Análisis de performance del servidor

Node Profiling

Sin console.log()

```
[Summary]:
  ticks  total  nonlib   name
     34   0.8%  100.0%  JavaScript
      0   0.0%    0.0%  C++
     18   0.4%   52.9%  GC
   4218  99.2%          Shared libraries
```

Con console.log()

```
[Summary]:
  ticks  total  nonlib   name
     45   0.9%   95.7%  JavaScript
      0   0.0%    0.0%  C++
     40   0.8%   85.1%  GC
   4826  99.0%          Shared libraries
      2   0.0%          Unaccounted
```

Análisis:

Mediante el uso del built-in profiler de Node se puede observar que la cantidad de ticks (shared libraries) para la ruta info que ejecuta el console.log es mayor. Esto indica una mayor cantidad de ciclos de ejecución, lo que implica un mayor costo de recursos y una latencia en los tiempos de respuesta aumentada en relacion a la ruta info que no ejecuta dicha función.

Node inspect

sin console.log()

Self Time		Total Time		Function
0.2 ms	0.01 %	172.4 ms	8.25 %	▸ ondata
3.0 ms	0.14 %	161.6 ms	7.74 %	▸ getInfo
4.1 ms	0.20 %	157.5 ms	7.54 %	▸ compileChildren

```
56   0.6 ms  async function getInfo(req,res){
57               try{
58   0.1 ms          const info = {
59                       argumentos: args,
60   0.1 ms              sistema: process.platform,
61                       node: process.version,
62                       memoria: process.memoryUsage().rss,
63                       path: process.execPath,
64   0.3 ms              processID: process.pid,
65   0.1 ms              folder: process.cwd(),
66                       cpus: numCPUs
67                   }
68   0.5 ms          if(Object.values(info).length){
69   0.1 ms              if(config.clog){
70                           console.log(`info:`, info)
71                       }
72   0.7 ms              res.render('info', {info})
73               }
74   0.5 ms      }catch(err){logger.error(err)}
75           };
```

Con console.log()

Self Time		Total Time		Function
0.7 ms	0.02 %	0.7 ms	0.02 %	▸ getKeys
9.2 ms	0.28 %	1114.5 ms	33.87 %	▸ getInfo
0.5 ms	0.01 %	1.0 ms	0.03 %	▸ getHighWaterMark

```
56   0.5 ms  async function getInfo(req,res){
57               try{
58                   const info = {
59   0.2 ms              argumentos: args,
60   0.1 ms              sistema: process.platform,
61                       node: process.version,
62   0.5 ms              memoria: process.memoryUsage().rss,
63   0.1 ms              path: process.execPath,
64                       processID: process.pid,
65   0.1 ms              folder: process.cwd(),
66                       cpus: numCPUs
67                   }
68   0.3 ms          if(Object.values(info).length){
69                       if(config.clog){
70   2.5 ms                  console.log(`info:`, info)
71                       }
72   4.4 ms              res.render('info', {info})
73                   }
74   0.5 ms      }catch(err){logger.error(err)}
75           };
```

Análisis:

Los datos observados mediante Node DevTools del browser dan cuenta del aumento en los tiempos de respuesta del la ruta /info. Se puede apreciar cómo afecta a la función getInfo() la inclusión de un console.log() en 2.5ms por ciclo.

Autocannon

Sin console.log()

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	94 ms	6118 ms	9669 ms	9679 ms	5495.92 ms	3047.08 ms	9685 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	0	0	9	35	10	8.36	1
Bytes/Sec	0 B	0 B	19.7 kB	76.7 kB	21.9 kB	18.3 kB	2.19 kB

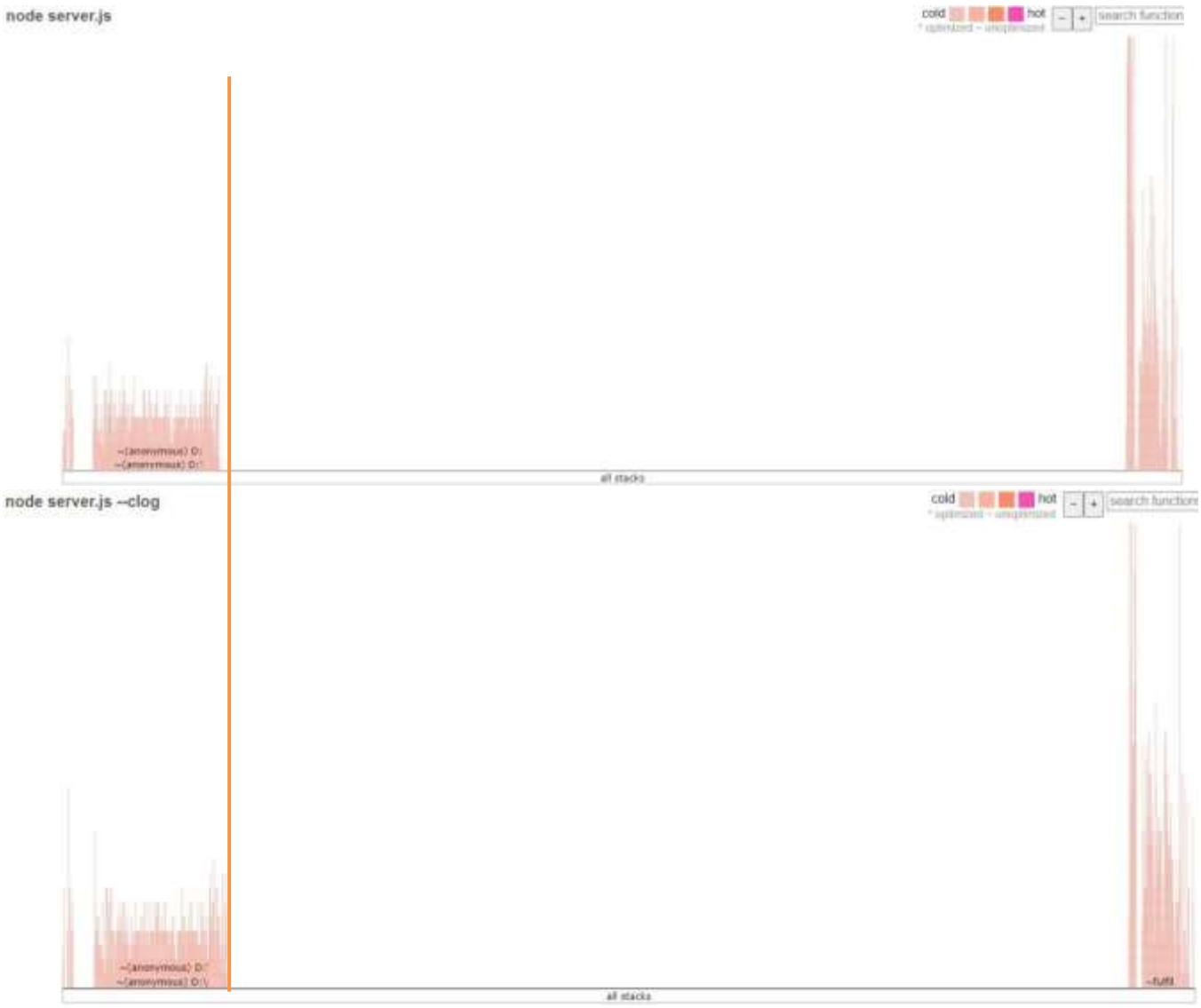
Con console.log()

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	111 ms	6204 ms	9691 ms	9698 ms	5531.42 ms	3050.47 ms	9701 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	0	0	9	35	10	8.58	1
Bytes/Sec	0 B	0 B	19.9 kB	77.4 kB	22.1 kB	19 kB	2.21 kB

Análisis:
A través de autocannon fueron creadas 100 conexiones durante 20 segundos. Como resultado del test se puede observar una mayor latencia en la ruta /info que ejecuta console.log() () (P99 = 9698ms vs P99 = 9678ms s/console.log()). Por otro lado se puede observar que la cantidad promedio de request procesadas se establecio en 35 por segundo en ambos casos

0x flamegraph



Análisis:
Se puede observar que la ruta /info con console.log() posee un mayor extensión a nivel horizontal en la gráfica de procesos bloqueantes (lado izquierdo) indicando un mayor tiempo en la ejecución de los procesos.

Conclusión:

Mediante las diferentes instancias de testeo se pudo determinar que para la ruta /info que ejecuta un simple console.log() existe una diferencia en los tiempos de procesamiento de los request, debido a esta ejecución bloqueante el servidor da cuenta de una mínima perdida de performance aumentado el costo en recursos y aumentando su latencia en cuanto a tiempos de respuesta.