

1) Perfilamiento del servidor con -prof

Artillery – Bloqueante

```
artillery_prof-bloq.txt U x
desafio14 > tests > --prof > artillery > artillery_prof-bloq.txt
1 Statistical profiling result from artillery_bloq-v8.log, (2891 ticks, 0 unaccounted, 0 excluded).
2
3 [Shared libraries]:
4   ticks  total  nonlib   name
5   2620   90.6%          C:\WINDOWS\SYSTEM32\ntdll.dll
6    265    9.2%          D:\Archivos de Programa\nodejs\node.exe
7
8 [JavaScript]:
9   ticks  total  nonlib   name
10    1     0.0%  16.7% LazyCompile: *toNamespacedPath node:path:618:19
11    1     0.0%  16.7% LazyCompile: *slowCases node:internal/util:165:19
12    1     0.0%  16.7% Function: ^validateEncoding node:internal/validators:198:26
13    1     0.0%  16.7% Function: ^query C:\Users\itinc\Desktop\Coder\BackEnd\desafios-backend-32070-coderhouse\node_modules\express\lib\middleware\query.js:39:24
14    1     0.0%  16.7% Function: ^normalizeParseOptions C:\Users\itinc\Desktop\Coder\BackEnd\desafios-backend-32070-coderhouse\node_modules\qs\lib\parse.js:204:59
15    1     0.0%  16.7% Function: ^createWriteHead C:\Users\itinc\Desktop\Coder\BackEnd\desafios-backend-32070-coderhouse\node_modules\on-headers\index.js:24:26
16
17 [C++]:
18   ticks  total  nonlib   name
19
20 [Summary]:
21   ticks  total  nonlib   name
22    6     0.2% 100.0% JavaScript
23    0     0.0%   0.0% C++
24   19     0.7% 316.7% GC
25  2885   99.8%          Shared libraries
26
27 [C++ entry points]:
28   ticks  cpp  total   name
```

Artillery – NoBloqueante

```
artillery_prof-nobloq.txt U X
desafio14 > tests > --prof > artillery > artillery_prof-nobloq.txt
1  Statistical profiling result from artillery_nobloq-v8.log, (1803 ticks, 0 unaccounted, 0 excluded).
2
3  [Shared libraries]:
4  | ticks  total  nonlib   name
5  | 1646   91.3%          C:\WINDOWS\SYSTEM32\ntdll.dll
6  | 155     8.6%          D:\Archivos de Programa\nodejs\node.exe
7
8  [JavaScript]:
9  | ticks  total  nonlib   name
10 | 1       0.1%   50.0%   Function: ^resOnFinish node:_http_server:788:21
11 | 1       0.1%   50.0%   Function: ^onstat C:\Users\itinc\Desktop\Coder\BackEnd\desafios-backend-32070-coderhouse\node_modules\send\index.js:717:33
12
13 [C++]:
14 | ticks  total  nonlib   name
15
16 [Summary]:
17 | ticks  total  nonlib   name
18 | 2       0.1%   100.0%   JavaScript
19 | 0       0.0%    0.0%   C++
20 | 9       0.5%   450.0%   GC
21 | 1801    99.9%          Shared libraries
22
23 [C++ entry points]:
24 | ticks  cpp    total   name
```

Se observó que el proceso no bloqueante lleva un 37% menos de ticks a diferencia del proceso bloqueante.

Comparación log de Test de Artillery: Bloqueante | NoBloqueante

```
result_artillery_bloq.txt U X
desafio14 > tests > --prof > artillery > result_artillery_bloq.txt
1 Running scenarios...
2 Phase started: unnamed (index: 0, duration: 1s) 23:30:55(-0300)
3
4 Phase completed: unnamed (index: 0, duration: 1s) 23:30:56(-0300)
5
6 All VUs finished. Total time: 5 seconds
7
8 -----
9 Summary report @ 23:30:59(-0300)
10 -----
11
12 http.codes.200: ..... 1000
13 http.request_rate: ..... 178/sec
14 http.requests: ..... 1000
15 http.response_time:
16   min: ..... 2
17   max: ..... 124
18   median: ..... 49.9
19   p95: ..... 96.6
20   p99: ..... 117.9
21 http.responses: ..... 1000
22 vusers.completed: ..... 50
23 vusers.created: ..... 50
24 vusers.created_by_name.0: ..... 50
25 vusers.failed: ..... 0
26 vusers.session_length:
27   min: ..... 761.9
28   max: ..... 1674.6
29   median: ..... 1436.8
30   p95: ..... 1587.9
31   p99: ..... 1620
32

result_artillery_nobloq.txt U X
desafio14 > tests > --prof > artillery > result_artillery_nobloq.txt
1 Running scenarios...
2 Phase started: unnamed (index: 0, duration: 1s) 23:33:52(-0300)
3
4 Phase completed: unnamed (index: 0, duration: 1s) 23:33:53(-0300)
5
6 All VUs finished. Total time: 5 seconds
7
8 -----
9 Summary report @ 23:33:56(-0300)
10 -----
11
12 http.codes.200: ..... 1000
13 http.request_rate: ..... 457/sec
14 http.requests: ..... 1000
15 http.response_time:
16   min: ..... 1
17   max: ..... 42
18   median: ..... 10.1
19   p95: ..... 19.1
20   p99: ..... 23.8
21 http.responses: ..... 1000
22 vusers.completed: ..... 50
23 vusers.created: ..... 50
24 vusers.created_by_name.0: ..... 50
25 vusers.failed: ..... 0
26 vusers.session_length:
27   min: ..... 127.9
28   max: ..... 338.8
29   median: ..... 295.9
30   p95: ..... 333.7
31   p99: ..... 340.4
32
```

Se observa que en el proceso no bloqueante la tasa de respuesta es significativamente menor.

Autocannon – Bloqueante

```
desafio14 > tests > --prof > autocannon > autocannon_prof-bloq.txt
58
59 [C++]:
60 | ticks total nonlib name
61
62 [Summary]:
63 | ticks total nonlib name
64 | 60 3.0% 98.4% JavaScript
65 | 0 0.0% 0.0% C++
66 | 39 2.0% 63.9% GC
67 | 1922 96.9% Shared libraries
68 | 1 0.1% Unaccounted
69
70 [C++ entry points]:
71 | ticks cpp total name
72
73 [Bottom up (heavy) profile]:
74 Note: percentage shows a share of a particular caller in the total
75 amount of its parent calls.
76 Callers occupying less than 1.0% are not shown.
77
```

Autocannon - NoBloqueante

```
desafio14 > tests > --prof > autocannon > autocannon_prof-nobloq.txt
84
85 [C++]:
86 | ticks total nonlib name
87
88 [Summary]:
89 | ticks total nonlib name
90 | 116 6.5% 99.1% JavaScript
91 | 0 0.0% 0.0% C++
92 | 102 5.8% 87.2% GC
93 | 1655 93.4% Shared libraries
94 | 1 0.1% Unaccounted
95
96 [C++ entry points]:
97 | ticks cpp total name
98
99 [Bottom up (heavy) profile]:
100 Note: percentage shows a share of a particular caller in the total
101 amount of its parent calls.
102 Callers occupying less than 1.0% are not shown.
```

De la misma manera, en el proceso no bloqueante la cantidad de ticks es menor. aunque por otro lado la diferencia ambos procesos tambien es menor.

Comparación log de Test de Autocannon: Bloqueante | NoBloqueante: Se observa que la cantidad de requests en 20 segundos en el proceso no bloqueante es 4 veces mayor.

```
PS C:\Users\itinc\Desktop\Coder\BackEnd\desafios-backend-32070-coderhouse\desafio14> autocannon -c 100 -d 20 http://localhost:8080/info
Running 20s test @ http://localhost:8080/info
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	129 ms	177 ms	279 ms	283 ms	184.48 ms	43.78 ms	396 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	399	399	568	680	539.96	82.88	399
Bytes/Sec	307 kB	307 kB	438 kB	525 kB	416 kB	64.2 kB	307 kB

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

11k requests in 20.13s, 8.33 MB read

```
PS C:\Users\itinc\Desktop\Coder\BackEnd\desafios-backend-32070-coderhouse\desafio14> ^C
```

```
PS C:\Users\itinc\Desktop\Coder\BackEnd\desafios-backend-32070-coderhouse\desafio14> |
```

```
PS C:\Users\itinc\Desktop\Coder\BackEnd\desafios-backend-32070-coderhouse\desafio14> autocannon -c 100 -d 20 http://localhost:8080/info
Running 20s test @ http://localhost:8080/info
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	35 ms	40 ms	69 ms	80 ms	42.85 ms	10.15 ms	154 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	1400	1400	2401	2589	2310.56	286.64	1400
Bytes/Sec	1.08 MB	1.08 MB	1.86 MB	2 MB	1.79 MB	222 kB	1.08 MB

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

46k requests in 20.08s, 35.7 MB read

```
PS C:\Users\itinc\Desktop\Coder\BackEnd\desafios-backend-32070-coderhouse\desafio14> |
```

2) Perfilamiento del servidor con –inspect

Artillery – Bloqueante

or de perfiles Fuentes Memoria					
Pesado (de abajo hacia arriba) 🔍 ✕ ↺					
Tiempo individual ▼		Tiempo total		Función	
24385.0 ms		24385.0 ms		(idle)	
590.3 ms	23.30%	1226.5 ms	48.42%	▼ consoleCall	
590.3 ms	23.30%	1226.5 ms	48.42%	▼ (anónimas)	info.router.js:21
590.3 ms	23.30%	1226.5 ms	48.42%	▼ handle	layer.js:86
590.3 ms	23.30%	1226.5 ms	48.42%	▼ next	route.js:116
590.3 ms	23.30%	1226.5 ms	48.42%	▼ dispatch	route.js:98
590.3 ms	23.30%	1226.5 ms	48.42%	▼ handle	layer.js:86
590.3 ms	23.30%	1226.5 ms	48.42%	▼ (anónimas)	index.js:280
590.3 ms	23.30%	1226.5 ms	48.42%	▼ process_params	index.js:338
590.3 ms	23.30%	1226.5 ms	48.42%	▼ next	index.js:177
590.3 ms	23.30%	1226.5 ms	48.42%	▼ handle	index.js:136
590.3 ms	23.30%	1226.5 ms	48.42%	▶ router	index.js:46
560.1 ms	22.11%	560.1 ms	22.11%	▶ writeUtf8String	
121.5 ms	4.80%	121.5 ms	4.80%	▶ writev	
80.8 ms	3.19%	80.8 ms	3.19%	(garbage collector)	
43.3 ms	1.71%	1666.7 ms	65.80%	▶ initialize	initialize.js:51
42.2 ms	1.67%	42.2 ms	1.67%	(program)	
21.3 ms	0.84%	38.6 ms	1.52%	▶ nextTick	node:internal/p...ask_queues:104
19.5 ms	0.77%	98.5 ms	3.89%	▶ session	index.js:179
19.1 ms	0.75%	12256.9 ms	483.89%	▶ handle	layer.js:86
18.7 ms	0.74%	33.1 ms	1.31%	▶ stat	node:fs:1452
18.5 ms	0.73%	18.5 ms	0.73%	▶ getColorDepth	node:internal/tty:106
18.0 ms	0.71%	35.9 ms	1.42%	▶ Hash	node:internal/crypto/hash:58
17.7 ms	0.70%	17.7 ms	0.70%	▶ Hash	
17.7 ms	0.70%	173.0 ms	6.83%	▶ cookieParser	index.js:44
17.6 ms	0.70%	49.0 ms	1.94%	▶ hash	index.js:596
17.1 ms	0.68%	603.6 ms	23.83%	▶ send	response.js:111
16.4 ms	0.65%	27.3 ms	1.08%	▶ writeHead	node: http_server:269
15.8 ms	0.62%	1587.5 ms	62.67%	▶ (anónimas)	info.router.js:21
15.1 ms	0.60%	23.8 ms	0.94%	▶ set	memory.js:119
14.5 ms	0.57%	80.8 ms	3.19%	▶ compression	index.js:59
13.8 ms	0.54%	10699.7 ms	422.41%	▶ next	index.js:177
13.2 ms	0.52%	15.0 ms	0.59%	▶ parse	index.js:106
12.8 ms	0.50%	12.8 ms	0.50%	▶ digest	
11.7 ms	0.46%	190.5 ms	7.52%	▶ expressInit	init.js:29
11.3 ms	0.45%	11.3 ms	0.45%	▶ status	response.js:67
11.2 ms	0.44%	11.2 ms	0.44%	▶ setSession	memory.js:164

Aquí se observa que la función menos performante es consoleCall, ejecutada en info.router.js:21

Artillery – NoBloqueante

Fuentes		Memoria		
Pesado (de abajo hacia arriba)				
Tiempo individual		Tiempo total		Función
23130.6 ms		23130.6 ms		(idle)
159.6 ms	13.11%	159.6 ms	13.11%	writev
101.1 ms	8.31%	101.1 ms	8.31%	(garbage collector)
33.8 ms	2.77%	33.8 ms	2.77%	(program)
32.4 ms	2.66%	350.8 ms	28.82%	initialize
22.0 ms	1.81%	174.9 ms	14.37%	cookieParser
21.9 ms	1.80%	102.5 ms	8.42%	session
19.3 ms	1.58%	48.2 ms	3.96%	hash
19.0 ms	1.56%	2791.8 ms	229.34%	next
18.4 ms	1.51%	3013.1 ms	247.52%	handle
17.2 ms	1.42%	72.1 ms	5.92%	compression
16.2 ms	1.33%	16.2 ms	1.33%	Hash
14.5 ms	1.19%	25.7 ms	2.11%	writeHead
14.1 ms	1.15%	30.3 ms	2.49%	Hash
13.3 ms	1.09%	20.7 ms	1.70%	Store.createSession
12.8 ms	1.05%	12.8 ms	1.05%	getSession
12.7 ms	1.04%	20.9 ms	1.71%	nextTick
12.7 ms	1.04%	464.4 ms	38.15%	send
12.6 ms	1.03%	21.2 ms	1.74%	set
11.7 ms	0.96%	11.7 ms	0.96%	digest
9.9 ms	0.81%	191.3 ms	15.72%	expressInit
9.4 ms	0.77%	497.6 ms	40.88%	handle
9.0 ms	0.74%	681.7 ms	56.00%	callbackTrampoline
8.6 ms	0.71%	179.4 ms	14.74%	writevGeneric
8.2 ms	0.67%	32.5 ms	2.67%	write_
7.9 ms	0.65%	8.0 ms	0.66%	onHeaders
7.9 ms	0.65%	13.1 ms	1.08%	setHeader
7.8 ms	0.64%	7.8 ms	0.64%	stringify
7.7 ms	0.63%	7.7 ms	0.63%	memoryUsage
7.3 ms	0.60%	7.3 ms	0.60%	stat
7.3 ms	0.60%	312.9 ms	25.70%	authenticate
7.1 ms	0.59%	166.2 ms	13.65%	processTicksAndRejections
7.1 ms	0.59%	117.7 ms	9.67%	writeHead
6.8 ms	0.56%	11.1 ms	0.91%	_storeHeader
6.6 ms	0.54%	13.6 ms	1.12%	update
6.3 ms	0.52%	34.9 ms	2.87%	pipe
6.1 ms	0.50%	355.3 ms	28.94%	init

Artillery en Info.router.js

Bloqueante

```
info.router.js x
1  const { Router } = require("express");
2  const routerInfo = Router();
3  const CPUQty = require('os').cpus().length;
4  const logger = require('../utils/logger')
5
6  //Ruta utilizada para probar no bloqueante
7  // routerInfo.get('/', (req, res) => {
8  //   logger.info(`Ruta: ${req.originalUrl}, Método: ${req.method}`)
9  //   res.status(200).json({
10 //     argv: process.argv.slice(2),
11 //     SO: process.platform,
12 //     version: process.version,
13 //     memory: process.memoryUsage(),
14 //     execPath: process.execPath,
15 //     proyectPath: process.cwd(),
16 //     processID: process.pid,
17 //     CPUQty: CPUQty
18 //   })
19 // })
20
21 routerInfo.get('/', (req, res) => {
22   // logger.info(`Ruta: ${req.originalUrl}, Método: ${req.method}`)
23   const info = {
24     argv: process.argv.slice(2),
25     SO: process.platform,
26     version: process.version,
27     memory: process.memoryUsage(),
28     execPath: process.execPath,
29     proyectPath: process.cwd(),
30     processID: process.pid,
31     CPUQty: CPUQty
32   }
33   //Utilizado para probar bloqueante
34   console.log(
35     `argv: ${info.argv}` + "\n",
36     `SO: ${info.SO}` + "\n",
37     `version: ${info.version}` + "\n",
38     `memory: ${info.memory}` + "\n",
39     `execPath: ${info.execPath}` + "\n",
40     `proyectPath: ${info.proyectPath}` + "\n",
41     `processID: ${info.processID}` + "\n",
42     `CPUQty: ${info.CPUQty}` + "\n",
43   );
44   res.status(200).send(info)
45 })
46
47 module.exports = { routerInfo, CPUQty };
```

NoBloqueante

```
info.router.js x
1  const { Router } = require("express");
2  const routerInfo = Router();
3  const CPUQty = require('os').cpus().length;
4  const logger = require('../utils/logger')
5
6  //Ruta utilizada para probar no bloqueante
7  // routerInfo.get('/', (req, res) => {
8  //   logger.info(`Ruta: ${req.originalUrl}, Método: ${req.method}`)
9  //   res.status(200).json({
10 //     argv: process.argv.slice(2),
11 //     SO: process.platform,
12 //     version: process.version,
13 //     memory: process.memoryUsage(),
14 //     execPath: process.execPath,
15 //     proyectPath: process.cwd(),
16 //     processID: process.pid,
17 //     CPUQty: CPUQty
18 //   })
19 // })
20
21 routerInfo.get('/', (req, res) => {
22   // logger.info(`Ruta: ${req.originalUrl}, Método: ${req.method}`)
23   const info = {
24     argv: process.argv.slice(2),
25     SO: process.platform,
26     version: process.version,
27     memory: process.memoryUsage(),
28     execPath: process.execPath,
29     proyectPath: process.cwd(),
30     processID: process.pid,
31     CPUQty: CPUQty
32   }
33   //Utilizado para probar bloqueante
34   // console.log(
35   //   `argv: ${info.argv}` + "\n",
36   //   `SO: ${info.SO}` + "\n",
37   //   `version: ${info.version}` + "\n",
38   //   `memory: ${info.memory}` + "\n",
39   //   `execPath: ${info.execPath}` + "\n",
40   //   `proyectPath: ${info.proyectPath}` + "\n",
41   //   `processID: ${info.processID}` + "\n",
42   //   `CPUQty: ${info.CPUQty}` + "\n",
43   // );
44   res.status(200).send(info)
45 })
46
47 module.exports = { routerInfo, CPUQty };
```

La línea 44, toma aproximadamente 3,3 veces mas tiempo en el proceso bloqueante que en el no bloqueante.

Autocannon – Bloqueante

Or de perfiles				Fuentes	Memoria
Pesado (de abajo hacia arriba) ▼					
Tiempo individual ▼		Tiempo total		Función	
13027.5 ms		13027.5 ms		(idle)	
4557.7 ms	27.04%	9522.5 ms	56.49%	▼ consoleCall	
4557.7 ms	27.04%	9522.5 ms	56.49%	▸ (anónimas)	
4437.3 ms	26.32%	4437.3 ms	26.32%	▸ writeUtf8String	
821.2 ms	4.87%	821.2 ms	4.87%	▸ writev	
359.2 ms	2.13%	359.2 ms	2.13%	(garbage collector)	
338.0 ms	2.01%	12803.2 ms	75.95%	▸ initialize	
215.7 ms	1.28%	215.7 ms	1.28%	(program)	
149.5 ms	0.89%	149.5 ms	0.89%	▸ getColorDepth	
142.0 ms	0.84%	269.9 ms	1.60%	▸ nextTick	
129.6 ms	0.77%	292.7 ms	1.74%	▸ hash	
127.9 ms	0.76%	143.5 ms	0.85%	▸ parse	
127.0 ms	0.75%	182.3 ms	1.08%	▸ set	
115.6 ms	0.69%	12176.3 ms	72.23%	▸ (anónimas)	
115.3 ms	0.68%	4523.2 ms	26.83%	▸ send	
110.7 ms	0.66%	163.1 ms	0.97%	▸ writeHead	
109.0 ms	0.65%	405.9 ms	2.41%	▸ compression	
105.3 ms	0.62%	105.3 ms	0.62%	▸ asyncTaskScheduled	
103.9 ms	0.62%	93151.4 ms	552.60%	▸ handle	
100.6 ms	0.60%	100.6 ms	0.60%	▸ Hash	
94.8 ms	0.56%	81175.9 ms	481.56%	▸ next	
89.5 ms	0.53%	1050.8 ms	6.23%	▸ session	
86.2 ms	0.51%	86.2 ms	0.51%	▸ stringify	
80.1 ms	0.48%	80.1 ms	0.48%	▸ status	
73.6 ms	0.44%	174.6 ms	1.04%	▸ Hash	
69.3 ms	0.41%	1153.0 ms	6.84%	▸ cookieParser	
68.9 ms	0.41%	70.0 ms	0.42%	▸ serialize	
65.0 ms	0.39%	65.0 ms	0.39%	▸ digest	
64.0 ms	0.38%	303.0 ms	1.80%	▸ store.generate	
63.3 ms	0.38%	1237.3 ms	7.34%	▸ expressInit	
60.8 ms	0.36%	73.1 ms	0.43%	▸ set maxAge	
57.2 ms	0.34%	1457.5 ms	8.65%	▸ writeHead	
56.2 ms	0.33%	124.0 ms	0.74%	▸ header	
53.7 ms	0.32%	979.3 ms	5.81%	▸ writevGeneric	
53.4 ms	0.32%	58.1 ms	0.34%	▸ checkInvalidHeaderChar	
51.4 ms	0.30%	164.8 ms	0.98%	▸ write_	
50.3 ms	0.29%	50.3 ms	0.29%	▸ init	

Al igual que con Artillery, se observa que la función menos performante es consoleCall.

Autocannon – NoBloqueante

Pesado (de abajo hacia arriba) ▾ 🔍 ✕ ↺					
Tiempo individual ▾		Tiempo total		Función	
6233.1 ms		6233.1 ms		(idle)	
2579.9 ms	13.29%	2579.9 ms	13.29%	▸ writev	
834.8 ms	4.30%	7117.3 ms	36.66%	▸ initialize	initialize.js:51
784.9 ms	4.04%	784.9 ms	4.04%	(program)	
690.7 ms	3.56%	690.7 ms	3.56%	(garbage collector)	
424.4 ms	2.19%	683.1 ms	3.52%	▸ nextTick	node:internal/p...ask_queues:104
416.5 ms	2.15%	1302.0 ms	6.71%	▸ compression	index.js:59
388.0 ms	2.00%	3364.9 ms	17.33%	▸ session	index.js:179
371.9 ms	1.92%	481.7 ms	2.48%	▸ set	memory.js:119
331.3 ms	1.71%	3816.9 ms	19.66%	▸ cookieParser	index.js:44
314.7 ms	1.62%	58180.3 ms	299.64%	▸ next	index.js:177
304.1 ms	1.57%	740.5 ms	3.81%	▸ hash	index.js:596
303.4 ms	1.56%	469.9 ms	2.42%	▸ writeHead	node: http_server:269
289.7 ms	1.49%	321.7 ms	1.66%	▸ parse	index.js:106
255.1 ms	1.31%	255.1 ms	1.31%	▸ Hash	
254.3 ms	1.31%	820.2 ms	4.22%	▸ store.generate	index.js:158
233.2 ms	1.20%	4110.9 ms	21.17%	▸ expressInit	init.js:29
232.0 ms	1.19%	232.0 ms	1.19%	▸ asyncTaskScheduled	
231.4 ms	1.19%	231.4 ms	1.19%	▸ stringify	response.js:1145
224.6 ms	1.16%	9859.0 ms	50.78%	▸ send	response.js:111
216.7 ms	1.12%	598.8 ms	3.08%	▸ write_	node: http_outgoing:730
211.5 ms	1.09%	7511.1 ms	38.68%	▸ jsonParser	json.js:98
197.9 ms	1.02%	197.9 ms	1.02%	▸ digest	
190.6 ms	0.98%	63174.8 ms	325.36%	▸ handle	layer.js:86
170.2 ms	0.88%	170.2 ms	0.88%	▸ stat	
169.1 ms	0.87%	414.4 ms	2.13%	▸ header	response.js:777
159.5 ms	0.82%	2998.9 ms	15.44%	▸ writevGeneric	node:internal/s...se_commons:126
158.2 ms	0.81%	289.5 ms	1.49%	▸ _send	node: http_outgoing:319
154.1 ms	0.79%	174.9 ms	0.90%	▸ checkInvalidHeaderChar	node: http_common:232
152.6 ms	0.79%	370.5 ms	1.91%	▸ setHeader	node: http_outgoing:574
148.4 ms	0.76%	149.6 ms	0.77%	▸ onHeaders	index.js:56
143.7 ms	0.74%	166.4 ms	0.86%	▸ _storeHeader	node: http_outgoing:374
141.5 ms	0.73%	145.5 ms	0.75%	▸ serialize	index.js:101
139.8 ms	0.72%	142.4 ms	0.73%	▸ parseurl	index.js:35
136.9 ms	0.71%	136.9 ms	0.71%	▸ getHeader	node: http_outgoing:590
136.5 ms	0.70%	5272.1 ms	27.15%	▸ json	response.js:250
122.6 ms	0.68%	1560.4 ms	8.55%	▸ send	node: http_outgoing:822

Autocannon en Info.router.js

Bloqueante

Fuentes	Memoria
info.router.js X	
1	const { Router } = require("express");
2	const routerInfo = Router();
3	const CPUQty = require('os').cpus().length;
4	const logger = require('../utils/logger')
5	
6	//Ruta utilizada para probar no bloqueante
7	// routerInfo.get('/', (req, res) => {
8	// logger.info(`Ruta: \${req.originalUrl}, Método: \${req.method}`)
9	// res.status(200).json({
10	// argv: process.argv.slice(2),
11	// SO: process.platform,
12	// version: process.version,
13	// memory: process.memoryUsage(),
14	// execPath: process.execPath,
15	// proyectPath: process.cwd(),
16	// processID: process.pid,
17	// CPUQty: CPUQty
18	// })
19	// })
20	
21	0.6 ms routerInfo.get('/', (req, res) => {
22	// logger.info(`Ruta: \${req.originalUrl}, Método: \${req.method}`)
23	0.3 ms const info = {
24	6.4 ms argv: process.argv.slice(2),
25	0.5 ms SO: process.platform,
26	0.2 ms version: process.version,
27	1.4 ms memory: process.memoryUsage(),
28	0.9 ms execPath: process.execPath,
29	1.2 ms proyectPath: process.cwd(),
30	0.8 ms processID: process.pid,
31	CPUQty: CPUQty
32	}
33	//Utilizado para probar bloqueante
34	18.7 ms console.log(
35	10.9 ms `argv: \${info.argv}` + "\n",
36	0.2 ms `SO: \${info.SO}` + "\n",
37	`version: \${info.version}` + "\n",
38	3.0 ms `memory: \${info.memory}` + "\n",
39	0.2 ms `execPath: \${info.execPath}` + "\n",
40	0.2 ms `proyectPath: \${info.proyectPath}` + "\n",
41	3.1 ms `processID: \${info.processID}` + "\n",
42	0.2 ms `CPUQty: \${info.CPUQty}` + "\n",
43);
44	66.0 ms res.status(200).send(info)
45	0.2 ms })
46	
47	module.exports = { routerInfo, CPUQty };

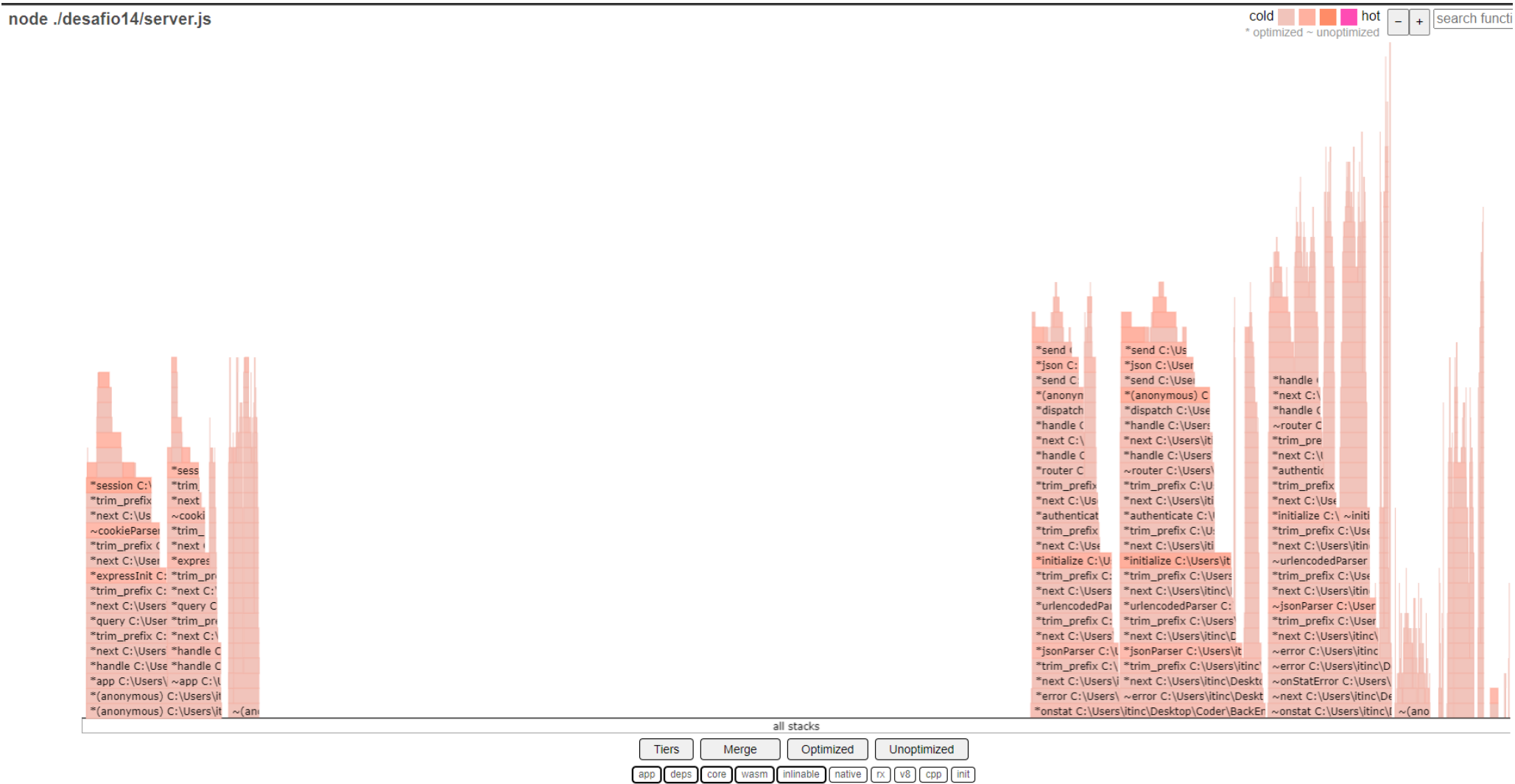
NoBloqueante

info.router.js X	
1	const { Router } = require("express");
2	const routerInfo = Router();
3	const CPUQty = require('os').cpus().length;
4	const logger = require('../utils/logger')
5	
6	//Ruta utilizada para probar no bloqueante
7	// routerInfo.get('/', (req, res) => {
8	// logger.info(`Ruta: \${req.originalUrl}, Método: \${req.method}`)
9	// res.status(200).json({
10	// argv: process.argv.slice(2),
11	// SO: process.platform,
12	// version: process.version,
13	// memory: process.memoryUsage(),
14	// execPath: process.execPath,
15	// proyectPath: process.cwd(),
16	// processID: process.pid,
17	// CPUQty: CPUQty
18	// })
19	// })
20	
21	1.6 ms routerInfo.get('/', (req, res) => {
22	// logger.info(`Ruta: \${req.originalUrl}, Método: \${req.method}`)
23	1.1 ms const info = {
24	10.1 ms argv: process.argv.slice(2),
25	0.8 ms SO: process.platform,
26	0.7 ms version: process.version,
27	4.7 ms memory: process.memoryUsage(),
28	1.6 ms execPath: process.execPath,
29	0.6 ms proyectPath: process.cwd(),
30	0.4 ms processID: process.pid,
31	CPUQty: CPUQty
32	}
33	//Utilizado para probar bloqueante
34	// console.log(
35	// `argv: \${info.argv}` + "\n",
36	// `SO: \${info.SO}` + "\n",
37	// `version: \${info.version}` + "\n",
38	// `memory: \${info.memory}` + "\n",
39	// `execPath: \${info.execPath}` + "\n",
40	// `proyectPath: \${info.proyectPath}` + "\n",
41	// `processID: \${info.processID}` + "\n",
42	// `CPUQty: \${info.CPUQty}` + "\n",
43	//);
44	58.5 ms res.status(200).send(info)
45	1.7 ms })
46	
47	module.exports = { routerInfo, CPUQty };

Al igual como sucedió utilizando --prof el test con Autocannon arroja una diferencia en los procesos significativamente menor.

3) Diagrama de Flama con 0x con carga emulada por Autocannon

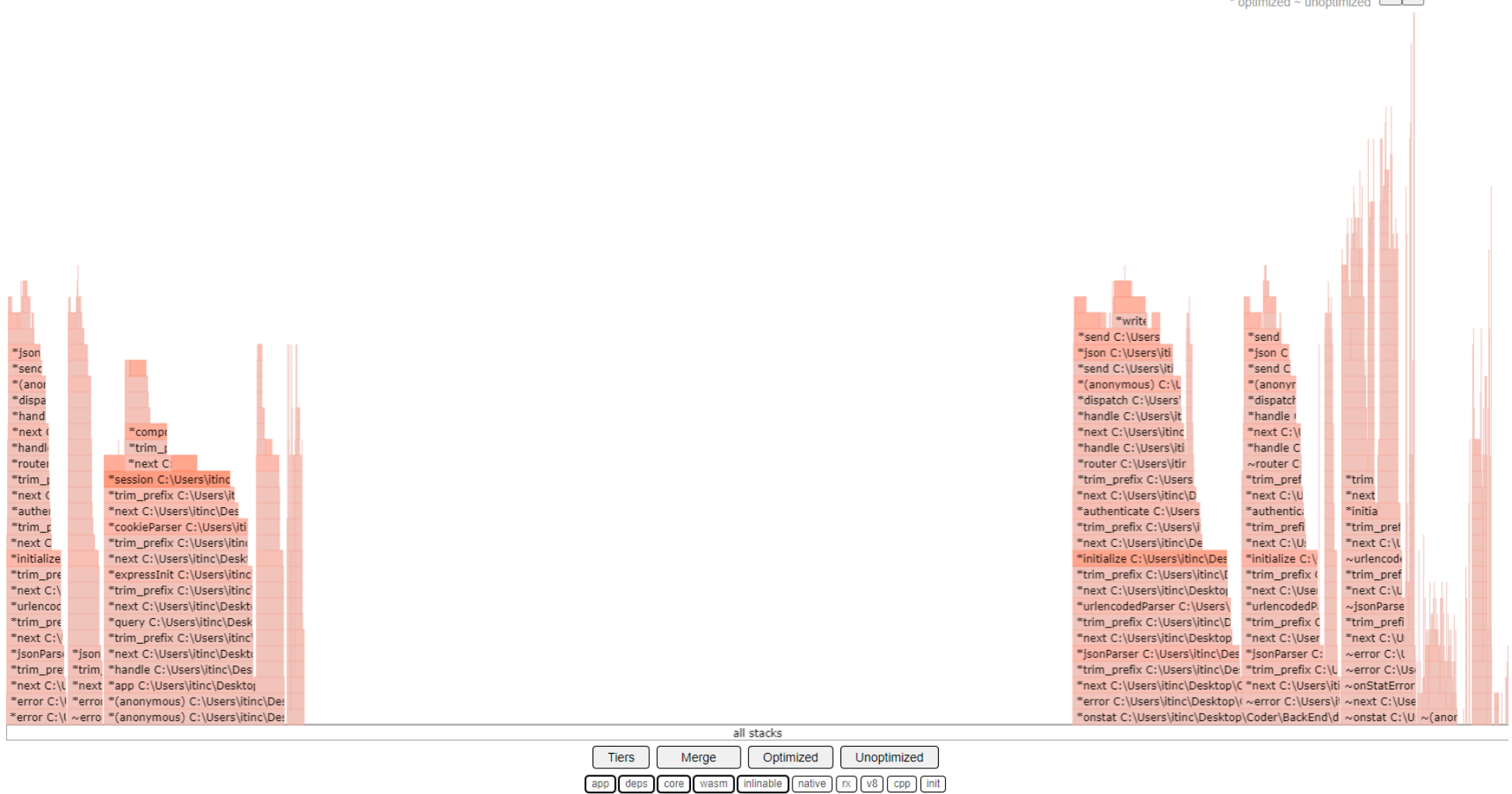
Bloqueante



No Bloqueante

node ./desafio14/server.js

cold hot - + search functi
* optimized ~ unoptimized



Conclusión:

De los test realizados se observó que a grandes escalas, emplear procesos **No Bloqueantes** puede mejorar considerablemente la respuesta del servidor ante múltiples conexiones y solicitudes. Evitando en muchos casos entrar en un estado de “suspensión” hasta finalizar una tarea específica.