Taller 3 - Pruebas Headless y No-Headless

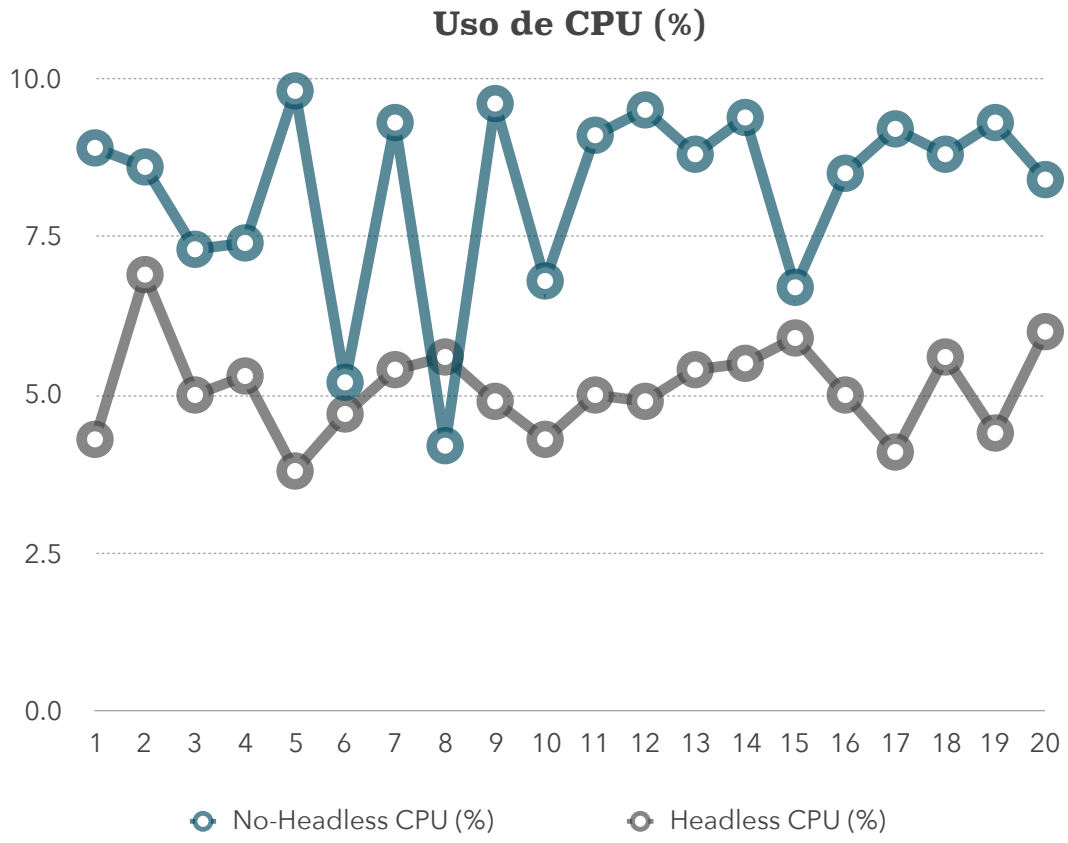
Diego Alfonso Prieto Torres - Febrero 21 de 2018

Protractor

Rendimiento de Protractor:

Se ejecutaron 20 veces las pruebas desarrolladas en protractor para la aplicación **Tour of Heroes**. Para cada ejecución se hará la prueba de forma **"headless"** y **"No-Headless"**. El objetivo fue medir el consumo de CPU, RAM y tiempo de ejecución.

Uso de CPU (%)

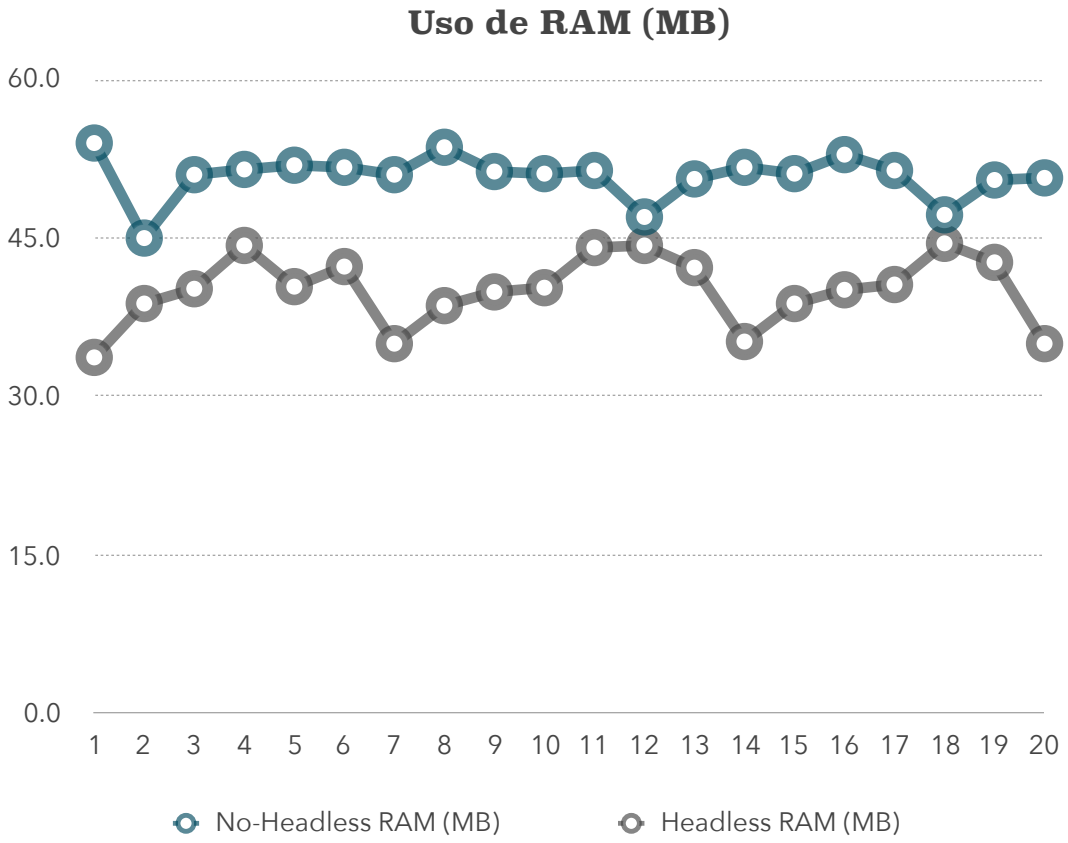


Es evidente que Protractor consume muchos menos recursos de procesador en términos del porcentaje de CPU que usa para ejecutar las pruebas.

En promedio de forma No-Headless, protractor consume 8.2% de CPU, con un máximo de 9.8% presentado durante las pruebas; además el comportamiento es bastante irregular

logrando que sea poco predecible el consumo de recursos. Por el otro lado, de forma Headless, Protractor mantuvo un consumo promedio de 5.1% de CPU, alcanzando un máximo de 6.9% que es de hecho inferior al promedio de No-Headless.

Consumo de Memoria RAM (MB)



Aunque Protractor consume menos memoria RAM si se ejecuta de modo Headless, tan solo hubo una variación en promedio de 10,7 MB, y vale la pena recalcar que ambas medidas se mantuvieron estables durante las pruebas.

Tiempo de ejecución (Seg)

17 segundos fue el tiempo utilizado por Protector para ejecutar las pruebas independiente del tipo de ejecución que se uso. Por tal razón este item no fue tenido en cuenta para tomar desiciones.

Conclusión

Sin lugar a dudas, Protractor ejecutado de modo Headless consume muchos menos recursos, se pueden ahorrar hasta 3.2% de uso de CPU y 10.7 MB en memoria RAM, lo cual es un ahorro significativo en pruebas de larga duración sobretodo si se están ejecutando en ambientes cloud.

Por otro lado la estabilidad de los tiempos de ejecución da seguridad sobre la herramienta y permite calcular y determinar la duración de un ciclo de pruebas.

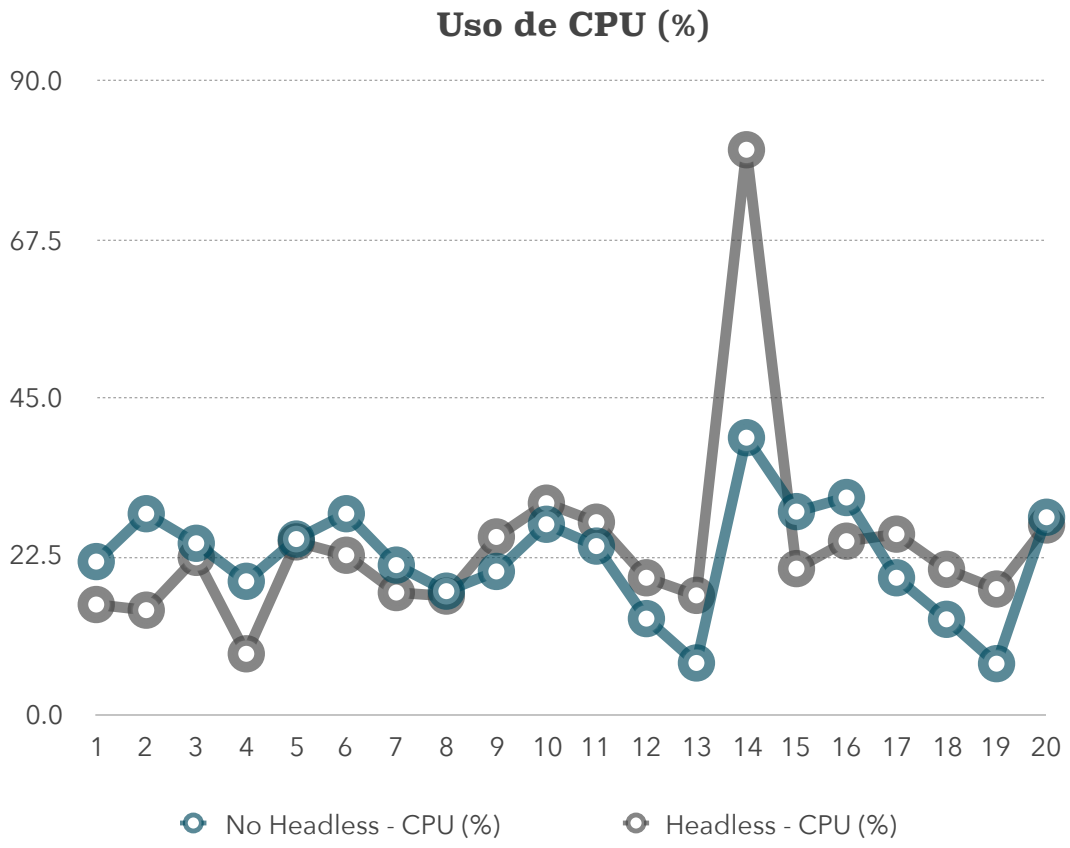
Si no es importante ver la ejecución de las pruebas sino solamente el resultado, se aconseja usar Protractor de modo Headless.

Web Driver IO

Rendimiento de WebDriver IO:

Se ejecutaron 20 veces las pruebas desarrolladas en WebDriver IO para la aplicación **losestudiantes.com**. Para cada ejecución se hará la prueba de forma **"headless"** y **"No-Headless"**. El objetivo fue medir el consumo de CPU, RAM y tiempo de ejecución.

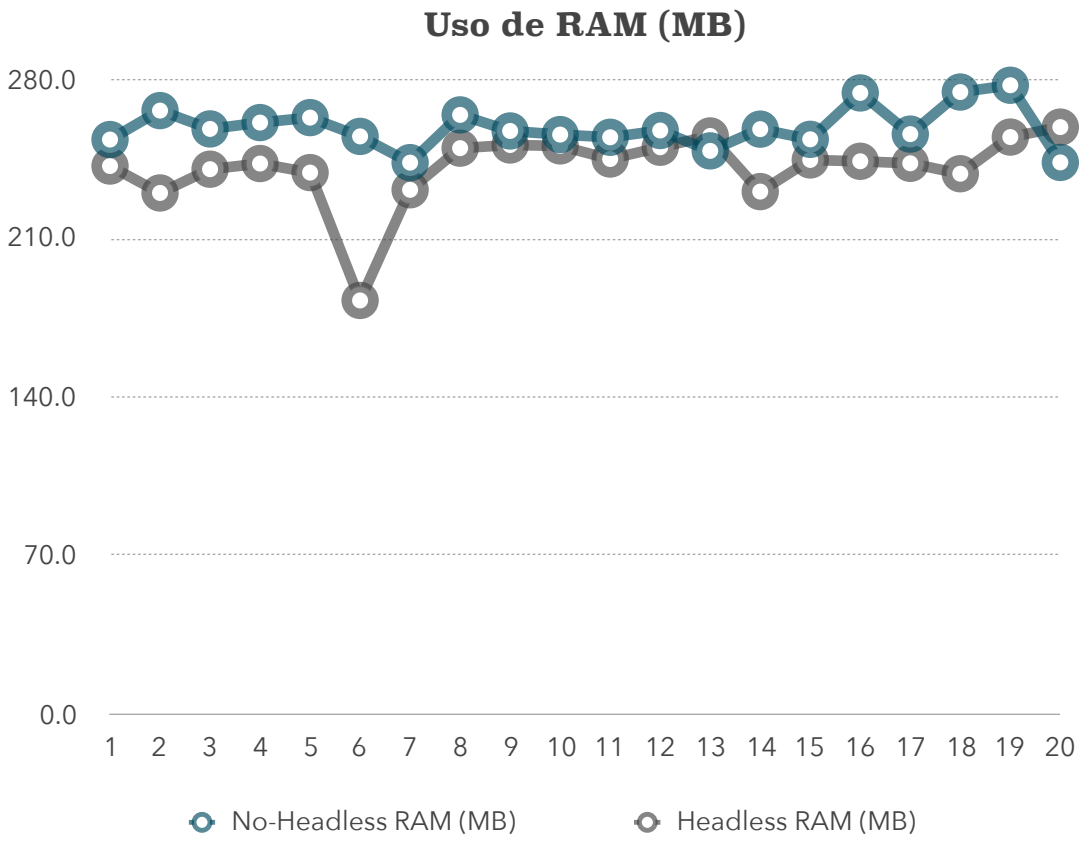
Uso de CPU (%)



Aunque por poco, WebDriver IO consume menos recursos en términos de CPU si se ejecuta de forma No-Headless, tan solo 1,8 % de mejora y esto se debe a que de forma Headless, WebDriver IO debe levantar unos procesos de Chrome conocidos como Headless que aumentan la carga inicial.

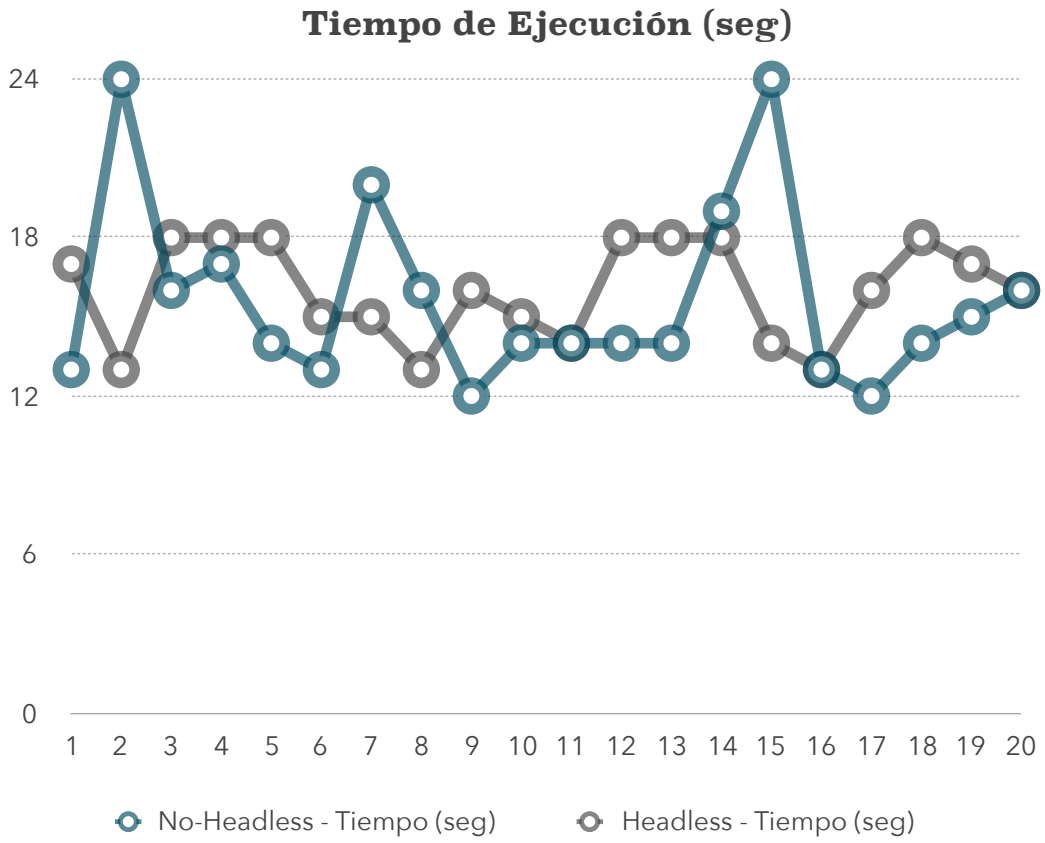
El pico que se presenta en la gráfica fue debido a una ejecución errónea de WebDriver IO que produjo una gran carga sobre el procesador.

Consumo de Memoria RAM (MB)



Tampoco hay una gran diferencia en el consumo de memoria RAM ya que tan solo por 17,6 MB WebDriver IO tiene un menor consumo si se se ejecuta de forma Headless. Esto se produce debido a que el proceso de la interfaz gráfica de chrome no se levanta, pero los helpers del motor de renderizado siguen funcionando.

Tiempo de ejecución (Seg)



En cuanto al tiempo de ejecución, ambos se comportaron de forma muy irregular, aunque en promedio ambas ejecuciones reportaron tiempos cercanos a los 16 segundos, los máximos y mínimos estuvieron entre 12 segundos y 24 segundos. Esto se debe a que el servidor de selenium no tiene tiempos de respuesta fijos para la interacción con el mismo.

Conclusión

Es irrelevante tomar una decisión en este punto porque el comportamiento es muy similar, una vez más, si el hecho es correr las pruebas en un servidor, usar Headless es más útil pq consume menos recursos de RAM que puede llegar a ser costoso incluso en servidores en la Nube.

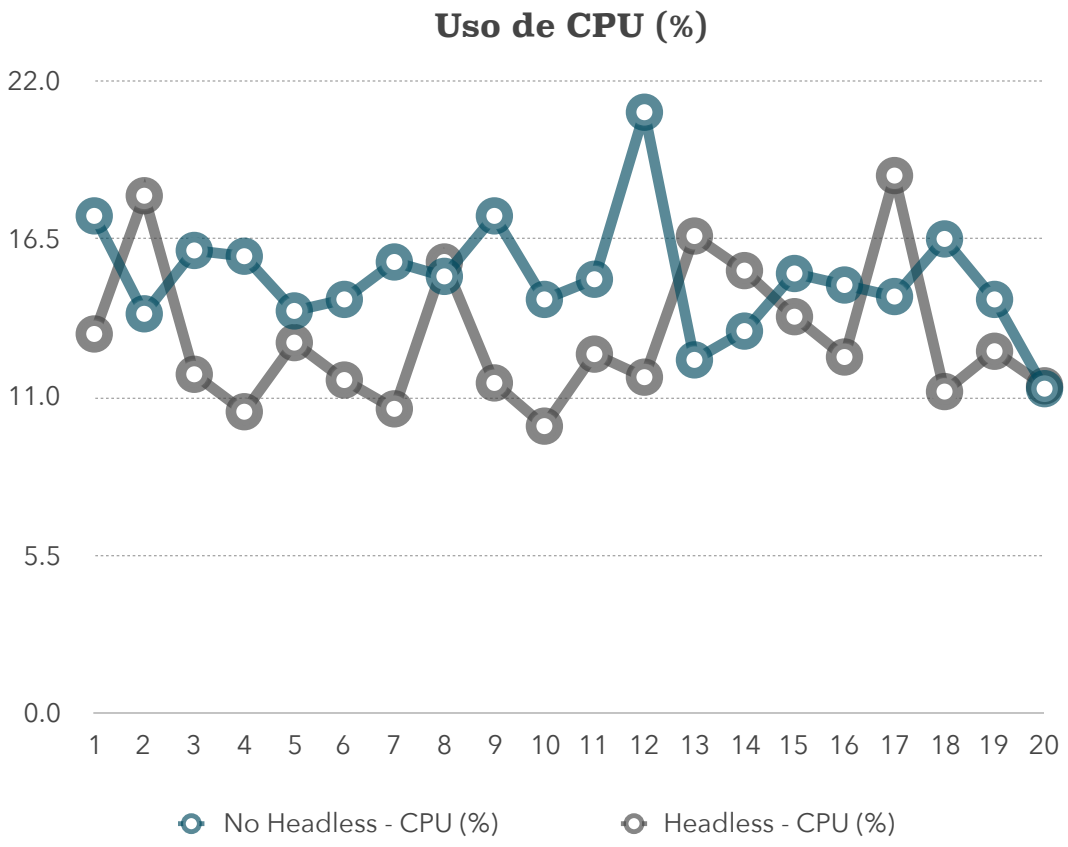
Lo que si preocupa es que los tiempos de ejecución varíen de esa forma, ya que resulta muy difícil poder predecir los tiempos de ejecución y calcular el tiempo necesario para ejecutar un ciclo de pruebas más grande de forma completa. Puede llegar a duplicar el tiempo de ejecución y si hablamos en el termino de días es un retraso enorme.

Nigthwatch

Rendimiento de Nigthwatch:

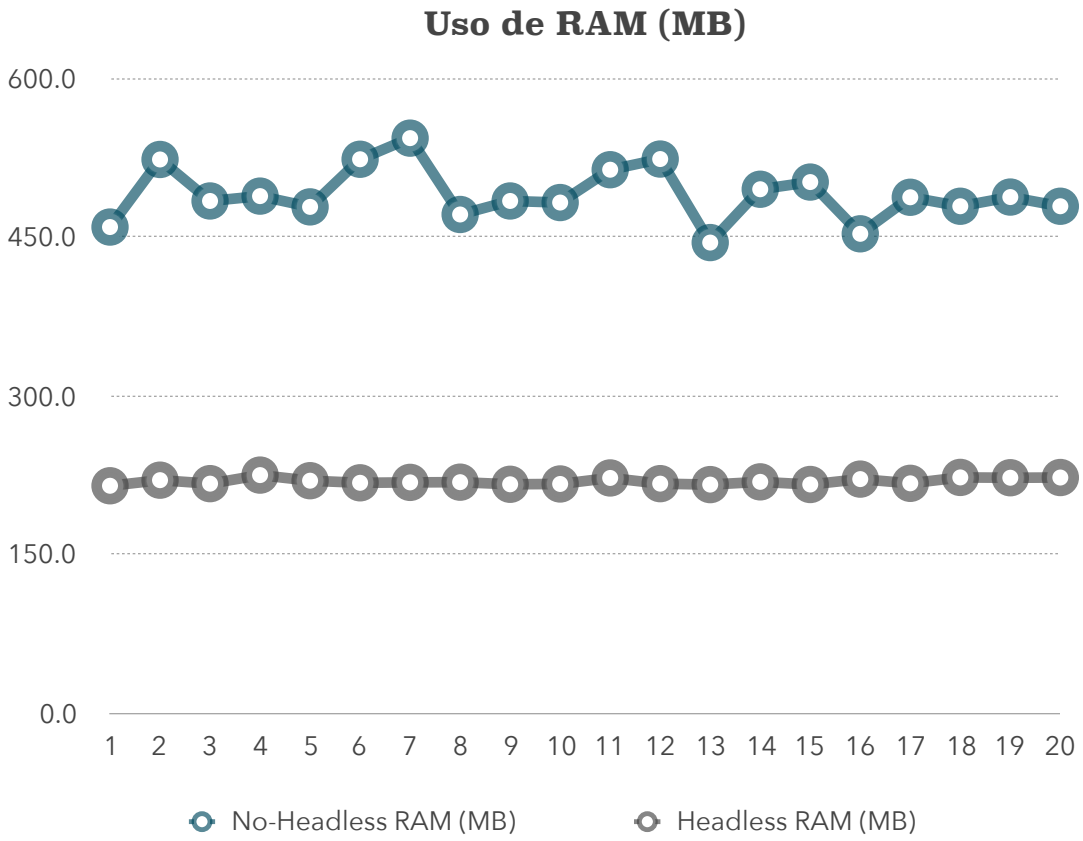
Se ejecutaron 20 veces las pruebas desarrolladas en Nigthwatch para la aplicación **losestudiantes.com**. Para cada ejecución se hará la prueba de forma **"headless"** y **"No-Headless"**. El objetivo fue medir el consumo de CPU, RAM y tiempo de ejecución.

Uso de CPU (%)



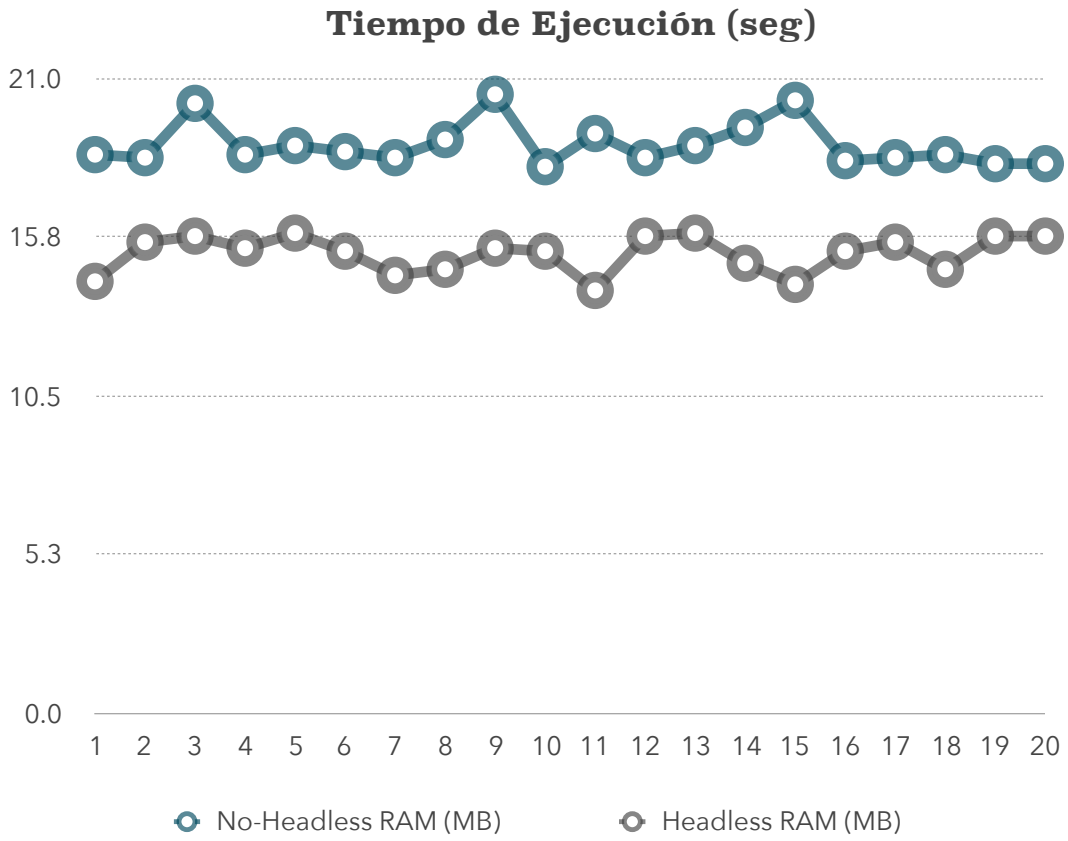
En promedio, Nigthwatch consume 15,1 % de CPU si se ejecuta de forma NoHeadless y 13,1 % si se ejecuta de forma Headless; sin embargo, como lo demuestra la gráfica ese consumo es muy inestable y varia bastante alcanzando el 21 % de carga al CPU o descendiendo hasta el 10%.

Consumo de Memoria RAM (MB)



Nigthwatch consume en promedio 490 MB de RAM si se ejecuta de forma NoHeadless y tan solo 219 MB si se ejecuta de forma Headless, no solo hay una mejora significativa, sino además el comportamiento Headless es mucho más estable y predecible. Lo que ocurre es que al no ser Headless, el browser que se levanta consume RAM de forma aleatoria por distintos procesos adicionales que debe soportar.

Tiempo de ejecución (Seg)



En este item, la diferencia es clara, aunque ambos tiempos de ejecución son estables, en promedio de forma NoHeadless se tarda 19 segundos en terminar las pruebas, mientras que de forma Headless le toma 15 segundos. En gran medida se debe a la guarda **waitForVisible** usada en el código de la prueba que obliga a repintar en la pantalla los elementos y no deja avanzar la prueba hasta que estos este visibles. De la forma Headless no tenemos ese problema.

Conclusión

Sin lugar a dudas, Nigthwatch consume muchos menos recursos si se ejecuta de forma headless, ahorrando 2% de CPU y hasta 271 MB de memoria RAM, así que se recomienda usarlo de esta forma.

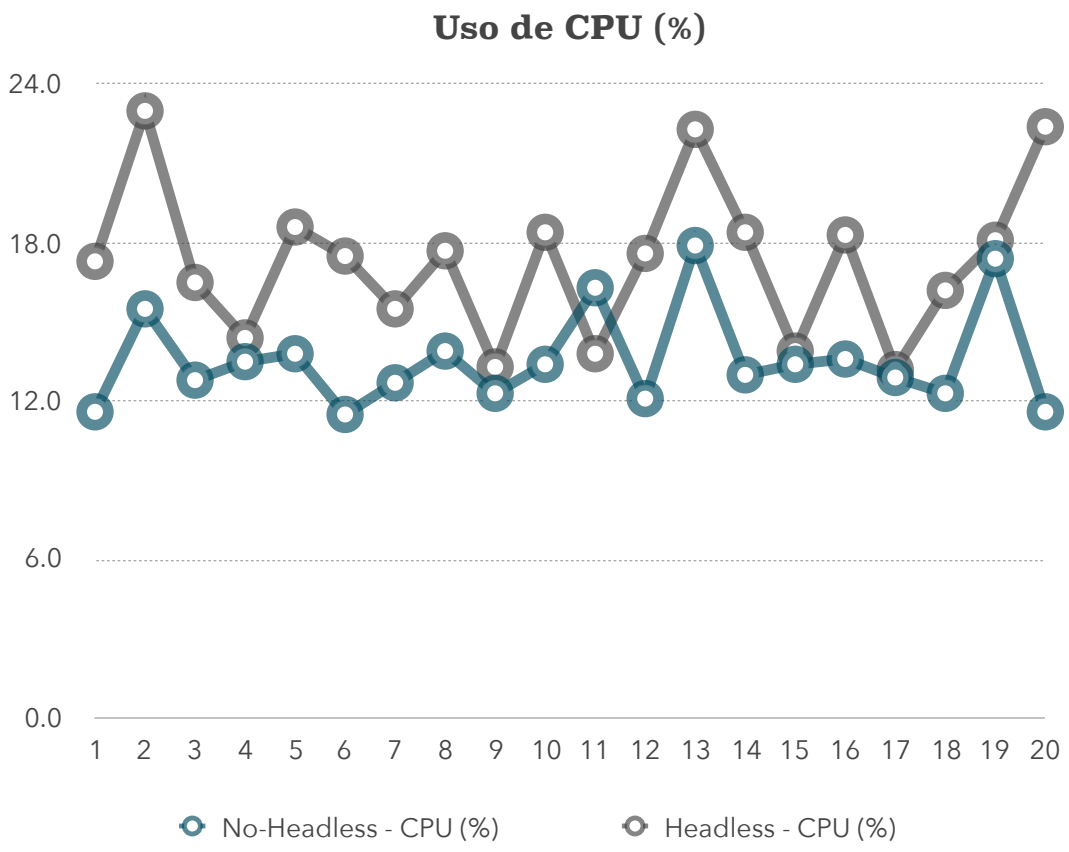
Sin embargo no es idempotente y la misma prueba puede fallar si los selectores que se usaron para llegar hasta los elementos fueron basados en clases o identificadores. Debe usarse selectores basados en XPath.

Cypress

Rendimiento de WebDriver IO:

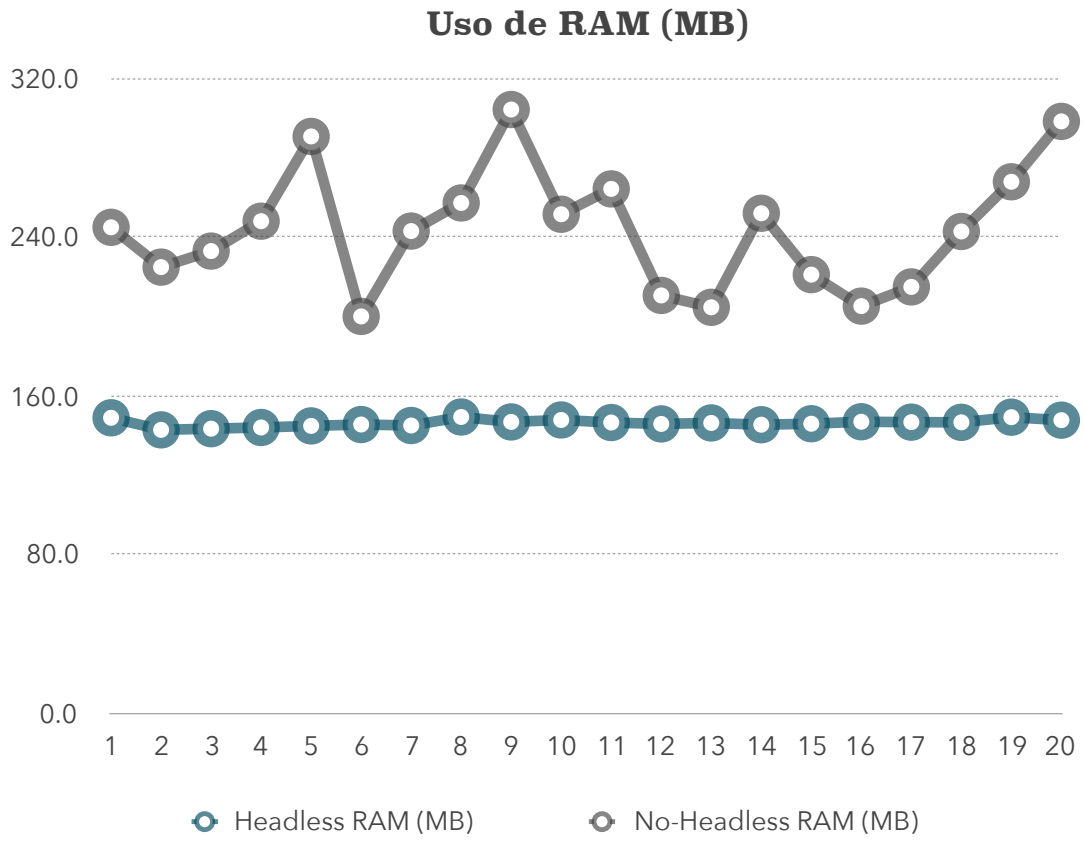
Se ejecutaron 20 veces las pruebas desarrolladas en Cypress para la aplicación **losestudiantes.com**. Para cada ejecución se hará la prueba de forma **"headless"** y **"No-Headless"**. El objetivo fue medir el consumo de CPU, RAM y tiempo de ejecución.

Uso de CPU (%)



Cypress en promedio consume 13,6 % de CPU si se ejecuta desde la interfaz gráfica, y 17.3 % si se ejecuta de forma Headless, la diferencia de 3,7 % esta en la recopilación de vídeos y fotos que hace Cypress cuando se ejecuta Headless.

Consumo de Memoria RAM (MB)

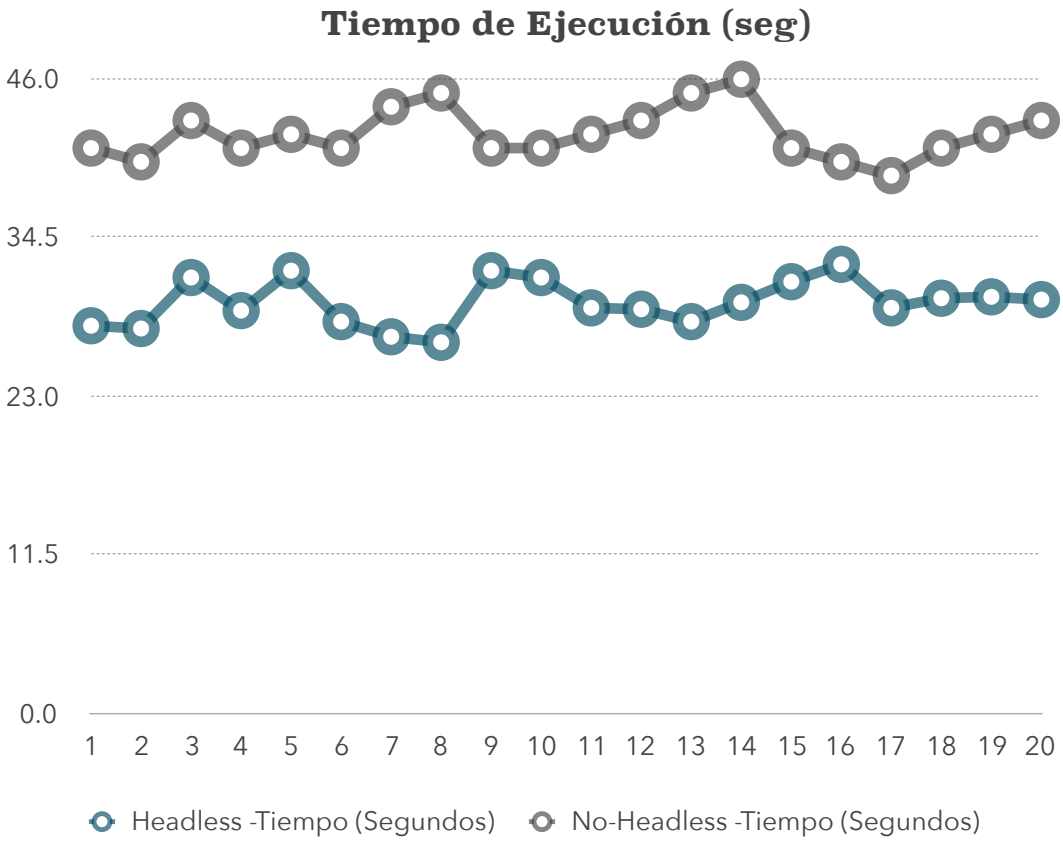


Medir el consumo de memoria RAM con cypress para la parte Headless fue complicada debido a que la recopilación de vídeo y screenshots se ejecutan en un momento posterior a la prueba y en otro hilo de ejecución. Sin embargo se logro obtener esta medida aproximada. Donde el consumo de RAM es muy superior de forma headless, llegando a los 244 MB en promedio versus los 146 MB que usa la interfaz gráfica.

Esto se debe a que cuando Cypress inicia su interfaz gráfica el consumo de RAM se estabiliza y cualquier ejecución posterior solo incrementa un poco el consumo. Por otro lado de forma Headless es totalmente imprevisible, ya que si la ejecución falla o algo anormal sucede, Cypress toma un screenshot y al final recopila el vídeo.

Adicionalmente cypress tiene configuraciones para evitar grabar el vídeo y tomar los screenshots, que reducen mucho el consumo de CPU y Memoria, aunque no fueron tenidos en cuenta para esta prueba es importante aclarar que el mal rendimiento es debido a estos procesos adicionales.

Tiempo de ejecución (Seg)



En los tiempos de ejecución si hay una diferencia significativa de aproximadamente 12 segundos, aunque ambas medidas tienen un comportamiento estable, Cypress toma mucho más tiempo ejecutando las pruebas de forma Headless y no es debido a la captura del vídeo o screenshots, ya que estos se hacen después de la ejecución de pruebas. Realmente lo que ocurre es que Cypress debe levantar su propio motor de renderizado basado en electron; este proceso se hace solo 1 vez cuando se abre la interfaz gráfica, pero Headless se hace por cada ejecución.

Conclusión

Indudablemente Cypress consume muchos más recursos si se ejecuta de forma headless, bien sea de CPU, memoria RAM o tiempo, aunque agrega un valor adicional que es el vídeo y el screenshot de fallos que no ofrecen otras herramientas.

Usar Cypress de modo headless aunque es más costoso agrega más valor a la documentación y evidencia de la ejecución de pruebas.

Anexos

Url Repositorios

Taller 3
https://github.com/daprieto1/MISO-4208-Workshops/tree/master/workshop-3

Taller 2
https://github.com/daprieto1/MISO-4208-Workshops/tree/master/workshop-2

Taller 1
https://github.com/daprieto1/MISO-4208-Workshops/tree/master/workshop-1

Datos Protractor

	No-Headless			Headless		
Ejecución	CPU (%)	RAM (MB)	Tiempo (seg)	CPU (%)	RAM (MB)	Tiempo (seg)
1	8.9	54.0	17	4.3	33.7	17
2	8.6	45.0	17	6.9	38.8	17
3	7.3	51.0	17	5.0	40.2	17
4	7.4	51.5	17	5.3	44.3	17
5	9.8	51.9	17	3.8	40.4	17
6	5.2	51.7	17	4.7	42.3	17
7	9.3	51.0	17	5.4	35.0	17
8	4.2	53.6	17	5.6	38.6	17
9	9.6	51.3	17	4.9	39.9	17
10	6.8	51.1	17	4.3	40.3	17
11	9.1	51.4	17	5.0	44.1	17
12	9.5	47.0	17	4.9	44.3	17
13	8.8	50.6	17	5.4	42.2	17
14	9.4	51.7	17	5.5	35.2	17
15	6.7	51.1	17	5.9	38.8	17
16	8.5	52.9	17	5.0	40.1	17
17	9.2	51.4	17	4.1	40.6	17
18	8.8	47.2	17	5.6	44.5	17
19	9.3	50.5	17	4.4	42.7	17
20	8.4	50.7	17	6.0	35.0	17
MAX	9.8	54.0	17.0	6.9	44.5	17.0
MIN	4.2	45.0	17.0	3.8	33.7	17.0
AVG	8.2	50.8	17.0	5.1	40.1	17.0

Datos WebDriver IO

Ejecución	CPU (%)	RAM (MB)	Tiempo (seg)	CPU (%)	RAM (MB)	Tiempo (seg)
1	21.8	253.8	13	15.7	242.1	17
2	28.6	266.6	24	14.9	230.2	13
3	24.4	258.5	16	22.4	240.7	18
4	19.0	261.3	17	8.7	243.2	18
5	25.0	263.5	14	24.6	239.2	18
6	28.6	255.2	13	22.7	182.7	15
7	21.3	243.6	20	17.4	231.6	15
8	17.6	264.7	16	16.9	250.0	13
9	20.4	257.6	12	25.3	251.4	16
10	27.1	256.0	14	30.1	251.0	15
11	24.0	254.8	14	27.4	245.2	14
12	13.7	257.8	14	19.5	250.4	18
13	7.4	248.8	14	17.0	255.4	18
14	39.4	258.4	19	80.3	230.8	18
15	28.9	253.9	24	20.8	244.8	14
16	30.9	274.4	13	24.7	244.2	13
17	19.5	256.2	12	25.7	243.3	16
18	13.6	274.8	14	20.7	238.7	18
19	7.3	277.8	15	17.9	254.9	17
20	28.1	243.7	16	27.0	259.3	16
MAX	39.4	277.8	24.0	80.3	259.3	18.0
MIN	7.3	243.6	12.0	8.7	182.7	13.0
AVG	22.3	259.1	15.7	24.0	241.5	16.0

Datos Nigthwatch

No-Headless				Headless		
Ejecución	CPU (%)	RAM (MB)	Tiempo (seg)	CPU (%)	RAM (MB)	Tiempo (seg)
1	17.3	460.0	18.5	13.2	215.0	14.3
2	13.9	524.0	18.4	18.0	220.3	15.6
3	16.1	484.7	20.2	11.8	217.0	15.8
4	15.9	489.3	18.5	10.5	225.1	15.4
5	14.0	479.1	18.8	12.9	219.8	15.9
6	14.4	524.2	18.6	11.6	217.8	15.3
7	15.7	544.2	18.4	10.6	218.2	14.5
8	15.2	471.9	19.0	15.7	218.4	14.7
9	17.3	484.7	20.5	11.5	216.4	15.4
10	14.4	483.2	18.1	10.0	216.7	15.3
11	15.1	514.3	19.2	12.5	222.4	14.0
12	20.9	524.4	18.4	11.7	217.0	15.8
13	12.3	445.3	18.8	16.6	216.1	15.9
14	13.3	495.8	19.4	15.4	218.7	14.9
15	15.3	502.6	20.3	13.8	216.2	14.2
16	14.9	453.6	18.3	12.4	221.3	15.3
17	14.5	488.2	18.4	18.7	217.3	15.6
18	16.5	479.7	18.5	11.2	222.9	14.7
19	14.4	488.4	18.2	12.6	222.6	15.8
20	11.3	479.7	18.2	11.4	222.8	15.8
MAX	20.9	544.2	20.5	18.7	225.1	15.9
MIN	11.3	445.3	18.1	10.0	215.0	14.0
AVG	15.1	490.9	18.8	13.1	219.1	15.2

Datos Cypress

No-Headless				Headless		
Ejecución	CPU (%)	RAM (MB)	Tiempo (seg)	CPU (%)	RAM (MB)	Tiempo (seg)
1	11.6	149.0	28.1	17.3	245.2	41
2	15.5	142.9	27.9	23.0	225.1	40
3	12.8	143.5	31.6	16.5	233.2	43
4	13.5	144.1	29.2	14.4	248.3	41
5	13.8	144.9	32.1	18.6	291.1	42
6	11.5	145.6	28.4	17.5	200.2	41
7	12.7	145.1	27.3	15.5	243.3	44
8	13.9	149.4	26.9	17.7	257.5	45
9	12.3	146.9	32.1	13.3	304.7	41
10	13.4	147.9	31.6	18.4	251.8	41
11	16.3	146.6	29.4	13.8	264.6	42
12	12.1	145.9	29.3	17.6	210.8	43
13	17.9	146.4	28.4	22.3	204.8	45
14	13.0	145.5	29.8	18.4	252.3	46
15	13.4	146.0	31.3	13.9	221.3	41
16	13.6	147.0	32.6	18.3	205.4	40
17	12.9	146.8	29.4	13.2	215.1	39
18	12.3	146.7	30.1	16.2	243.1	41
19	17.4	149.3	30.2	18.1	268.2	42
20	11.6	147.8	30.0	22.4	298.7	43
MAX	17.9	149.4	32.6	23.0	304.7	46.0
MIN	11.5	142.9	26.9	13.2	200.2	39.0
AVG	13.6	146.4	29.8	17.3	244.2	42.1

Protractor Ejecución Paralela

Yo adapte las pruebas de Protractor para la ejecución en paralelo, la razón es que por ahora es el framework más estable de los 4 que hemos trabajado. A continuación se mencionan los pasos para ejecutar protractor en paralelo.

Pasos:

1. Instalar **parallel-protractor**
2. Crear el archivo conf.js como esta especificado en la documentación de la herramienta
3. Con el comando **node -r parallel-protractor node_modules/.bin/protractor** se ejecutan las pruebas en paralelo.

El funcionamiento de esta librería es muy sencillo, por cada bloque **describe** la herramienta simula un archivo **spec** completamente diferente, lo que ocurre es que protractor por defecto ejecuta archivos deparados de forma paralela; así que esta simulación permite dividir un solo archivo y ejecutarlo de forma paralela reuniendo los tiempos de ejecución.

El desarrollador debe ser consciente que si solo cuenta con 1 bloque **describe** en cada **spec** la herramienta no alterará el resultado.

Las pruebas secuencia les y headless con Protractor tomaron 17 segundos en ejecutarse, usando la herramienta **parallel-protractor**, ese tiempo se redijo a 5,352 segundos, que es el tiempo que le toma al mayor **describe** en ser ejecutado.