Loggers, Gzip y análisis de performance

1) Perfilamiento con –prof de node.js

Artillery

Sin console log

```
Statistical profiling result from sin-consoleLog-v8.log, (1876 ticks, 11 unaccounted,
0 excluded).

 [Shared libraries]:
   ticks  total  nonlib   name
     22    1.2%          /usr/lib/system/libsystem_c.dylib
     20    1.1%          /usr/lib/system/libsystem_malloc.dylib
     18    1.0%          /usr/lib/system/libsystem_pthread.dylib
     14    0.7%          /usr/lib/libc++.1.dylib
     13    0.7%          /usr/lib/system/libsystem_kernel.dylib
      3    0.2%          /usr/lib/system/libsystem_platform.dylib
      2    0.1%          /usr/lib/libc++abi.dylib
      1    0.1%          /usr/lib/system/libdyld.dylib
```

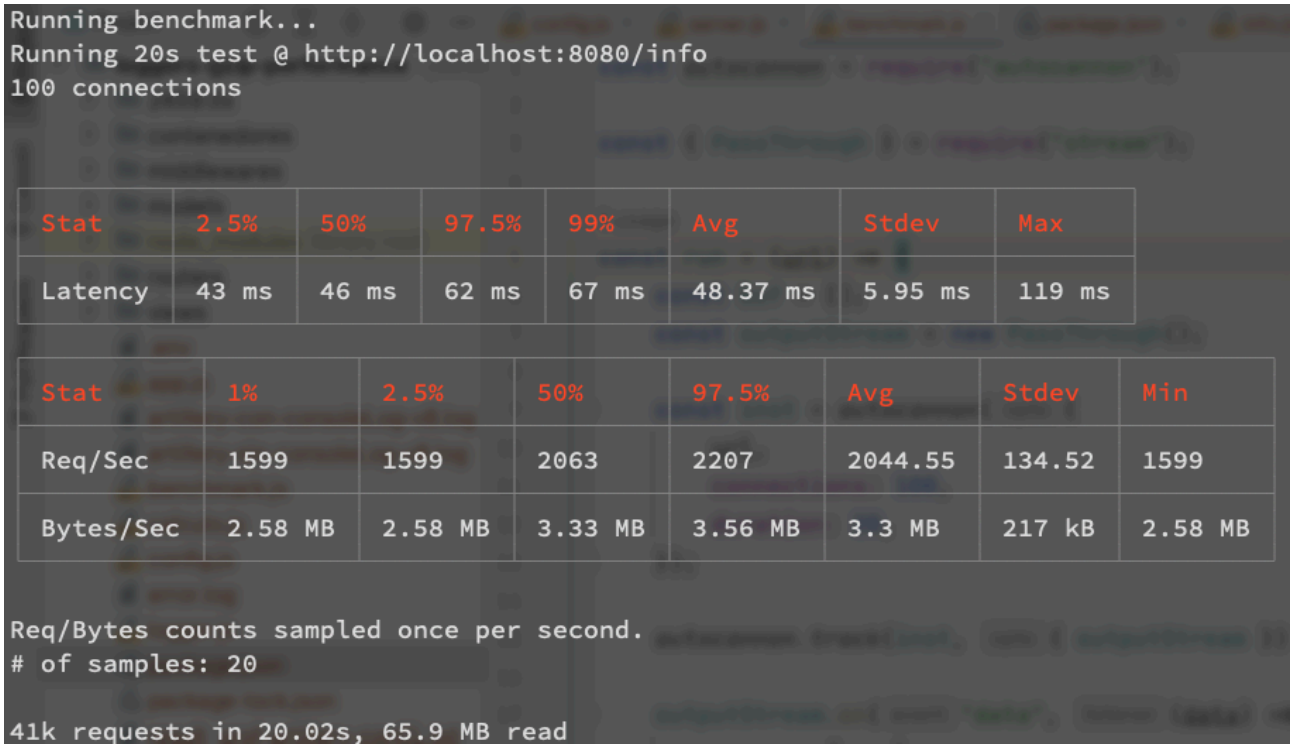
Con console.log

```
Statistical profiling result from con-consoleLog-v8.log, (2061 ticks, 13 unaccounted,
0 excluded).

 [Shared libraries]:
   ticks  total  nonlib   name
     25    1.2%          /usr/lib/system/libsystem_pthread.dylib
     19    0.9%          /usr/lib/system/libsystem_kernel.dylib
     18    0.9%          /usr/lib/system/libsystem_malloc.dylib
     13    0.6%          /usr/lib/system/libsystem_c.dylib
     11    0.5%          /usr/lib/libc++.1.dylib
      2    0.1%          /usr/lib/system/libsystem_platform.dylib
```

Autocannon

Sin console.log



```
Running benchmark...
Running 20s test @ http://localhost:8080/info
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	43 ms	46 ms	62 ms	67 ms	48.37 ms	5.95 ms	119 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	1599	1599	2063	2207	2044.55	134.52	1599
Bytes/Sec	2.58 MB	2.58 MB	3.33 MB	3.56 MB	3.3 MB	217 kB	2.58 MB

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

41k requests in 20.02s, 65.9 MB read
```

Con console.log

```
Running benchmark...
Running 20s test @ http://localhost:8080/info
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	46 ms	47 ms	60 ms	68 ms	48.79 ms	5.5 ms	136 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	1464	1464	2077	2101	2028.4	141.45	1464
Bytes/Sec	2.36 MB	2.36 MB	3.35 MB	3.39 MB	3.27 MB	228 kB	2.36 MB

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

41k requests in 20.02s, 65.4 MB read
```

2) Perfilamiento con node inspect

Sin console.log

info.js ×

```
1                const {Router} = require('express')
2                const infoWebRouter = Router()
3                const yargs = require('yargs/yargs')(process.argv.slice(2))
4                const args = yargs.argv
5                const logger = require('../../logger')
6
7                infoWebRouter.get('/info', (req, res)=>{
8                    const info = {
9                        entrada: args,
10     0.1 ms        plataforma: process.platform,
11                   version: process.version,
12                   memoria: process.memoryUsage().rss,
13                   path: process.cwd(),
14                   pid: process.pid,
15               }
16               // console.log(info)
17     1.3 ms    res.render('info',{info})
18           })
19
20           module.exports = infoWebRouter
```

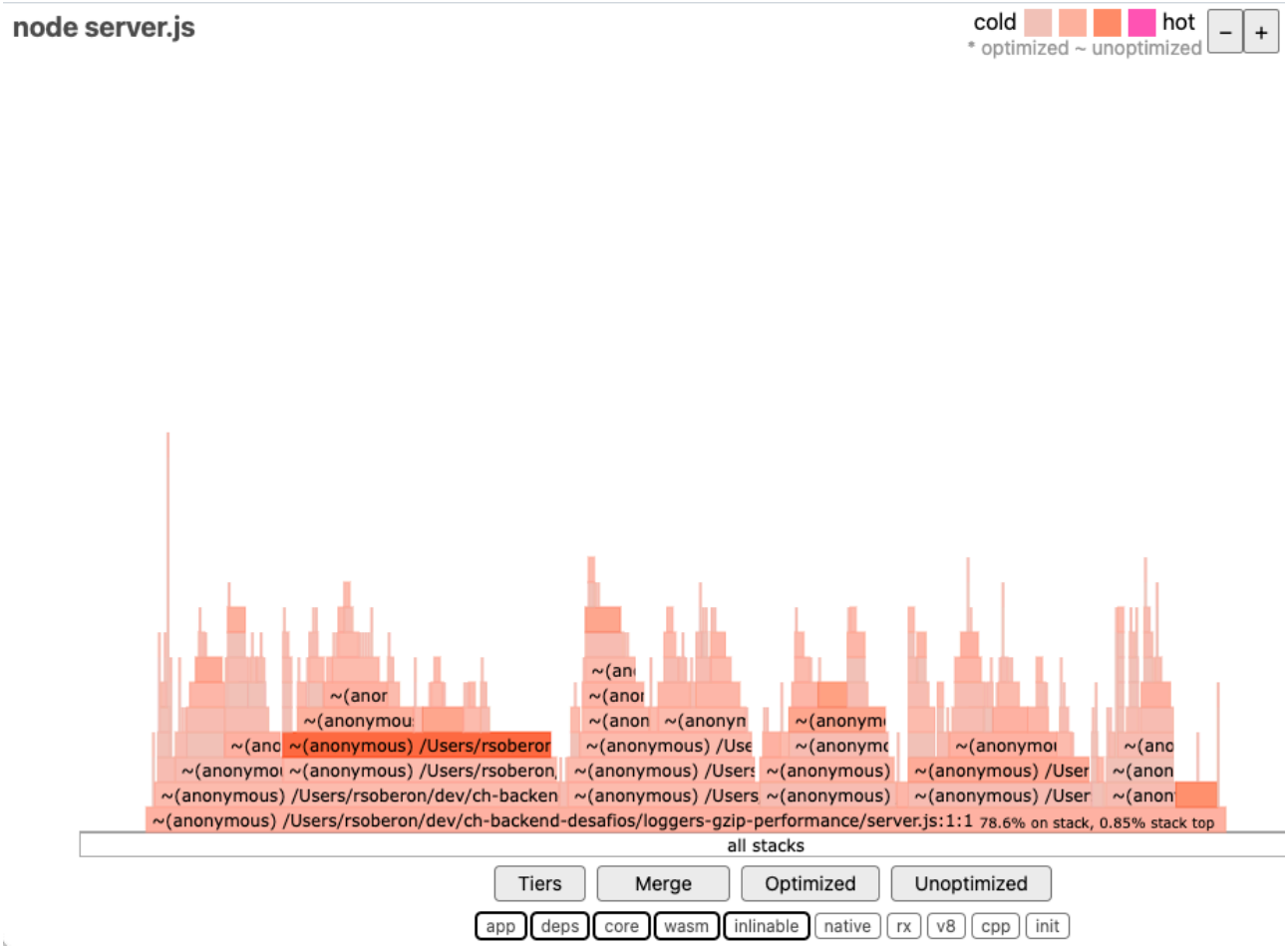
Con console.log

info.js

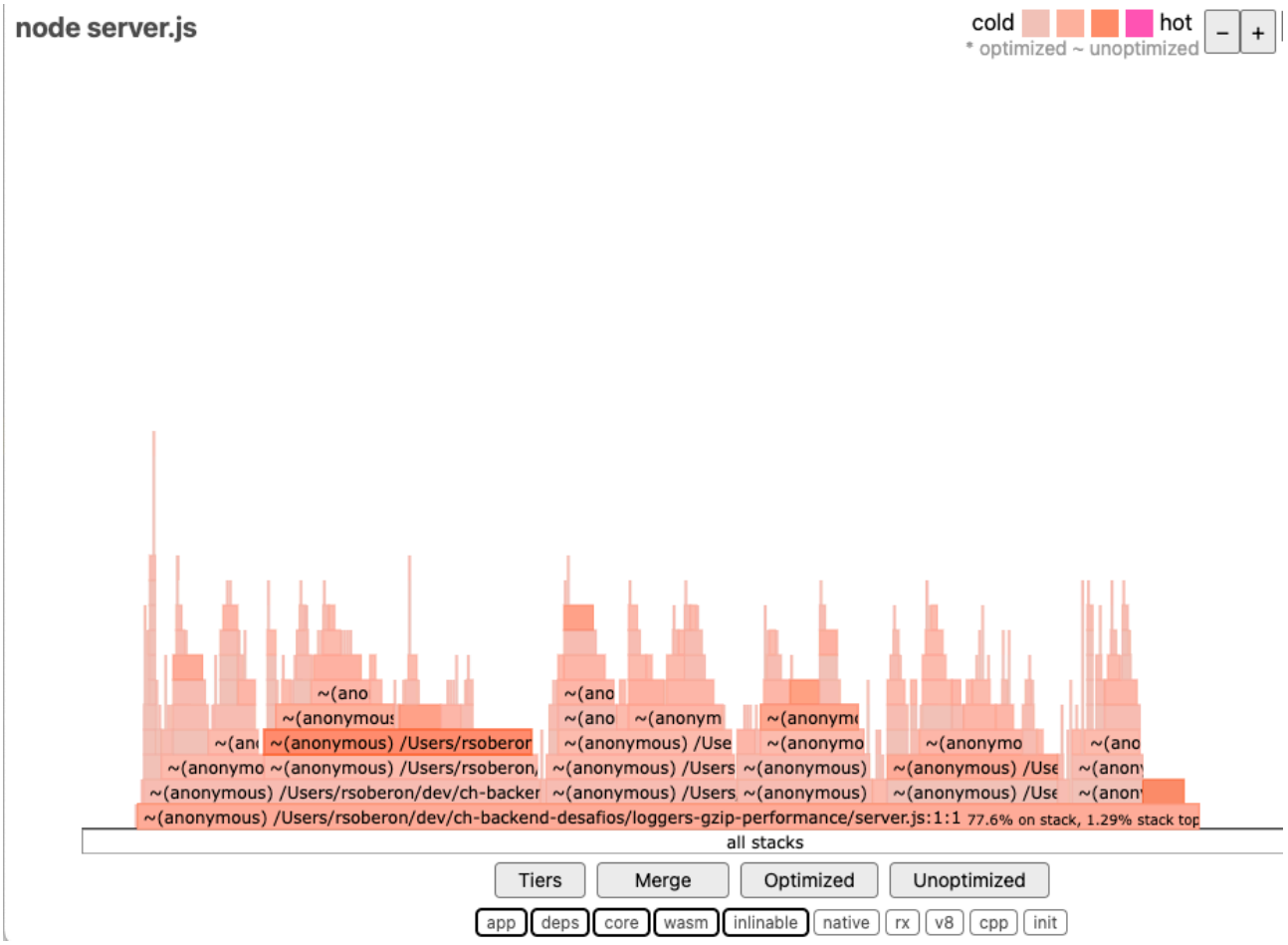
```
1                const {Router} = require('express')
2                const infoWebRouter = Router()
3                const yargs = require('yargs/yargs')(process.argv.slice(2))
4                const args = yargs.argv
5                const logger = require('../../logger')
6
7                infoWebRouter.get('/info', (req, res)=>{
8          0.1 ms    const info = {
9                        entrada: args,
10                       plataforma: process.platform,
11         0.3 ms        version: process.version,
12                       memoria: process.memoryUsage().rss,
13                       path: process.cwd(),
14                       pid: process.pid,
15                   }
16         0.3 ms    console.log(info)
17         0.5 ms    res.render('info',{info})
18               })
19
20               module.exports = infoWebRouter
```

3) Diagramas de flama con 0x

Sin console.log



Con console.log



Conclusión

A partir de la información recolectada, se puede observar y concluir que los procesos bloqueantes, como el de *console.log*, suponen una pérdida de performance. Si bien en las pruebas realizadas no tuvo un gran impacto para el usuario, al incluir este tipo de acciones en múltiples lugares de nuestro código, podría empezar a ser notorias y tener un impacto en la experiencia final.