



Entregable 14 – Loggers, gzip y análisis de performance

Informe de rendimiento

Comisión: 30945

Alumno: Gonzalo Leonel Gramajo

Tutor: Emiliano Pérez

Usando console.log

Test usando “ -- prof ”

Comando para iniciar node en modo profiler:

```
node --prof server.js
```

Comando para test con Artillery:

```
artillery quick --count 20 -n 50 "http://localhost:8080/info" >
result_bloq.txt
```

En el archivo result_console.txt se destaca la siguiente información:

```
All VUs finished. Total time: 4 minutes, 8 seconds
-----
Summary report @ 20:58:02 (-0300)
-----
errors.ETIMEDOUT:..... 10
http.codes.200:..... 655
http.request_rate:..... 2/sec
http.requests:..... 665
http.response_time:
  min:..... 1098
  max:..... 9938
  median:..... 4403.8
  p95:..... 7117
  p99:..... 8692.8
http.responses:..... 655
vusers.completed:..... 10
vusers.created:..... 20
```

```

vusers.created_by_name.0:..... 20
vusers.failed:..... 10
vusers.session_length:
  min:..... 224599.3
  max:..... 240882.8
  median:..... 235711.1
  p95:..... 240473
  p99:..... 240473

```

Análisis del test con “--prof - process”


Statistical profiling result from isolate-0000029994996230-47012-v8.log,
(25629 ticks, 0 unaccounted, 0 excluded).

```

[Summary]:
  ticks  total  nonlib   name
    0     0.0%    NaN%  JavaScript
    0     0.0%    NaN%   C++
    3     0.0% Infinity%   GC
25629 100.0%             Shared libraries

```

Test usando “--inspect”



```

121 // INFO (FORK O CLUSTER)
122 app.get('/info', (req, res) => {
123   let toSend = {nucleos: numCPUs, port: PORT}
124   if (SMODE == 'FORK'){
125     const child = fork('./src/utls/infoChild.js');
126     child.send(toSend)
127     child.on('message', (info) => {
128       res.render('pages/info',{body: info});
129     });
130   } else {
131     let info = routes.getInfo(toSend)
132     res.render('pages/info',{body: info});
133   }
134 })
135

```

Sin usar console.log

Test usando “--prof”

Comando para iniciar node en modo profiler:

```
node --prof server.js
```

Comando para test con Artillery:

```
artillery quick --count 20 -n 50 "http://localhost:8080/info" >
result_noConsole.txt
```

En el archivo result_noConsole.txt se destaca la siguiente información:

```

All VUs finished. Total time: 4 minutes, 48 seconds
-----
Summary report @ 00:33:45(-0300)
-----
errors.ETIMEDOUT:..... 9
http.codes.200:..... 835
http.request_rate:..... 2/sec
http.requests:..... 844
http.response_time:
  min:..... 1673
  max:..... 9978
  median:..... 5378.9
  p95:..... 7865.6
  p99:..... 9230.4
http.responses:..... 835

```

```

vusers.completed:..... 11
vusers.created:..... 20
vusers.created_by_name.0:..... 20
vusers.failed:..... 9
vusers.session_length:
  min:..... 264277.3
  max:..... 281293.8
  median:..... 276611.1
  p95:..... 282199.2
  p99:..... 282199.2

```

Análisis del test con “--prof - process”

Statistical profiling result from isolate-0000029519594140-39176-v8.log,
(16482 ticks, 0 unaccounted, 0 excluded).

[Summary]:

ticks	total	nonlib	name
0	0.0%	NaN%	JavaScript
0	0.0%	NaN%	C++
0	0.0%	NaN%	GC
16482	100.0%		Shared libraries

Test usando “--inspect”

```

121 // INFO (FORK 0 CLUSTER)
122 app.get('/info', (req, res) => {
123   0.1 ms let toSend = {nucleos: numCPUs, port: PORT}
124   0.3 ms if (SMODE == 'FORK'){
125   4.9 ms   const child = fork('./src/utls/infoChild.js');
126   1.0 ms   child.send(toSend)
127   0.6 ms   child.on('message', (info) => {
128   9.9 ms     res.render('pages/info',{body: info});
129   });
130   } else {
131     let info = routes.getInfo(toSend)
132     1.0 ms res.render('pages/info',{body: info});
133   }
134 })
135

```

```

135 // PRUEBA SIN FORK NI CLUSTER
136 app.get('/info', (req, res) => {
137   1.7 ms let toSend = {nucleos: numCPUs, port: PORT, path: req.path, method: req.method}
138   2.4 ms let info = routes.getInfo(toSend)
139   18.1 ms res.render('pages/info',{body: info});
140 })
141
142

```

```

41
42 function getInfo (data) {
43   1.1 ms myLogs.showInfo(data.path, data.method)
44   1.0 ms let info = {
45   0.4 ms   entryArg: data.port,
46   1.9 ms   execPath: process.execPath,
47   0.5 ms   os: process.platform,
48   0.4 ms   idProcess: process.pid,
49   0.4 ms   nodeVersion: process.version,
50   0.4 ms   directorio: process.cwd(),
51   2.3 ms   memory: process.memoryUsage().rss,
52   numCPUs: data.nucleos
53   };
54
55   return info
56 }
57

```

Test usando Autocannon y 0x

```
6  "scripts": {
7    "test": "node benchmark.js",
8    "start": "0x server.js"
9  },
```

Running all benchmarks in parallel...
Running 20s test @ http://localhost:8080/infoConsole
10 connections

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	32 ms	37 ms	61 ms	408 ms	47.07 ms	91 ms	1582 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	0	0	236	291	210	76.38	81
Bytes/Sec	0 B	0 B	501 kB	617 kB	445 kB	162 kB	172 kB

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

4k requests in 20.05s, 8.91 MB read

Running 20s test @ http://localhost:8080/infoNoConsole
10 connections

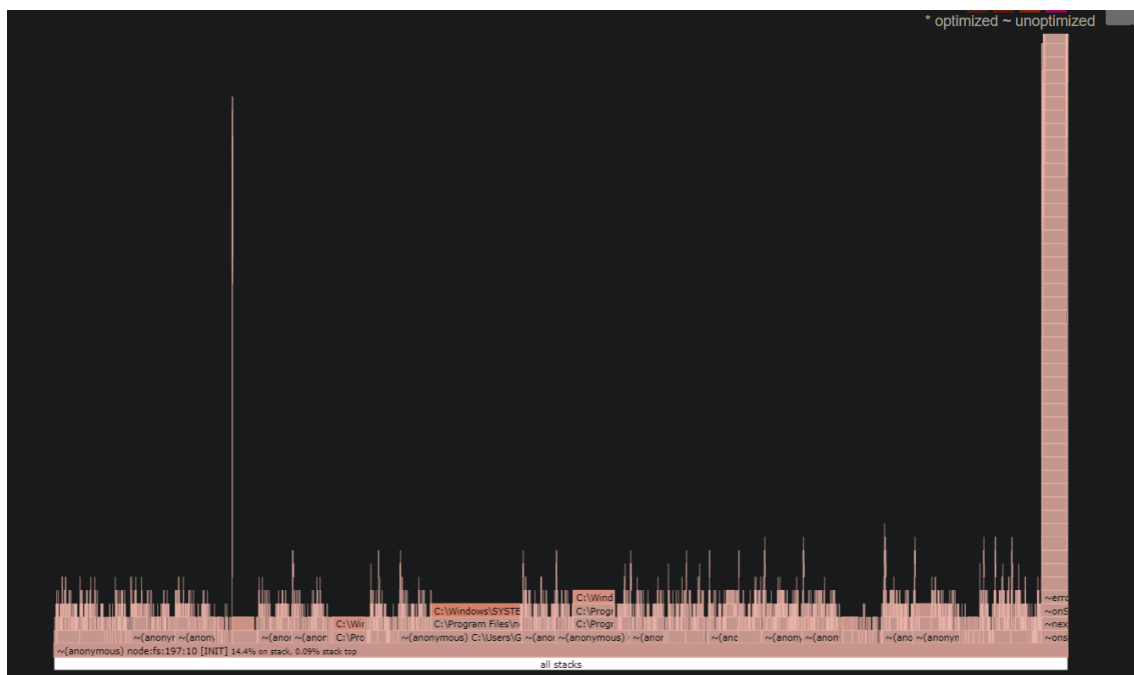
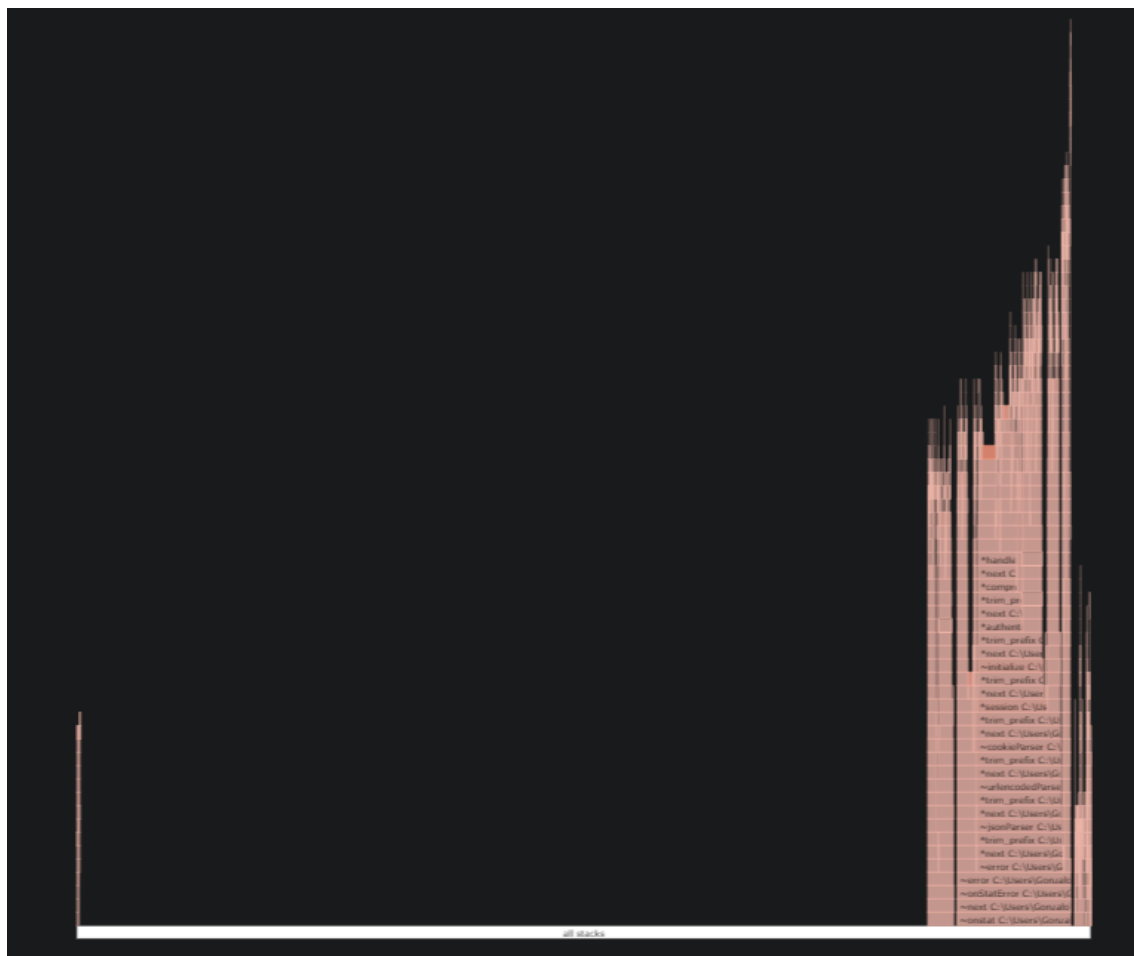
Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	32 ms	37 ms	59 ms	407 ms	47.13 ms	91.1 ms	1583 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	0	0	238	295	209.9	76.83	82
Bytes/Sec	0 B	0 B	505 kB	626 kB	445 kB	163 kB	174 kB

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

4k requests in 20.13s, 8.9 MB read

Grafico de llama obtenido:



Conclusión:

Los parámetros expuestos indican que el uso de `console.log` baja el rendimiento de la aplicación significativamente. Se puede ver esto claramente con los ticks usando `console.log` (25629 ticks) contra los ticks sin `console.log` (16482).

El test con Google DevTools en modo `--inspect` muestra que a pesar de que se elimina el `console.log`, el tiempo de ejecución incrementa, pero el método `res.render` disminuye en cuanto a su tiempo de ejecución. Mi hipótesis era de que al no usar `console.log`, iba a tardar mucho menos, pero fue al revés y estimo que es gracias al modo FORK. Para terminar de derogar esta hipótesis, ejecuté el programa sin FORK y evidentemente, el tiempo de `res.render` aumentó considerablemente.

En el test usando `0x`, se puede ver que sin `console.log`, los tiempos son menores por muy poco, esto se debe a que los req por segundo son mayores sin usar `console.log`, pero nuevamente, por una diferencia ínfima en este caso, siendo la mínima de 81 req/seg sin `console.log` y 82 req/seg, con `console.log` para los parámetros especificados (100 conexiones en 20 segundos). El gráfico de llama muestra una gran optimización.

Una complicación que encontré al efectuar los tests, es que el motor `v8` daba errores usando Autocannon, sin embargo al renderizar la página no mostraba ninguno, al igual que el servidor que se ejecutaba de satisfactoriamente. Al no poder encontrar solución a esto, elimine el fork, y recién se pudo efectuar el test. Algo parecido sucedió con al usar la opción `--inspect` y Google DevTools; la sección "run" no se encontraba y supuestamente, también es culpa de "v8" y FORK, por lo que procedí a buscar mi `server.js` sección por sección. Al ejecutar sin fork, pude encontrar mi archivo `route.js` que contenía la función de obtención de la "info".