1) Perfilamiento del servidor con –prof

Artillery – Bloqueante

artillery_prof-bloq.txt U

desafio14 > tests > --prof > artillery > artillery_prof-bloq.txt

```
Statistical profiling result from artillery_bloq-v8.log, (2891 ticks, 0 unaccounted, 0 excluded).

 [Shared libraries]:
   ticks  total  nonlib   name
    2620   90.6%          C:\WINDOWS\SYSTEM32\ntdll.dll
     265    9.2%          D:\Archivos de Programa\nodejs\node.exe

 [JavaScript]:
   ticks  total  nonlib   name
       1    0.0%   16.7%  LazyCompile: *toNamespacedPath node:path:618:19
       1    0.0%   16.7%  LazyCompile: *slowCases node:internal/util:165:19
       1    0.0%   16.7%  Function: ^validateEncoding node:internal/validators:198:26
       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
       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
       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

 [C++]:
   ticks  total  nonlib   name

 [Summary]:
   ticks  total  nonlib   name
       6    0.2%  100.0%  JavaScript
       0    0.0%    0.0%  C++
      19    0.7%  316.7%  GC
    2885   99.8%          Shared libraries

 [C++ entry points]:
   ticks    cpp   total   name
```

Artillery – NoBloqueante

artillery_prof-nobloq.txt U

desafio14 > tests > --prof > artillery > artillery_prof-nobloq.txt

```
Statistical profiling result from artillery_nobloq-v8.log, (1803 ticks, 0 unaccounted, 0 excluded).

 [Shared libraries]:
   ticks  total  nonlib   name
    1646   91.3%          C:\WINDOWS\SYSTEM32\ntdll.dll
     155    8.6%          D:\Archivos de Programa\nodejs\node.exe

 [JavaScript]:
   ticks  total  nonlib   name
       1    0.1%   50.0%  Function: ^resOnFinish node:_http_server:788:21
       1    0.1%   50.0%  Function: ^onstat C:\Users\itinc\Desktop\Coder\BackEnd\desafios-backend-32070-coderhouse\node_modules\send\index.js:717:33

 [C++]:
   ticks  total  nonlib   name

 [Summary]:
   ticks  total  nonlib   name
       2    0.1%  100.0%  JavaScript
       0    0.0%    0.0%  C++
       9    0.5%  450.0%  GC
    1801   99.9%          Shared libraries

 [C++ entry points]:
   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

desafio14 > tests > --prof > artillery > result_artillery_bloq.txt

```
Running scenarios...
Phase started: unnamed (index: 0, duration: 1s) 23:30:55(-0300)

Phase completed: unnamed (index: 0, duration: 1s) 23:30:56(-0300)

All VUs finished. Total time: 5 seconds

--------------------------------
Summary report @ 23:30:59(-0300)
--------------------------------

http.codes.200: ..................................................................... 1000
http.request_rate: .................................................................. 178/se
http.requests: ...................................................................... 1000
http.response_time:
  min: .............................................................................. 2
  max: .............................................................................. 124
  median: ........................................................................... 49.9
  p95: .............................................................................. 96.6
  p99: .............................................................................. 117.9
http.responses: ..................................................................... 1000
vusers.completed: ................................................................... 50
vusers.created: ..................................................................... 50
vusers.created_by_name.0: ........................................................... 50
vusers.failed: ...................................................................... 0
vusers.session_length:
  min: .............................................................................. 761.9
  max: .............................................................................. 1674.6
  median: ........................................................................... 1436.8
  p95: .............................................................................. 1587.9
  p99: .............................................................................. 1620
```

result_artillery_nobloq.txt U

desafio14 > tests > --prof > artillery > result_artillery_nobloq.txt

```
Running scenarios...
Phase started: unnamed (index: 0, duration: 1s) 23:33:52(-0300)

Phase completed: unnamed (index: 0, duration: 1s) 23:33:53(-0300)

All VUs finished. Total time: 5 seconds

--------------------------------
Summary report @ 23:33:56(-0300)
--------------------------------

http.codes.200: ..................................................................... 1000
http.request_rate: .................................................................. 457/se
http.requests: ...................................................................... 1000
http.response_time:
  min: .............................................................................. 1
  max: .............................................................................. 42
  median: ........................................................................... 10.1
  p95: .............................................................................. 19.1
  p99: .............................................................................. 23.8
http.responses: ..................................................................... 1000
vusers.completed: ................................................................... 50
vusers.created: ..................................................................... 50
vusers.created_by_name.0: ........................................................... 50
vusers.failed: ...................................................................... 0
vusers.session_length:
  min: .............................................................................. 127.9
  max: .............................................................................. 338.8
  median: ........................................................................... 295.9
  p95: .............................................................................. 333.7
  p99: .............................................................................. 340.4
```

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

Autocannon – Bloqueante

autocannon_prof-bloq.txt U

desafio14 > tests > --prof > autocannon > autocannon_prof-bloq.txt

```
[C++]:
  ticks  total  nonlib   name

[Summary]:
  ticks  total  nonlib   name
     60   3.0%   98.4%  JavaScript
      0   0.0%    0.0%  C++
     39   2.0%   63.9%  GC
   1922  96.9%          Shared libraries
      1   0.1%          Unaccounted

[C++ entry points]:
  ticks    cpp   total   name

[Bottom up (heavy) profile]:
 Note: percentage shows a share of a particular caller in the total
 amount of its parent calls.
 Callers occupying less than 1.0% are not shown.
```

Autocannon - NoBloqueante

autocannon_prof-nobloq.txt U

desafio14 > tests > --prof > autocannon > autocannon_prof-nobloq.txt

```
[C++]:
  ticks  total  nonlib   name

[Summary]:
  ticks  total  nonlib   name
    116   6.5%   99.1%  JavaScript
      0   0.0%    0.0%  C++
    102   5.8%   87.2%  GC
   1655  93.4%          Shared libraries
      1   0.1%          Unaccounted

[C++ entry points]:
  ticks    cpp   total   name

[Bottom up (heavy) profile]:
 Note: percentage shows a share of a particular caller in the total
 amount of its parent calls.
 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.

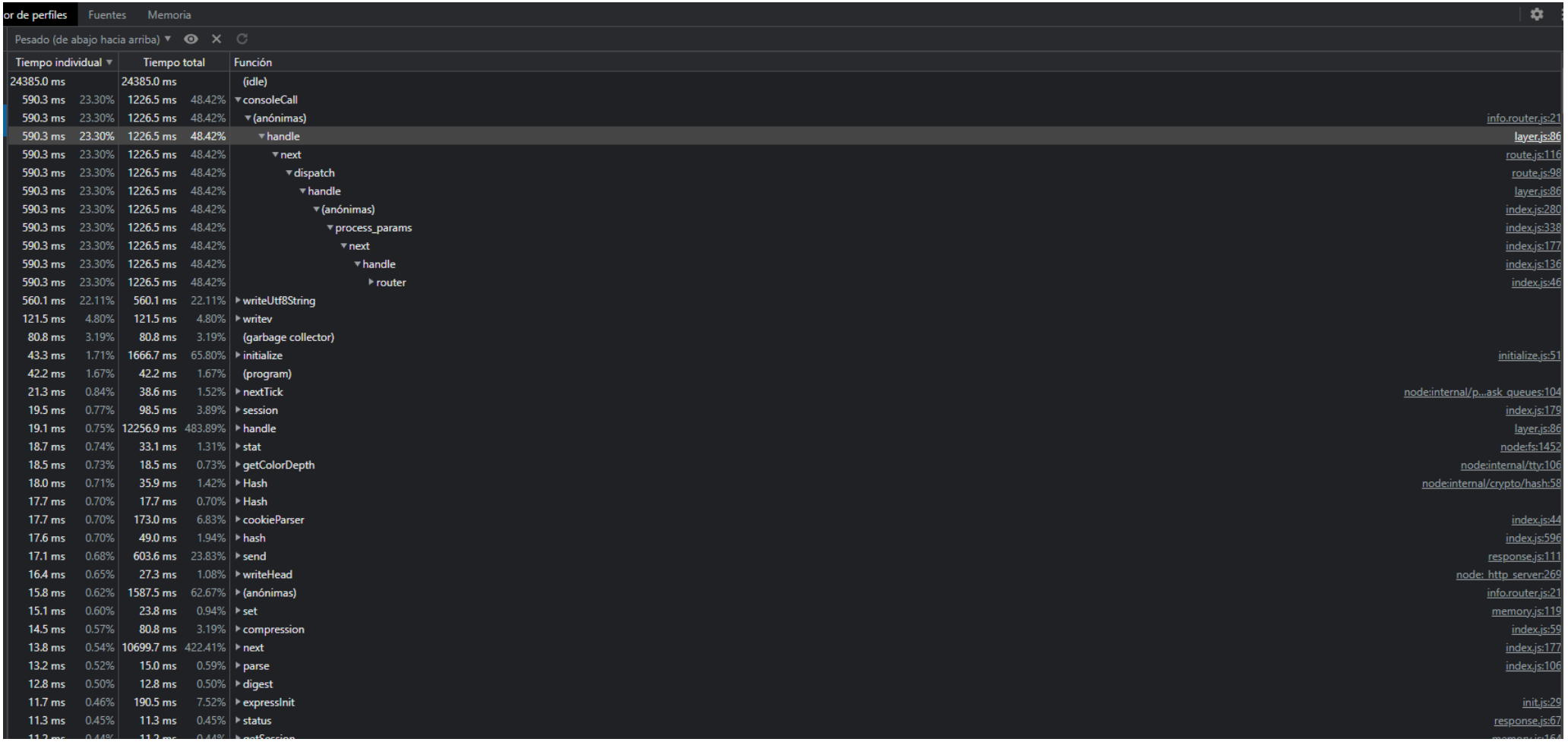
Comparación log de Test de Autocannon: Bloqueante | NoBloqueante: Se observa que la cantidad de requests en 20 segundos en el proceso no bloqueante es 4 veces mayor.

```
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>
```

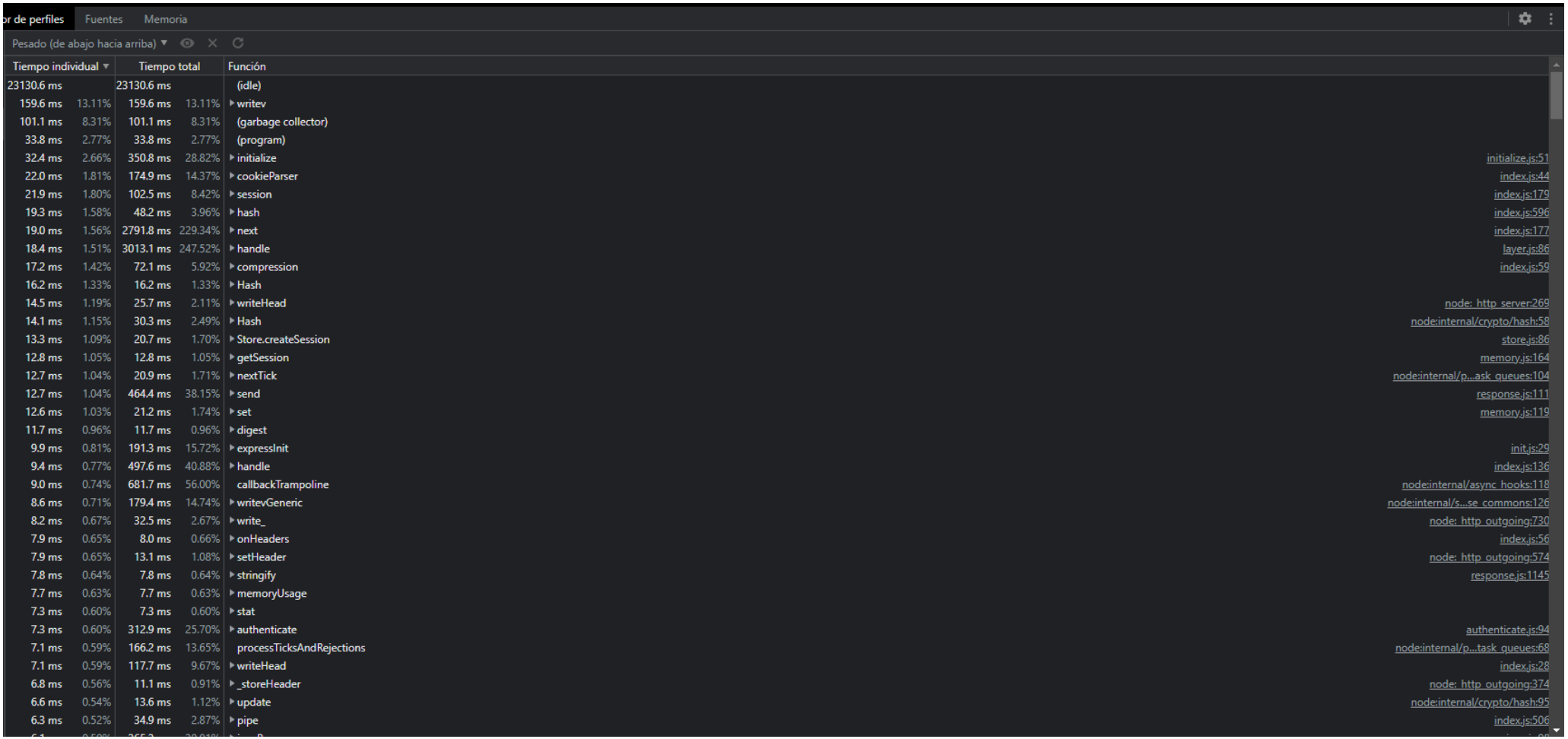
2) Perfilamiento del servidor con –inspect

Artillery – Bloqueante



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

Artillery – NoBloqueante



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	initialize.js:51
22.0 ms	1.81%	174.9 ms	14.37%	cookieParser	index.js:44
21.9 ms	1.80%	102.5 ms	8.42%	session	index.js:179
19.3 ms	1.58%	48.2 ms	3.96%	hash	index.js:596
19.0 ms	1.56%	2791.8 ms	229.34%	next	index.js:177
18.4 ms	1.51%	3013.1 ms	247.52%	handle	layer.js:86
17.2 ms	1.42%	72.1 ms	5.92%	compression	index.js:59
16.2 ms	1.33%	16.2 ms	1.33%	Hash	
14.5 ms	1.19%	25.7 ms	2.11%	writeHead	node: http_server:269
14.1 ms	1.15%	30.3 ms	2.49%	Hash	node:internal/crypto/hash:58
13.3 ms	1.09%	20.7 ms	1.70%	Store.createSession	store.js:86
12.8 ms	1.05%	12.8 ms	1.05%	getSession	memory.js:164
12.7 ms	1.04%	20.9 ms	1.71%	nextTick	node:internal/p...ask_queues:104
12.7 ms	1.04%	464.4 ms	38.15%	send	response.js:111
12.6 ms	1.03%	21.2 ms	1.74%	set	memory.js:119
11.7 ms	0.96%	11.7 ms	0.96%	digest	
9.9 ms	0.81%	191.3 ms	15.72%	expressInit	init.js:29
9.4 ms	0.77%	497.6 ms	40.88%	handle	index.js:136
9.0 ms	0.74%	681.7 ms	56.00%	callbackTrampoline	node:internal/async_hooks:118
8.6 ms	0.71%	179.4 ms	14.74%	writevGeneric	node:internal/s...se_commons:126
8.2 ms	0.67%	32.5 ms	2.67%	write_	node: http_outgoing:730
7.9 ms	0.65%	8.0 ms	0.66%	onHeaders	index.js:56
7.9 ms	0.65%	13.1 ms	1.08%	setHeader	node: http_outgoing:574
7.8 ms	0.64%	7.8 ms	0.64%	stringify	response.js:1145
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	authenticate.js:94
7.1 ms	0.59%	166.2 ms	13.65%	processTicksAndRejections	node:internal/p...task_queues:68
7.1 ms	0.59%	117.7 ms	9.67%	writeHead	index.js:28
6.8 ms	0.56%	11.1 ms	0.91%	_storeHeader	node: http_outgoing:374
6.6 ms	0.54%	13.6 ms	1.12%	update	node:internal/crypto/hash:95
6.3 ms	0.52%	34.9 ms	2.87%	pipe	index.js:506

Artillery en Info.router.js

Bloqueante

info.router.js

```
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  0.2 ms  const info = {
24  0.7 ms      argv: process.argv.slice(2),
25  0.1 ms      SO: process.platform,
26  0.1 ms      version: process.version,
27  0.5 ms      memory: process.memoryUsage(),
28  0.4 ms      execPath: process.execPath,
29              proyectPath:process.cwd(),
30              processID: process.pid,
31  0.1 ms      CPUqty: CPUqty
32      }
33      //Utilizado para probar bloqueante
34  2.5 ms  console.log(
35  1.5 ms      `argv: ${info.argv}` + "\n",
36  0.1 ms      `SO: ${info.SO}` + "\n",
37              `version: ${info.version}` + "\n",
38  0.5 ms      `memory: ${info.memory}` + "\n",
39              `execPath: ${info.execPath}` + "\n",
40              `proyectPath: ${info.proyectPath}` + "\n",
41  0.2 ms      `processID: ${info.processID}` + "\n",
42              `CPUqty: ${info.CPUqty}` + "\n",
43      );
44  8.7 ms  res.status(200).send(info)
45  })
46
47  module.exports = { routerInfo, CPUqty };
```

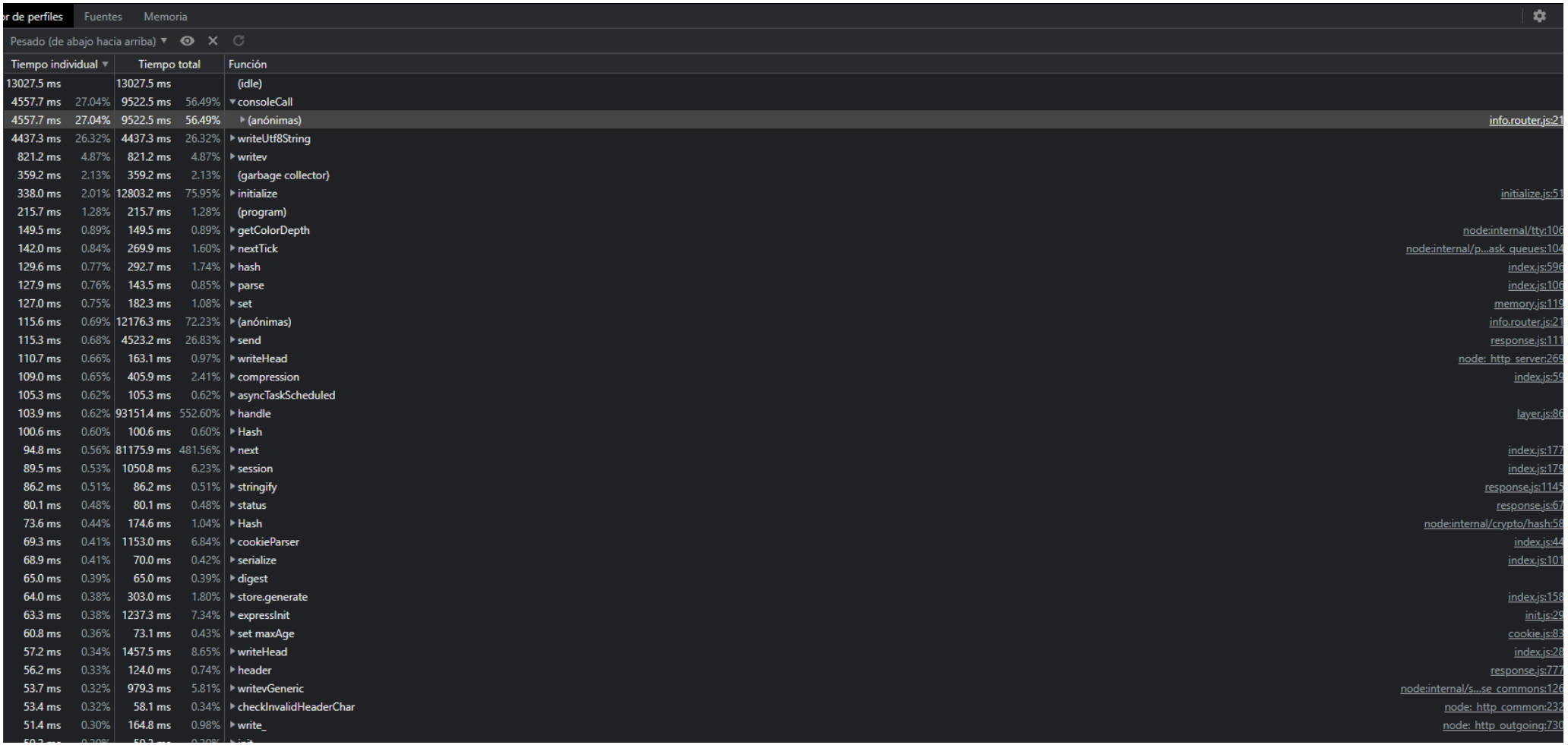
NoBloqueante

info.router.js

```
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  0.3 ms  const info = {
24  0.8 ms      argv: process.argv.slice(2),
25  0.1 ms      SO: process.platform,
26  0.2 ms      version: process.version,
27  0.7 ms      memory: process.memoryUsage(),
28  0.3 ms      execPath: process.execPath,
29  0.1 ms      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  2.6 ms  res.status(200).send(info)
45  })
46
47  module.exports = { routerInfo, CPUqty };
```

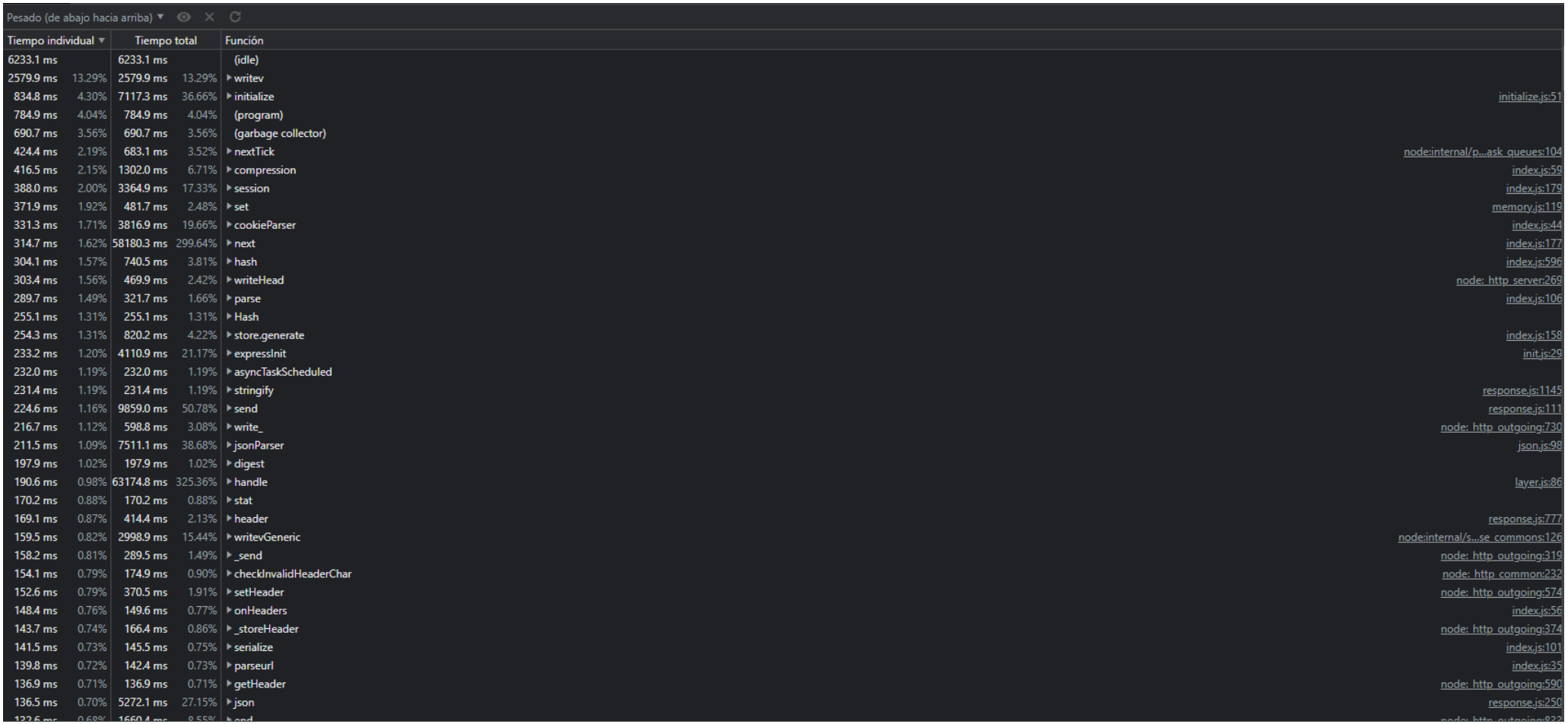
La línea 44, toma aproximadamente 3,3 veces mas tiempo en el proceso bloqueante que en el no bloqueante.

Autocannon – Bloqueante



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

Autocannon – NoBloqueante

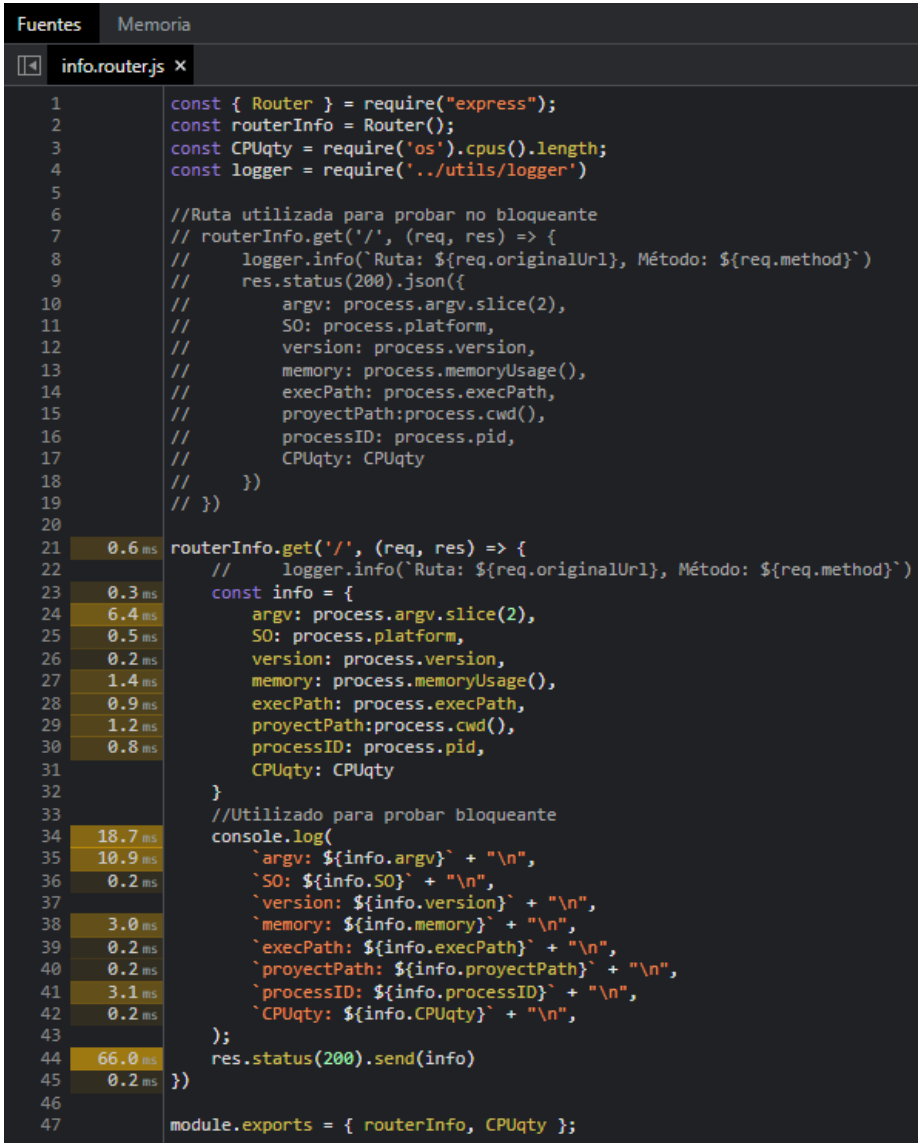


Pesado (de abajo hacia arriba)

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

Autocannon en Info.router.js

Bloqueante

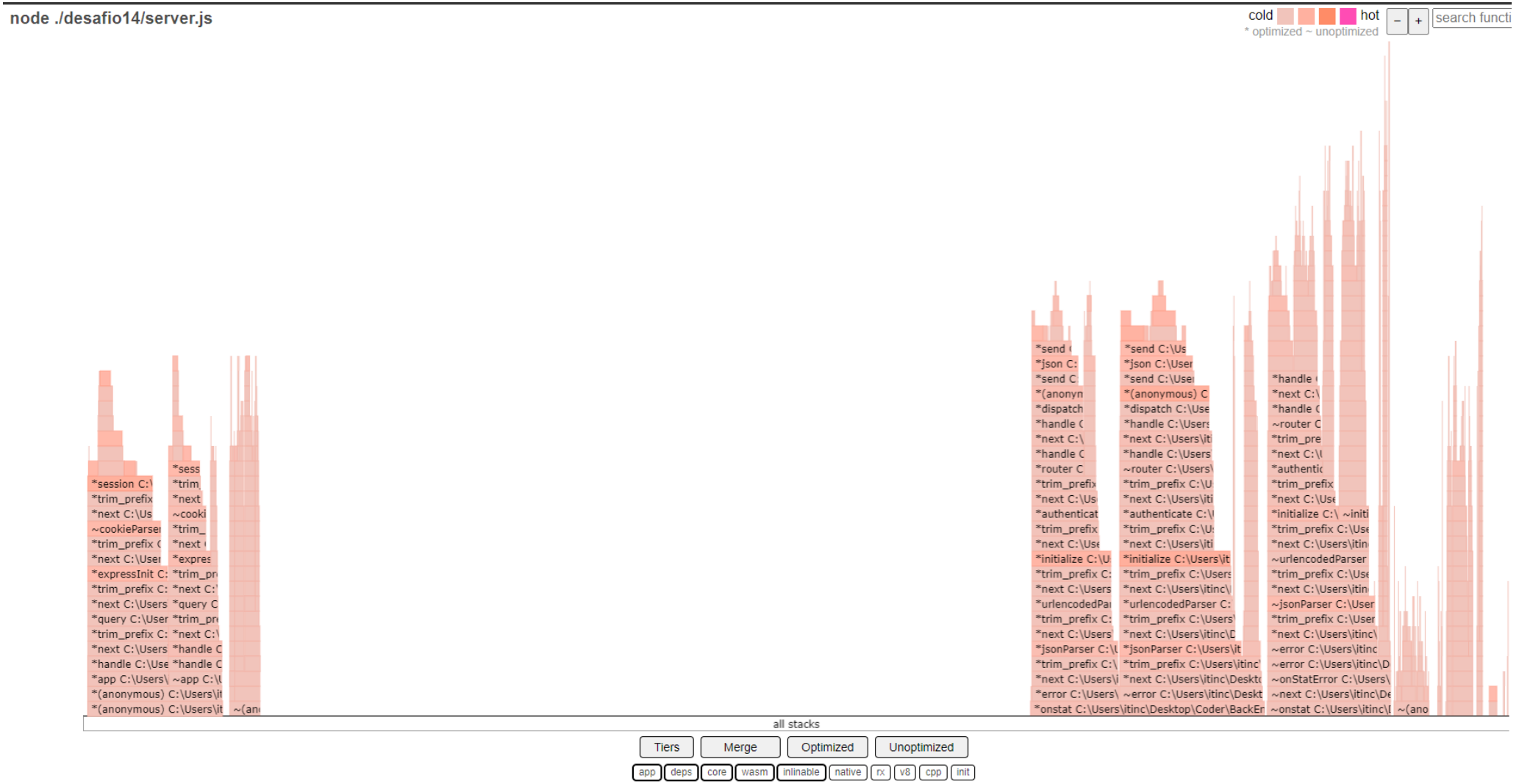


NoBloqueante

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

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.