ANÁLISIS

1)

Prendemos el servidor en modo Fork con el comando:

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

Ahora abrimos otra terminal, en la carpeta del proyecto, y procedemos a usar el test de carga:

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

Primero, debemos renombrar el archivo isolate como *log-v8.log* y antes de decodificarlo, en el archivo "*controllers\infoController.js*" comentamos la línea 15, donde ser realiza un console.log(info).

ahora ejecutamos lo siguiente:

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

Va a hacer un test de 20 request con 50 usuarios a la url especificada. Y el resultado lo va a guardar en el archivo result_sin_log.txt

Cambiar el nombre del archivo isolate como sinlog-v8.log

Pasamos ahora a decodificar los archivos log que se crearon:

```
node --prof-process log-v8.log > result_prof-log.txt
```

```
node --prof-process sinlog-v8.log > result_prof-sinlog.txt
```

Se crean entonces los archivos result_prof-log.txt y result_prof-sinlog.txt con la información de los primeros archivos decodificada.

Comparación

```
Statistical profiling result from log-v8.log, (2789 ticks, 0 unaccounted, 0 excluded).

 [Shared libraries]:
   ticks  total  nonlib   name
    2111   75.7%          C:\Windows\SYSTEM32\ntdll.dll
     644   23.1%          C:\Program Files (x86)\nodejs\node.exe
       2    0.1%          C:\Windows\System32\KERNELBASE.dll
```

result_prof-sinlog.txt

```
Statistical profiling result from sinlog-v8.log, (1808 ticks, 0 unaccounted, 0 excluded).

 [Shared libraries]:
   ticks  total  nonlib   name
    1282   70.9%          C:\Windows\SYSTEM32\ntdll.dll
     487   26.9%          C:\Program Files (x86)\nodejs\node.exe
       3    0.2%          C:\Windows\System32\KERNELBASE.dll

 [JavaScript]:
```

Vemos que en Shared libraries el proceso no bloqueante (es decir, sin console.log())se lleva muchos menos ticks de los que se lleva en el proceso bloqueante.

2)

Prendemos el servidor con el comando: `node --inspect ./server.js`

Abrimos DevTools en el navegador, y en la pestaña profiler ejecutamos “iniciar” o “start”. Una vez hecho esto, puedo volver a la consola y correr nuevamente los comandos del test de carga artillery que mencionamos anteriormente. Para los procesos bloqueante y no bloqueante.

En la vista del código podemos ver los milisegundos de cada función y ver las que están demorando la ejecución de la aplicación.

infoController.js

```
import config from "../config.js";
import os from "os";

const getInfo = (req, res) => {
  const info = {
    argumentos: Object.entries(config.ARG),
    SO: process.platform,
    v_node: process.version,
    rss: process.memoryUsage().rss,
    path: process.title,
    process_id: process.pid,
    carpeta_raiz: process.cwd(),
    cant_procesadores: os.cpus().length,
  };
  console.log(info);
  res.render("info", { info });
};

export { getInfo };
```

infoController.js

```
import config from "../config.js";
import os from "os";

const getInfo = (req, res) => {
  const info = {
    argumentos: Object.entries(config.ARG
    SO: process.platform,
    v_node: process.version,
    rss: process.memoryUsage().rss,
    path: process.title,
    process_id: process.pid,
    carpeta_raiz: process.cwd(),
    cant_procesadores: os.cpus().length,
  };
  // console.log(info);
  res.render("info", { info });
};

export { getInfo };
```

Caso 1: Con loggeo. Tiempo total estimado: 54.3 ms.

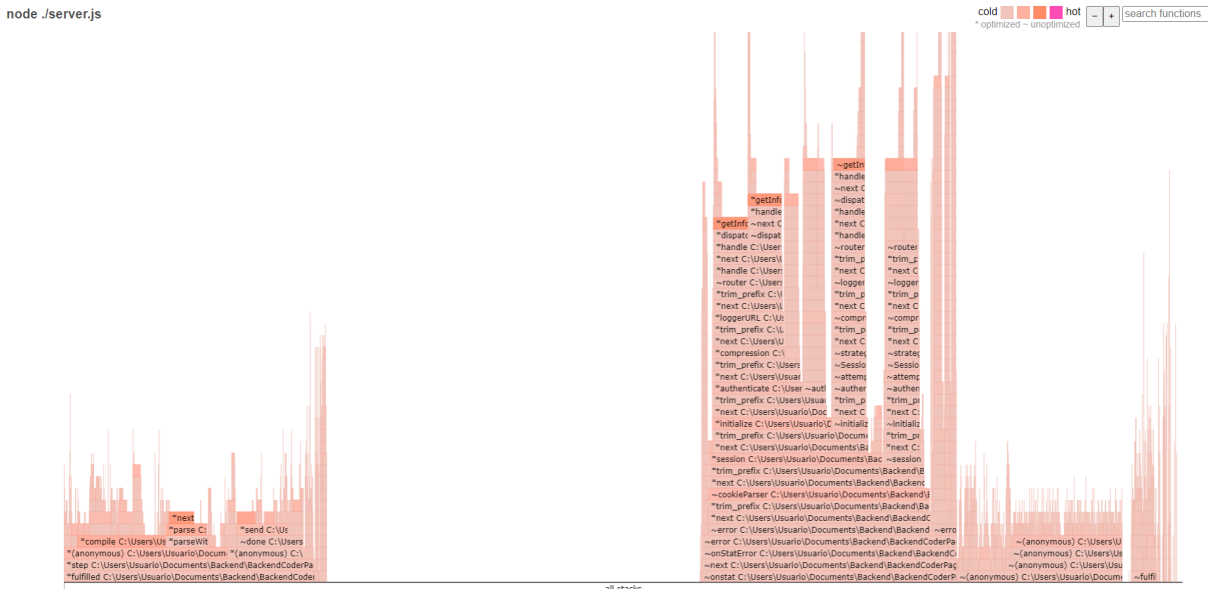
Caso 2: Sin loggeo. (Recordar en el archivo “*controllers\infoController.js*” comentamos la línea 15) Tiempo total estimado: 46.4 ms.

3)

Prendemos el servidor con el comando: `npm start`

Finalmente, ejecutamos los test con el comando: `npm test`

Caso 1: Con loggeo.



Caso 2: Sin loggeo. (Recordar en el archivo “*controllers\infoController.js*” comentamos la línea 15)

Comparación

Para el caso 2 vemos que los procesos tienen prácticamente todos el mismo color, propio del tipo no bloqueante. En el caso 1 los procesos de arriba son los que bloquean a los de abajo y son los que están en color más oscuro.

Conclusión

Observando todos los resultados obtenidos en cada test, se puede observar una gran diferencia en la performance al tener procesos bloqueantes y no bloqueantes. En aquellos test donde se realizaba un console.log(info), se puede observar que esto último demoraba la ejecución de la aplicación, ya que se trata de una función bloqueante. Caso contrario, en aquellos test donde estaba comentada la línea donde logueaba “info”, se vio una mejora en la ejecución de la aplicación.
Entonces, se puede concluir que al existir procesos no bloqueantes mejorará la ejecución de la aplicación.