

ANÁLISIS COMPLETO DE PERFORMANCE

- 1) Análisis con node --prof
 - a) Resultado del análisis del proceso bloqueante:

```
[Summary]:
  ticks  total  nonlib   name
    75    3.7%   97.4%  JavaScript
     0    0.0%    0.0%    C++
    28    1.4%   36.4%    GC
  1959   96.2%           Shared libraries
     2    0.1%           Unaccounted
```

Salida de Artillery:

```
-----
Summary report @ 18:28:47 (-0300)
-----

http.codes.200: ..... 16000
http.request_rate: ..... 353/sec
http.requests: ..... 16000
http.response_time:
  min: ..... 2
  max: ..... 248
  median: ..... 106.7
  p95: ..... 135.7
  p99: ..... 147
http.responses: ..... 16000
vusers.completed: ..... 80
vusers.created: ..... 80
vusers.created_by_name.0: ..... 80
vusers.failed: ..... 0
vusers.session_length:
  min: ..... 20550.2
  max: ..... 22085.3
  median: ..... 21813.5
  p95: ..... 22254.1
  p99: ..... 22254.1
```

Resultado Autocannon:

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	45 ms	49 ms	69 ms	77 ms	50.65 ms	6.76 ms	107 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	1709	1709	1963	2109	1955.2	111.09	1709
Bytes/Sec	2.62 MB	2.62 MB	3 MB	3.23 MB	2.99 MB	170 kB	2.61 MB

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

39k requests in 20.05s, 59.8 MB read

b) Resultado del análisis del proceso no bloqueante:

```
[Summary]:
  ticks  total  nonlib   name
    64    4.3%   95.5%  JavaScript
     0    0.0%    0.0%    C++
    18    1.2%   26.9%    GC
  1438   95.5%
     3    0.2%   Unaccounted
```

Salida de Artillery:

```
-----
Summary report @ 18:28:00(-0300)
-----

http.codes.200: ..... 16000
http.request_rate: ..... 1143/sec
http.requests: ..... 16000
http.response_time:
  min: ..... 1
  max: ..... 66
  median: ..... 27.9
  p95: ..... 40.9
  p99: ..... 55.2
http.responses: ..... 16000
vusers.completed: ..... 80
vusers.created: ..... 80
vusers.created_by_name.0: ..... 80
vusers.failed: ..... 0
vusers.session_length:
  min: ..... 5043.2
  max: ..... 5880.9
  median: ..... 5711.5
  p95: ..... 5826.9
  p99: ..... 5826.9
```

Resultado Autocannon:

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	16 ms	19 ms	35 ms	42 ms	19.64 ms	4.76 ms	67 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	3015	3015	5111	5403	4974.65	497.94	3015
Bytes/Sec	4.62 MB	4.62 MB	7.82 MB	8.27 MB	7.61 MB	761 kB	4.61 MB

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

100k requests in 20.05s, 152 MB read

c) Conclusiones

Se puede observar diferencia en la performance entre el proceso bloqueante y el no bloqueante, si bien el perfilado de Node muestra solo un 26% mejor performance, en la salida de Artillery se puede notar la gran diferencia con una tasa de pedidos de mas del triple y una media de tiempo de respuesta 60% menor. Esto ultimo se respalda con los resultados de Autocannon que muestran 3 veces mas pedidos en el mismo tiempo y mas del doble de pedidos en promedio.

2) Perfilamiento con node –inspect

a) Proceso bloqueante:

Self Time	Total Time	Function
14158.4 ms	14158.4 ms	(idle)
8314.9 ms 37.47 %	8314.9 ms 37.47 %	▸ writeUtf8String
7525.8 ms 33.92 %	15440.1 ms 69.59 %	▼ consoleCall
7525.8 ms 33.92 %	15440.1 ms 69.59 %	▸ getInfoLogPage
415.2 ms 1.87 %	415.2 ms 1.87 %	(garbage collector)
326.7 ms 1.47 %	326.7 ms 1.47 %	▸ writev
325.4 ms 1.47 %	1820.2 ms 8.20 %	▸ initialize
265.6 ms 1.20 %	265.6 ms 1.20 %	(program)
233.0 ms 1.05 %	339.0 ms 1.53 %	▸ nextTick
179.2 ms 0.81 %	73395.1 ms 330.78 %	▸ next

Codigo:

```

37
38 4.8 ms const getInfoLogPage = async (req, res) => {
39   try {
40     5.5 ms   const html = buildInfoPage();
41     15.0 ms   console.log(html);
42     27.5 ms   res.end(html);
43   } catch (error) {
44     logger.error(error.message);
45     2.1 ms   res.status(HTTP_STATUS_ERROR_BAD_REQUEST).send(error.message);
46   }
47 }
48 };
```

b) Proceso no bloqueante:

Self Time	Total Time	Function
19688.2 ms	19688.2 ms	(idle)
873.2 ms 14.31 %	873.2 ms 14.31 %	▸ writeUtf8String
415.3 ms 6.80 %	2083.7 ms 34.14 %	▸ initialize
308.0 ms 5.05 %	308.0 ms 5.05 %	(garbage collector)
206.1 ms 3.38 %	3003.1 ms 49.21 %	▸ compression
200.5 ms 3.29 %	200.5 ms 3.29 %	(program)
193.3 ms 3.17 %	193.3 ms 3.17 %	▸ writev
176.9 ms 2.90 %	245.0 ms 4.01 %	▸ nextTick
151.8 ms 2.49 %	28198.3 ms 462.05 %	▸ next
134.0 ms 2.20 %	2733.3 ms 44.79 %	▸ session

c) Conclusiones

Como en el punto uno se nota diferencia entre el proceso no bloqueante contra el bloqueante insumiendo este ultimo 15s extra de ejecución.

3) Diagrama de flama

a) Bloqueante:
node src/server.js



b) No bloqueante:

node src/server.js



c) Conclusiones

Como en los casos anteriores se observa una notable diferencia entre ambos procesos, mostrando como las operaciones del proceso bloqueante se extienden por mas tiempo.

4) Conclusión general

Las herramientas testeadas demuestran ser muy útiles a la hora de poder determinar el desempeño de una aplicación y poder aislar posibles focos de bajo rendimiento en la misma.