Arquitectura de Software

75.73

Trabajo práctico

2º Cuatrimestre 2021

Apellido y nombre	**Padrón**	**E-mail**
Petrucci, Maximiliano	95872	mpetrucci@fi.uba.ar
Martin, Debora	90934	dmartin@fi.uba.ar

Introducción

El presente trabajo práctico consiste en la comparación y análisis de los distintos atributos de calidad en dos servidores HTTP bajo distintas condiciones para distintos tipos de tareas que pueda llevar a cabo un servidor.

Para llevar adelante estas tareas se utilizarán tecnologías auxiliares como Docker, Nginx, Grafana y cAdvisor, entre otras. Estas tecnologías nos ayudarán a simular distintos escenarios, así como también a recolectar diferentes datos necesarios para nuestro análisis.

Entonces, nuestro objetivo final será utilizar las distintas herramientas para poder comparar dos entornos diferentes, bajo las mismas condiciones, y analizar cómo varían los atributos de calidad en cada caso.

Servicios disponibles

Cada nodo tendrá los siguientes endpoints:

Tendremos 3 servicios a los cuales se podrán acceder:

- **/ping**: este servicio da una respuesta rápida y sin procesamiento. Representa un healthcheck.
- **/load**: este servicio hace una simulación de procesamiento y responde al finalizar
- **/sync**: este servicio consume otro (bbox provisto por la cátedra) que responde **sincrónicamente**
- **/async**: este servicio consume otro (bbox provisto por la cátedra) que responde **asincrónicamente**

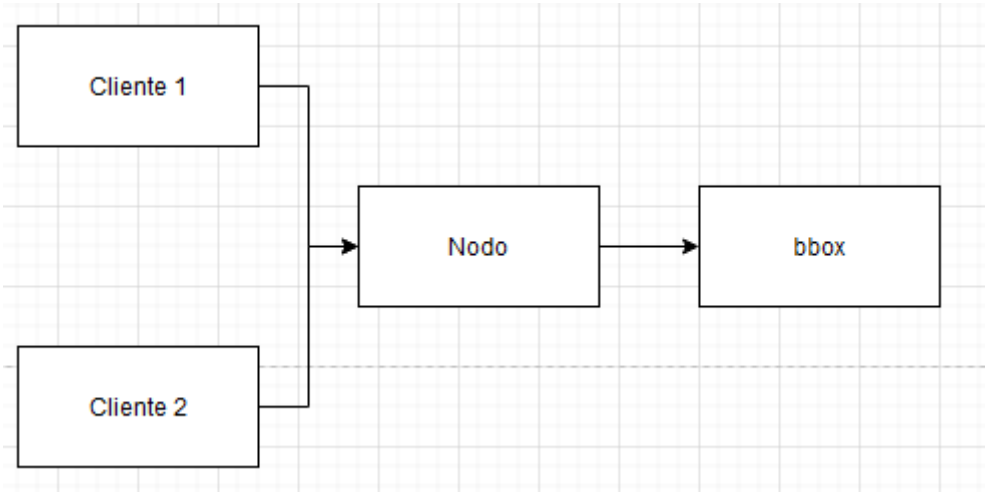
Además, se crearán dos configuraciones distintas

Configuración

Para realizar las pruebas, se plantearon distintas configuraciones de deployment. Por un lado, tenemos el deploy en un solo container; esto se replicará para ambos servidores. Por otro, mediante nginx se simulará un balanceador de carga aplicado sólo a uno de los servidores.

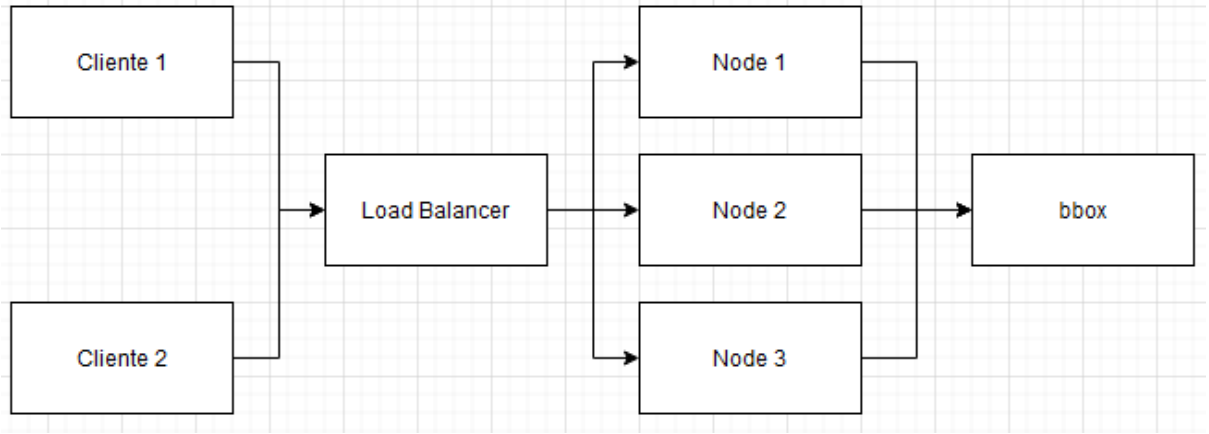
De esta forma, podremos comparar cómo se comportan las distintas tecnologías de los servidores si estuviesen desplegados en un único nodo cada uno. Además podemos determinar qué ventajas y desventajas tiene un balanceador de carga (varios nodos) contra un único container desplegado para un mismo servidor.

Para los casos de nodo único, la configuración es la siguiente:



Este nodo, por sí solo, puede resolver los escenarios de **/ping** y **/load**. Para los endpoints bbox1 y bbox2 consume al servicio bbox.

El esquema de 3 nodos replicados es el siguiente:



Cada cliente interactúa con un balanceador de carga que se encarga de distribuir las solicitudes en los distintos nodos replicados. Finalmente, consumen un servicio bbox (único y no replicado)

Escenarios

A continuación se plantean los distintos escenarios a los cuales serán sometidas nuestras aplicaciones. Por cada uno analizaremos distintas propiedades que creamos interesantes, tales como los diferentes tiempos de ejecución (promedio, máximo, mediana), la relación entre pedidos finalizados contra pedidos totales realizados en determinado tiempo, etc.

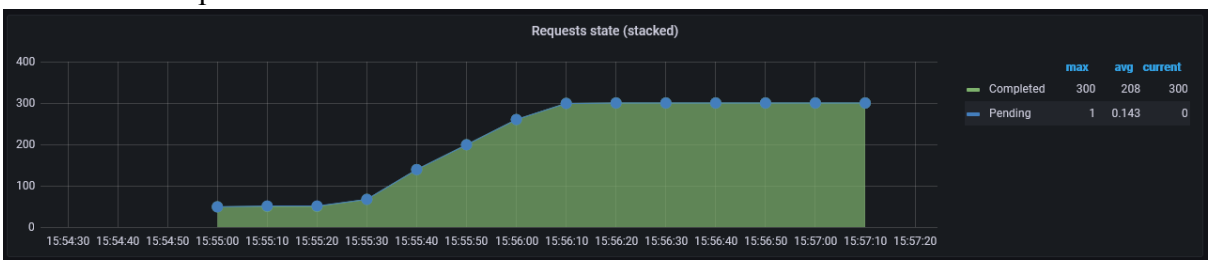
Escenario 1 - ping

El primer escenario que analizaremos se basa en realizar múltiples pedidos al servicio ping de nuestra aplicación. Al ser una operación rápida, esperamos que en todos los casos los tiempos se mantengan constantes, en el orden de los microsegundos.
Veamos entonces, los resultados obtenidos en las pruebas.

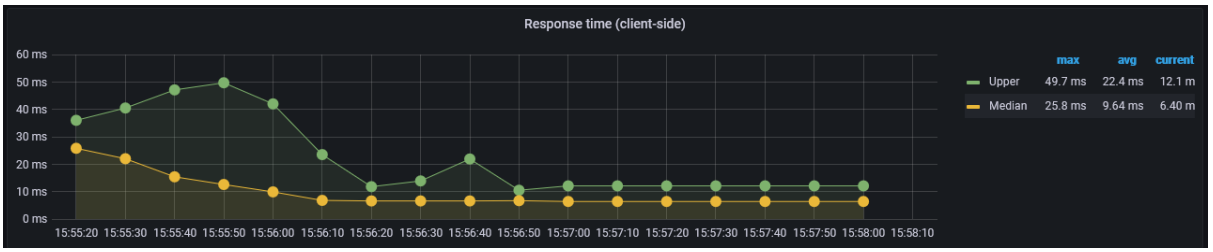
Para todos los casos utilizaremos las mismas fases de requests a los servidores. Inicialmente, por 30 segundos, se realizarán 5 requests por segundo en forma constante. Luego, por 120 segundos, se incrementará en forma lineal la cantidad de requests por segundo de 5 a 50. Finalmente, se mantendrá una etapa de carga alta por 60 segundos, en la que se realizarán 30 requests por segundo.
Este mismo análisis se intentó hacer con una rampa de 5 a 50 y luego la etapa de carga alta con 50 pero por consola el Artillery indicaba un alto uso de la CPU y según lo explicado en clase y leído en el "manual" de este programa dice que esto puede generar una degradación de la información obtenida lo que puede verse representado como errores en la medición por lo que preferimos bajar la cantidad de request y tener información más fiable.

Un nodo

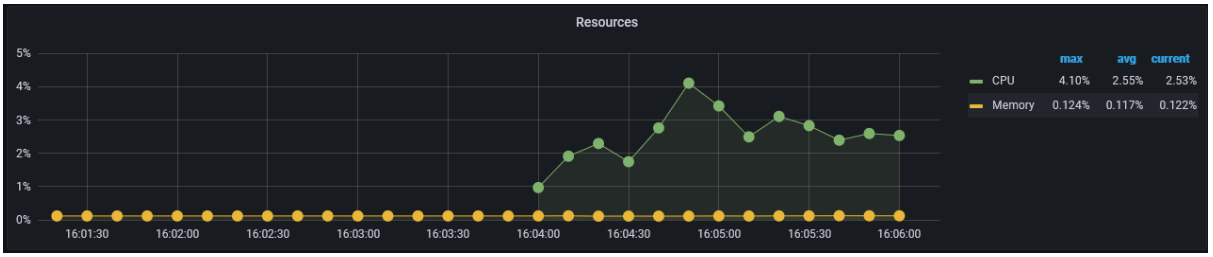
Arranquemos analizando el comportamiento por parte de los servidores en Node.
Vamos a mostrar el estado de los request, el tiempo de respuesta al request y el impacto en la CPU.
Comencemos con el estado de los request. Nuestra hipótesis para este caso y en los 2 casos siguientes es que no se va a generar ningún problema significante ya que el request es sumamente simple.



Como era de esperarse, todos los request fueron completados con éxito y no quedó ninguno pendiente.
Pasemos a la parte del tiempo de respuesta. Esperamos no ver un tiempo de respuesta mayor a los 50ms sino algo estaría mal.



Obviando el inicio del escenario, el máximo tiempo de respuesta está entre los 50 ms. Lo llamativo es el comienzo que se puede asociar a la inicialización del escenario, momento en el cual el servidor pide recursos a la máquina. Cuando comienza la rampa, lógicamente empieza a aumentar el tiempo de respuesta. También notamos que cuando cambia de fase se genera un pico (15:56:40). Esto puede ser por el procesamiento que debe realizar artillery para cambiar de fase. Luego en la etapa de request constantes el tiempo baja y la cosa se normaliza.
Por último analicemos el uso de CPU.



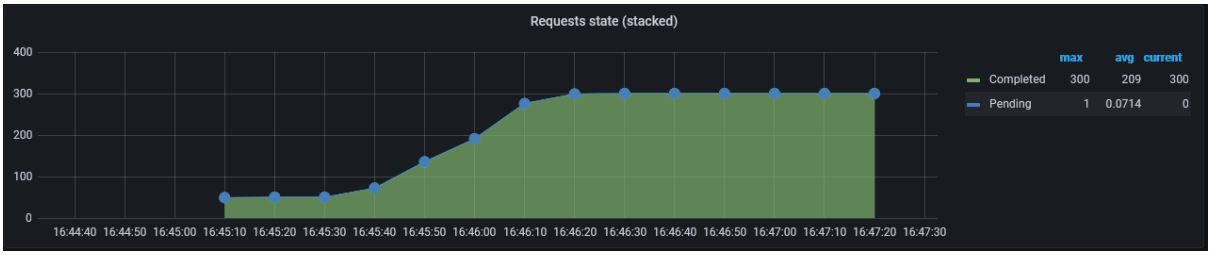
Se muestra lo esperable que es un aumento en el uso del CPU cuando comienza la rampa y un descenso cuando se estabiliza la cantidad de requests.
Este es el resultado que mostró Artillery por consola:

```
All virtual users finished
Summary report @ 16:06:10(-0300) 2021-10-21
  Scenarios launched:  2662
  Scenarios completed: 2662
  Requests completed:  2662
  Mean response/sec: 20.31
  Response time (msec):
    min: 2
    max: 25
    median: 3
    p95: 11
    p99: 14
  Scenario counts:
    Ping: 2662 (100%)
  Codes:
    200: 2662
```

Se pudieron enviar todos los requests generados sin problemas. El tiempo máximo fue menor al que nos mostró Grafana, eso puede haber sido el agregado de la comunicación entre los containers (Graphite, Grafana, Artillery). El 99% de los request se realizó en 14 ms o menos lo cual es aceptable.

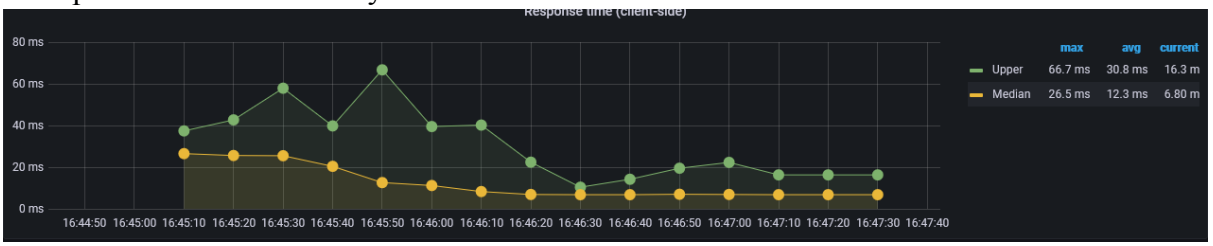
3 nodos replicados

Cambiemos un poco y probemos con 3 servidores de Node corriendo al mismo tiempo con un load balancer.
Comencemos con el estado de los request. En esta parte vamos a mostrar solamente 1 de los 3 gráficos ya que eran idénticos y no aportaba nada al análisis.
Al igual que con un solo servidor todos los request deberían completarse sin problemas mientras todo funcione correctamente.



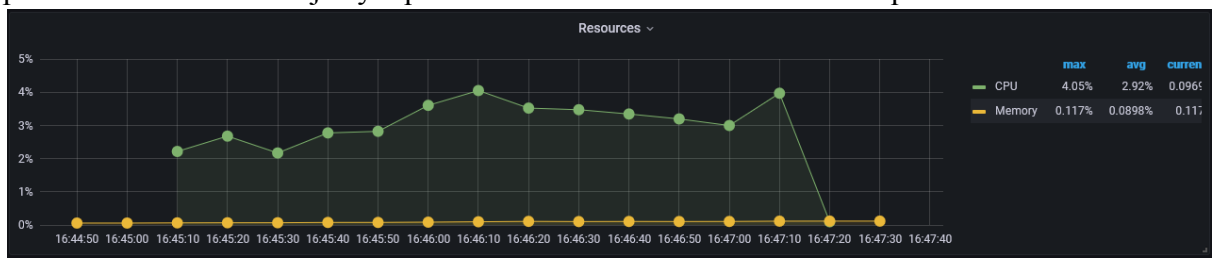
Como se puede ver, todos se completaron correctamente.

Si miramos el tiempo de respuesta esperaríamos lo mismo que para el caso anterior.
No esperamos resultados muy distintos a los obtenidos con 1 servidor corriendo.



Son resultados parecidos pero con diferencias: los tiempos de respuesta incrementaron un poco ya que esta vez pasan los 60ms. Esto debe ser producto de estar corriendo 3 servidores que se reparten los request a estar corriendo 1 solo

Por último veamos el uso de CPU. Vemos que no supera el 4% del uso como el caso anterior. Esperamos que el comportamiento sea parecido al obtenido cuando había un solo servidor pero con valores más bajos ya que ahora se estaban dividiendo los requests entre 3.



Efectivamente, se comportan de la misma manera. Un aumento del uso cuando comienza la rampa y se disminuye cuando todo se estabiliza.

Este es el resultado que mostró Artillery por consola:

```
All virtual users finished
Summary report @ 16:47:15(-0300) 2021-10-21
  Scenarios launched:  2645
  Scenarios completed: 2645
  Requests completed:  2645
  Mean response/sec: 20.18
  Response time (msec):
    min: 1
    max: 28
    median: 4
    p95: 11
    p99: 16
  Scenario counts:
    Ping: 2645 (100%)
  Codes:
    200: 2645
```

Los resultados son muy parecidos a los obtenidos con 1 solo servidor.

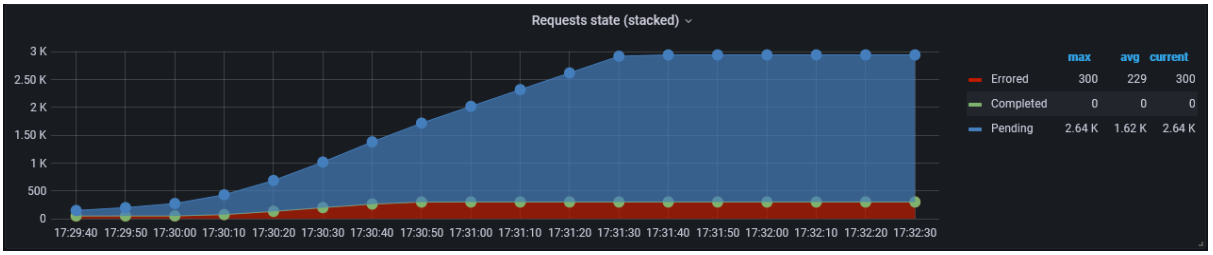
Servicio 2 - load

En este segundo escenario analizaremos el alto procesamiento. Esperaremos encontrar errores dado que el procesador está ocupado resolviendo una tarea y no puede atender la otra que llega.

Para ambos casos utilizaremos las mismas fases de requests a los servidores. Inicialmente, por 30 segundos, se realizarán 5 requests por segundo en forma constante. Luego, por 40 segundos, se incrementará en forma lineal la cantidad de requests por segundo de 5 a 30. Finalmente, se mantendrá una etapa de carga alta por 60 segundos, en la que se realizarán 30 requests por segundo.

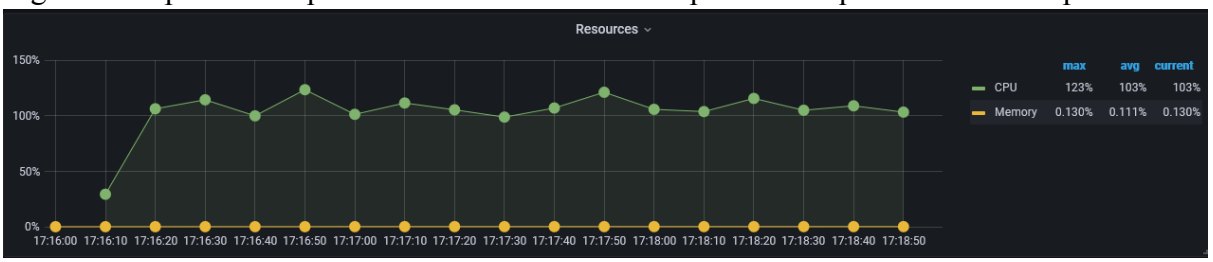
Un nodo

A continuación se muestra una gráfica de la cantidad de requests en función del tiempo pedidas al servidor:

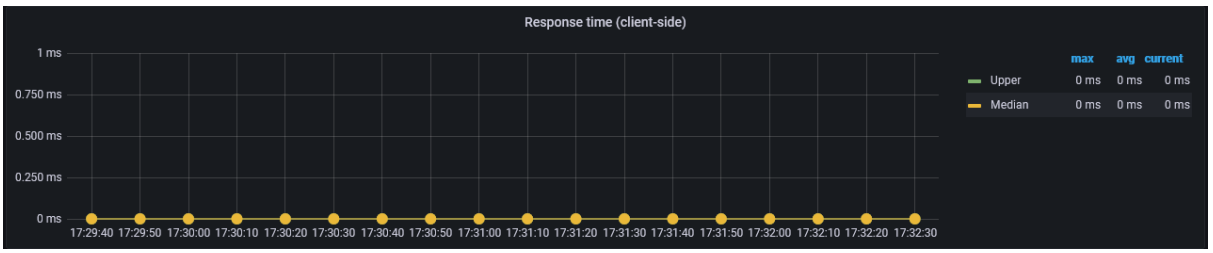


Se puede ver que hay requests que fallan. Como dijimos anteriormente, esto se debe a que el procesador está ocupado resolviendo las primeras requests y no puede atender las siguientes.

Si analizamos el consumo del CPU, el cual debería ser alto en todo momento, vemos efectivamente que se cumple la predicción. Se mantiene en un valor casi constante dado que desde el principio hasta el final el servidor está resolviendo todos los pedidos que le fueron llegando. Se puede ver que al comienzo no estaba ocupado dado que no tenía nada para hacer.



Por último, el response time se ve a 0. Sinceramente, no le encontramos respuesta a que el response time sea 0 dado que es alto porque las requests fallan (¿Grafana cuenta como 0 los NaN que aparecían en consola?)

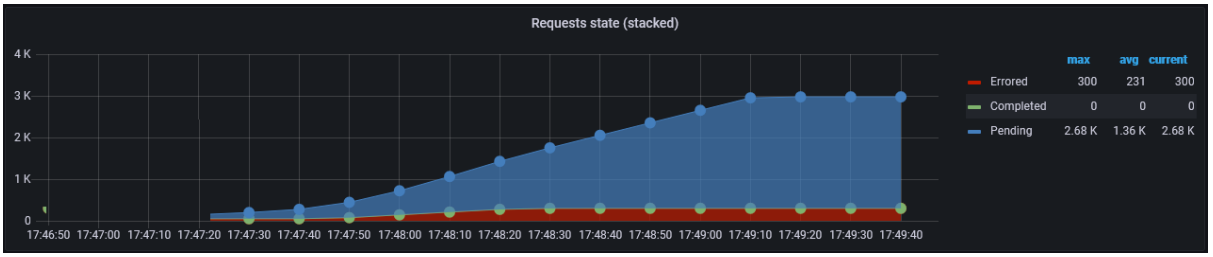


Este es el resultado en consola:

```
All virtual users finished
Summary report @ 17:31:42(-0300) 2021-10-21
  Scenarios launched:  2639
  Scenarios completed: 0
  Requests completed:  0
  Mean response/sec: 18.71
  Response time (msec):
    min: NaN
    max: NaN
    median: NaN
    p95: NaN
    p99: NaN
  Scenario counts:
    Load: 2639 (100%)
  Errors:
    ETIMEDOUT: 2639
```

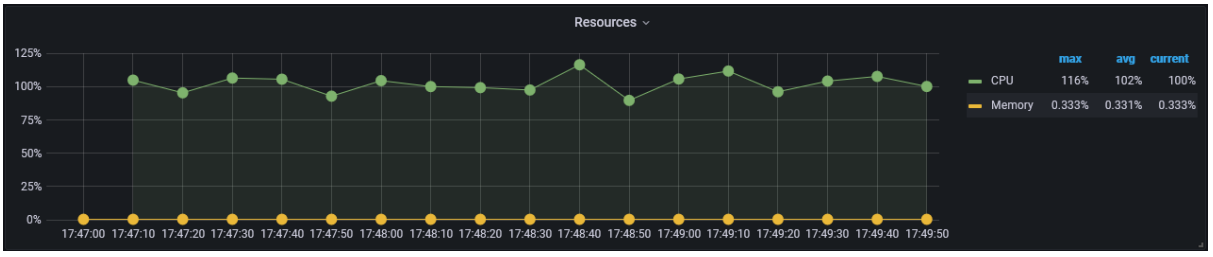
3 nodos replicados

Pasemos a analizar la segunda configuración: nginx con 1 load balancer y 3 nodos atrás repartiéndose las requests.



Se encuentran errores en la configuración del servidor replicado y atendiendo procesos en paralelo. Esto se debe a que el servicio que estamos probando es sincrónico, por lo que el procesador no puede resolver peticiones en paralelo. Esto demuestra que nuestro servicio no es escalable porque si bien con más nodos lograremos aceptar más peticiones, nunca llegaremos a responderlas.

En cuanto al uso del CPU, no esperamos ver diferencias con el esquema de un solo nodo dado que el procesador que resuelve las peticiones es el mismo.



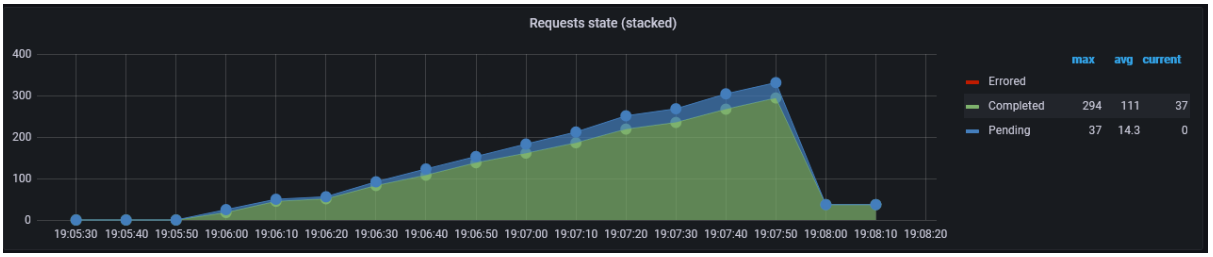
Estos fueron los resultados que mostró la consola:

```
All virtual users finished
Summary report @ 17:49:16(-0300) 2021-10-21
  Scenarios launched:  2675
  Scenarios completed: 0
  Requests completed:  0
  Mean response/sec: 18.96
  Response time (msec):
    min: NaN
    max: NaN
    median: NaN
    p95: NaN
    p99: NaN
  Scenario counts:
    Load: 2675 (100%)
  Errors:
    ETIMEDOUT: 2675
```

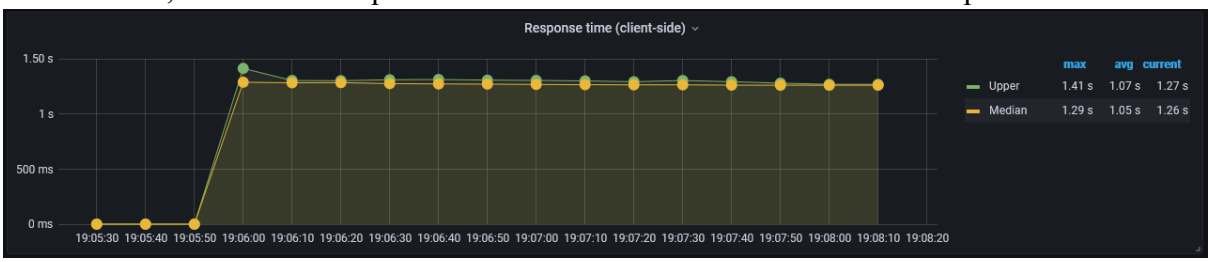
Escenario 3 - bbox1

En este escenario analizaremos cómo se comporta el endpoint de bbox, que consume a la aplicación bbox (provista por la cátedra). En base a los resultados analizaremos si este endpoint es asincrónico o no comparándolo con bbox.
La configuración del escenario será una simple rampa de 120 segundos (2 minutos) en los que arribará 1 request por segundo al principio y terminará con 30 requests al terminar el proceso.

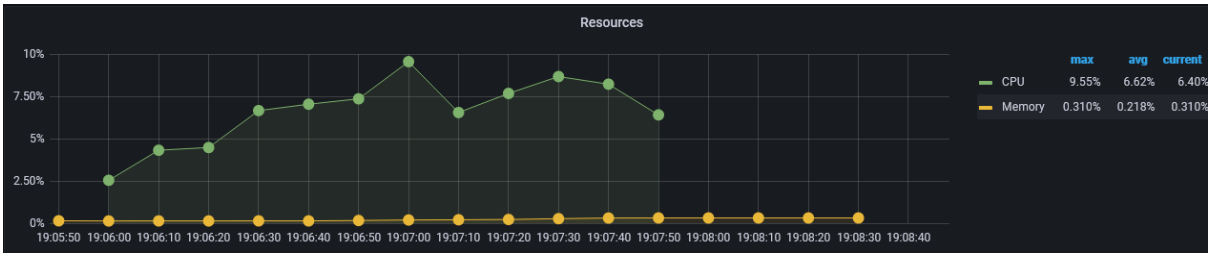
Un nodo



Vemos que las requests van subiendo y a medida que llegan requests van quedando algunas pendientes. Esto indica que el servicio presenta una demora por lo que no es escalable.
Por otro lado, si vemos el response time del mismo nos encontraremos con problemas:



El response time se mantiene constante. El servicio no demora al responder las requests por lo que pareciera que las peticiones anteriores no generan ninguna demora en las siguientes.
Por último, si miramos el uso del CPU podremos tener más información sobre lo que está pasando detrás de este servicio:

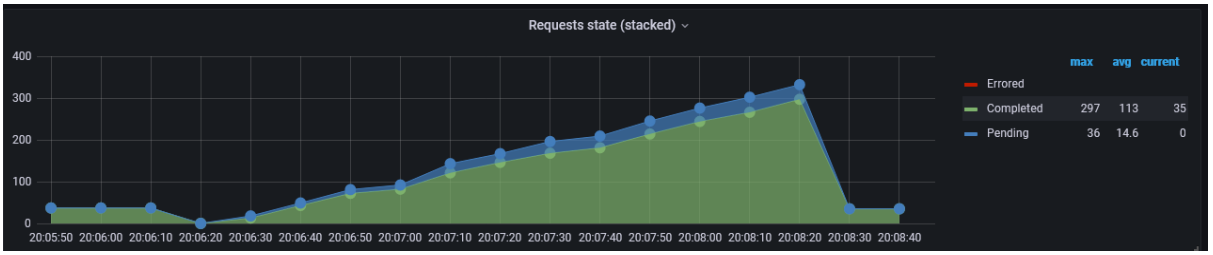


Se genera un consumo en el servicio pero no es tan considerable respecto al ping.

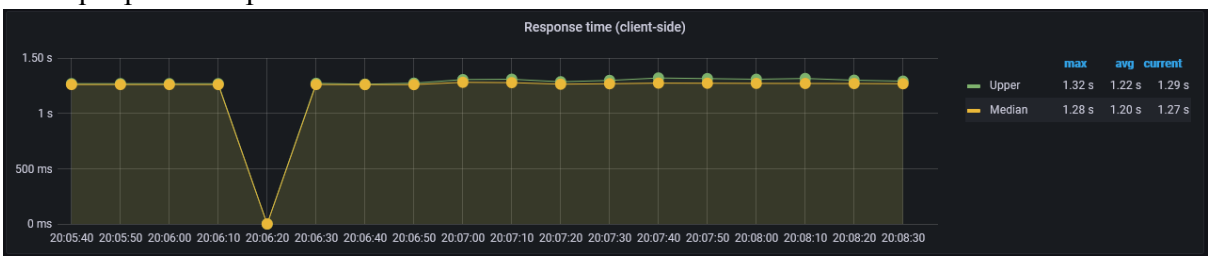
Antes de pasar al otro servicio, veamos que pasa con este en un esquema de 3 nodos detrás de un load balancer:

<u>3 nodos replicados</u>

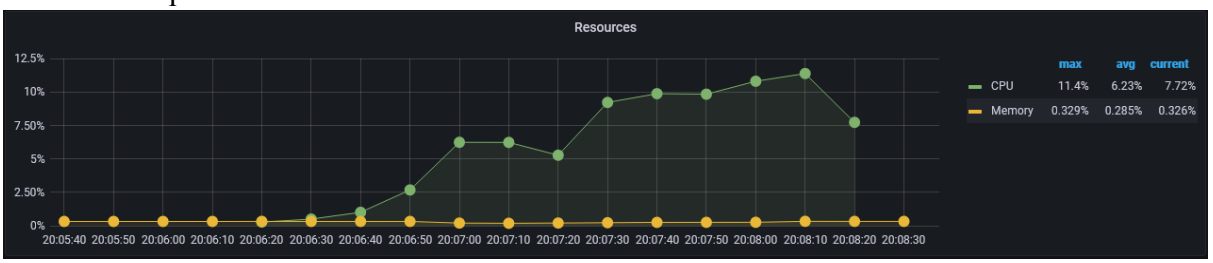
Esperamos que las requests respondan del mismo modo que con la configuración anterior dado que tenemos un nodo recibiendo las peticiones. Podemos notar acá también que a medida que crecen las requests crece la cantidad de peticiones pendientes.



En cuanto al tiempo de respuesta, si bien tenemos 3 nodos en lugar de hay que recordar que todos los nodos estarán consultando a un único nodo bbox por lo que al final todo dependerá de lo que pueda responder el nodo bbox:



Vemos que se mantiene constante salvo un pico invertido (¿podrá ser un error?).
En cuanto a los recursos de la máquina, esperamos ver algo similar, creciendo a medida que crecen los requests.



Esta es la respuesta de la consola:

```
All virtual users finished
Summary report @ 20:08:33(-0300) 2021-10-21
  Scenarios launched:  1882
  Scenarios completed: 1882
  Requests completed:  1882
  Mean response/sec: 14.44
  Response time (msec):
    min: 1254
    max: 1297
    median: 1261
    p95: 1274
    p99: 1281
  Scenario counts:
    Bbox: 1882 (100%)
  Codes:
    200: 1882
```

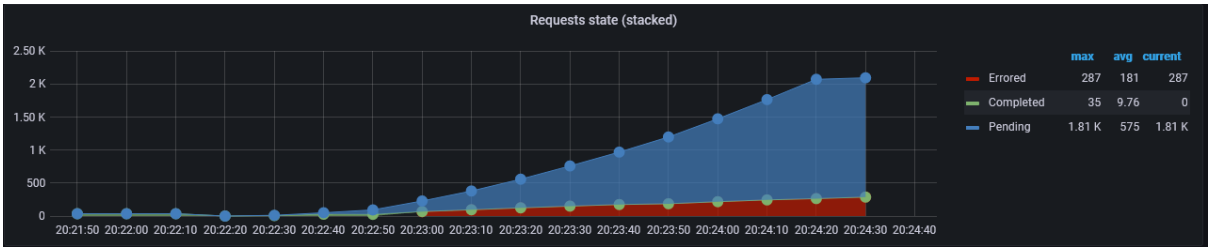
Dado que este servicio no presenta errores ni se ve una modificación en el response time, tiene grandes probabilidades de ser el servicio asincrónico. Veremos qué pasa con el otro servicio.

Escenario 4 - bbox2

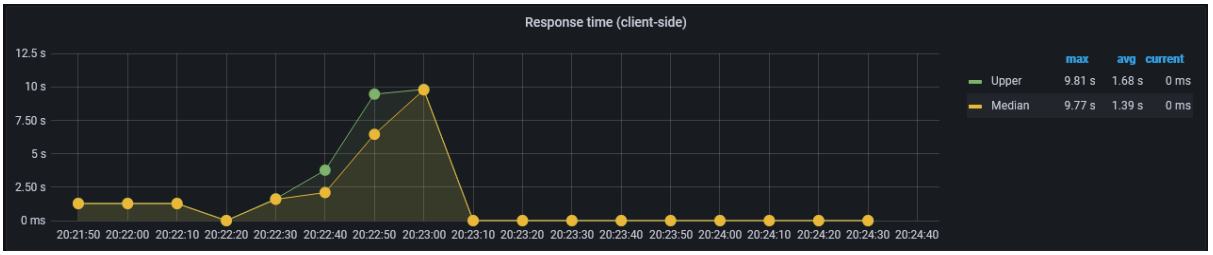
Probaremos la misma configuración anterior pero con este servicio para lograr detectar cual es el sincrónico y cual el asincrónico:

<u>Un nodo</u>

Acá ya vemos un resultado que nos puede decir mucho: a medida que saturamos el servicio hay requests que quedan con errores y el número de requests pendientes son altas.

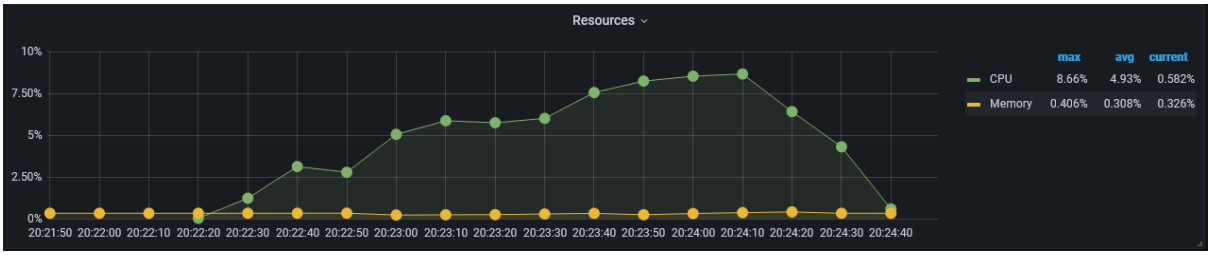


Para los response time esperamos ver un pico de saturación dado que después de una cierta cantidad de solicitudes el servicio falla.



Efectivamente es lo que esperábamos ver. Se puede ver que al 20:23:00 el servicio tiene un pico y luego cae dejando sin respuesta al resto.

Finalmente en cuanto al consumo de CPU creemos que habrá un consumo regulado y creciente:

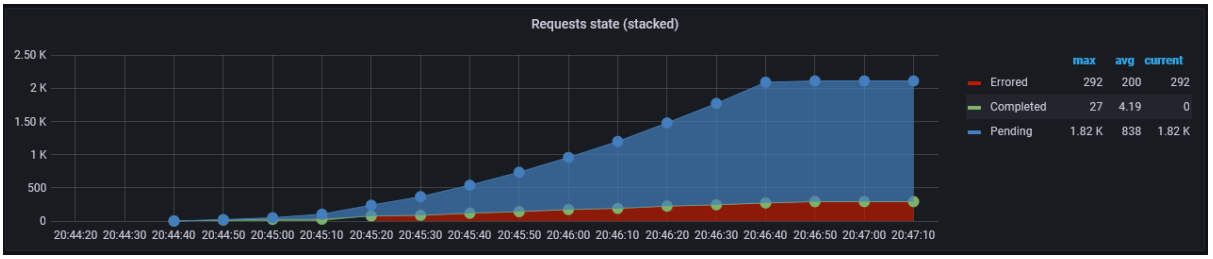


Los resultados de la consola fueron los siguientes:

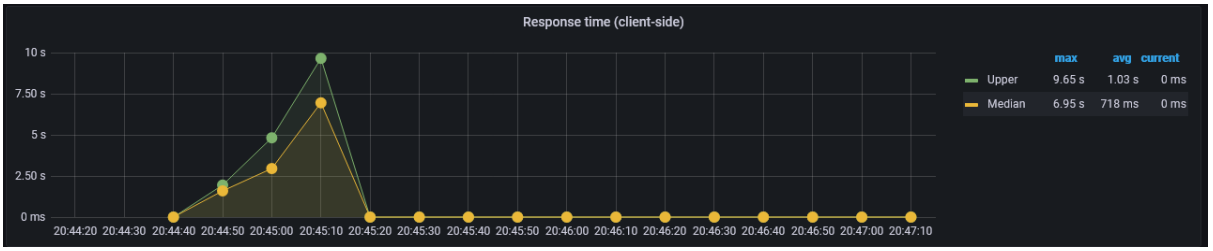
```
All virtual users finished
Summary report @ 20:24:33(-0300) 2021-10-21
  Scenarios launched:  1867
  Scenarios completed: 61
  Requests completed:  61
  Mean response/sec: 14.65
  Response time (msec):
    min: 1562
    max: 9793
    median: 3536
    p95: 9580.9
    p99: 9788.9
  Scenario counts:
    Bbox: 1867 (100%)
  Codes:
    200: 61
  Errors:
    ETIMEDOUT: 1806
```

<u>3 nodos replicados</u>

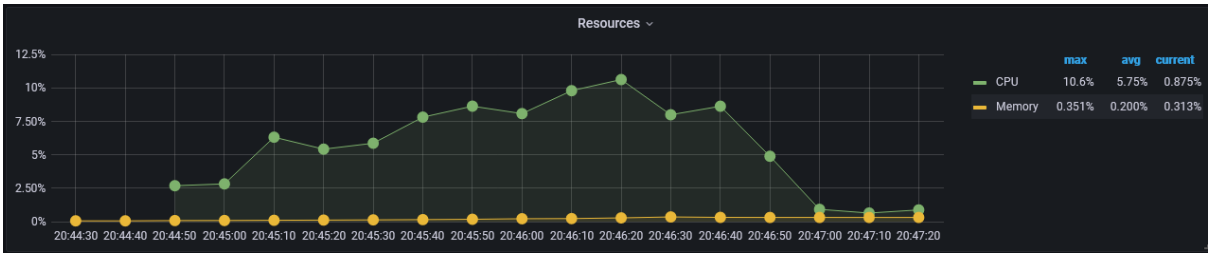
Para esta prueba esperamos ver algo similar a lo anterior porque quien “realiza el cálculo” es bbox y no cada uno de los nodos que replicamos.



Como preveíamos, se da similar al caso anterior.
Luego continúa todo muy similar al caso anterior:



Se vuelve a repetir el pico y la saturación del servicio y en el consumo del CPU también hay algo similar:



Por último tenemos los datos que devolvió la consola:

```
All virtual users finished
Summary report @ 20:46:54(-0300) 2021-10-21
  Scenarios launched:  1884
  Scenarios completed: 67
  Requests completed:  67
  Mean response/sec: 14.44
  Response time (msec):
    min: 1567
    max: 9635
    median: 3570
    p95: 9438.7
    p99: 9633.8
  Scenario counts:
    Bbox: 1884 (100%)
  Codes:
    200: 67
  Errors:
    ETIMEDOUT: 1817
```

Conclusiones

El análisis de atributos de calidad no es una ciencia exacta. Hay que analizar cada caso en particular, necesitando comparar qué se pierde y qué se gana al realizar determinado cambio: desde cambiar las tecnologías o realizar un refactor del código, hasta modificar la infraestructura agregando nodos para un balanceador de carga.

Como pudimos ver, no siempre agregar nodos para procesar en paralelo nos puede mejorar la performance de un servicio ya que depende de cómo está hecho el servicio en sí y, por supuesto, también influye si se consume un servicio tercero (caso del bbox).