Trabajo Práctico Grupal Nº1
75.73
Arquitectura de Software

Grupo *"La Covideira"*

2do Cuatrimestre 2021

Docentes

Christian Calónico - Guillermo Rugilo - Mariano D'Ascanio

Alumnos

Nombre	Padrón
Fernández Theillet, Nicolás Pablo	88599
Pagani, Maximiliano	94754
Alvarez, Nicolás	93503
Orive, Ma. Sol	91351

1. Introducción

1.1 Objetivo

El objetivo principal del presente trabajo práctico es comparar algunas tecnologías, ver cómo diversos casos impactan en los atributos de calidad y probar o al menos sugerir qué cambios se podrían hacer para mejorarlos. En menor lugar, aprender a usar una variedad de tecnologías útiles y muy usadas actualmente.

1.2 Consigna

Implementar un servicio HTTP en Node.js-Express. Someter distintos tipos de endpoints a diversas intensidades/escenarios de carga en algunas configuraciones de deployment, tomar mediciones y analizar resultados.

1.3 Consideraciones

Algunas consideraciones respecto a lo que esperamos, a priori, encontrar y/o comprobar durante el trabajo en cuestión:

- El servicio sincrónico debería presentar dificultades para absorber un número creciente de carga o bien mucha carga en simultáneo.
- El servicio asincrónico debería escalar en mejor medida respecto al servicio sincrónico.
- Escalar por N nodos/instancias replicadas del servidor de la app Node debería aumentar la capacidad total del sistema para atender requests en un factor cercano a N respecto a un solo nodo/instancia.

2. Escenarios

2.1.Condiciones de los escenarios

Cuando se alcanza el 80% del uso del CPU, Artillery devuelve un warning[1] indicando que a partir de ese punto los resultados son imprecisos. Es por esto que los escenarios han sido armados de forma tal de no alcanzar dicho punto.

2.2.Escenarios

Cada uno de los siguientes escenarios será ejecutado y comparado teniendo en cuenta los distintos modo de deployment y los endpoints pertinentes, a saber:

Modos de deployment:

- Un nodo:
  Un solo proceso, un solo container.
- Replicado:
  Replicado en múltiples containers, con load balancing a nivel de nginx. En nuestras pruebas utilizaremos un aumento en la escala a 3 containers de la app de Node.

Endpoints:

- Ping :
  - Respuesta de un valor constante (rápido y de procesamiento mínimo)
  - Healthcheck básico
- Proxy sincrónico:
  - Invocación a servicio sincrónico provisto por la cátedra.
  - Aproximación al consumo de servicio real sincrónico.
- Proxy asincrónico:
  - Invocación a servicio asincrónico provisto por la cátedra.
  - Aproximación al consumo de servicio real asincrónico.
- Intensivo:
  - Loop de cierto tiempo (lento y de alto procesamiento)
  - Cálculos pesados sobre los datos (ej: algoritmos pesados, o simplemente muchos cálculos).

Escenario 1 (Plain- X request en 1 segundo) [plain-x-request.yaml]

phases:
- **name**: Plain-x-request
- **duration**: 1
- **arrivalCount**: [30 - 1000]

[1] https://artillery.io/docs/faq/#high-cpu-warnings

X request en 1 segundo.

La idea de este escenario es probar con distintos request en 1 segundo para poder ver si se saturan los distintos servicios(endpoints) así como también poder obtener aproximadamente cuanto tiempo tarda en responder cada servicio.

Escenario 2 (Constant) [constant_arrival_rate.yaml]

phases:
- **name**: Constant
- **duration**: 300
- **arrivalRate**: 30

X (arrivalRate) usuarios en simultáneo por segundo, haciendo request por Y (duration) segundos.

Nos vamos a dar una idea de cómo maneja la carga y los recursos cada uno de los endpoints a medida que pasa el tiempo con una cantidad fija de usuarios. Podríamos detectar de qué tipo se trata c/u de los endpoints

Escenario 3 (Rampa) [ramp-request.yaml]

phases:
- **duration**: 90
- **arrivalRate**: 1
- **rampTo**: 20

Crece de a Y usuarios en simultáneo hasta llegar al ramp Z.

Nos vamos a dar una idea de cómo maneja la carga y los recursos cada uno de los endpoints de manera escalada, observando cual es el límite a medida que avanza el tiempo y la cantidad de requests.

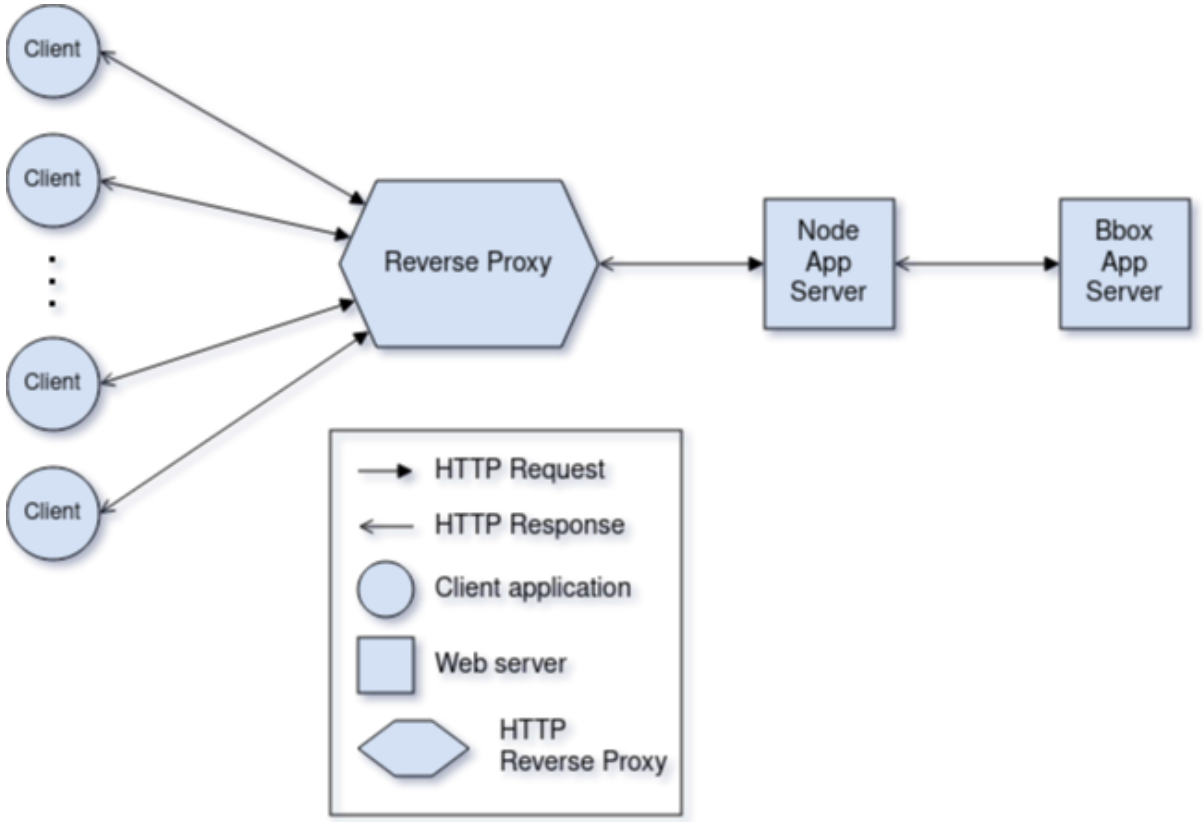
Nota Bene:

*Para que el **gráfico vuelva a cero**, en las pruebas a los escenarios se le agrega la siguiente phase:*

```
- name: Zero
  duration: 10
  arrivalRate: 0
```

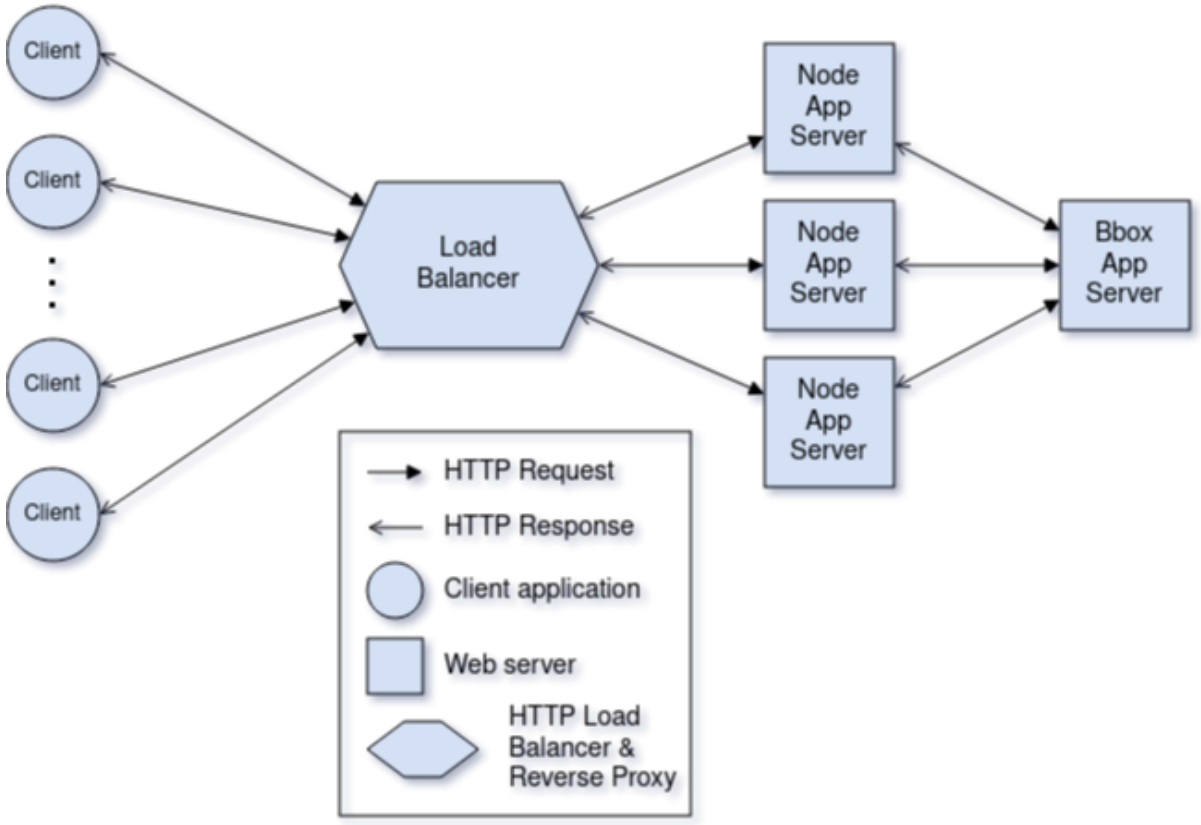
3. Vista de componentes y conectores

Caso con sólo un nodo o instancia del servidor Node.js



El proxy intermedio implementado con Nginx en este caso actúa sólo como un reverse proxy puesto que interactúa con un único nodo o instancia del servidor Node.

Caso con múltiples nodos o instancias replicadas del servidor Node.js



Al contar con múltiples nodos replicados de nuestro servidor Node, en este caso el proxy intermedio implementado con Nginx, además de funcionar como reverse proxy que interactúa con los servidores en nombre del cliente, también realiza la función de balancear la carga entrante entre los distintos servidores disponibles.

4. Análisis y caracterización

4.1. Sincrónico / Asincrónico

Para detectar cuál de los dos servicios es sincrónico y cuál es asincrónico, correremos un escenario de prueba de tipo *ramp* [bbox-sync-test.yaml], en el que se irán aumentando lenta pero progresivamente la cantidad de requests que se van realizando cada segundo hacia cada servicio. De esta forma podremos evaluar la respuesta ante una carga creciente. En el caso del servicio sincrónico, esperamos detectarlo observando un aumento repentino de requests pendientes o con errores en el punto en que la capacidad de procesamiento sincrónica comience a ser superada por la tasa de concurrencia de clientes. En el caso del asincrónico, esto no sucedería, o al menos no tan pronto como en el sincrónico.

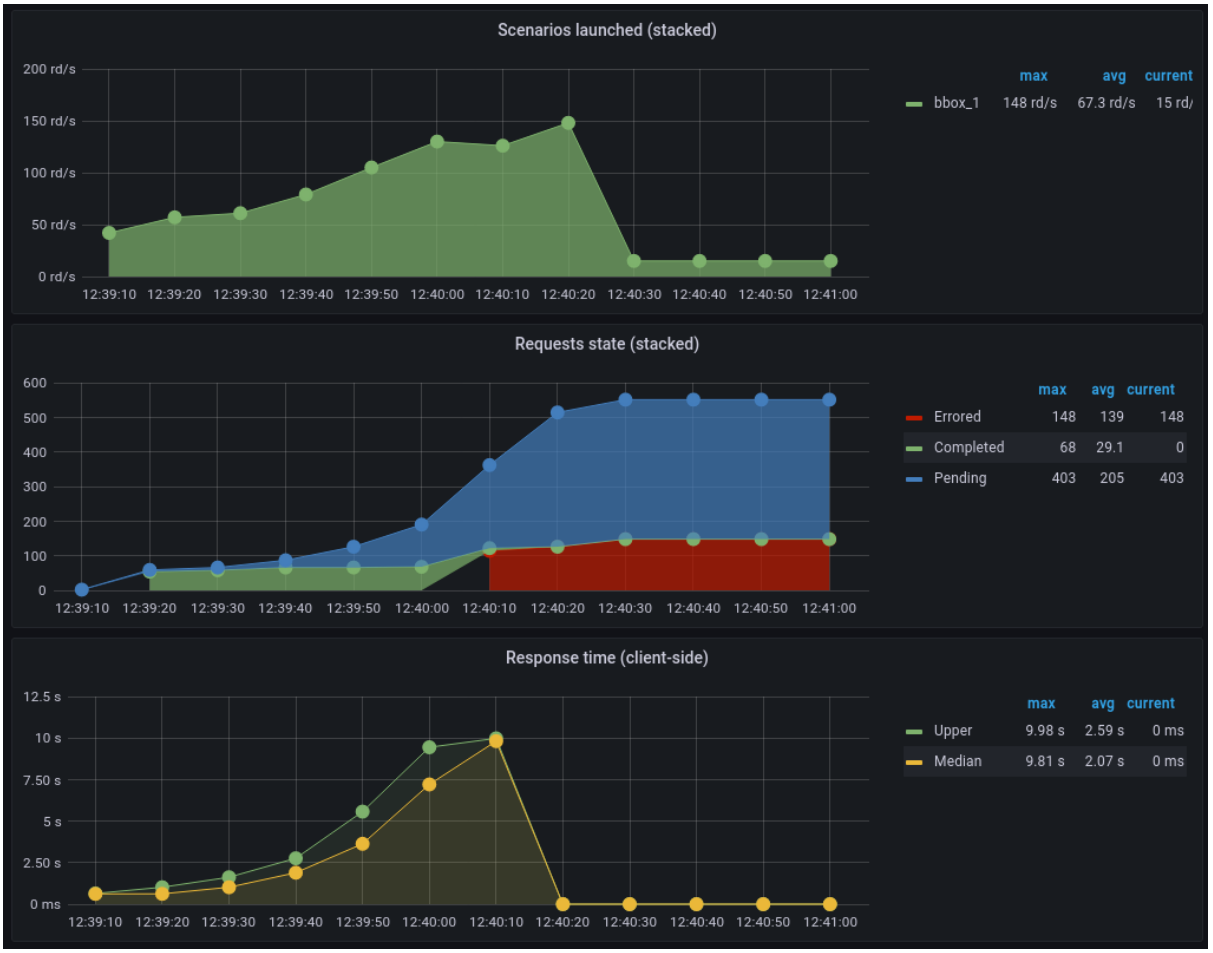
Entonces, con una etapa de:

[bbox-sync-test.yaml]
name: Ramp
duration: 120
arrivalRate: 1
rampTo: 20

Vamos incrementando la carga desde 1 request por segundo hasta 20 requests concurrentes por segundo durante 120 segundos, y obtenemos que:

- Servicio 1 (puerto 9090 en el servidor bbox) en http://localhost:5555/bbox/1

```
All virtual users finished
Summary report @ 12:40:33(-0300) 2021-10-20
  Scenarios launched:  801
  Scenarios completed: 398
  Requests completed:  398
  Mean response/sec: 7.28
  Response time (msec):
    min: 605
    max: 9979
    median: 1546.5
    p95: 8679.6
    p99: 9812.5
  Scenario counts:
    bbox 1: 801 (100%)
  Codes:
    200: 398
  Errors:
    ETIMEDOUT: 403
```



- Servicio 2 (puerto 9091 en el servidor bbox) en http://localhost:5555/bbox/2

```
All virtual users finished
Summary report @ 12:50:47(-0300) 2021-10-20
  Scenarios launched:  1255
  Scenarios completed: 1255
  Requests completed:  1255
  Mean response/sec: 9.64
  Response time (msec):
    min: 1053
    max: 1096
    median: 1059
    p95: 1064
    p99: 1070
  Scenario counts:
    bbox: 1255 (100%)
  Codes:
    200: 1255
```

Como podemos observar en los resultados, al aumentar la carga en el servicio 1, inicialmente el sistema se comportó correctamente, si bien algunas requests estaban pendientes y obtenían su respuesta con alguna demora. Sin embargo avanzados unos pocos segundos de esta etapa inicial, se observa que el sistema comienza a tener dificultades para procesar las peticiones, generando un crecimiento importante en el tiempo de respuesta por cliente y en la cantidad de peticiones pendientes y con errores, llegando finalmente a un punto de saturación en el que se evidencia un colapso y todas las requests (inclusive las pendientes) no son respondidas y obtienen un error.

En cambio, en los resultados del servicio 2, vemos que el sistema se comportó correctamente en toda la etapa de prueba, pudiendo responder todas las peticiones a medida que se iba aumentando la carga, sin ningún error y con tiempos de respuestas estables en toda la prueba.

De esta forma, podemos deducir que el servicio 1 en http://localhost:5555/bbox/1 corresponde al servicio sincrónico, ya que evidencia una limitación en el procesamiento concurrente provocada por el bloqueo existente durante el mismo (que le impide atender otras peticiones en ese lapso), y el servicio 2 en http://localhost:5555/bbox/2 corresponde al servicio asincrónico, ya que atender una request no provoca un bloqueo durante todo el tiempo de respuesta, pudiendo mientras tanto atender otras peticiones.

4.2. Cantidad de workers (servicio sincrónico)

Un sencillo método para la detección de workers es testear los tiempos de respuesta de requests concurrentes, que se irán incrementando. En el punto en que el tiempo de respuesta de un request comience a ser prácticamente el doble del esperado, tendremos un indicativo de que dicho request está en un estado de espera la mitad del tiempo dado que se llegó al punto en el que todos los workers del servicio están ocupados.

Como ya identificamos que el servicio http://localhost:5555/bbox/1 es el servicio sincrónico, las pruebas se efectuarán a dicho endpoint. Utilizaremos la herramienta ApacheBench de la siguiente manera:

ab -n X -c X http://localhost:5555/bbox/1

Se comienza de a un request concurrente, es decir X = 1, y en cada paso se incrementará en 1 la cantidad de requests enviadas y concurrentes:

X	Tiempos obtenidos
1	Connection Times (ms) min mean[+/-sd] median max Connect: 0 0 0.0 0 0 Processing: 609 609 0.0 609 609 Waiting: 609 609 0.0 609 609 Total: 609 609 0.0 609 609
2	Percentage of the requests served within a certain time (ms) 50% 609 66% 609 75% 609 80% 609 90% 609 95% 609 98% 609 99% 609 100% 609 (longest request)

3	Percentage of the requests served within a certain time (ms) 50% 604 66% 604 75% 605 80% 605 90% 605 95% 605 98% 605 99% 605 100% 605 (longest request)
4	Percentage of the requests served within a certain time (ms) 50% 612 66% 612 75% 614 80% 614 90% 614 95% 614 98% 614 99% 614 100% 614 (longest request)
5	Percentage of the requests served within a certain time (ms) 50% 608 66% 609 75% 609 80% 611 90% 611 95% 611 98% 611 99% 611 100% 611 (longest request)
6	Percentage of the requests served within a certain time (ms) 50% 619 66% 619 75% 620 80% 620 90% 1215 95% 1215 98% 1215 99% 1215 100% 1215 (longest request)

7	Percentage of the requests served within a certain time (ms) 50% 620 66% 621 75% 1216 80% 1216 90% 1217 95% 1217 98% 1217 99% 1217 100% 1217 (longest request)

Como podemos ver, a partir de X = 6 se introduce una request con un tiempo de respuesta de 1215 que impacta a partir del percentil 90 (1 en 6 es el 16,6%). Esto nos dice que todos los workers ya están ocupados atendiendo las primeras 5 requests, y la última es puesta en espera un tiempo cercano a los ~635 ms hasta que es atendida.

Con X = 7 se introduce una segunda request que tampoco puede ser procesada en los primeros ~620 ms, por lo que a partir del percentil 75 ya vemos ese impacto (2 en 7 es el 28,5%).

Dado los datos obtenidos, podemos concluir que en el caso del servicio sincrónico http://localhost:5555/bbox/1 la cantidad de workers procesando requests es de 5. Al enviar concurrentemente un número mayor a dicha cantidad, la respuesta de N - 5 requests será como mínimo el doble del tiempo demorado por un request en condiciones normales que fuese atendida inmediatamente por algún worker.

4.3. Demora en responder

Vamos a establecer como hipótesis, que el tiempo de demora en responder es aquel que corresponde al escenario ideal, es decir, sin carga extra ni congestión; el servidor en un mismo momento sólo se dedica la atención y procesamiento de la request enviada.

Para detectar la demora de los distintos endpoints utilizamos curl y ab.

Usando curl:

curl -o /dev/null -s -w 'Tiempo total request 1: %{time_total}\n' <url>

En este caso es un valor poco confiable ya que se trata de una sola prueba para cada servicio.

Caso	Un Nodo	Múltiples nodos (3)

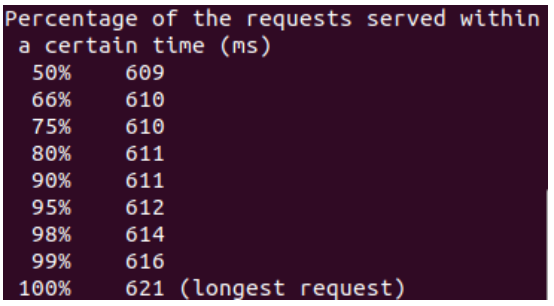
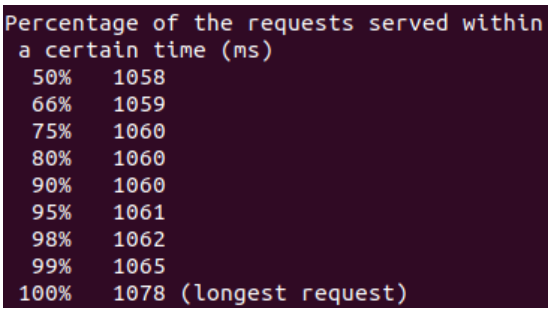
Ping	Curl: **0.017432 s**	Curl: **0.010413 s**
Proxy sincrónico	Curl: **0.690412 s**	Curl: **0.644440 s**
Proxy asincrónico	Curl: **1.086559 s**	Curl: **1.081083 s**
Intensivo	Curl: **5.068637 s**	Curl: **5.007982 s**

Usando ApacheBench:

Para tener una medida más confiable y representativa del tiempo de demora en responder de cada servicio, realizaremos pruebas con mayor cantidad de requests (500), y tomaremos el percentil p_{95}.

Se prueba de a un request a la vez (-c 1) y con un solo nodo de servidor. Esto nos dará el tiempo de respuesta en condiciones ideales, en las cuales no hay congestión ni tiempos de espera. Cada request es atendida inmediatamente.

ab -n 500 -c 1 http://localhost:5555/bbox/1
ab -n 500 -c 1 http://localhost:5555/bbox/2

Servicio	**p_{95} (ms)**	**Resultados en 500 requests**
http://localhost:5555/bbox/1 Proxy sincrónico	**612 ms**	
http://localhost:5555/bbox/2 Proxy asincrónico	**1061 ms**	

4.4. Escenarios: comparación entre pruebas

4.4.1 Escenario 1

Resumen

Se puede observar que en las pruebas obtuvimos que bbox/1 tiene un cierta capacidad limitada para manejar request en 1 segundo. En cambio bbox/2 respondió todos los request propuestos.

Podríamos pensar que bbox/1 se estaría manejando de manera sincrónica dado la cantidad de request sin responder, incluso en un número tan pequeño de request.

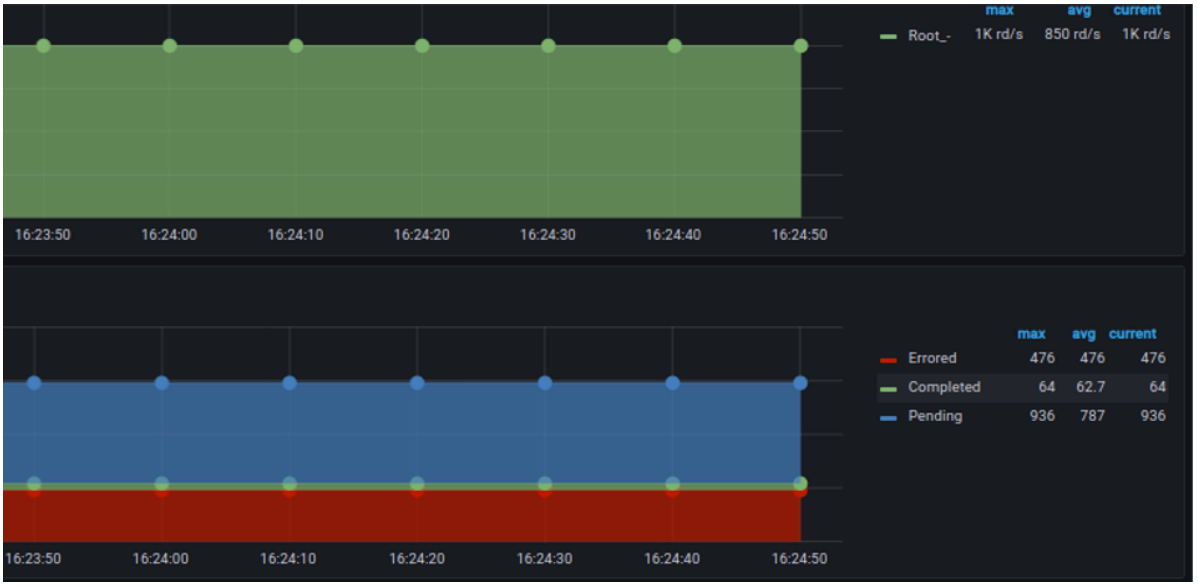
3 nodos:

1000 request:

Se puede observar que en esta cantidad de request bbox/1 empieza a perder un gran porcentaje.

En el caso de bbox/2 responde a todos los request.

bbox/1



bbox/2


```
All virtual users finished
Summary report @ 17:31:09(-0300) 2021-10-18
  Scenarios launched:  1000
  Scenarios completed: 1000
  Requests completed:  1000
  Mean response/sec: 39.95
  Response time (msec):
    min: 1047
    max: 1837
    median: 1143
    p95: 1540.5
    p99: 1725
  Scenario counts:
    Root (/): 1000 (100%)
  Codes:
    200: 1000
```

100 request:

Se puede observar que en esta cantidad de request bbox/1 pierde al menos la mitad de los request. En este caso quedaron como pending en grafana pero nunca fueron resueltos.

- **bbox/1**

- **bbox/2**

50 request:

En ambos casos ambos, bbox/1 y bbox/2 responden bien a esa cantidad de request

<u>1 nodo</u>

En el caso de 1 nodo, los resultados son similares a 3 nodos, es decir que bbox/1 empieza a dejar de responder request ya a los 100 request.

4.4.2 Escenario 2

Arrival rate : 5 (5 request por segundo, que son 50 request en cada snapshot de métricas)

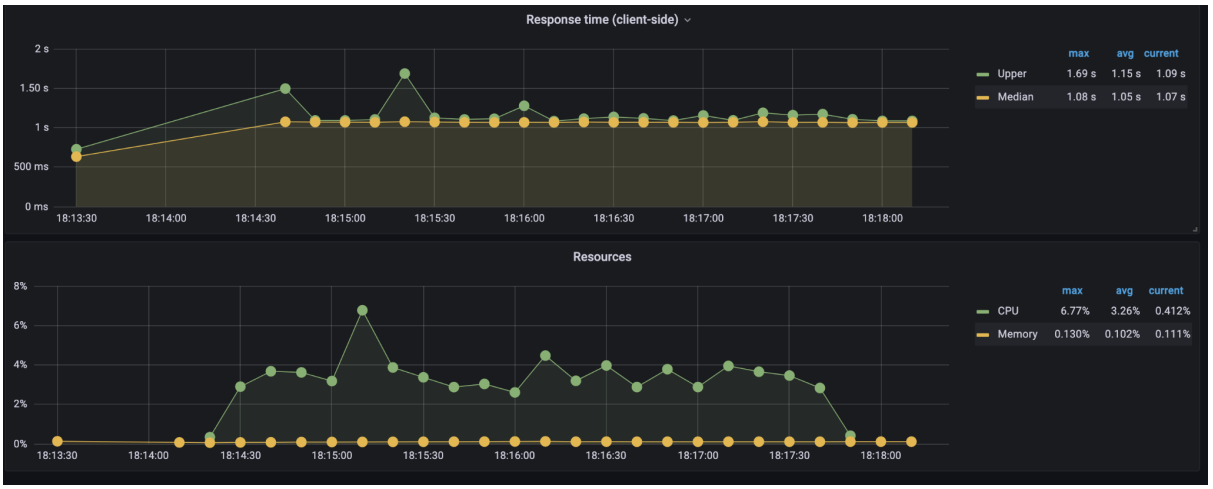
1 nodo - /bbox/1

Bajo esta carga de entrada se ve que el endpoint no tiene problemas para manejar los request, todos son procesados dentro del tiempo esperado e incluso los recursos no se ven estresados siendo CPU con máximo del 9%.


```
Summary report @ 18:08:39(-0300) 2021-10-19
  Scenarios launched:  1000
  Scenarios completed: 1000
  Requests completed:  1000
  Mean response/sec: 4.76
  Response time (msec):
    min: 607
    max: 1141
    median: 618
    p95: 690
    p99: 835.5
  Scenario counts:
    Root (/): 1000 (100%)
  Codes:
    200: 1000
```

1 nodo - /bbox/2

Bajo las mismas condiciones que en el endpoint anterior, se observa la misma situación general de todos los parámetros. También podemos ver que este proceso lleva más tiempo que el proceso anterior siendo el p99 de 1100 ms en comparación del endpoint anterior que tardaba el p99 800 ms.

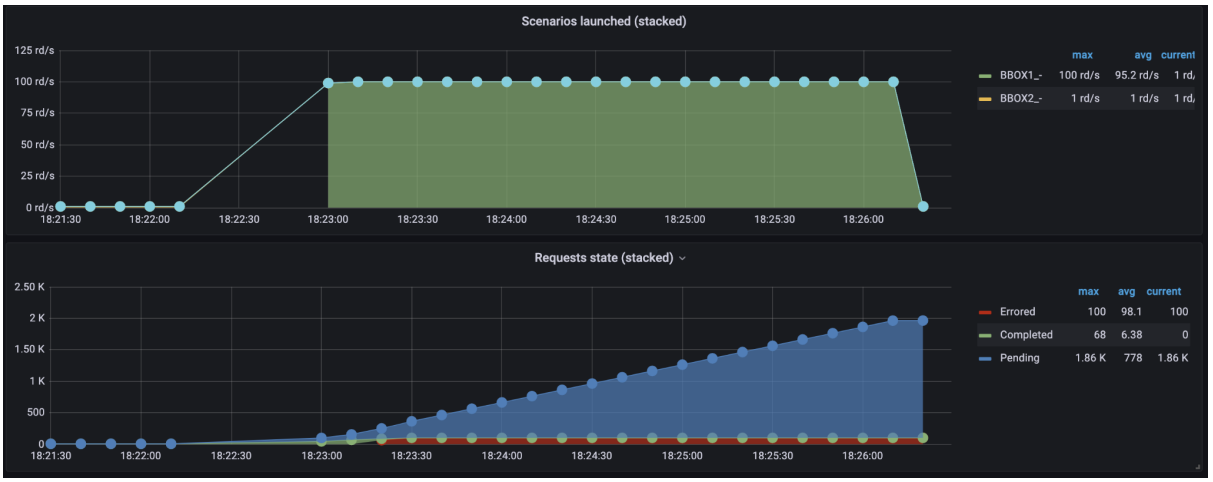


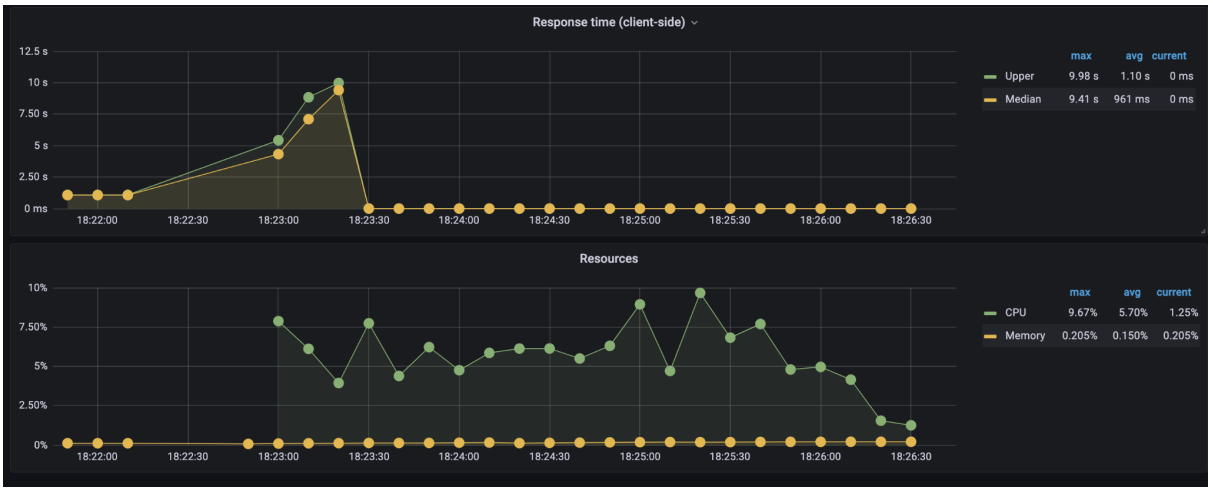

```
Summary report @ 18:17:58(-0300) 2021-10-19
  Scenarios launched:  1000
  Scenarios completed: 1000
  Requests completed:  1000
  Mean response/sec: 4.75
  Response time (msec):
    min: 1058
    max: 1597
    median: 1065
    p95: 1088
    p99: 1184.5
  Scenario counts:
    BBOX2 (/): 1000 (100%)
  Codes:
    200: 1000
```

Arrival rate : 10

1 nodo - /bbox/1

En este caso podemos ver cómo se empieza a degradar la performance. La mayoría de los request no pueden ser procesados en el tiempo y se empiezan a acumular.
En el tercer gráfico se puede ver cómo el tiempo de respuesta es cero ya que no llegan a responderse los request que entran y terminan saliendo por timeout.





Resumen de la ejecución:

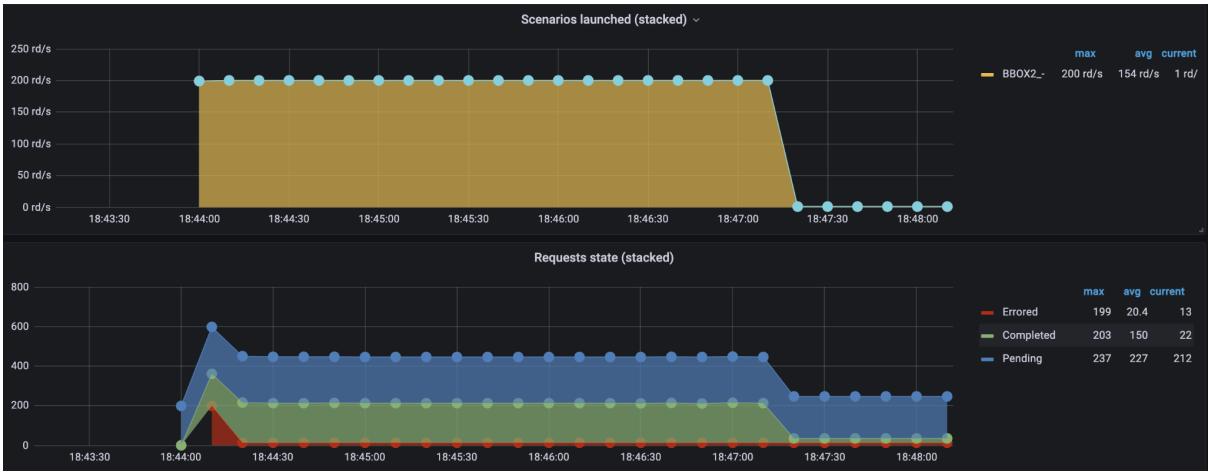
```
Summary report @ 18:26:23(-0300) 2021-10-19
  Scenarios launched:  2000
  Scenarios completed: 136
  Requests completed:  136
  Mean response/sec: 9.51
  Response time (msec):
    min: 3089
    max: 9978
    median: 6605
    p95: 9661.9
    p99: 9947.9
  Scenario counts:
    BBOX1 (/): 2000 (100%)
  Codes:
    200: 136
  Errors:
    ETIMEDOUT: 1864
```

1 nodo - /bbox/2

Este endpoint performa mucho mejor que el anterior ya que llegando a 100 request, se procesan todos mientras que en el caso anterior se perdían. De hecho se puede observar que el tiempo que tardan en procesarse en promedio los request, no difere mucho del escenario anterior con la mitad de req/s.

```
Summary report @ 18:36:47(-0300) 2021-10-19
  Scenarios launched:  2000
  Scenarios completed: 2000
  Requests completed:  2000
  Mean response/sec: 9.51
  Response time (msec):
    min: 1057
    max: 2570
    median: 1070
    p95: 1313
    p99: 1840
  Scenario counts:
    BBOX2 (/): 2000 (100%)
  Codes:
    200: 2000
```

Con fines de estresar /bbox/2 corremos un nuevo escenario con arrival rate de 20, que representa el doble de lo corrido en el caso anterior.
Acá se puede ver que ya no todos los request son devueltos sino que arranca el timeout pero, a diferencia de /bbox/1 los request no quedan esperando sino que se mantiene constante en el tiempo.



Resumen:

```
All virtual users finished
Summary report @ 18:47:16(-0300) 2021-10-19
  Scenarios launched:  4000
  Scenarios completed: 3788
  Requests completed:  3788
  Mean response/sec: 19
  Response time (msec):
    min: 1056
    max: 2753
    median: 1073
    p95: 1465.2
    p99: 2031
  Scenario counts:
    BBOX2 (/): 4000 (100%)
  Codes:
    200: 3788
  Errors:
    ETIMEDOUT: 212
```

3 nodos /bbox/2

Agregar 3 nodos en node no hizo que este endpoint (/bbox/2) o /bbox/1 cambien su comportamiento de ninguna manera.

Resumen:

```
Summary report @ 18:57:22(-0300) 2021-10-19
  Scenarios launched:  4000
  Scenarios completed: 3772
  Requests completed:  3772
  Mean response/sec: 19
  Response time (msec):
    min: 1056
    max: 9883
    median: 1089
    p95: 4320.7
    p99: 8371.5
  Scenario counts:
    BBOX2 (/): 4000 (100%)
  Codes:
    200: 3772
  Errors:
    ETIMEDOUT: 228
```

4.4.3 Escenario 3

Una única instancia Node.js

- Endpoint /heavy

  Dada la naturaleza del endpoint (bloqueante, simula una operación pesada en carga del cpu durante 5 segundos), se cambiaron los valores del escenario descritos en la sección 2.2 a los siguientes:

  name: Ramp
  duration: 30
  arrivalRate: 1
  rampTo: 2

  Resultados:

```
All virtual users finished
Summary report @ 16:55:11(-0300) 2021-10-20
  Scenarios launched:  47
  Scenarios completed: 2
  Requests completed:  2
  Mean response/sec: 1.15
  Response time (msec):
    min: 5011
    max: 9548
    median: 7279.5
    p95: 9548
    p99: 9548
  Scenario counts:
    Root (/): 47 (100%)
  Codes:
    200: 2
  Errors:
    ETIMEDOUT: 45
```

Como podemos ver sólo soporta un request cada 5 segundos. En la prueba sólo pudo procesar 2 request, la primera (5 s de procesamiento) y la segunda (5 s de espera + 5 s de procesamiento). Todas las demás requests no pudieron ser procesadas y se devolvieron con error.

- Endpoint /bbox/1 (servicio sincrónico)

  Resultados:

```
All virtual users finished
Summary report @ 16:09:48(-0300) 2021-10-20
  Scenarios launched:  960
  Scenarios completed: 323
  Requests completed:  323
  Mean response/sec: 9.52
  Response time (msec):
    min: 606
    max: 9990
    median: 1815
    p95: 8902.8
    p99: 9847.3
  Scenario counts:
    Root (/): 960 (100%)
  Codes:
    200: 323
  Errors:
    ETIMEDOUT: 637
```

Al ser un servicio sincrónico con una cantidad limitada de workers (5), podemos ver que luego de cierto punto de aumento de carga, el sistema comienza a atrasarse en las respuestas a las peticiones, finalizando en un colapso al poco tiempo en el que ninguna petición es atendida correctamente.

- Endpoint /bbox/2 (servicio asincrónico)

Resultados:

```
All virtual users finished
Summary report @ 16:16:50(-0300) 2021-10-20
  Scenarios launched:  972
  Scenarios completed: 972
  Requests completed:  972
  Mean response/sec: 9.72
  Response time (msec):
    min: 1054
    max: 1215
    median: 1060
    p95: 1065
    p99: 1070
  Scenario counts:
    Root (/): 972 (100%)
  Codes:
    200: 972
```

Dado que observamos que la respuesta se mantiene estable y no se ve afectada la operación del servicio, aumentamos la pendiente de la rampa para probar con una carga más potente:

name: Ramp
duration: 120
arrivalRate: 1
rampTo: 90

Resultados:

```
All virtual users finished
Summary report @ 16:22:28(-0300) 2021-10-20
  Scenarios launched:  5578
  Scenarios completed: 5578
  Requests completed:  5578
  Mean response/sec: 42.71
  Response time (msec):
    min: 1052
    max: 1216
    median: 1058
    p95: 1062
    p99: 1068
  Scenario counts:
    Root (/): 5578 (100%)
  Codes:
    200: 5578
```

Como vemos que el sistema todavía se comporta de forma correcta y estable, aumentamos de manera agresiva la pendiente de la rampa, generando un aumento significativo de la carga recibida durante 2 minutos hasta 500 requests por segundo:

name: Ramp
duration: 120
arrivalRate: 1
rampTo: 500

Resultados:

```
All virtual users finished
Summary report @ 16:27:02(-0300) 2021-10-20
  Scenarios launched:  30380
  Scenarios completed: 30380
  Requests completed:  30380
  Mean response/sec: 232.01
  Response time (msec):
    min: 1051
    max: 1213
    median: 1056
    p95: 1067
    p99: 1082
  Scenario counts:
    Root (/): 30380 (100%)
  Codes:
    200: 30380
```

El servicio no se ve afectado por grandes aumentos o "burst" en la cantidad de request, estimamos se debe a su rasgo asincrónico.

Múltiples (3) instancias Node.js

- Endpoint /heavy

  Dada la naturaleza del endpoint (bloqueante, simula una operación pesada en carga del cpu durante 5 segundos), se cambiaron los valores del escenario descritos en la sección 2.2 a los siguientes:

  name: Ramp
  duration: 30
  arrivalRate: 1
  rampTo: 2

  Resultados:

```
All virtual users finished
Summary report @ 16:48:36(-0300) 2021-10-20
  Scenarios launched:  45
  Scenarios completed: 8
  Requests completed:  8
  Mean response/sec: 1.11
  Response time (msec):
    min: 5008
    max: 8028
    median: 7015.5
    p95: 8028
    p99: 8028
  Scenario counts:
    Root (/): 45 (100%)
  Codes:
    200: 8
  Errors:
    ETIMEDOUT: 37
```

A diferencia del mismo caso con un único nodo/instancia de servidor node, el sistema puede procesar 3 requests en paralelo, vemos que logra atender sólo a 5 peticiones. Pese a este leve aumento de las peticiones atendidas, su escalabilidad sigue siendo baja, ya que igualmente los servicios siguen siendo sincrónicos. Una

concurrencia mayor a 3 requests cada 5 segundos ya generará un colapso del servicio.

- Endpoint /bbox/1 (servicio sincrónico)

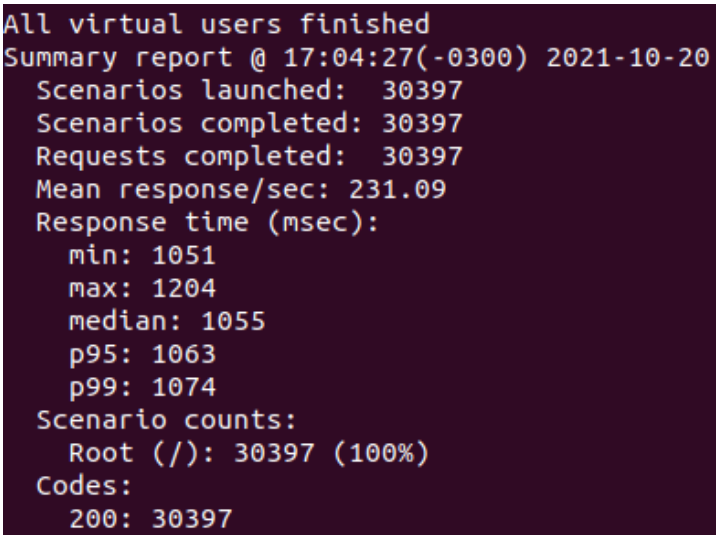
  Resultados:

```
All virtual users finished
Summary report @ 16:33:46(-0300) 2021-10-20
  Scenarios launched:  951
  Scenarios completed: 332
  Requests completed:  332
  Mean response/sec: 9.39
  Response time (msec):
    min: 608
    max: 9892
    median: 1474.5
    p95: 8736.6
    p99: 9801.1
  Scenario counts:
    Root (/): 951 (100%)
  Codes:
    200: 332
  Errors:
    ETIMEDOUT: 619
```

En este caso vemos que aumentar los nodos/instancias de los servidores Node no generó un aumento en la performance percibida por los usuarios, ya que el comportamiento y tratamiento de requests durante la rampa fue similar al caso de un único nodo/instancia, colapsando rápidamente. Más tarde concluiremos que, a diferencia de lo que creíamos, es un resultado lógico puesto que los servicios de bbox no escalaron junto a los servidores node, aún se conserva un único nodo de atención de peticiones.

- Endpoint /bbox/2 (servicio asincrónico)

  Resultados:



El servicio /bbox/2 no tiene problemas para escalar y poder absorber un número creciente de peticiones efectuadas, ya sea con un nodo o con varios nodos de servidores Node.

5. Conclusiones

- Inicialmente creíamos que escalando la cantidad de servidores Node iba a mejorar la performance del servicio sincrónico bbox/1. Pero resultó que no, y esto se debe a que el servicio bbox sólo es redirigido por los nodos de la app de Node, no se atiende allí, terminando los requests en un único punto de procesamiento que no estaba escalando.

- Hay que tener mucho cuidado con los servicios sincrónicos, pueden generar bloqueos en la atención de futuras requests, aumentando la espera o directamente devolviendo un error en la misma.

- Escalar los servidores de node no va a dar ningún improvement si el servicio depende de otro componente que no escala a la par, como en el caso de los bbox/1 y bbox/2, que dependían de un servidor en un único contenedor bbox **(cuello de botella)**.
  La escalabilidad de un sistema, muchas veces depende de la escalabilidad conjunta de cada subsistema o componente.

- Altos tiempos de respuesta (fuera de lo esperado) puede ser un indicativo de mucha congestión en el sistema e inclusive una alerta temprana para un próximo colapso.

- Para operaciones de procesamiento complejo (/heavy) en la mayoría de los casos es recomendable utilizar servicios asincrónicos para no perder disponibilidad.