



Enunciados

Consigna 1

Incorporar al proyecto de servidor de trabajo la compresión gzip. Verificar sobre la ruta /info con y sin compresión, la diferencia de cantidad de bytes devueltos en un caso y otro.

Luego implementar logguego (con alguna librería vista en clase) que registre lo siguiente:

- Ruta y método de todas las peticiones recibidas por el servidor (info)
- Ruta y método de las peticiones a rutas inexistentes en el servidor (warning)
- Errores lanzados por las apis de mensajes y productos, únicamente (error)

Considerar el siguiente criterio:

- Loggear todos los niveles a consola (info, warning y error)
- Registrar sólo los logs de warning a un archivo llamada warn.log
- Enviar sólo los logs de error a un archivo llamada error.log

Consigna 2

Luego, realizar el análisis completo de performance del servidor con el que venimos trabajando.

Vamos a trabajar sobre la ruta '/info', en modo fork, agregando ó extrayendo un console.log de la información colectada antes de devolverla al cliente. Además desactivaremos el child_process de la ruta '/randoms'

Para ambas condiciones (con o sin console.log) en la ruta '/info' OBTENER:

El perfilamiento del servidor, realizando el test con --prof de node.js. Analizar los resultados obtenidos luego de procesarlos con --prof-process. Utilizaremos como test de carga Artillery en línea de comandos, emulando 50 conexiones concurrentes con 20 request por cada una. Extraer un reporte con los resultados en archivo de texto.

Consigna 3

Luego utilizaremos Autocannon en línea de comandos, emulando 100 conexiones concurrentes realizadas en un tiempo de 20 segundos. Extraer un reporte con los resultados (puede ser un print screen de la consola)

El perfilamiento del servidor con el modo inspector de node.js --inspect. Revisar el tiempo de los procesos menos performantes sobre el archivo fuente de inspección.

Consigna 4

El diagrama de flama con 0x, emulando la carga con Autocannon con los mismos parámetros anteriores.

Consigna 1

Perfilamiento /info

```
mkdir perfilamiento_info
cd perfilamiento_info
artillery quick -c 50 -n 20 "http://localhost:8080/info" > artillery_info.txt
node --prof-process isolate-0x61b90c0-12093-v8.log > prof_info.txt
```

Perfilamiento /info_debug

```
mkdir perfilamiento_info_debug
cd perfilamiento_info_debug
artillery quick -c 50 -n 20 "http://localhost:8080/info_debug" > artillery_info_debug.txt
node --prof-process isolate-0x61b90c0-12093-v8.log > prof_info_debug.txt
```

Consigna 2

Ejecucion

Se instancia el server en modo **inspect**

```
node --inspect main.js -p 8080 -m fork
```

Se realiza un benchmark empleando **autocannon**

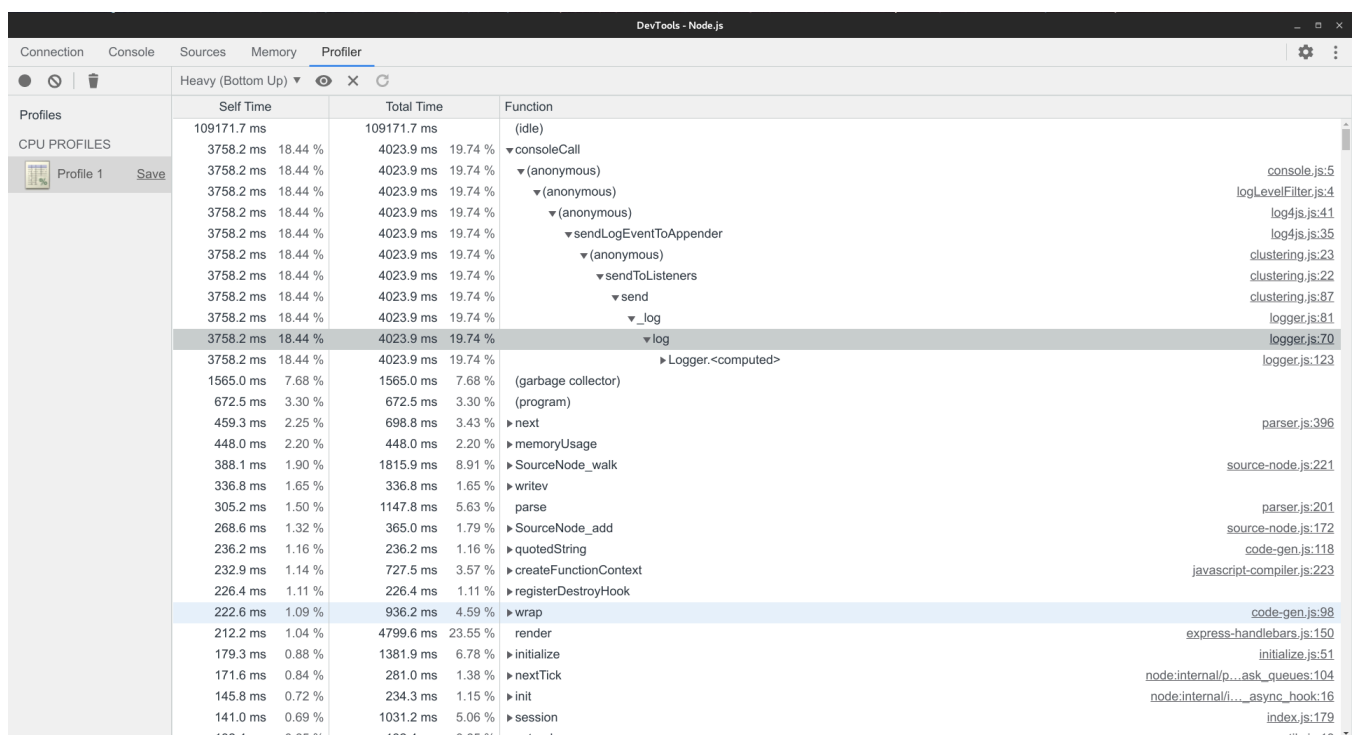
```
node benchmark_individual.js http://localhost:8080/info
```

Luego se configura en Chrome un profiling especifico para cada ruta.

Resultado

Ruta /info

Se observa a continuación el resultado de los tiempos requeridos en la ruta "/info"



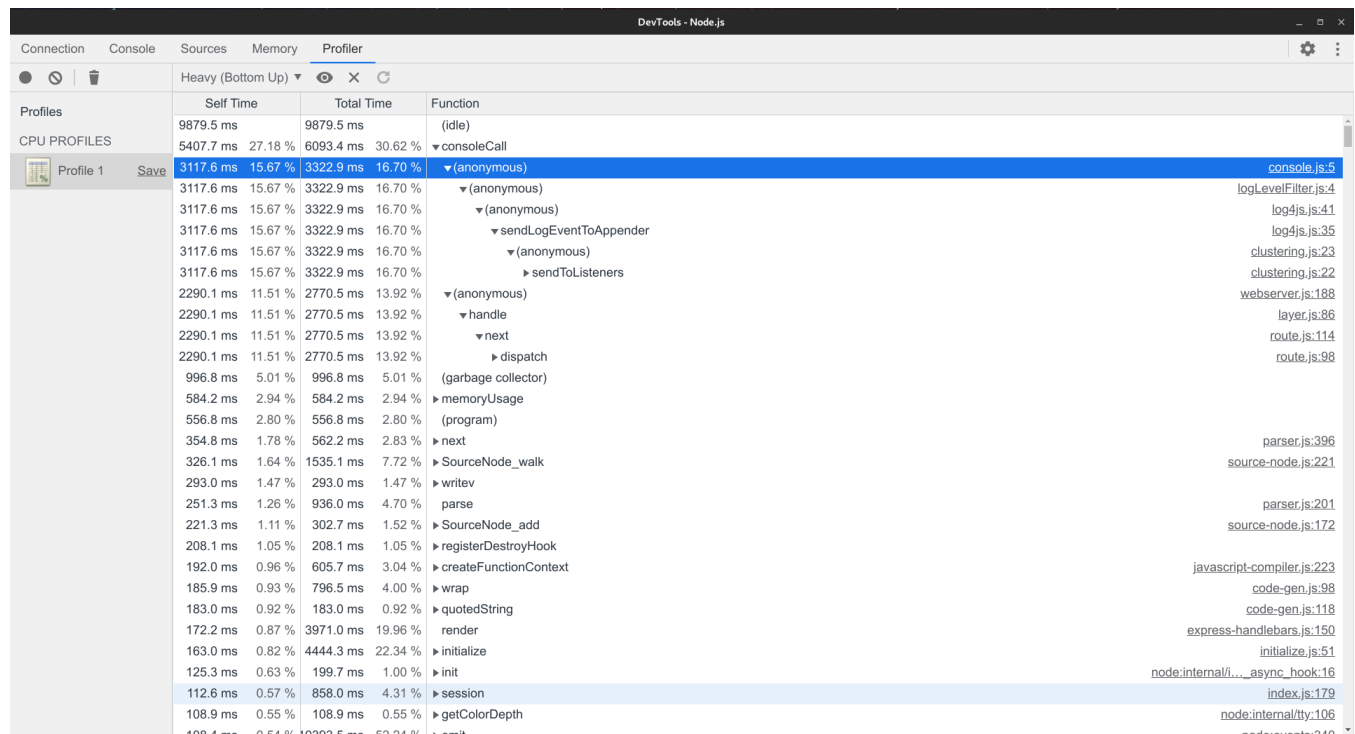
Self Time	Total Time	Function
109171.7 ms	109171.7 ms	(idle)
3758.2 ms 18.44 %	4023.9 ms 19.74 %	▼ consoleCall
3758.2 ms 18.44 %	4023.9 ms 19.74 %	▼ (anonymous)
3758.2 ms 18.44 %	4023.9 ms 19.74 %	▼ (anonymous)
3758.2 ms 18.44 %	4023.9 ms 19.74 %	▼ (anonymous)
3758.2 ms 18.44 %	4023.9 ms 19.74 %	▼ sendLogEventToAppender
3758.2 ms 18.44 %	4023.9 ms 19.74 %	▼ (anonymous)
3758.2 ms 18.44 %	4023.9 ms 19.74 %	▼ sendToListeners
3758.2 ms 18.44 %	4023.9 ms 19.74 %	▼ send
3758.2 ms 18.44 %	4023.9 ms 19.74 %	▼ _log
3758.2 ms 18.44 %	4023.9 ms 19.74 %	▼ log
3758.2 ms 18.44 %	4023.9 ms 19.74 %	▶ Logger.<computed>
1565.0 ms 7.68 %	1565.0 ms 7.68 %	(garbage collector)
672.5 ms 3.30 %	672.5 ms 3.30 %	(program)
459.3 ms 2.25 %	698.8 ms 3.43 %	▶ next
448.0 ms 2.20 %	448.0 ms 2.20 %	▶ memoryUsage
388.1 ms 1.90 %	1815.9 ms 8.91 %	▶ SourceNode_walk
336.8 ms 1.65 %	336.8 ms 1.65 %	▶ writev
305.2 ms 1.50 %	1147.8 ms 5.63 %	parse
268.6 ms 1.32 %	365.0 ms 1.79 %	▶ SourceNode_add
236.2 ms 1.16 %	236.2 ms 1.16 %	▶ quotedString
232.9 ms 1.14 %	727.5 ms 3.57 %	▶ createFunctionContext
226.4 ms 1.11 %	226.4 ms 1.11 %	▶ registerDestroyHook
222.6 ms 1.09 %	936.2 ms 4.59 %	▶ wrap
212.2 ms 1.04 %	4799.6 ms 23.55 %	render
179.3 ms 0.88 %	1381.9 ms 6.78 %	▶ initialize
171.6 ms 0.84 %	281.0 ms 1.38 %	▶ nextTick
145.8 ms 0.72 %	234.3 ms 1.15 %	▶ init
141.0 ms 0.69 %	1031.2 ms 5.06 %	▶ session
132.4 ms 0.65 %	132.4 ms 0.65 %	▶ extend

El archivo completo se adjunta a continuación:

- [ruta_info.cpubprofile](#)

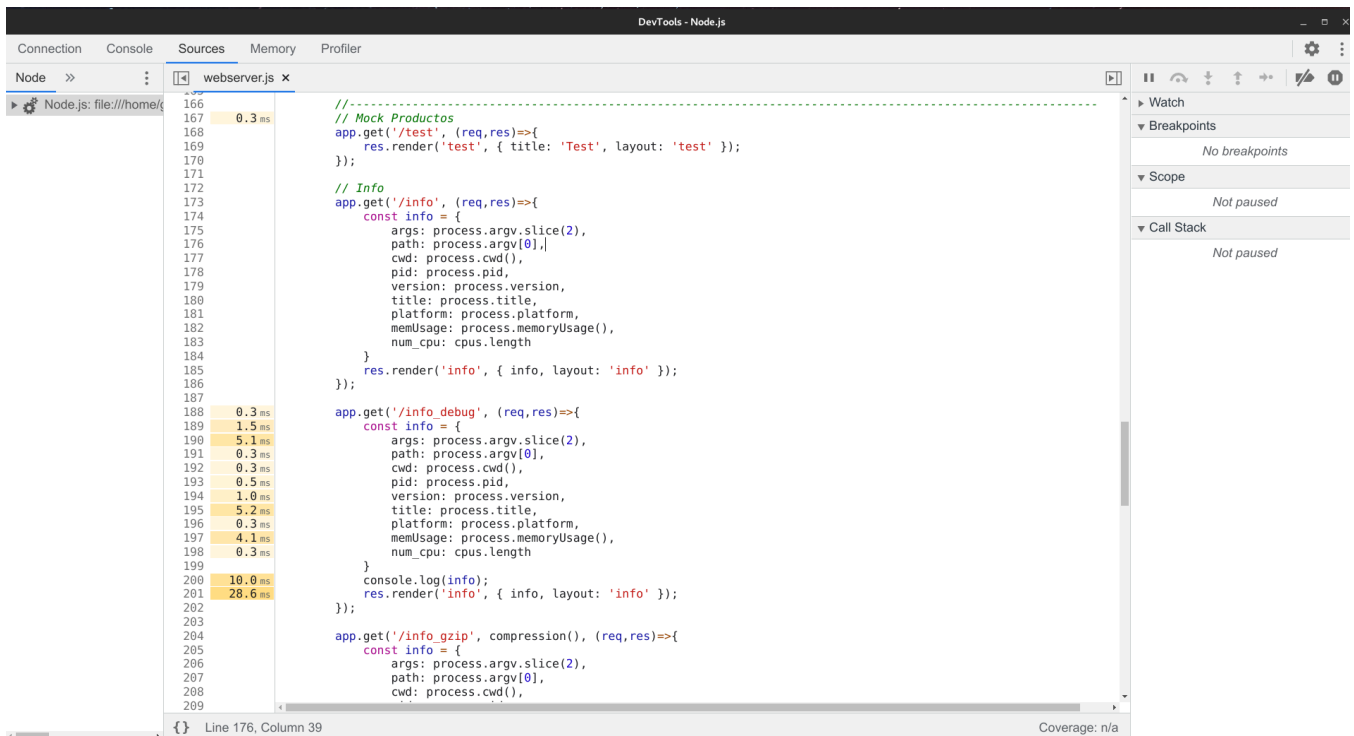
Ruta /info_debug

Se observa a continuación el resultado de los tiempos requeridos en la ruta "/info_debug"



Profiles	Self Time	Total Time	Function
CPU PROFILES	9879.5 ms	9879.5 ms	(idle)
	5407.7 ms 27.18 %	6093.4 ms 30.62 %	▼ consoleCall
Profile 1 Save	3117.6 ms 15.67 %	3322.9 ms 16.70 %	▼ (anonymous) console.js:5
	3117.6 ms 15.67 %	3322.9 ms 16.70 %	▼ (anonymous) logLevelFilter.js:4
	3117.6 ms 15.67 %	3322.9 ms 16.70 %	▼ (anonymous) log4js.js:41
	3117.6 ms 15.67 %	3322.9 ms 16.70 %	▼ sendLogEventToAppender log4js.js:35
	3117.6 ms 15.67 %	3322.9 ms 16.70 %	▼ (anonymous) clustering.js:23
	3117.6 ms 15.67 %	3322.9 ms 16.70 %	▶ sendToListeners clustering.js:22
	2290.1 ms 11.51 %	2770.5 ms 13.92 %	▼ (anonymous) webserver.js:188
	2290.1 ms 11.51 %	2770.5 ms 13.92 %	▼ handle layer.js:86
	2290.1 ms 11.51 %	2770.5 ms 13.92 %	▼ next route.js:114
	2290.1 ms 11.51 %	2770.5 ms 13.92 %	▶ dispatch route.js:98
	996.8 ms 5.01 %	996.8 ms 5.01 %	(garbage collector)
	584.2 ms 2.94 %	584.2 ms 2.94 %	▶ memoryUsage
	556.8 ms 2.80 %	556.8 ms 2.80 %	(program)
	354.8 ms 1.78 %	562.2 ms 2.83 %	▶ next
	326.1 ms 1.64 %	1535.1 ms 7.72 %	▶ SourceNode_walk
	293.0 ms 1.47 %	293.0 ms 1.47 %	▶ writev
	251.3 ms 1.26 %	936.0 ms 4.70 %	parse
	221.3 ms 1.11 %	302.7 ms 1.52 %	▶ SourceNode_add
	208.1 ms 1.05 %	208.1 ms 1.05 %	▶ registerDestroyHook
	192.0 ms 0.96 %	605.7 ms 3.04 %	▶ createFunctionContext
	185.9 ms 0.93 %	796.5 ms 4.00 %	▶ wrap
	183.0 ms 0.92 %	183.0 ms 0.92 %	▶ quotedString
	172.2 ms 0.87 %	3971.0 ms 19.96 %	render
	163.0 ms 0.82 %	4444.3 ms 22.34 %	▶ initialize
	125.3 ms 0.63 %	199.7 ms 1.00 %	▶ init
	112.6 ms 0.57 %	858.0 ms 4.31 %	▶ session
	108.9 ms 0.55 %	108.9 ms 0.55 %	▶ getColorDepth
	108.4 ms 0.54 %	10302.5 ms 52.24 %	▶ emit

Luego, analizando el detalle de este analisis, se observa el consumo de "console.log".



El archivo completo se adjunta a continuación:

- [ruta_info_debug.cpubuprofile](#)

Consigna 3

Ejecucion Conjunta

Para ejecutar el server con el profiling de 0x:

```
0x main.js -p 8080 -m fork
```

Luego para disparar las pruebas en simultaneo de ambas rutas:

```
node benchmark.js
```

Resultado

```
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	385 ms	582 ms	1426 ms	1744 ms	670.06 ms	282.68 ms	2342 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	0	0	126	268	148.25	65.34	99
Bytes/Sec	0 B	0 B	397 kB	844 kB	467 kB	206 kB	312 kB

Req/Bytes counts sampled once per second.

```
3k requests in 20.07s, 9.33 MB read
Running 20s test @ http://localhost:8080/info_debug
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	387 ms	581 ms	1980 ms	2134 ms	681.09 ms	341.36 ms	2307 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	0	0	128	300	145	71.69	37
Bytes/Sec	0 B	0 B	403 kB	945 kB	456 kB	226 kB	116 kB

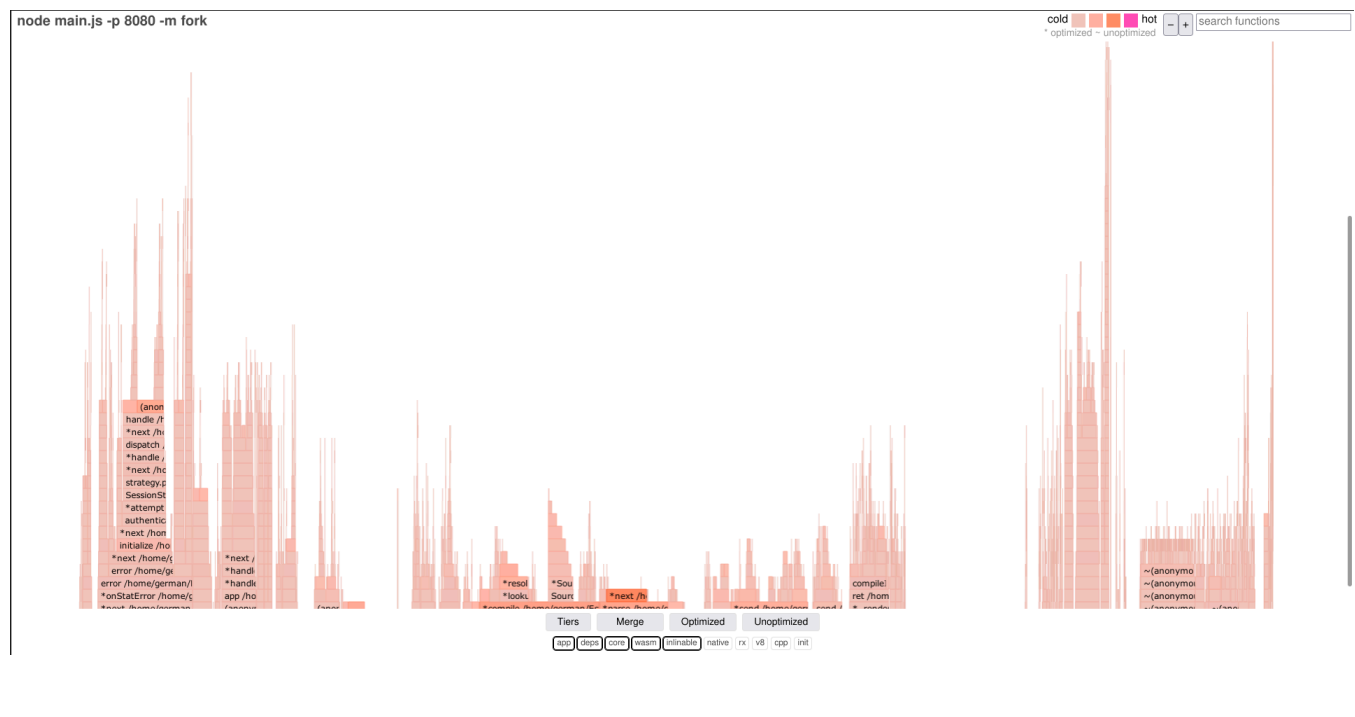
Req/Bytes counts sampled once per second.

```
3k requests in 20.06s, 9.13 MB read
```

Se adjunta a continuación el diagrama de flama asociado a dicho prueba.

[Diagrama de Flama](#)

El diagrama de flama obtenido es el siguiente:



Consigna 4 - 0x solo ruta INFO

Ejecucion

Para ejecutar el server con el profiling de 0x:

```
0x main.js -p 8080 -m fork
```

Luego para disparar las pruebas en simultaneo de ambas rutas:

```
node benchmark_individual.js http://localhost:8080/info
```

Resultado

```
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	198 ms	237 ms	400 ms	637 ms	257.14 ms	70.47 ms	875 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	136	136	400	475	385.85	80.19	136
Bytes/Sec	428 kB	428 kB	1.26 MB	1.5 MB	1.21 MB	252 kB	428 kB

Req/Bytes counts sampled once per second.

8k requests in 20.05s, 24.3 MB read

El diagrama de flama de resultado se obtiene en el siguiente archivo:

```
Flame_Result_INFO
```

Se adjunta a continuación el diagrama de flama asociado a dicho prueba.

Diagrama de Flama

El diagrama de flama obtenido es el siguiente:

Resultado

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

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	204 ms	247 ms	404 ms	663 ms	269.5 ms	74.26 ms	908 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	120	120	387	459	368.3	77.59	120
Bytes/Sec	378 kB	378 kB	1.22 MB	1.44 MB	1.16 MB	244 kB	378 kB

Req/Bytes counts sampled once per second.

7k requests in 20.05s, 23.2 MB read

El diagrama de flama de resultado se obtiene en el siguiente archivo:

Flame_Result_INFO_Debug

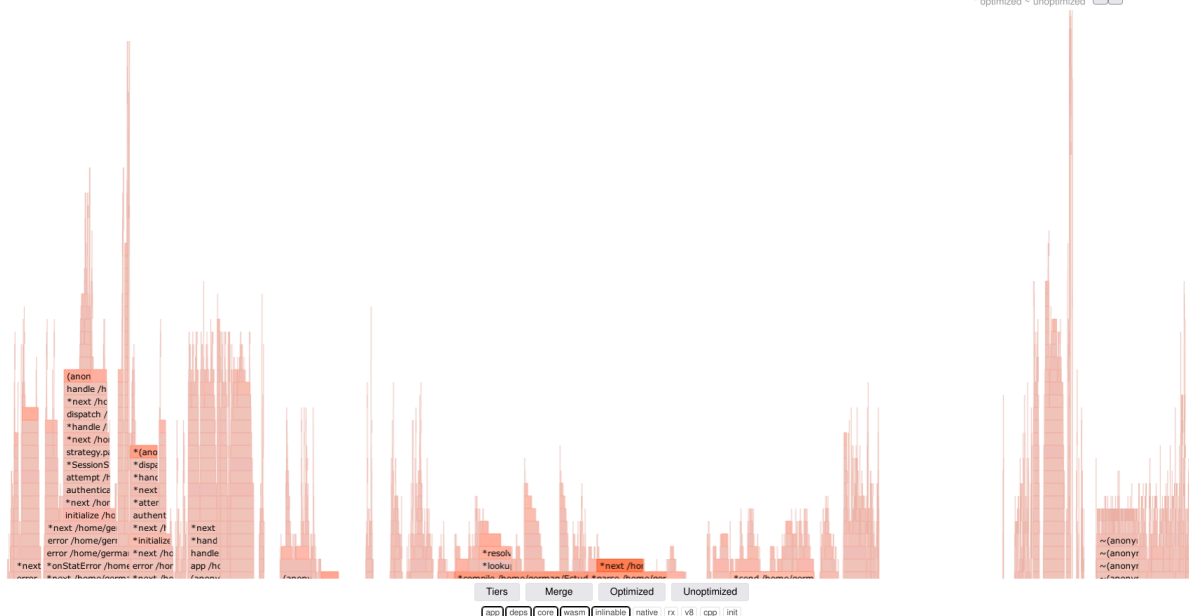
Se adjunta a continuación el diagrama de flama asociado a dicho prueba.

Diagrama de Flama

El diagrama de flama obtenido es el siguiente:

node main.js -p 8080 -m fork

cold hot
* optimized - unoptimized



Conclusiones

En base al análisis de perfilamiento se observa que la ruta "/info_debug" consume una mayor cantidad de recursos como consecuencia de la operación "console.log" que se emplea para el debug del proceso.

node main.js -p 8080 -m fork

cold hot
* optimized - unoptimized



```
171
172 // Info
173 app.get('/info', (req,res)=>{
174     const info = {
175         args: process.argv.slice(2),
176         path: process.argv[0],
177         cwd: process.cwd(),
178         pid: process.pid,
179         version: process.version,
180         title: process.title,
181         platform: process.platform,
182         memUsage: process.memoryUsage(),
183         num_cpu: cpus.length
184     }
185     res.render('info', { info, layout: 'info' });
186 });
187
188 app.get('/info_debug', (req,res)=>{
189     const info = {
190         args: process.argv.slice(2),
191         path: process.argv[0],
192         cwd: process.cwd(),
193         pid: process.pid,
194         version: process.version,
195         title: process.title,
196         platform: process.platform,
197         memUsage: process.memoryUsage(),
198         num_cpu: cpus.length
199     }
200     console.log(info);
201     res.render('info', { info, layout: 'info' });
202 });
203
```