

Profiling utilizando “node –prof” y artillery para realizar el test de carga.

(En el repositorio se encuentran los dos archivos correspondientes con la decodificación del .log completo)

```
Terminal Help
JS index.js package.json flamegraph.html result-prof-console.txt result-prof-no-console.txt

result-prof-console.txt
23 1 0.0% 0.2% LazyCompile: * storeHeader node:http_outgoing:373:
24 1 0.0% 0.2% LazyCompile: *Writable.write node:internal/streams/
25
26 [C++]:
27 ticks total nonlib name
28 2 0.1% 0.4% fwrite
29 2 0.1% 0.4% __write
30 1 0.0% 0.2% cfree
31 1 0.0% 0.2% _IO_file_sync
32
33 [Summary]:
34 ticks total nonlib name
35 23 1.1% 4.9% JavaScript
36 6 0.3% 1.3% C++
37 136 6.6% 28.9% GC
38 1602 77.3% Shared libraries
39 442 21.3% Unaccounted
40
41 [C++ entry points]:
42 ticks cpp total name
43 1 25.0% 0.0% fwrite
44 1 25.0% 0.0% cfree
45 1 25.0% 0.0% __write
46 1 25.0% 0.0% _IO_file_sync
47
48 [Bottom up (heavy) profile]:
49 Note: percentage shows a share of a particular caller in the total
50 amount of its parent calls.
51 Callers occupying less than 1.0% are not shown.
52
53 ticks parent name
54 1599 77.1% /home/nacchogonza/.nvm/versions/node/v15.11.0/bin/node
55 819 51.2% /home/nacchogonza/.nvm/versions/node/v15.11.0/bin/node
56 110 13.6% LazyCompile: <anonymous> file:///home/nacchogonza/Escr

result-prof-no-console.txt
15 [C++]:
16 ticks total nonlib name
17 3 0.2% 0.6% __write
18 2 0.1% 0.4% std::basic_ostream<char, std::char_traits<char> >5
19 2 0.1% 0.4% fwrite
20 1 0.1% 0.2% readlink
21 1 0.1% 0.2% pthread_sigmask
22 1 0.1% 0.2% _GI__pthread_mutex_unlock
23 1 0.1% 0.2% _GI__pthread_mutex_lock
24 1 0.1% 0.2% _IO_file_xsputn
25
26 [Summary]:
27 ticks total nonlib name
28 10 0.5% 1.8% JavaScript
29 12 0.6% 2.2% C++
30 144 7.3% 26.5% GC
31 1426 72.4% Shared libraries
32 521 26.5% Unaccounted
33
34 [C++ entry points]:
35 ticks cpp total name
36 1 100.0% 0.1% TOTAL
37
38 [Bottom up (heavy) profile]:
39 Note: percentage shows a share of a particular caller in the total
40 amount of its parent calls.
41 Callers occupying less than 1.0% are not shown.
42
43 ticks parent name
44 1425 72.4% /home/nacchogonza/.nvm/versions/node/v15.11.0/bin/node
45 676 47.4% /home/nacchogonza/.nvm/versions/node/v15.11.0/bin/node
46 76 11.2% LazyCompile: <anonymous> file:///home/nacchogonza/Escr
47 76 100.0% LazyCompile: ~handle /home/nacchogonza/Escritorio/bac
48 76 100.0% LazyCompile: ~next /home/nacchogonza/Escritorio/hac
```

En este caso podemos apreciar que el número de ticks es mayor en el caso de la ruta que cuenta con “console.log”, similar a lo que vimos en clase para el caso bloqueante vs. No bloqueante.

No me queda claro porque aparece una fila “Unaccounted” en este caso

Profiling con Autocannon + Inspect Google Chrome

- Resultado del test de carga con Autocannon

```
nacchogonza@nacchogonza-Lenovo-V330-14IKB:~/Escritorio/backend-mern-clase-32$ npm run autocannon
> entregable-22@1.0.0 autocannon
> node ./src/benchmark.js

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



| Stat    | 2.5%   | 50%    | 97.5%  | 99%    | Avg       | Stdev   | Max    |
|---------|--------|--------|--------|--------|-----------|---------|--------|
| Latency | 320 ms | 375 ms | 525 ms | 537 ms | 387.03 ms | 63.9 ms | 546 ms |



| Stat      | 1%     | 2.5%   | 50%    | 97.5%  | Avg    | Stdev | Min    |
|-----------|--------|--------|--------|--------|--------|-------|--------|
| Req/Sec   | 198    | 198    | 253    | 300    | 255    | 37.85 | 198    |
| Bytes/Sec | 136 kB | 136 kB | 174 kB | 206 kB | 175 kB | 26 kB | 136 kB |



Req/Bytes counts sampled once per second.

5k requests in 19.07s, 3.5 MB read
Running 20s test @ http://localhost:8080/info-console
100 connections



| Stat    | 2.5%   | 50%    | 97.5%  | 99%    | Avg       | Stdev    | Max    |
|---------|--------|--------|--------|--------|-----------|----------|--------|
| Latency | 321 ms | 372 ms | 522 ms | 531 ms | 389.89 ms | 58.73 ms | 544 ms |



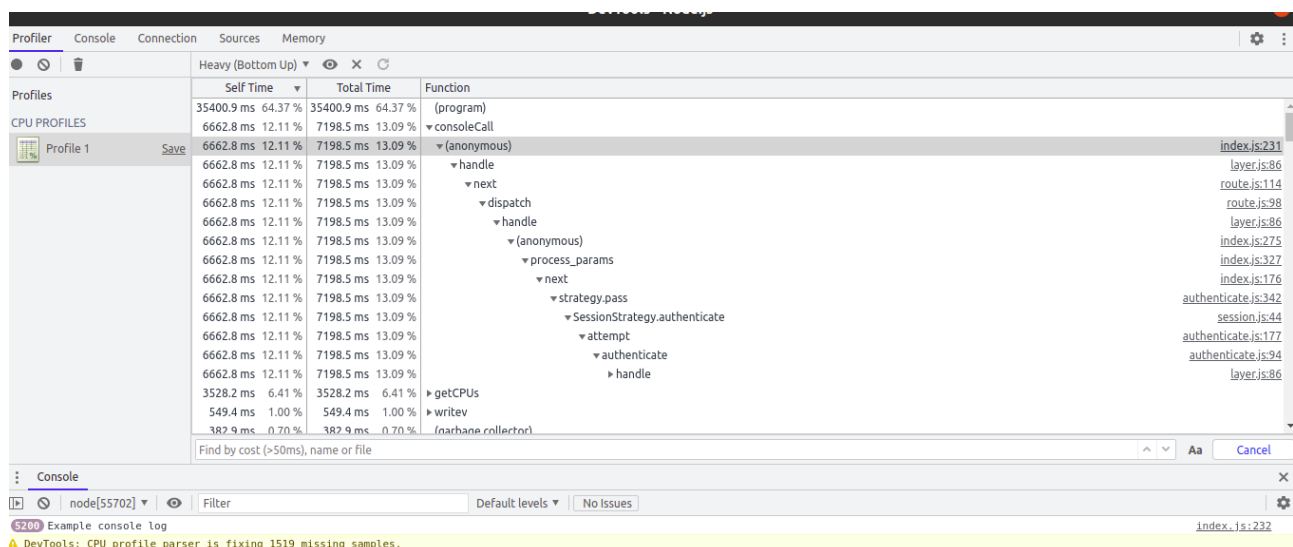
| Stat      | 1%     | 2.5%   | 50%    | 97.5%  | Avg    | Stdev   | Min    |
|-----------|--------|--------|--------|--------|--------|---------|--------|
| Req/Sec   | 188    | 188    | 256    | 300    | 255    | 37.67   | 188    |
| Bytes/Sec | 129 kB | 129 kB | 176 kB | 206 kB | 175 kB | 25.8 kB | 129 kB |



Req/Bytes counts sampled once per second.

5k requests in 19.09s, 3.5 MB read
nacchogonza@nacchogonza-Lenovo-V330-14IKB:~/Escritorio/backend-mern-clase
nacchogonza@nacchogonza-Lenovo-V330-14IKB:~/Escritorio/backend-mern-clase-32$
```

- Capturas del Inspector de Chrome: En primer lugar los procesos de la app (ordenados de mayor a menor por tiempo de duracion) y luego el detalle en el codigo de los tiempos de duraci3n en los casos con y sin console.log



ProfilerConsoleConnectionSourcesMemory

>

index.js x

Node.js: file

216

217

218

2190.3 ms

22020.2 ms

22120.6 ms

2221.2 ms

2230.5 ms

2242.9 ms

2251.2 ms

2262.4 ms

22714.6 ms

228

229

230

2310.5 ms

23237.2 ms

23345.0 ms

23443.1 ms

2351.5 ms

2361.5 ms

2370.7 ms

2382.0 ms

2391.9 ms

24018.5 ms

241

2420.2 ms

243

244

}); /*

/* Process Info Route */

app.get("/info", (req, res) => {

res.send({

Argumentos de entrada: \${process.argv}

Plataforma: \${process.platform}

Version de Node: \${process.version}

ID del proceso: \${process.pid}

Directorio de Trabajo: \${process.cwd()}

Directorio de Ejecucion: \${process.argv[0]}

Cantidad de procesadores en el : \${os.cpus().length}

});

});

app.get("/info-console", [(req, res) => {

console.log("Example console log");

res.send({

Argumentos de entrada: \${process.argv}

Plataforma: \${process.platform}

Version de Node: \${process.version}

ID del proceso: \${process.pid}

Directorio de Trabajo: \${process.cwd()}

Directorio de Ejecucion: \${process.argv[0]}

Cantidad de procesadores en el : \${os.cpus().length}

});

});

app.get("/infozip", compression(), (req, res) => {

Line 231, Column 28

Coverage: n/a

Console

node[55702] Filter Default levels No Issues

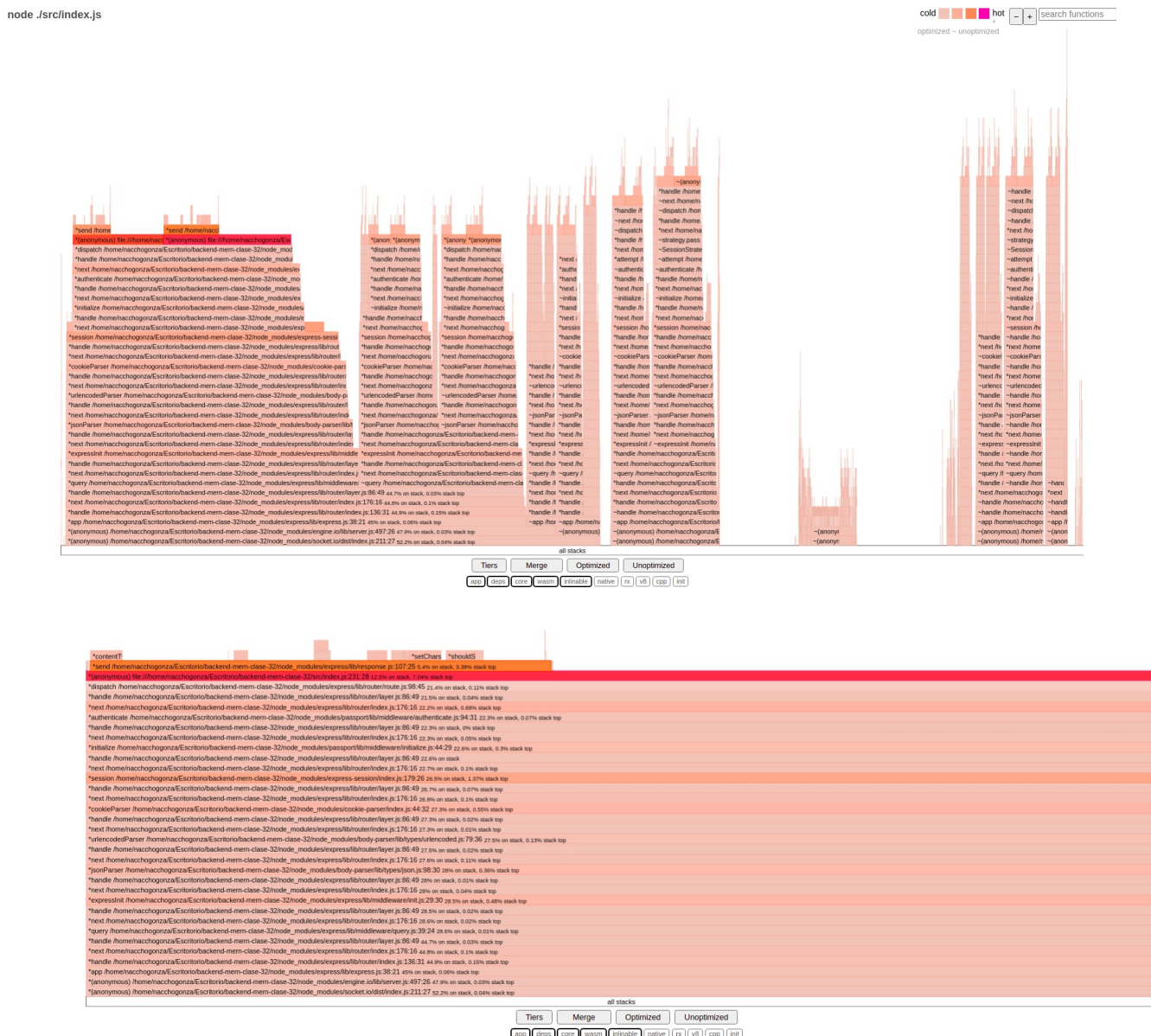
5280 Example console log

DevTools: CPU profile parser is fixing 1519 missing samples.

index.js:232

Diagrama de Flama con 0x

- Del lado izquierdo y mas duradero vemos el caso con el console.log, en el cual se marca como HOT el código a partir de la línea 231 del código (correspondiente al endpoint del /info-console)
- Debajo podemos ver en más detalle el proceso del caso del endpoint de console.log



A modo de conclusión, creo que en cada uno de los casos, tanto en las capturas como en los archivos de resultados de test de carga que quedaron en el repositorio, podemos observar como los tiempos de repuesta son mayores en el caso del endpoint con “console.log”, demostrando asi como cada pequeño proceso que forme parte de la respuesta que se da desde el servidor va a generar un mayor tiempo de respuesta para poder realizar la acción determinada

Por otra parte, creo que es muy buena la comparación visual que puede hacerse en el diagrama de flamas con 0x, que nos demuestra de forma muy simple el incremento de tiempo de un simple “console.log” y como en ese caso aparece el proceso calificado como Hot que aumenta asi los tiempos de respuesta