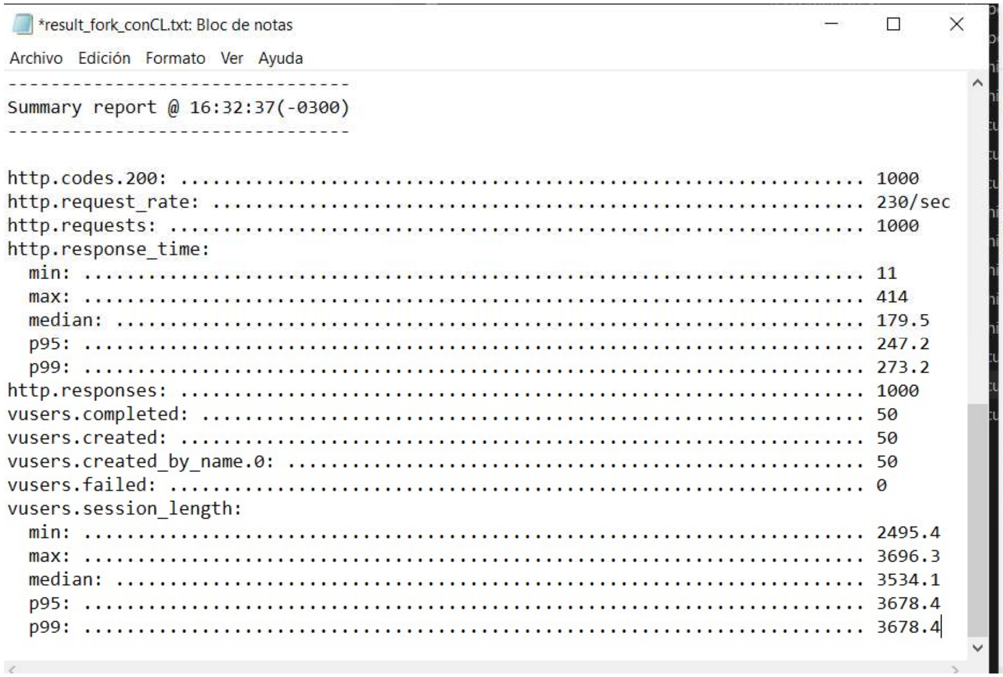
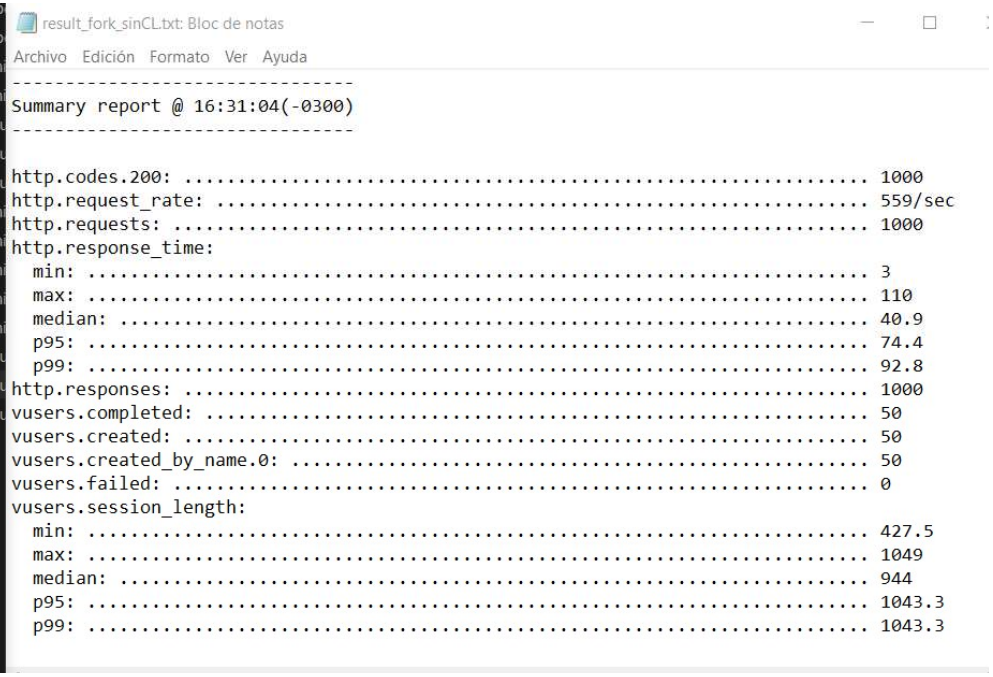
Análisis con la salida de Artillery:


*result_fork_conCL.txt: Bloc de notas
Archivo Edición Formato Ver Ayuda

```
--------------------------------
Summary report @ 16:32:37(-0300)
--------------------------------

http.codes.200: ............................................................. 1000
http.request_rate: .......................................................... 230/sec
http.requests: .............................................................. 1000
http.response_time:
  min: ...................................................................... 11
  max: ...................................................................... 414
  median: ................................................................... 179.5
  p95: ...................................................................... 247.2
  p99: ...................................................................... 273.2
http.responses: ............................................................. 1000
vusers.completed: ........................................................... 50
vusers.created: ............................................................. 50
vusers.created_by_name.0: ................................................... 50
vusers.failed: .............................................................. 0
vusers.session_length:
  min: ...................................................................... 2495.4
  max: ...................................................................... 3696.3
  median: ................................................................... 3534.1
  p95: ...................................................................... 3678.4
  p99: ...................................................................... 3678.4
```


result_fork_sinCL.txt: Bloc de notas
Archivo Edición Formato Ver Ayuda

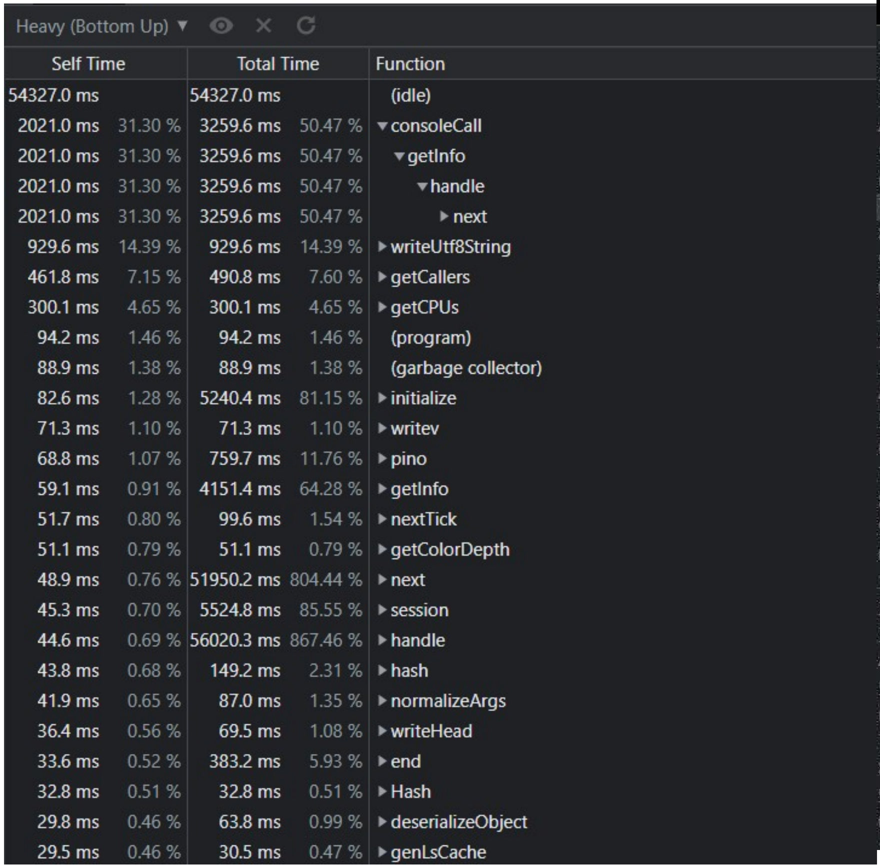
```
--------------------------------
Summary report @ 16:31:04(-0300)
--------------------------------

http.codes.200: ............................................................. 1000
http.request_rate: .......................................................... 559/sec
http.requests: .............................................................. 1000
http.response_time:
  min: ...................................................................... 3
  max: ...................................................................... 110
  median: ................................................................... 40.9
  p95: ...................................................................... 74.4
  p99: ...................................................................... 92.8
http.responses: ............................................................. 1000
vusers.completed: ........................................................... 50
vusers.created: ............................................................. 50
vusers.created_by_name.0: ................................................... 50
vusers.failed: .............................................................. 0
vusers.session_length:
  min: ...................................................................... 427.5
  max: ...................................................................... 1049
  median: ................................................................... 944
  p95: ...................................................................... 1043.3
  p99: ...................................................................... 1043.3
```

Analisis con la salida de Artillery+ Built-in Profiler:

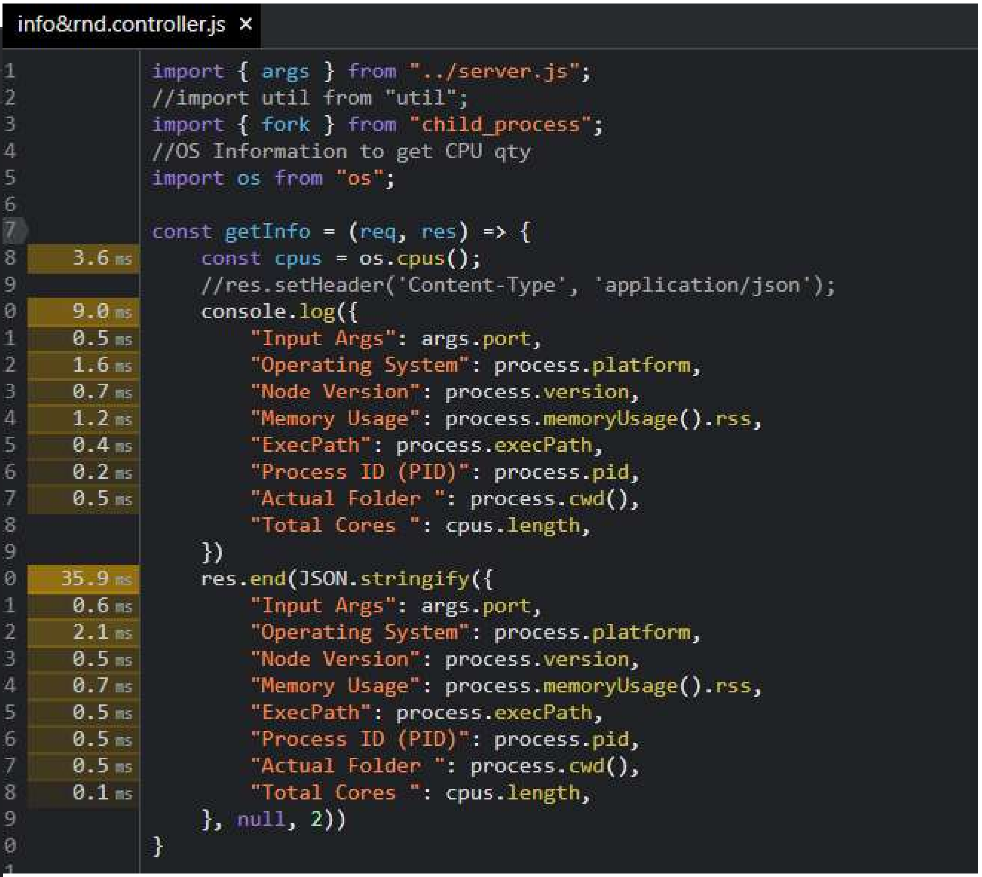
CON Console Log:	*SIN Console Log:*
[Shared libraries]: ticks total nonlib name 18902 96.2% C:\Windows\SYSTEM32\ntdll.dll 696 3.5% C:\Program Files\nodejs\node.exe 11 0.1% C:\Windows\System32\KERNELBASE.dll 7 0.0% C:\Windows\System32\KERNEL32.DLL 1 0.0% C:\Windows\System32\WS2_32.dll	[Shared libraries]: ticks total nonlib name 7189 93.1% C:\Windows\SYSTEM32\ntdll.dll 499 6.5% C:\Program Files\nodejs\node.exe 9 0.1% C:\Windows\System32\KERNELBASE.dll 3 0.0% C:\Windows\System32\KERNEL32.DLL
[Summary]: ticks total nonlib name 22 0.1% 95.7% JavaScript 0 0.0% 0.0% C++ 16 0.1% 69.6% GC 19617 99.9% Shared libraries 1 0.0% Unaccounted	[Summary]: ticks total nonlib name 20 0.3% 100.0% JavaScript 0 0.0% 0.0% C++ 21 0.3% 105.0% GC 7700 99.7% Shared libraries

Informe *con* **Inspect:**

CON Console Log

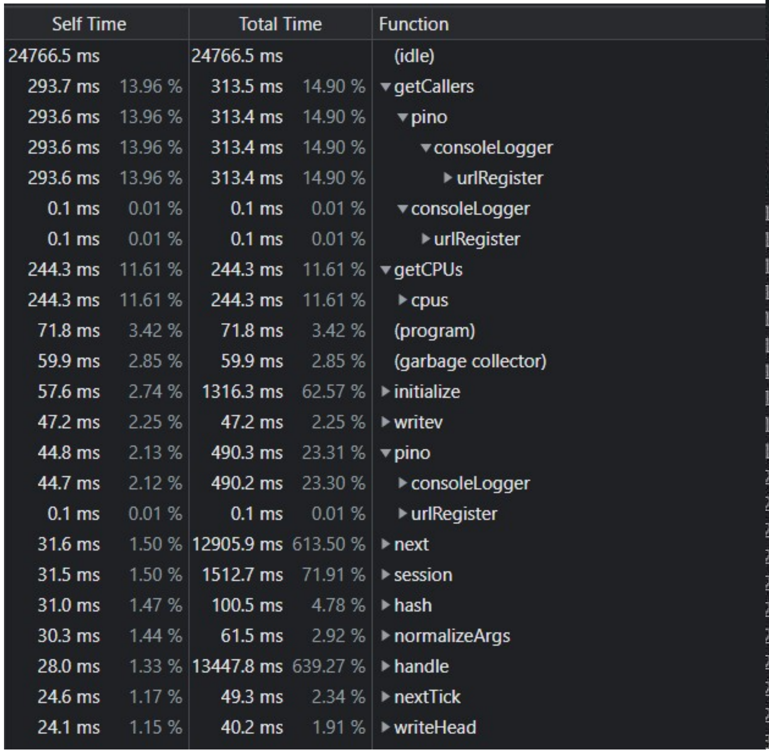


Self Time		Total Time		Function
54327.0 ms		54327.0 ms		(idle)
2021.0 ms	31.30 %	3259.6 ms	50.47 %	▼consoleCall
2021.0 ms	31.30 %	3259.6 ms	50.47 %	▼getInfo
2021.0 ms	31.30 %	3259.6 ms	50.47 %	▼handle
2021.0 ms	31.30 %	3259.6 ms	50.47 %	▶next
929.6 ms	14.39 %	929.6 ms	14.39 %	▶writeUtf8String
461.8 ms	7.15 %	490.8 ms	7.60 %	▶getCallers
300.1 ms	4.65 %	300.1 ms	4.65 %	▶getCPUs
94.2 ms	1.46 %	94.2 ms	1.46 %	(program)
88.9 ms	1.38 %	88.9 ms	1.38 %	(garbage collector)
82.6 ms	1.28 %	5240.4 ms	81.15 %	▶initialize
71.3 ms	1.10 %	71.3 ms	1.10 %	▶writev
68.8 ms	1.07 %	759.7 ms	11.76 %	▶pino
59.1 ms	0.91 %	4151.4 ms	64.28 %	▶getInfo
51.7 ms	0.80 %	99.6 ms	1.54 %	▶nextTick
51.1 ms	0.79 %	51.1 ms	0.79 %	▶getColorDepth
48.9 ms	0.76 %	51950.2 ms	804.44 %	▶next
45.3 ms	0.70 %	5524.8 ms	85.55 %	▶session
44.6 ms	0.69 %	56020.3 ms	867.46 %	▶handle
43.8 ms	0.68 %	149.2 ms	2.31 %	▶hash
41.9 ms	0.65 %	87.0 ms	1.35 %	▶normalizeArgs
36.4 ms	0.56 %	69.5 ms	1.08 %	▶writeHead
33.6 ms	0.52 %	383.2 ms	5.93 %	▶end
32.8 ms	0.51 %	32.8 ms	0.51 %	▶Hash
29.8 ms	0.46 %	63.8 ms	0.99 %	▶deserializeObject
29.5 ms	0.46 %	30.5 ms	0.47 %	▶genLsCache

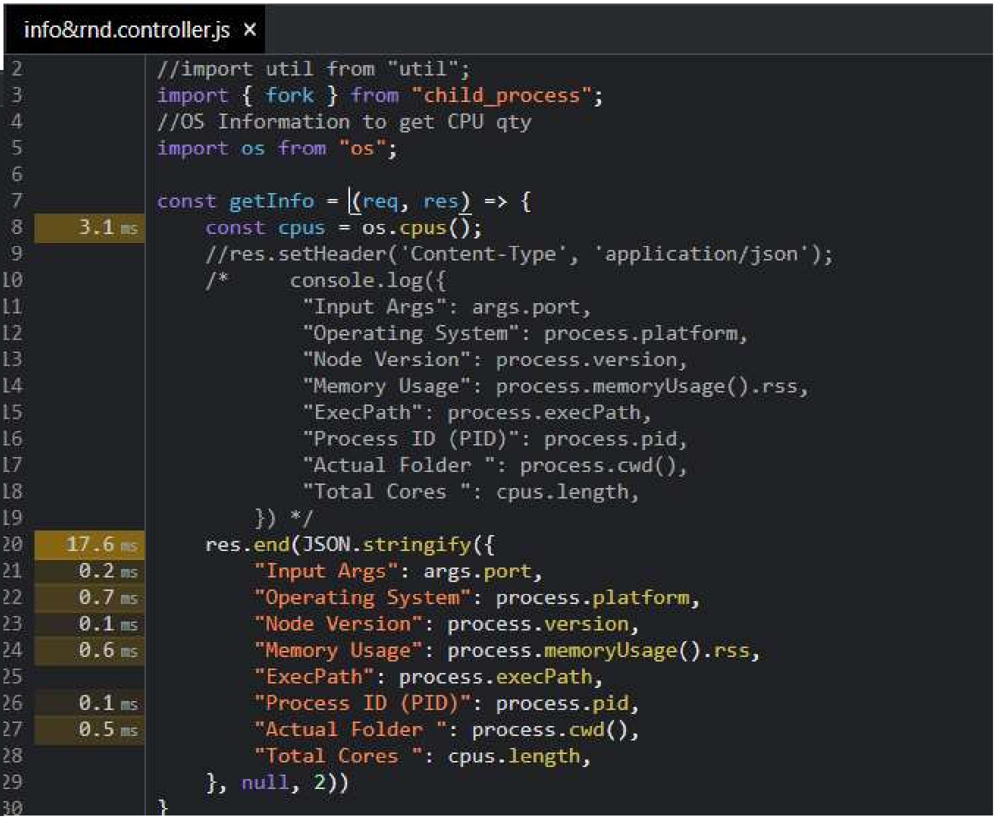


```
           import { args } from "../server.js";
           //import util from "util";
           import { fork } from "child_process";
           //OS Information to get CPU qty
           import os from "os";

           const getInfo = (req, res) => {
  3.6 ms       const cpus = os.cpus();
               //res.setHeader('Content-Type', 'application/json');
  9.0 ms       console.log({
  0.5 ms           "Input Args": args.port,
  1.6 ms           "Operating System": process.platform,
  0.7 ms           "Node Version": process.version,
  1.2 ms           "Memory Usage": process.memoryUsage().rss,
  0.4 ms           "ExecPath": process.execPath,
  0.2 ms           "Process ID (PID)": process.pid,
  0.5 ms           "Actual Folder ": process.cwd(),
                   "Total Cores ": cpus.length,
               })
 35.9 ms       res.end(JSON.stringify({
  0.6 ms           "Input Args": args.port,
  2.1 ms           "Operating System": process.platform,
  0.5 ms           "Node Version": process.version,
  0.7 ms           "Memory Usage": process.memoryUsage().rss,
  0.5 ms           "ExecPath": process.execPath,
  0.5 ms           "Process ID (PID)": process.pid,
  0.5 ms           "Actual Folder ": process.cwd(),
  0.1 ms           "Total Cores ": cpus.length,
               }, null, 2))
           }
```

SIN Console Log



Self Time		Total Time		Function
24766.5 ms		24766.5 ms		(idle)
293.7 ms	13.96 %	313.5 ms	14.90 %	▼getCallers
293.6 ms	13.96 %	313.4 ms	14.90 %	▼pino
293.6 ms	13.96 %	313.4 ms	14.90 %	▼consoleLogger
293.6 ms	13.96 %	313.4 ms	14.90 %	▶urlRegister
0.1 ms	0.01 %	0.1 ms	0.01 %	▼consoleLogger
0.1 ms	0.01 %	0.1 ms	0.01 %	▶urlRegister
244.3 ms	11.61 %	244.3 ms	11.61 %	▼getCPUs
244.3 ms	11.61 %	244.3 ms	11.61 %	▶cpus
71.8 ms	3.42 %	71.8 ms	3.42 %	(program)
59.9 ms	2.85 %	59.9 ms	2.85 %	(garbage collector)
57.6 ms	2.74 %	1316.3 ms	62.57 %	▶initialize
47.2 ms	2.25 %	47.2 ms	2.25 %	▶writev
44.8 ms	2.13 %	490.3 ms	23.31 %	▼pino
44.7 ms	2.12 %	490.2 ms	23.30 %	▶consoleLogger
0.1 ms	0.01 %	0.1 ms	0.01 %	▶urlRegister
31.6 ms	1.50 %	12905.9 ms	613.50 %	▶next
31.5 ms	1.50 %	1512.7 ms	71.91 %	▶session
31.0 ms	1.47 %	100.5 ms	4.78 %	▶hash
30.3 ms	1.44 %	61.5 ms	2.92 %	▶normalizeArgs
28.0 ms	1.33 %	13447.8 ms	639.27 %	▶handle
24.6 ms	1.17 %	49.3 ms	2.34 %	▶nextTick
24.1 ms	1.15 %	40.2 ms	1.91 %	▶writeHead

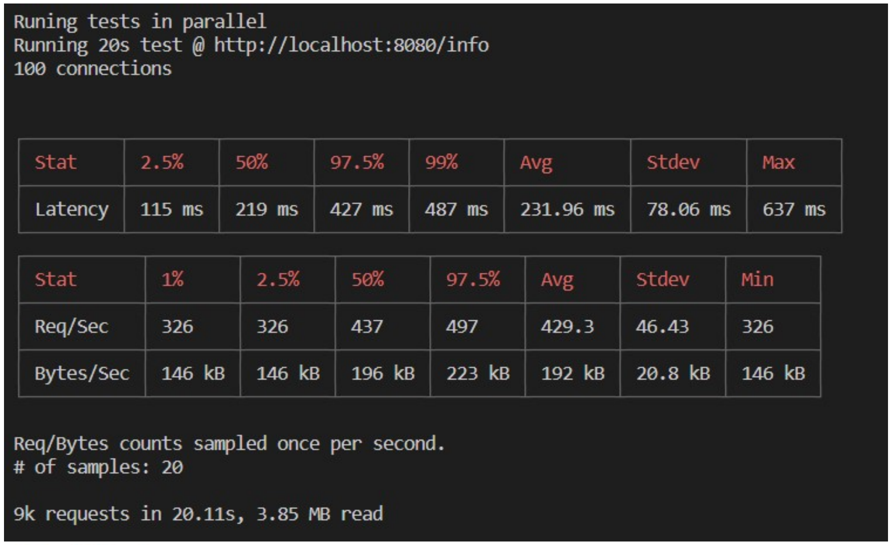


info&rnd.controller.js ×

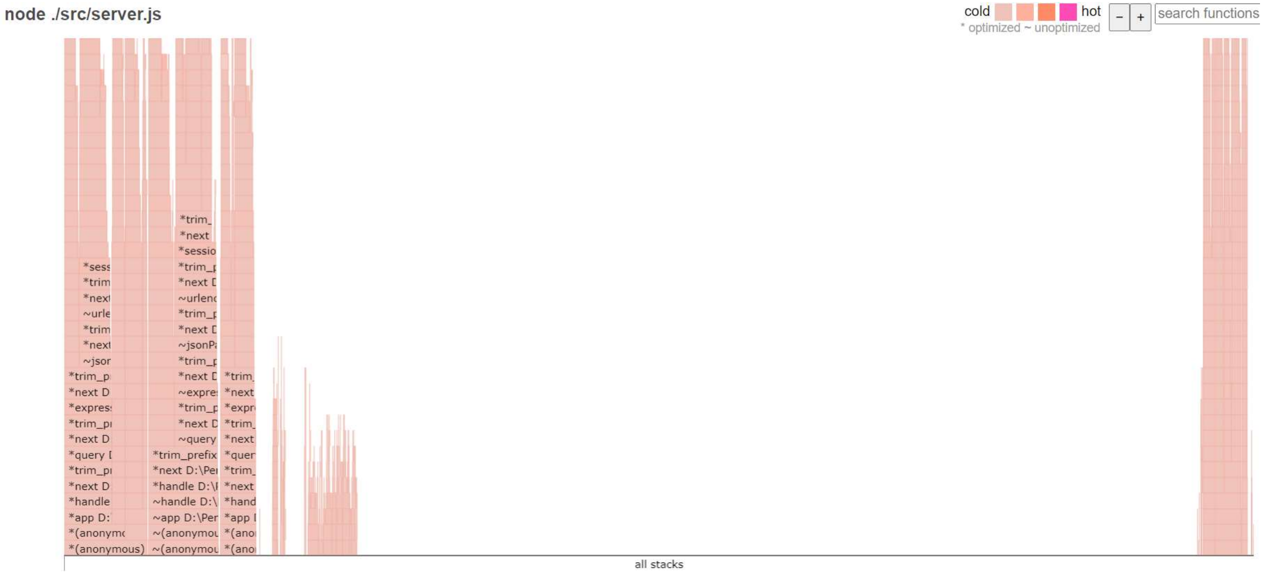
```
           //import util from "util";
           import { fork } from "child_process";
           //OS Information to get CPU qty
           import os from "os";

           const getInfo = (req, res) => {
  3.1 ms       const cpus = os.cpus();
               //res.setHeader('Content-Type', 'application/json');
               /*     console.log({
                        "Input Args": args.port,
                        "Operating System": process.platform,
                        "Node Version": process.version,
                        "Memory Usage": process.memoryUsage().rss,
                        "ExecPath": process.execPath,
                        "Process ID (PID)": process.pid,
                        "Actual Folder ": process.cwd(),
                        "Total Cores ": cpus.length,
                    }) */
 17.6 ms       res.end(JSON.stringify({
  0.2 ms           "Input Args": args.port,
  0.7 ms           "Operating System": process.platform,
  0.1 ms           "Node Version": process.version,
  0.6 ms           "Memory Usage": process.memoryUsage().rss,
                   "ExecPath": process.execPath,
  0.1 ms           "Process ID (PID)": process.pid,
  0.5 ms           "Actual Folder ": process.cwd(),
                   "Total Cores ": cpus.length,
               }, null, 2))
           }
```

Diagrama de Flama 0x con Autocannon

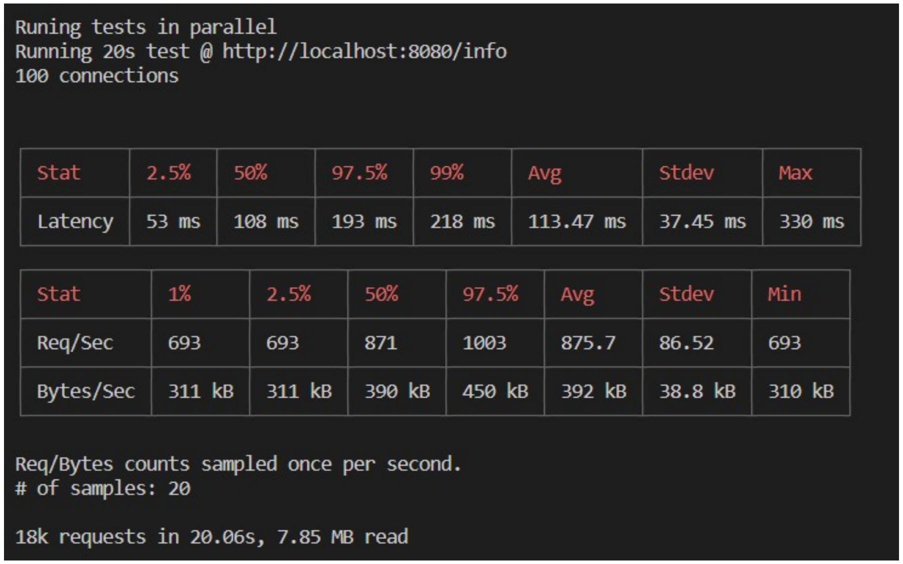
CON console Log



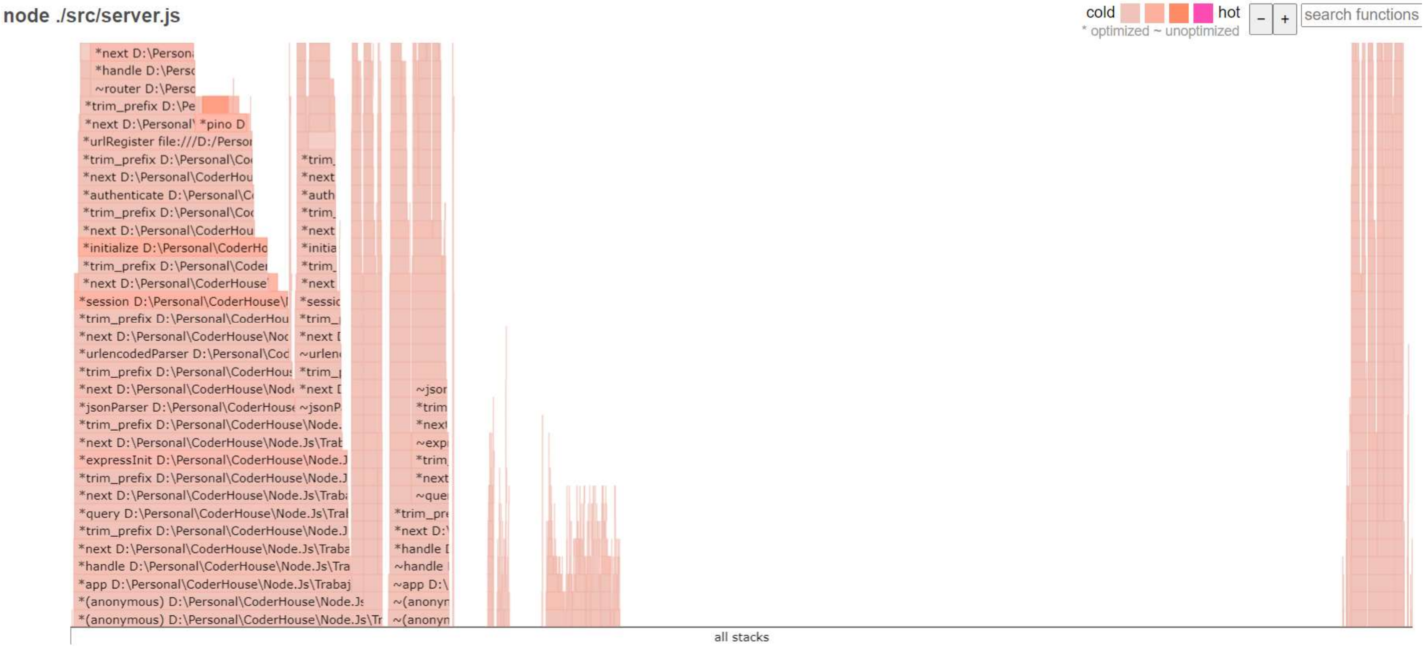
```
Runing tests in parallel
Running 20s test @ http://localhost:8080/info
100 connections


┌─────────┬────────┬────────┬────────┬────────┬───────────┬──────────┬────────┐
│ Stat    │ 2.5%   │ 50%    │ 97.5%  │ 99%    │ Avg       │ Stdev    │ Max    │
├─────────┼────────┼────────┼────────┼────────┼───────────┼──────────┼────────┤
│ Latency │ 115 ms │ 219 ms │ 427 ms │ 487 ms │ 231.96 ms │ 78.06 ms │ 637 ms │
└─────────┴────────┴────────┴────────┴────────┴───────────┴──────────┴────────┘
┌───────────┬────────┬────────┬────────┬────────┬────────┬─────────┬────────┐
│ Stat      │ 1%     │ 2.5%   │ 50%    │ 97.5%  │ Avg    │ Stdev   │ Min    │
├───────────┼────────┼────────┼────────┼────────┼────────┼─────────┼────────┤
│ Req/Sec   │ 326    │ 326    │ 437    │ 497    │ 429.3  │ 46.43   │ 326    │
├───────────┼────────┼────────┼────────┼────────┼────────┼─────────┼────────┤
│ Bytes/Sec │ 146 kB │ 146 kB │ 196 kB │ 223 kB │ 192 kB │ 20.8 kB │ 146 kB │
└───────────┴────────┴────────┴────────┴────────┴────────┴─────────┴────────┘

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

9k requests in 20.11s, 3.85 MB read
```



SIN Console Log:



```
Runing tests in parallel
Running 20s test @ http://localhost:8080/info
100 connections


| Stat    | 2.5%  | 50%    | 97.5%  | 99%    | Avg       | Stdev    | Max    |
| Latency | 53 ms | 108 ms | 193 ms | 218 ms | 113.47 ms | 37.45 ms | 330 ms |

| Stat      | 1%     | 2.5%   | 50%    | 97.5%  | Avg    | Stdev   | Min    |
| Req/Sec   | 693    | 693    | 871    | 1003   | 875.7  | 86.52   | 693    |
| Bytes/Sec | 311 kB | 311 kB | 390 kB | 450 kB | 392 kB | 38.8 kB | 310 kB |

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

18k requests in 20.06s, 7.85 MB read
```



CONCLUSION y Notas:

Se puede ver una perdida de performance cuando colocamos la sentencia de console.log antes de responder la request.

Para Artillery y el informe generado con profiler, se ve que la tasa de request es mayor son el CL y que los tiempos de respuesta, también.

Con Inspect de chrome , claramente vemos los tiempos dentro del método de GetInfo y se nota que , obviamente, tenemos mas líneas y mas tiempo de ejecución , con esto trayendo mas tiempo para la misma.

Para el informe con Autocannon , y 0x, vemos también esta diferencia, en la latencia y los B/seg enviados.

Viendo que la caída es grande, claramente se recomiendo no utilizar console.log fuera de un ámbito de desarrollo y test ya que multiplicar esta degradación de performance por cada console.log en el codigo bajaría drásticamente la misma.