 20 segundos.

```
> test@0.0.0 test
> node benchmark.js

Running all benchmarks in parallel ...
Running 20s test @ http://localhost:8080/info
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	112 ms	124 ms	181 ms	236 ms	131.73 ms	26.71 ms	405 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	400	400	798	899	754.85	103.7	400
Bytes/Sec	791 kB	791 kB	1.58 MB	1.78 MB	1.49 MB	205 kB	791 kB

```
Req/Bytes counts sampled once per second.
# of samples: 20

15k requests in 20.03s, 29.8 MB read
```

```
> test@0.0.0 test
> node benchmark.js

Running all benchmarks in parallel ...
Running 20s test @ http://localhost:8080/info
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	94 ms	97 ms	134 ms	186 ms	101.35 ms	17.44 ms	311 ms

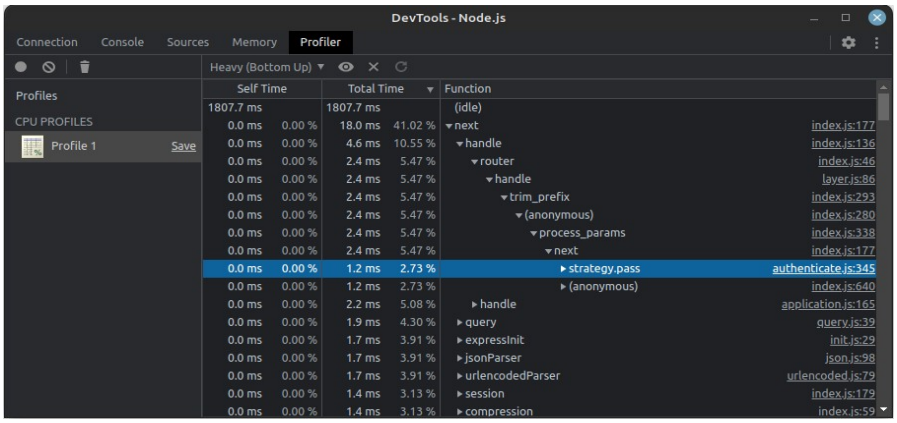
Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	535	535	1001	1091	979.9	110.69	535
Bytes/Sec	1.06 MB	1.06 MB	1.98 MB	2.16 MB	1.94 MB	219 kB	1.06 MB

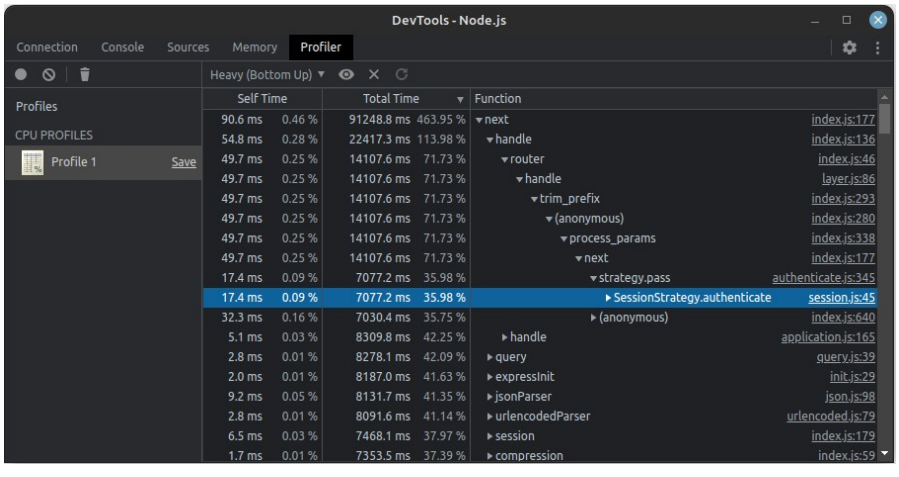
```
Req/Bytes counts sampled once per second.
# of samples: 20

20k requests in 20.02s, 38.7 MB read
```

(HACER ZOOM). A la izquierda la prueba con clg y a la derecha la prueba sin clg. Se puede ver que la prueba sin clg tiene una latencia promedio menor. Y que la prueba con clg pudo realizar menos request que la prueba sin clg dentro del mismo tiempo.

Cuarta prueba: Node inspect (performance/node-inspect)

Para usar el inspect de node realice dos pruebas, una con y una sin autocannon.





La primera es sin autocannon y la segunda con autocannon. En los dos casos se puede ver que el proceso que mas tiempo demora es un next y siguiéndolo para ver que procesos ejecuta en ambos casos llegamos a la estrategia de autenticación de passport.

Quinta prueba: 0x y autocannon (performance/0x)

Por ultimo para llegar a la prueba de 0x con autocannon tuve que levantar el server con express comentando todas las lineas que usaban fork o cluster porque 0x mostraba un error que decía que 0x no soporta fork.

Los gráficos y todas las imágenes y archivos usados se encuentran en la carpeta /*performance y dentro de esta una carpeta para cada prueba por separado.*

Conclusión:
Por ultimo, concluyo que en algunas pruebas se ve mucha diferencia en cuanto al rendimiento del server cuando sufre una gran carga, teniendo en cuenta que la diferencia entre cada prueba es solamente un mensaje por consola (console.log). Ademas, veo que es muy interesante evaluar el rendimiento de la app con una gran carga y también muy importante para corregir temas de optimización y de consumo de recursos. Y de esta manera no solo lograr una app mas rápida o mas ligera sino también de un menor tamaño o de un menor consumo de recursos para aprovechar al máximo el rendimiento de algún servicio de hosting o del mismo equipo en el que se ejecute la app.